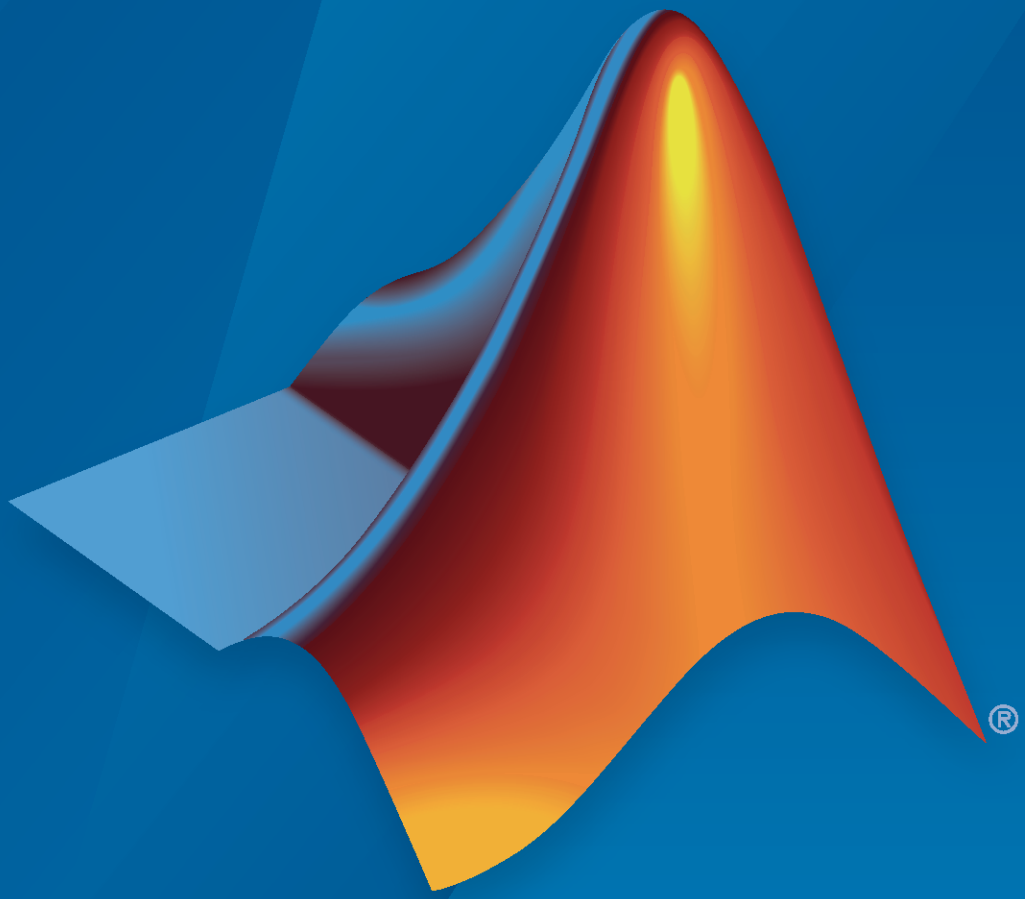
Computer Vision Toolbox™

Reference



MATLAB® & SIMULINK®

R2026a

MathWorks®

How to Contact MathWorks



Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`



Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Computer Vision Toolbox™ Reference

© COPYRIGHT 2011–2026 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

April 2011	Online only	Revised for Version 4.0 (Release 2011a)
September 2011	Online only	Revised for Version 4.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.0 (Release 2012a)
September 2012	Online only	Revised for Version 5.1 (Release R2012b)
March 2013	Online only	Revised for Version 5.2 (Release R2013a)
September 2013	Online only	Revised for Version 5.3 (Release R2013b)
March 2014	Online only	Revised for Version 6.0 (Release R2014a)
October 2014	Online only	Revised for Version 6.1 (Release R2014b)
March 2015	Online only	Revised for Version 6.2 (Release R2015a)
September 2015	Online only	Revised for Version 7.0 (Release R2015b)
March 2016	Online only	Revised for Version 7.1 (Release R2016a)
September 2016	Online only	Revised for Version 7.2 (Release R2016b)
March 2017	Online only	Revised for Version 7.3 (Release R2017a)
September 2017	Online only	Revised for Version 8.0 (Release R2017b)
March 2018	Online only	Revised for Version 8.1 (Release R2018a)
September 2018	Online only	Revised for Version 8.2 (Release R2018b)
March 2019	Online only	Revised for Version 9.0 (Release R2019a)
September 2019	Online only	Revised for Version 9.1 (Release R2019b)
March 2020	Online only	Revised for Version 9.2 (Release R2020a)
September 2020	Online only	Revised for Version 9.3 (Release R2020b)
March 2021	Online only	Revised for Version 10.0 (Release R2021a)
September 2021	Online only	Revised for Version 10.1 (Release R2021b)
March 2022	Online only	Revised for Version 10.2 (Release R2022a)
September 2022	Online only	Revised for Version 10.3 (Release R2022b)
March 2023	Online only	Revised for Version 10.4 (Release R2023a)
September 2023	Online only	Revised for Version 23.2 (R2023b)
March 2024	Online only	Revised for Version 24.1 (R2024a)
September 2024	Online only	Revised for Version 24.2 (R2024b)
March 2025	Online only	Revised for Version 25.1 (R2025a)
September 2025	Online only	Revised for Version 25.2 (R2025b)
March 2026	Online only	Revised for Version 26.1 (R2026a)

Contents

1 Blocks

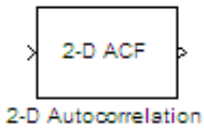
2 Objects

3 Functions

Blocks

2-D Autocorrelation

2-D autocorrelation of input matrix



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Autocorrelation block computes the 2-D autocorrelation of the input vector or matrix.

Ports

Input

Port_1 — Input array
vector | matrix

Input array, specified as a vector or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Output

Port_1 — Autocorrelation of input
vector | matrix

Autocorrelation of the input array, returned as a vector or matrix. The data type of the output is the same as that of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Parameters

For details on fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Algorithms

If the input $A$is a vector with dimension M-by-1 then the equation for 1-D discrete autocorrelation is:

where $0 \le n \le 2M - 1$.

The output is an autocorrelation vector of size $(2M - 1, 1)$.

If the input A is a matrix with dimension M-by-N then the equation for the 2-D discrete autocorrelation is:

where $0 \le i < 2M - 1$ and $0 \le j < 2N - 1$.

The dimension of the output autocorrelation matrix is $(2M - 1, 2N - 1)$.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

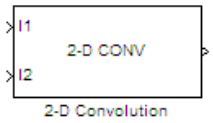
See Also

Blocks

2-D Correlation | 2-D Histogram | 2-D Mean | 2-D Median | 2-D Standard Deviation | 2-D Variance | 2-D Maximum | 2-D Minimum

2-D Convolution

Compute 2-D discrete convolution of two input matrices



Libraries:
Computer Vision Toolbox / Filtering

Description

The 2-D Convolution block computes the two-dimensional convolution of two input matrices. Assume that matrix A has dimensions (*Ma*, *Na*) and matrix B has dimensions (*Mb*, *Nb*). When the block calculates the full output size, the equation for the 2-D discrete convolution is:

$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) * B(i-m, j-n)$$

where $0 \le i < Ma + Mb - 1$ and $0 \le j < Na + Nb - 1$.

Ports

Input

I1 — Input matrix
matrix

Input matrix, specified as either a matrix of intensity values or a matrix that represents one plane of an RGB video stream.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

I2 — Input matrix
matrix

Input matrix, specified as either a matrix of intensity values or a matrix that represents one plane of an RGB video stream.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

Output — Convolution
matrix

Convolution of the input matrices, returned as a matrix.

Dependencies

- The dimensions of the output are dictated by the **Output size** parameter.
- If the data type of the input is floating point, the output of the block is also floating point.
- If `all(size(I1)<size(I2))`, the block returns an error.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main Tab

Output size — Output size
matrix

Dimensions of the output.

The table describes the block output, given the following input dimensions:

- **I1** — (*Ma*, *Na*)
- **I2** — (*Mb*, *Nb*)

Output size	Output	Output Dimension
`Full`	Full two-dimensional convolution	(*Ma*+*Mb*-1, *Na*+*Nb*-1).
`Same as input port I1`	Central part of the convolution with the same dimensions as the input at port **I1**	
`Valid`	Only the parts of the convolution that are computed without the zero-padded edges of any input.	(*Ma*-*Mb*+1, *Na*-*Nb*+1)

Normalized output — Normalized output
matrix

Normalize output by dividing the output by `sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2)))`, where `I1p` is the portion of the **I1** matrix that aligns with the **I2** matrix.

Note When you select the **Normalized output** check box, the block input cannot be fixed point.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Algorithms

2-D Convolution

In convolution, the value of an output element is computed as a weighted sum of neighboring elements.

For example, suppose the first input matrix represents an image and is defined as:

```
I1 = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
     11  18  25   2   9]
```

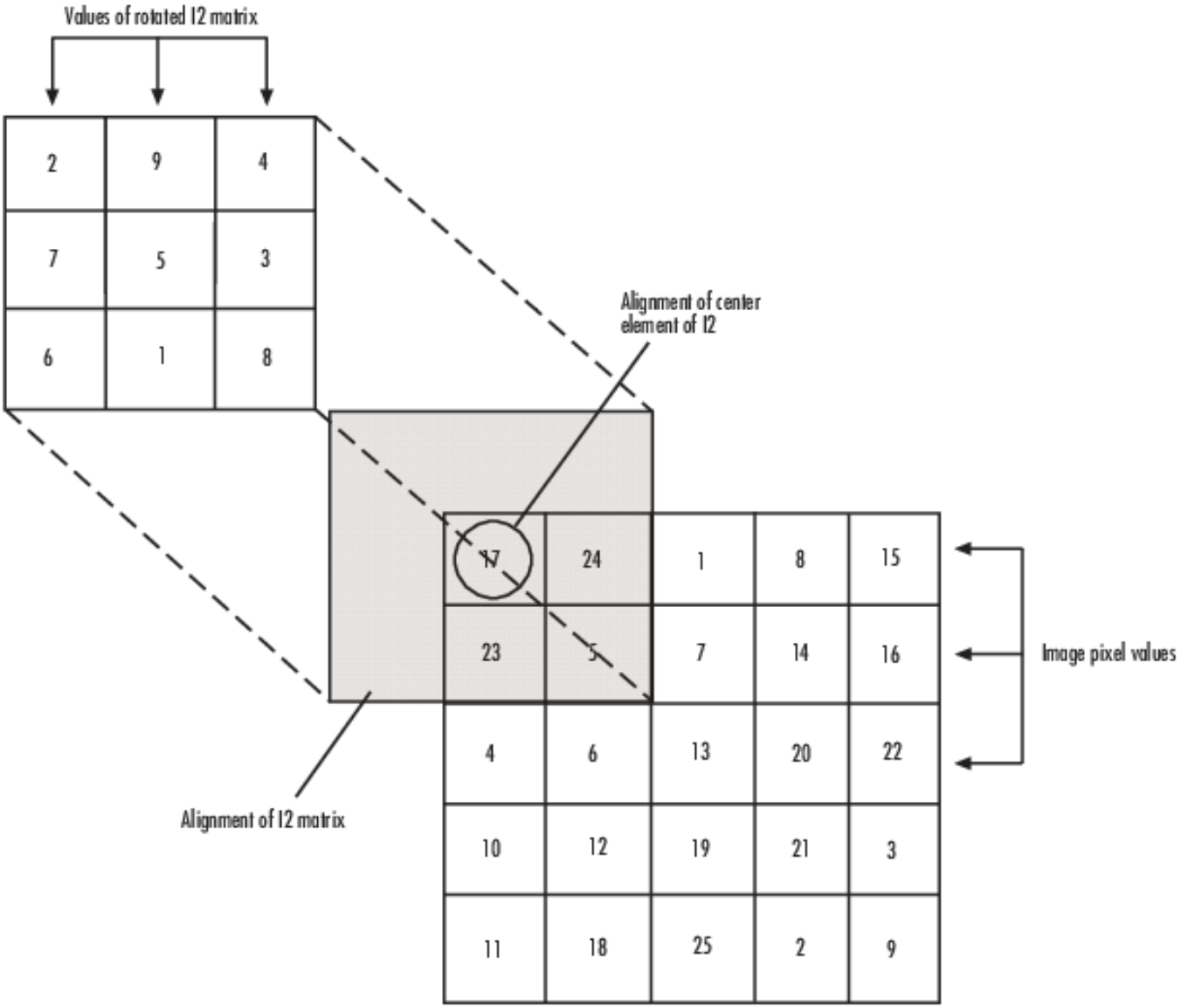
The second input matrix also represents an image and is defined as:

```
I2 = [8   1   6
     3   5   7
     4   9   2]
```

The following figure shows how to compute the (1,1) output element by using these steps:

1. Rotate the second input matrix, **I2**, 180 degrees around its center element.
2. Slide the center element of **I2** so that it lies on top of the (0,0) element of **I1**.
3. Multiply each element of the rotated **I2** matrix by the element of **I1** underneath.
4. Sum the individual products from step 3.

The (1,1) output element is $0 \cdot 2 + 0 \cdot 9 + 0 \cdot 4 + 0 \cdot 7 + 17 \cdot 5 + 24 \cdot 3 + 0 \cdot 6 + 23 \cdot 1 + 5 \cdot 8 = 220$.



Computing the (1,1) Output of Convolution

The normalized convolution of the (1,1) output element is `220/sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2)))` = 0.3459, where `I1p = [0 0 0; 0 17 24; 0 23 5]`.

Output Sizes

The equations below describe how the block computes the convolution depending on how you set the **Output size** parameter. For the cases below, the inputs are set as:

- **I1** — dimensions (4,3)
- **I2** — dimensions (2,2)
- When **Output size** is set to `Full`, the block uses the following equations:

$$C_{\text{full}_{\text{rows}}} = I1_{\text{rows}} + I2_{\text{rows}} - 1 = 5$$

$$C_{\text{full}_{\text{columns}}} = I1_{\text{columns}} + I2_{\text{columns}} - 1 = 4$$

The resulting matrix is:

$$C_{\text{full}} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \\ c_{40} & c_{41} & c_{42} & c_{43} \end{bmatrix}$$

- When **Output size** is set to `Same as input port I1`, the output is the central part of *Cfull* with the same dimensions as the input at port **I1**, (4,3). However, since a 4-by-3 matrix cannot be extracted from the exact center of *Cfull*, the block leaves more rows and columns on the top and left side of the *Cfull* matrix and outputs:

$$C_{\text{same}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \\ c_{41} & c_{42} & c_{43} \end{bmatrix}$$

- When the **Output size** is set to `Valid`, the block uses the following equations to determine the number of rows and columns of the output matrix:

$$C_{\text{valid}_{\text{rows}}} = I1_{\text{rows}} - I2_{\text{rows}} + 1 = 3$$

$$C_{\text{valid}_{\text{columns}}} = I1_{\text{columns}} - I2_{\text{columns}} + 1 = 2$$

In this case, it is always possible to extract the exact center of *Cfull*. Therefore, the block outputs:

$$C_{\text{full}} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

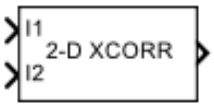
When you select the **Normalized output** check box, the block input cannot be fixed point.

See Also

2-D FIR Filter

2-D Correlation

Compute 2-D correlation of two input matrices



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Correlation block computes the two-dimensional cross-correlation between two input matrices.

Ports

Input

I1 — First input matrix
matrix

First input matrix, specified as an *M*-by-*N* matrix. The two input matrices must have data types in the same category such as `double`, `single`, unsigned integer (`uint8`,`uint16`,`uint32`,`uint64`), or integer (`int8`,`int16`,`int32`,`int64`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

I2 — Second input matrix
matrix

Second input matrix, specified as a *P*-by-*Q* matrix. The two input matrices must have data types in the same category such as `double`, `single`, unsigned integer (`uint8`,`uint16`,`uint32`,`uint64`), or integer (`int8`,`int16`,`int32`,`int64`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

Output

Port_1 — Calculated cross-correlation
scalar | vector | matrix

Calculated cross-correlation, returned as a scalar, vector, or matrix. The size of the cross-correlation output depends on the **Output size** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

Parameters

Main

Output size — Size of cross-correlation output

`Full` (default) | `Same as input port I1` | `Valid`

Size of the cross-correlation output.

- If you choose `Full`, the output has the dimensions (M+P-1)-by-(N+Q-1).
- If you choose `Same as input port I1`, the output has the same dimensions as the input at port **I1**. The block returns the central part of cross-correlation matrix, which is the same size as the input at port **I1**.
- If you choose `Valid`, the output has the dimensions (M-P+1)-by-(N-Q+1). The block returns only parts of the cross-correlation matrix that are computed without zero-padded edges.

Normalized output — Normalize cross-correlation output

`off` (default) | `on`

Select this parameter, if the data types of the inputs are floating-point. The values of the cross-correlation output are normalized to [0, 1].

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Algorithms

Given two input matrices, *I1* and *I2*, that are size M-by-N and P-by-Q, the 2-D cross-correlation value at the point `(k,l)` is given by

$$C(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I1(m,n)\overline{I2}(m+k,n+l).$$

The normalized cross-correlation value at the point `(k,l)` is calculated as

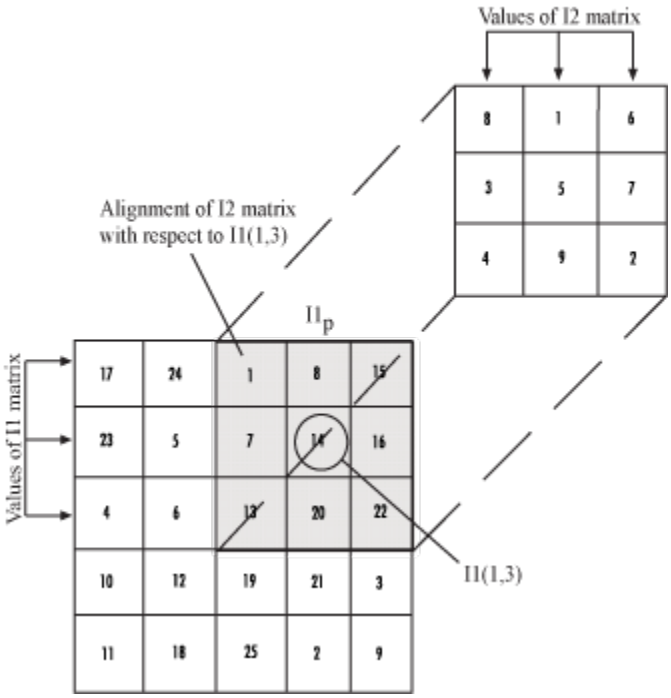
$$C_N(k,l) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I1(m,n)\overline{I2}(m+k,n+l)}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I1(m,n)^2}\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \overline{I2}(m+k,n+l)^2}},$$

where,

$0 \le k < M+P-1$

$0 \le l < N+Q-1$

Suppose *I1* and *I2* are matrices with dimensions (4,3) and (2,2). The following figure shows how the block computes cross-correlation value for the point `I1(1,3)`, which refers to the second row and fourth column in zero-based indexing.



The cross-correlation value for the point `I1(1,3)` is computed using these steps:

1. Slide the center element of *I2* so that it lies on top of the (1,3) element of *I1*.
2. Multiply each weight in *I2* by the element of *I1* underneath.
3. Sum the individual products from step 2.

The cross-correlation value for the point `I1(1,3)` is
$1 \cdot 8 + 8 \cdot 1 + 15 \cdot 6 + 7 \cdot 3 + 14 \cdot 5 + 16 \cdot 7 + 13 \cdot 4 + 20 \cdot 9 + 22 \cdot 2 = 585$.

The normalized cross-correlation value for the point `I1(1,3)` is

$$\frac{585}{\sqrt{\sum I1_p^2}\sqrt{\sum I2^2}} = 0.8070.$$

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

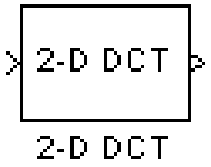
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

2-D Autocorrelation | 2-D Mean | 2-D Histogram | 2-D Median | 2-D Standard Deviation | 2-D Variance | 2-D Maximum | 2-D Minimum

2-D DCT

Compute 2-D discrete cosine transform (DCT)



Libraries:
Computer Vision Toolbox / Transforms

Description

The 2-D DCT block calculates the two-dimensional discrete cosine transform of an image. Suppose $f(x,y)$ is the input image of dimension M-by-N, the equation for the 2-D DCT is

$$F(m,n) = \frac{2}{\sqrt{MN}} C(m)C(n) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)m\pi}{2M} \cos\frac{(2y+1)n\pi}{2N}$$

where $C(m) = C(n) = 1/\sqrt{2}$ for $m, n = 0$ and $C(m), C(n) = 1$ otherwise.

The number of rows and columns of the input image must be power of 2. You can also use this block to compute 1-D DCT of a vector.

Ports

Input

Port_1 — Input data
matrix | vector

Input data, specified as a numeric matrix or a vector. The size of the input data must be power of 2.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Output

Port_1 — Output data
matrix | vector

Output data containing the DCT coefficients, returned as a matrix or vector. The size and the datatype of the output is same as that of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Parameters

Main Tab

Sine and cosine computation — Specify how the block computes the sine and cosine terms

`Table lookup` (default) | `Trigonometric fcn`

Use this parameter to specify how the block computes the sine and cosine terms in the DCT algorithm.

- If you select `Trigonometric fcn`, the block computes the sine and cosine values during the simulation.
- If you select `Table lookup`, the block computes and stores the trigonometric values before the simulation starts. In this case, the block requires extra memory. `Table lookup` is the default.

Data Types Tab

Fixed-point operational parameters

Rounding mode — Rounding mode for fixed-point representation

`Floor` (default) | `Ceiling` | `Convergent` | `Nearest` | `Round` | `Simplest` | `Zero`

Use this parameter to specify the rounding mode for the block to use when the specified data type and scaling cannot exactly represent the result of a fixed-point calculation. See Rounding Modes for more information on the available values.

Note The sine table values do not obey this parameter; they always round to `Nearest`.

Saturate on integer overflow — Specify overflow mode

off (default) | on

When you select this parameter, the block saturates the result of its fixed-point operation. When you clear this parameter, the block wraps the result of its fixed-point operation.

For details on saturate and wrap, see “Overflow Handling” for fixed-point operations.

Note The sine table values do not obey this parameter; instead, they are always saturated.

Fixed-point data types

Sine Table — Data type of sine table

`Inherit: Same word length as input` (default) | `fixdt(1,16)` | `<data type expression>`

Use this parameter to specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one.

The sine table values do not obey the **Rounding mode** and **Saturate on integer overflow** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output — Data type of product output

`Inherit: Inherit via internal rule` (default) | `Inherit: Same as input` | `fixdt(1,16,0)` | `<data type expression>`

Use this parameter to specify the product output data type. See “Fixed-Point Data Types” on page 1-17 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block.

Accumulator — Data type of accumulator

`Inherit: Inherit via internal rule` (default) | `Inherit: Same as input` | `Inherit: Same as product output` | `fixdt(1,16,0)` | `<data type expression>`

Use this parameter to specify the accumulator data type. See “Fixed-Point Data Types” on page 1-17 for illustrations depicting the use of the accumulator data type in this block.

Output — Data type of output

`Inherit: Inherit via internal rule` (default) | `Inherit: Same as input` | `fixdt(1,16,0)` | `<data type expression>`

Use this parameter to specify the output data type. See “Fixed-Point Data Types” on page 1-17 for illustrations depicting the use of the output data type in this block. When you set this parameter to

- `Inherit: Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{idealoutput} = WL_{input} + floor(\log_2(DCTlength - 1)) + 1$$

  $$FL_{idealoutput} = FL_{input}$$

  Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override
off (default) | on

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool.

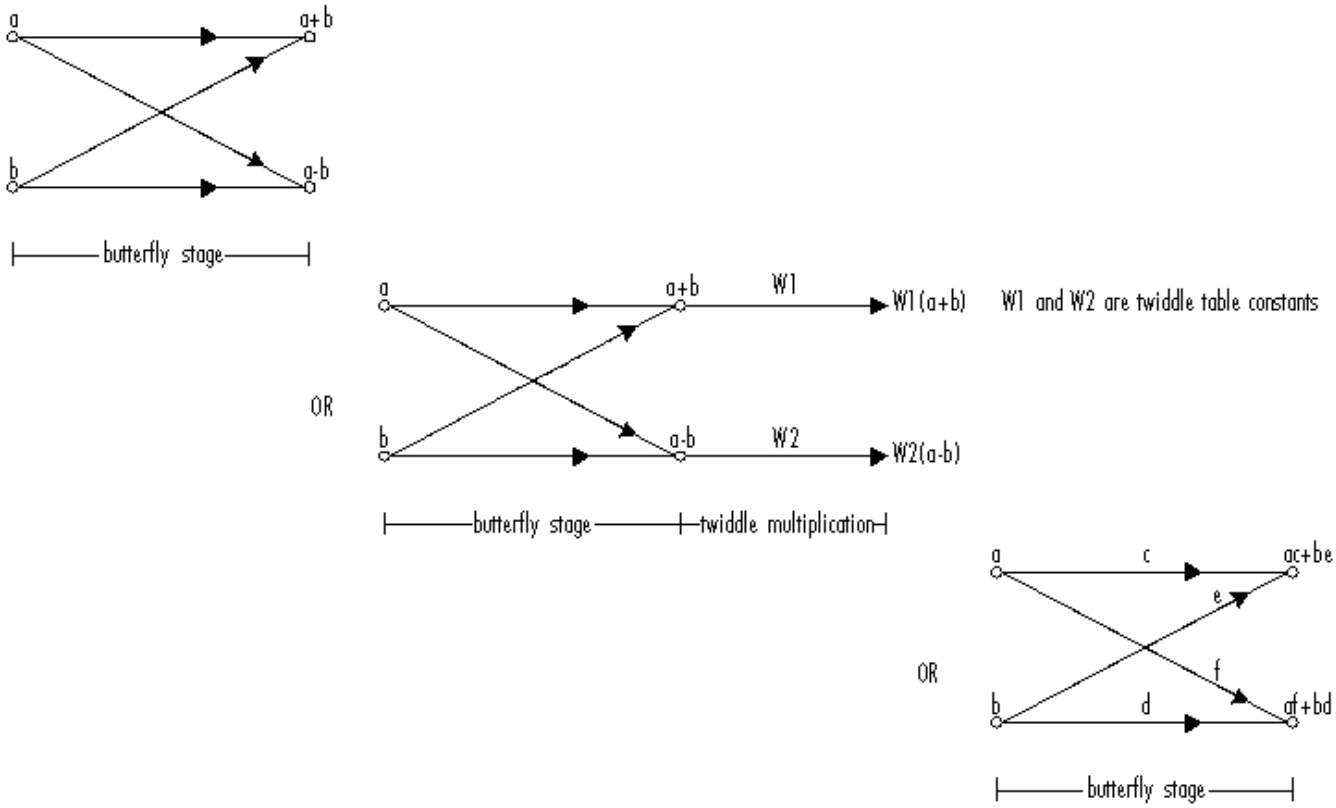
Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

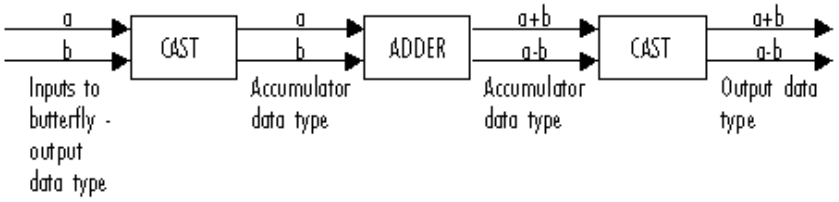
More About

Fixed-Point Data Types

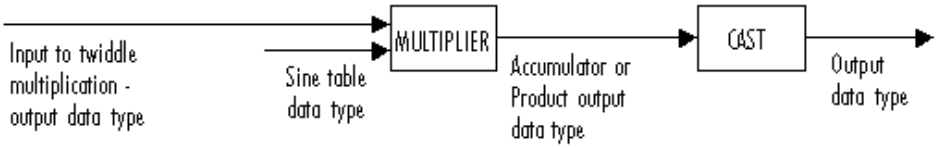
The following diagram shows the data types used in the 2-D DCT block for fixed-point signals. Inputs are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.



Butterfly Stage Data Types



Twiddle Multiplication Data Types



The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, refer to "Multiplication Data Types". You can set the sine table, product output, accumulator, and output data types in the block mask as discussed in the next section.

Version History

Introduced before R2006a

References

[1] Chen, W.H, C.H. Smith, and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Communications*, 25 (1977): 1004-1009.

[2] Wang, Z. "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, 32 (August 1984): 803-816.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

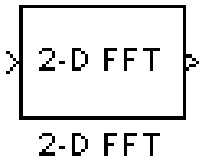
2-D IDCT | 2-D FFT | 2-D IFFT

Functions

`dct2` | `idct2`

2-D FFT

Compute 2-D fast Fourier transform (FFT)



Libraries:
Computer Vision Toolbox / Transforms

Description

The 2-D FFT block computes the discrete Fourier transform (DFT) of a two-dimensional input matrix using the fast Fourier transform (FFT) algorithm. The equation for the two-dimensional DFT $F(m, n)$ of an M-by-N input matrix, $f(x, y)$, is:

$$F(m,n) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j\frac{2\pi mx}{M}} e^{-j\frac{2\pi ny}{N}}$$

where $0 \le m \le M-1$ and $0 \le n \le N-1$.

The block supports FFT implementation based on the FFTW library and an implementation based on a collection of Radix-2 algorithms. You can either manually select one of these implementations or let the block select one automatically.

Ports

Input

Port_1 — Input data
vector | matrix

Input data, specified as a vector or matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Output

Port_1 — Output data
vector | matrix

Output data containing the 2-D FFT of the input, returned as a vector or matrix. The size and data type of the output are the same as those of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Parameters

Main

FFT implementation — FFT implementation

`Auto` (default) | `Radix-2` | `FFTW`

Specify the type of implementation to use to compute the FFT as one of these options:

- `FFTW` — Select this option to support an arbitrary-length input signal. The block restricts generated code with the `FFTW` implementation to host computers capable of running MATLAB®.
- `Radix-2` — Select this option to support bit-reversed processing, fixed and floating-point data, or portable C-code generation using "Simulink Coder". The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or, if possible, choose the `FFTW` implementation. For more information about the algorithms used by the `Radix-2` mode, see "Radix-2 Implementation" on page 1-23.
- `Auto` — Select this option to let the block choose the FFT implementation. For non-power-of-two transform lengths, the block restricts generated code to MATLAB host computers.

Output in bit-reversed order — Bit-reversed output

`off` (default) | `on`

Select this parameter to get the output data in bit-reversed order relative to the order of the input data. If you clear this parameter, the output data appears in linear order relative to the order of the input data. For more information on the bit-reversed order of output, see "Bit-Reversed Order" on page 1-21.

Dependencies

To enable this parameter, set the **FFT implementation** parameter to `Auto` or `Radix-2`.

Scale result by FFT length — Scale output data

`off` (default) | `on`

When you select this parameter, the block divides the output by the FFT length. This option is useful when you want the output of the FFT to stay in the same amplitude range as its input. This is particularly useful when working with fixed-point data types.

Data Types

For details on the fixed-point block parameters, see "Specify Fixed-Point Attributes for Blocks" (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see "Lock the Output Data Type Setting" (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

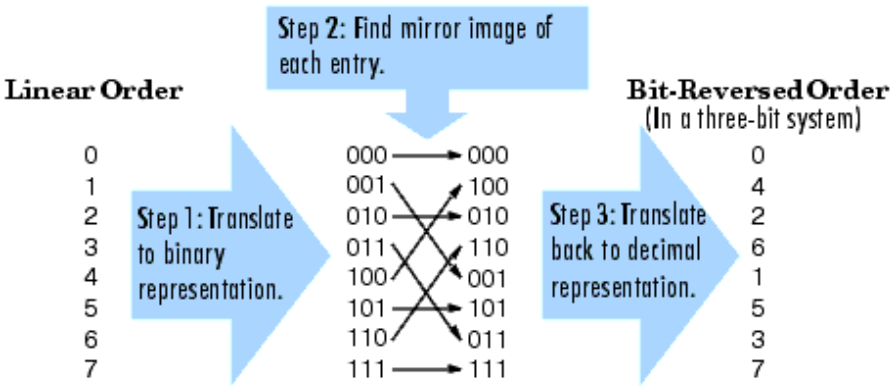
More About

Bit-Reversed Order

Two numbers are bit-reversed values of each other when the binary representation of one is the mirror image of the binary representation of the other. For example, in a three-bit system, one and four are bit-reversed values of each other because the three-bit binary representation of one, `001`, is the mirror image of the three-bit binary representation of four, `100`. The diagram shows the row indices in linear order. To put them in bit-reversed order:

1. Translate the indices into their binary representations with the minimum number of bits. In this example, the minimum number of bits is three because the binary representation of the largest row index, 7, is `111`.
2. Find the mirror image of each binary entry, and write it beside the original binary representation.
3. Translate each binary mirror image to its decimal representation.

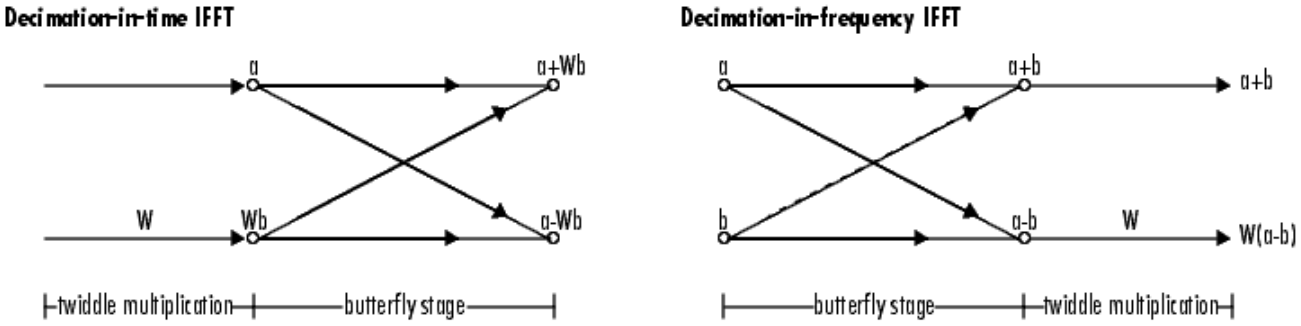
   The row indices now appear in bit-reversed order.

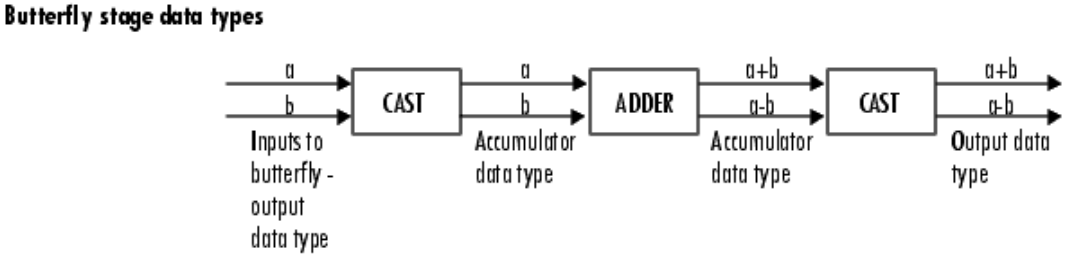


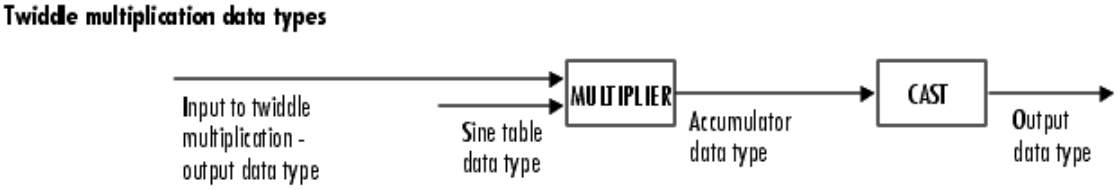
When you select the **Output in bit-reversed order** parameter of the 2-D FFT block, the block bit-reverses the order of both the rows and columns. All output values remain the same, but they appear in a different order.

Fixed-Point Data Types

These diagrams show the data types used in the 2-D FFT block for fixed-point signals. The block first casts inputs to the output data type and stores them in the output buffer. Each butterfly stage then processes signals in the accumulator data type, with the final butterfly casting its output back into the output data type. The block multiplies by a twiddle factor before each butterfly stage, in a decimation-in-time FFT, and after each butterfly stage in a decimation-in-frequency FFT.







The multiplier output appears in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, refer to “Multiplication Data Types”.

Algorithms

FFTW Implementation

The FFTW implementation provides an optimized FFT calculation, including support for power-of-two and non-power-of-two transform lengths in both simulation and code generation. Generated code using the FFTW implementation can run only on computers capable of running MATLAB. The input must be of a floating-point data type.

Radix-2 Implementation

The Radix-2 implementation supports bit-reversed processing, fixed or floating-point data, and enables the block to provide portable C-code generation using “Simulink Coder”. The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two.

The block implements one or more of these algorithms for Radix-2 implementation.

- Butterfly operation
- Double-signal algorithm
- Half-length algorithm
- Radix-2 decimation-in-time (DIT) algorithm
- Radix-2 decimation-in-frequency (DIF) algorithm

Radix-2 Algorithms for Real and Complex Signals

Complexity of Input	Output Ordering	Algorithms Used for FFT Computation
Complex	Linear	Bit-reversed operation and Radix-2 DIT
Complex	Bit-reversed	Radix-2 DIF
Real	Linear	Bit-reversed operation and Radix-2 DIT in conjunction with the half-length and double-signal algorithms
Real	Bit-reversed	Radix-2 DIF in conjunction with the half-length and double-signal algorithms

Radix-2 Optimization for Table of Trigonometric Values

In certain situations, the Radix-2 algorithm computes all the possible trigonometric values of the twiddle factor:

$$e^{j\frac{2\pi k}{K}},$$

where K is the greater value of either M or N, and k is an integer in the range `[0, K – 1]`. The block stores these values in a table and retrieves them during simulation. This table summarizes the number of table entries each for fixed-point and floating-point Radix-2 FFT algorithms:

Number of Table Entries for N-Point FFT	
Floating point	$3N/4$
Fixed point	N

Version History

Introduced before R2006a

References

[1] “FFTW Home Page.” Accessed February 23, 2022. https://www.fftw.org/.

[2] Frigo, M., and S.G. Johnson. “FFTW: An Adaptive Software Architecture for the FFT.” In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP ’98 (Cat. No.98CH36181)*, 3:1381–84. Seattle, WA, USA: IEEE, 1998. https://doi.org/10.1109/ICASSP.1998.681704.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:

- When these conditions apply, the executable generated from this block relies on prebuilt dynamic library files (`.dll` files) included with MATLAB:
  - **FFT implementation** is set to `FFTW`.
  - **Inherit FFT length from input dimensions** is cleared, and the length of each dimension of the input matrix is a power of two.

  Use the `packNGo` function to package the code generated from this block and all the relevant files in a compressed ZIP file. Using this ZIP file, you can relocate, unpack, and rebuild your project in another development environment where MATLAB is not installed.
- When the length of each dimension of the input matrix is a power of two, you can generate standalone C and C++ code from this block.

See Also

Blocks

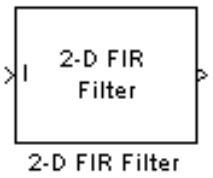
2-D IFFT | 2-D DCT | 2-D IDCT | 2-D FIR Filter

Functions

`fft2` | `ifft2` | `bitrevorder`

2-D FIR Filter

2-D FIR filter on input matrix



Libraries:
Computer Vision Toolbox / Filtering

Description

The 2-D FIR Filter block filters a 2-D grayscale image using a 2-D finite impulse response (FIR) filter represented as a 2-D coefficient matrix or a pair of separable filter coefficient vectors.

Ports

Input

I — Input image
matrix

Input image, specified as a numeric matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

H — Filter coefficients
matrix

Filter coefficients, specified as a numeric matrix. The input to the **H** port must have the same data type as the input to the **I** port.

Dependencies

To enable this port, set the **Coefficient source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

HV — Vertical filter coefficients
vector

Vertical filter coefficients, specified as a numeric vector. The input to the **HV** port must have the same data type as the input to the **I** port.

Dependencies

To enable this port, specify these parameter values.

- **Separable filter coefficients** — `on`
- **Coefficient source** — `Input port`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

HH — Horizontal filter coefficients
vector

Horizontal filter coefficients, specified as a numeric vector. The input to the **HH** port must have the same data type as the input to the **I** port.

Dependencies

To enable this port, specify these parameter values.

- **Separable filter coefficients** — `on`
- **Coefficient source** — `Input port`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

PVal — Pad value
scalar

Pad value, specified as a numeric scalar. The input to the **PVal** port must have the same data type as the input to the **I** port.

Dependencies

To enable this port, specify these parameter values.

- **Output size** — `Full` or `Same as input port I`
- **Padding options** — `Constant`
- **Pad value source** — `Input port`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`
Complex Number Support: Yes

Output

Output — Filtered output image
matrix

Filtered output image, returned as a numeric matrix of the same data type as the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main

Separable filter coefficients — Filter coefficients are separable

`off` (default) | `on`

Select this parameter if your filter coefficients are separable. Using separable filter coefficients reduces the number of calculations the block must perform to compute the output. You can use `isfilterseparable` to check if your filter is separable and compute the vertical and horizontal filter coefficients.

Coefficient source — Source of filter coefficients

`Specify via dialog` (default) | `Input port`

Specify the source of the filter coefficients as one of these options.

- `Specify via dialog` — Specify the filter coefficients using the **Coefficients** parameter or, if you select the **Separable filter coefficients** parameter, by using the **Vertical coefficients (across height)** and **Horizontal coefficients (across width)** parameters.
- `Input port` — Specify the filter coefficients using the **H** port, or if you select the **Separable filter coefficients** parameter, by using the **HV** and **HH** ports.

Coefficients — Filter coefficients

`[1 0; 0 -1]` (default) | numeric matrix

Specify the filter coefficients as a numeric matrix.

Dependencies

To enable this parameter, specify these parameter values.

- **Separable filter coefficients** — `off`
- **Coefficient source** — `Specify via dialog`

Vertical coefficients (across height) — Vertical filter coefficients

`[4 0]` (default) | numeric vector

Specify the vertical filter coefficients as a numeric vector.

Dependencies

To enable this parameter, specify these parameter values.

- **Separable filter coefficients** — `on`
- **Coefficient source** — `Specify via dialog`

Horizontal coefficients (across height) — Horizontal filter coefficients

`[4 0]` (default) | numeric vector

Specify the horizontal filter coefficients as a numeric vector.

Dependencies

To enable this parameter, specify these parameter values.

- **Separable filter coefficients** — `on`
- **Coefficient source** — `Specify via dialog`

Output size — Size of filtered image

`Full` (default) | `Same as input port I` | `Valid`

For an input image having dimensions *M*-by-*N* and a filter having dimensions *P*-by-*Q*, specify the size of the filtered image as one of these options.

- `Full` — The block uses padding to return a filtered image with full dimensions *(M+P–1)*-by-*(N+Q–1)*.
- `Same as input port I` — The block uses padding to return a filtered image with dimensions *M*-by-*N*, the same size as the input image.
- `Valid` — The block filters the input image only where the coefficient matrix fits entirely within it without padding, and returns a filtered image with dimensions *(M–P+1)*-by-*(N–Q+1)*.

Padding options — Padding method

`Constant` (default) | `Replicate` | `Symmetric` | `Circular`

Specify the padding method as one of these options.

- `Constant` — Pads the input image with a constant value.
- `Replicate` — Pads the input image by repeating its border values.
- `Symmetric` — Pads the input image by mirror-reflecting the image across its border.
- `Circular` — Pads the input image by assuming the image is periodic.

Dependencies

To enable this parameter, specify the **Output size** parameter as either `Full` or `Same as input port I`.

Pad value source — Source of padding value

`Specify via dialog` (default) | `Input port`

Specify the source of the padding value as one of these options.

- `Specify via dialog` — Specify the pad value using the **Pad value** parameter.
- `Input port` — Specify the pad value using the **PVal** port.

Dependencies

To enable this parameter, specify these parameter values.

- **Output size** — `Full` or `Same as input port I`

- **Padding options** — `Constant`

Pad value — Padding value

`0` (default) | numeric scalar

Specify the padding value as a numeric scalar.

Dependencies

To enable this parameter, specify these parameter values.

- **Output size** — `Full` or `Same as input port I`
- **Padding options** — `Constant`
- **Pad value source** — `Specify via dialog`

Filtering based on — Method of filtering

`Convolution` (default) | `Correlation`

Specify the method of filtering as `Convolution` or `Correlation`.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

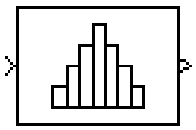
`isfilterseparable` | `imfilter`

Topics

“Periodic Noise Reduction”
“Video Stabilization”

2-D Histogram

Generate histogram from input



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Histogram block computes the frequencies of the elements in the input image. The block calculates the histogram values for either the entire input or for each column of the input image. It sorts all the of input values into bins according to their pixel values. The histogram value for a given bin represents the frequency of the input values bracketed by that bin. The block is also capable of computing a running histogram across a sequence of inputs. The block is also capable of computing a running histogram across a sequence of inputs.

Ports

Input

Port_1 — Input image
vector | matrix | 3-D array

Input image, specified as a vector, matrix, or 3-D array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Rst — Reset value
scalar

Reset value for an input sequence, specified as a scalar. This value dictates the event that causes the block to reset the running histogram. The sample time of the **Rst** input must be a positive integer and a multiple of the input sample time.

Dependencies

To enable this port, select the **Running histogram** parameter and set the **Trigger type** parameter to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero sample`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Output

Port_1 — Histogram output
vector | matrix | 3-D array

Histogram output, returned as a vector, matrix, or 3-D array.

If the **Find the histogram over** parameter is set to `Entire input`, then the output is always a vector. If the **Find the histogram over** parameter is set to `Each column`, then the output has the same number of dimension as the input.

Data Types: `single` | `double` | `uint32`

Parameters

Main

Lower limit of histogram — Lower limit of lowest-valued bin
`0` (default) | real-valued scalar

Specify a real-valued scalar for the lower boundary, B_m, of the lowest-valued bin. `NaN` and `inf` are not valid values for B_m.

Tunable: Yes

Upper limit of histogram — Upper limit of highest-valued bin
`1` (default) | real-valued scalar

Specify a real-valued scalar for the upper boundary, B_m, of the highest-valued bin. `NaN` and `inf` are not valid values for B_M.

Tunable: Yes

Number of bins — Number of bins in histogram
`256` (default) | real-valued scalar

Specify the number of bins in the histogram.

Find the histogram over — Dimension along which to compute histogram
`Entire input` (default) | `Each column`

Specify the dimension of the input image along which the block computes the histogram values. For any given value, *n*, of the **Number of bins** parameter, this parameter affects the output of the block as follows.

- `Entire input` — The block computes the histogram over the entire input. It outputs an *n*-element column vector.
- `Each column` — The block computes the histogram over each column of the input.
  - If the input is an *M*-by-*N* matrix, the block outputs an *n*-by-*N* matrix. The *j*th column of the output matrix contains the histogram for the data in the *j*th column of the *M*-by-*N* input matrix.
  - If the input is an *M*-by-*N*-by-*P* array, the block outputs an *n*-by-*N*-by-*P* array. If the input is a three-channel color image, the value of P is 3, and the block computes the histogram for each color channel separately.

Normalized — Normalize histogram output
`off` (default) | `on`

Select the **Normalized** parameter to normalize the output histogram value within a range of [0, 1].

Note Use of this parameter is not supported for fixed-point signals.

Running histogram — Frequency distribution for successive inputs
`off` (default) | `on`

Select the **Running histogram** parameter to compute the frequency distribution of both the past and present data for successive inputs. The block resets the histogram (by emptying all of the bins) when it detects a reset event at the optional **Rst** port. There's an additional dimension to the output size which is equal to $((T_{sim}/T_s)+1)$ where T_{sim} is the simulation time and T_s is the sample time for block execution. The output depends on the value of the **Find the histogram over** parameter.

- `Entire input` — The block computes a running histogram for the data in the first dimension of the input. The block outputs an *n*-by-1-by-$((T_{sim}/T_s)+1)$, where *n* is the value of **Number of bins** parameter.
- `Each column` — The block computes a running histogram for each column of the *M*-by-*N* matrix. The block outputs an *n*-by-*N*-by-$((T_{sim}/T_s)+1)$ array, where *n* is the value of the **Number of bins** parameter.

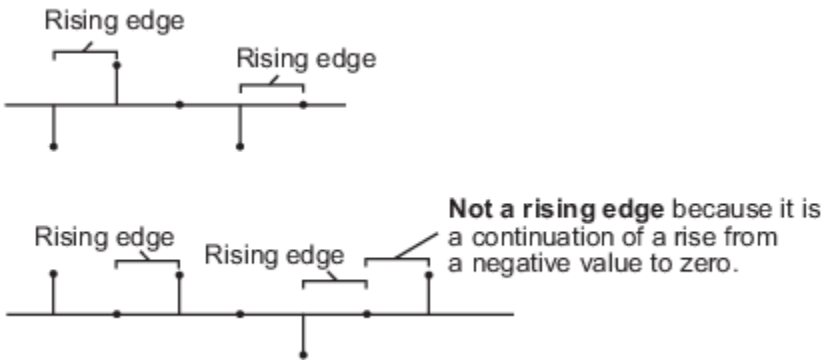
Note When the block is used in running mode and the input data type is non-floating point, the output of the histogram is stored as a `uint32` data type. The largest number that can be represented by this data type is $2^{32}-1$. If the range of the `uint32` data type is exceeded, the output data wraps back to 0.

Trigger type — Reset condition of running histogram
`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

Specify the event that, when detected at the optional **Rst** port, resets the running histogram. The reset signal and the input data signal must have the same rate.

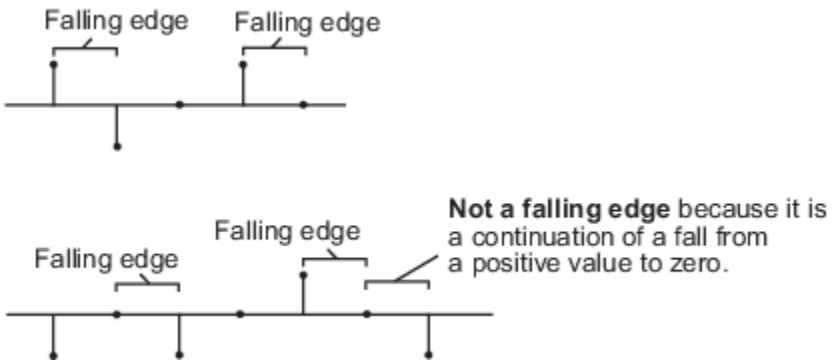
Specify the reset event as:

- `None` — Disables the **Rst** port
- `Rising edge` — Triggers a reset event when the **Rst** input does one of the following:
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` — Triggers a reset event when the **Rst** input does one of the following:
  - Falls from a positive value to either a negative value or zero

- Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` — Triggers a reset event when the **Rst** input is a rising edge or falling edge
- `Non-zero sample` — Triggers a reset event at each sample time that the **Rst** input is not zero

Dependencies

To enable this parameter, select the **Running histogram** parameter.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Algorithms

The histogram bins have equal width of:

$$\Delta = \frac{B_M - B_m}{n},$$

where

- B_M — Represents the upper boundary of the highest-valued bin.

- B_m — Represents the lower boundary of the lowest-valued bin.
- n — Represents the number of bins.

The centers are located at:

$$B_m + \left(k + \frac{1}{2}\right)\Delta \quad k = 0, 1, 2, \ldots, n - 1 .$$

For input values that fall between two bins, the block places them in the lower-valued bin. For input values greater than the upper limit of the histogram or less than the lower limit of the histogram, the block places them in the highest-valued or lowest-valued bin, respectively.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Generated code relies on `memcpy` or `memset` functions (string.h) under certain conditions.

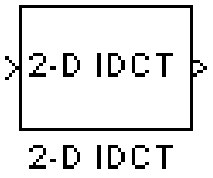
See Also

Functions

`histogram`

2-D IDCT

Compute 2-D inverse discrete cosine transform (IDCT)



Libraries:
Computer Vision Toolbox / Transforms

Description

The 2-D IDCT block calculates the two-dimensional inverse discrete cosine transform of the input signal. The equation for the two-dimensional IDCT of an input signal is:

$$f(x,y) = \frac{2}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} C(m)C(n)F(m,n)\cos\frac{(2x+1)m\pi}{2M}\cos\frac{(2y+1)n\pi}{2N},$$

where $F(m,n)$ is the discrete cosine transform (DCT) of the signal $f(x,y)$. If $m = n = 0$, then $C(m) = C(n) = 1/\sqrt{2}$. Otherwise $C(m) = C(n) = 1$.

Ports

Input

Port_1 — Input data
matrix | vector

Specify input data as a vector or matrix of intensity values. The number of elements in the input data must be a power of two.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Output

Port_1 — Output data
matrix | vector

Output data containing the 2-D IDCT of the input, returned as a matrix or vector. The size and data type of the output are the same as those of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Parameters

Main

Sine and cosine computation — How the block computes sine and cosine terms

`Table lookup` (default) | `Trigonometric fcn`

Specify how the block computes the sine and cosine terms to find the 2-D IDCT.

- `Table lookup` — The block computes and stores the trigonometric values before the simulation starts. This option requires more memory than the `Trigonometric fcn` option.
- `Trigonometric fcn` — The block computes the sine and cosine values during the simulation.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

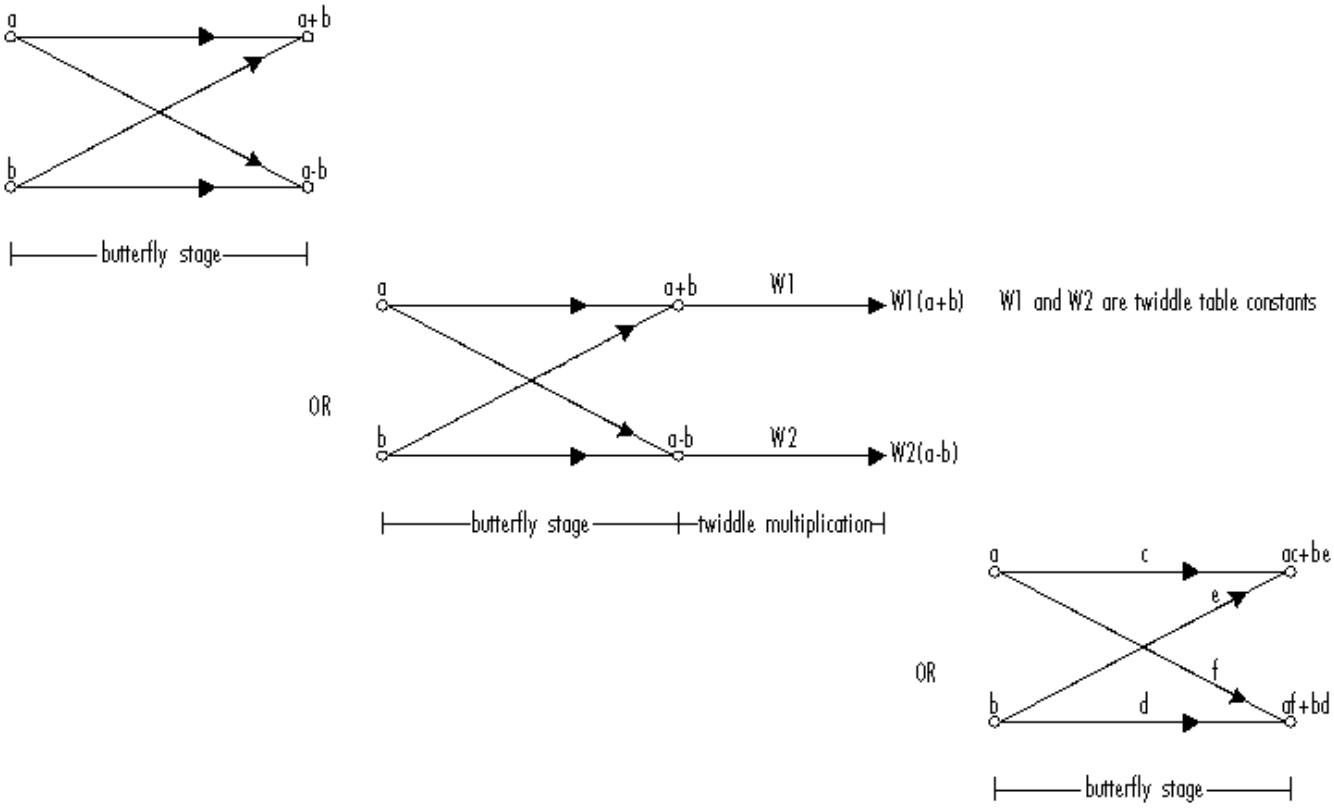
Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

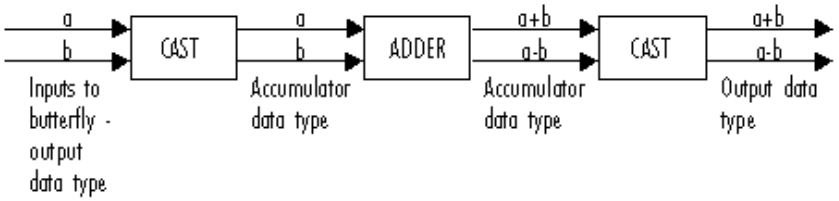
More About

Fixed-Point Data Types

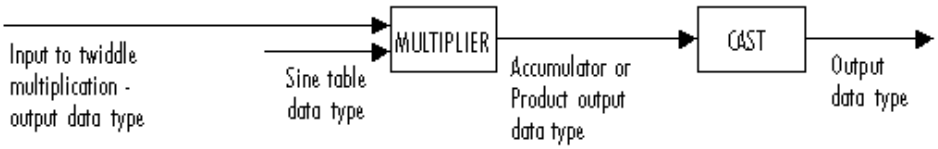
The following diagram shows the data types used in the 2-D IDCT block for fixed-point signals. Inputs are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.



Butterfly Stage Data Types



Twiddle Multiplication Data Types



When at least one of the inputs to the multiplier is real, the output of the multiplier is in the product output data type. When both of the inputs to the multiplier are complex, the multiplication result is in the accumulator data type. For more information on the multiplication of real and complex numbers, see “Multiplication Data Types”.

Version History

Introduced before R2006a

References

[1] Wen-Hsiung Chen, C. Smith, and S. Fralick. “A Fast Computational Algorithm for the Discrete Cosine Transform.” *IEEE Transactions on Communications* 25, no. 9 (September 1977): 1004–9. https://doi.org/10.1109/TCOM.1977.1093941.

[2] Zhongde Wang. “Fast Algorithms for the Discrete W Transform and for the Discrete Fourier Transform.” *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32, no. 4 (August 1984): 803–16. https://doi.org/10.1109/TASSP.1984.1164399.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

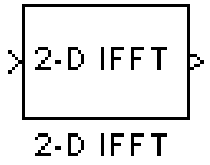
2-D DCT | 2-D FFT | 2-D IFFT

Functions

`dct2` | `idct2`

2-D IFFT

Compute 2-D inverse fast Fourier transform (IFFT)



Libraries:
Computer Vision Toolbox / Transforms

Description

The 2-D IFFT block computes the inverse discrete Fourier transform (IDFT) of a two-dimensional input matrix using the fast Fourier transform (FFT) algorithm. The equation for the 2-D IDFT $f(x, y)$ of an M-by-N input matrix, $F(m, n)$, is:

$$f(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)e^{j\frac{2\pi mx}{M}}e^{j\frac{2\pi ny}{N}},$$

where $0 \le x \le M-1$ and $0 \le y \le N-1$.

The block supports FFT implementation based on the FFTW library and an implementation based on a collection of Radix-2 algorithms. You can either manually select one of these implementations or let the block select one automatically.

Ports

Input

Port_1 — Input data
vector | matrix

Input data, specified as a vector or matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Output

Port_1 — Output data
vector | matrix

Output data containing the 2-D IFFT of the input, returned as a vector or matrix. The size and data type of the output are the same as those of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Parameters

Main

FFT implementation — FFT implementation

`Auto` (default) | `Radix-2` | `FFTW`

Specify the type of implementation to compute the FFT as one of these options:

- `FFTW` — Select this option to use to support an arbitrary-length input signal. The block restricts generated code with the `FFTW` implementation to host computers capable of running MATLAB.
- `Radix-2` — Select this option to support bit-reversed processing, fixed and floating-point data, or portable C-code generation using "Simulink Coder". The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two, or, if possible, choose the `FFTW` implementation. For more information about the algorithms used by the `Radix-2` mode, see "Radix-2 Implementation" on page 1-23.
- `Auto` — Select this option to let the block choose the FFT implementation. For non-power-of-two transform lengths, the block restricts generated code to MATLAB host computers.

Input is in bit-reversed order — Bit-reversed input

`off` (default) | `on`

Specify whether the input to the block is in bit-reversed order or linear order. Select this parameter when you specify the input in bit-reversed order. Otherwise, clear this parameter. The block yields invalid output when you do not set this parameter correctly. For more information on the bit-reversed order of input, see "Bit-Reversed Order" on page 1-42.

Dependencies

To enable this parameter, set the **FFT implementation** parameter to `Auto` or `Radix-2`.

Input is conjugate symmetric — Conjugate symmetric input

`on` (default) | `off`

Select this parameter to specify that the block input is conjugate symmetric, and you want real-valued outputs. Otherwise, clear this parameter.

The 2-D FFT block yields conjugate symmetric output when you input real-valued data. Taking the 2-D IFFT of a conjugate symmetric input matrix produces real-valued output. Therefore, if the input to the 2-D IFFT block is conjugate symmetric, and you select this parameter, the block produces real-valued outputs. When you select this parameter, the block optimizes its computation method. You cannot select this parameter for fixed-point.

If you specify conjugate symmetric input data and do not select this parameter, the block outputs complex-valued data with small imaginary parts. The block outputs invalid data if you select this parameter with input data that is not conjugate symmetric input data.

Divide output by product of FFT length in each input dimension — Scale output data

`on` (default) | `off`

Select this parameter to compute the scaled IFFT, which divides the output by the product of the dimension lengths of the FFT input dimension, as shown in this equation.

$$f(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)e^{j\frac{2\pi mx}{M}}e^{j\frac{2\pi ny}{N}},$$

where $0 \le x \le M-1$ and $0 \le y \le N-1$.

When you clear this parameter, the block computes the unscaled version of the IFFT, as shown in this equation.

$$f(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)e^{j\frac{2\pi mx}{M}}e^{j\frac{2\pi ny}{N}}$$

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

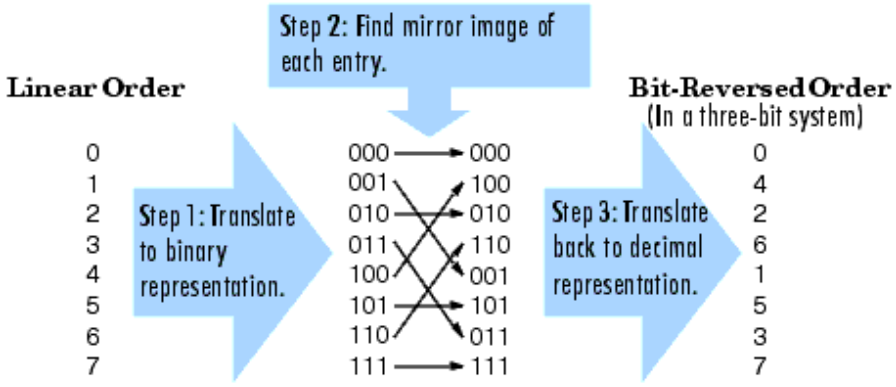
More About

Bit-Reversed Order

Two numbers are bit-reversed values of each other when the binary representation of one is the mirror image of the binary representation of the other. For example, in a three-bit system, one and four are bit-reversed values of each other because the three-bit binary representation of one, `001`, is the mirror image of the three-bit binary representation of four, `100`. The diagram shows the row indices in linear order. To put them in bit-reversed order:

1. Translate the indices into their binary representations with the minimum number of bits. In this example, the minimum number of bits is three because the binary representation of the largest row index, 7, is `111`.
2. Find the mirror image of each binary entry, and write it beside the original binary representation.
3. Translate each binary mirror image to its decimal representation.

   The row indices now appear in bit-reversed order.

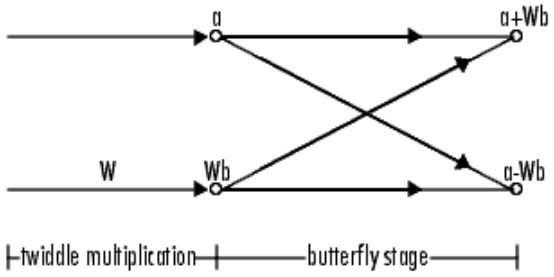


When you select the **Output in bit-reversed order** parameter of the 2-D FFT block, the block bit-reverses the order of both the rows and columns. All output values remain the same, but they appear in a different order.

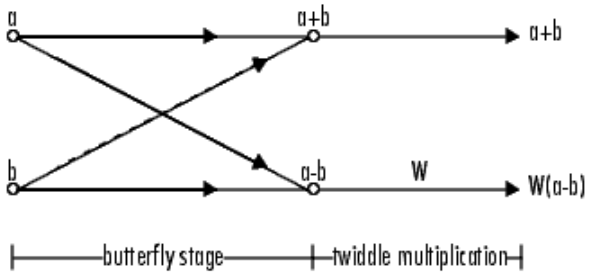
Fixed-Point Data Types

These diagrams show the data types used in the 2-D FFT block for fixed-point signals. The block first casts inputs to the output data type and stores them in the output buffer. Each butterfly stage then processes signals in the accumulator data type, with the final butterfly casting its output back into the output data type. The block multiplies by a twiddle factor before each butterfly stage, in a decimation-in-time FFT, and after each butterfly stage in a decimation-in-frequency FFT.

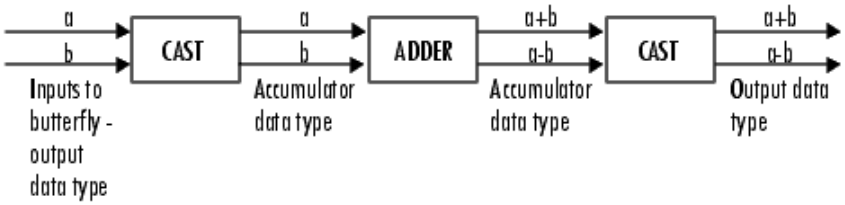
Decimation-in-time IFFT



Decimation-in-frequency IFFT



Butterfly stage data types



Twiddle multiplication data types



The multiplier output appears in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, refer to “Multiplication Data Types”.

Algorithms

FFTW Implementation

The FFTW implementation provides an optimized FFT calculation, including support for power-of-two and non-power-of-two transform lengths in both simulation and code generation. Generated code using the FFTW implementation can run only on computers capable of running MATLAB. The input must be of a floating-point data type.

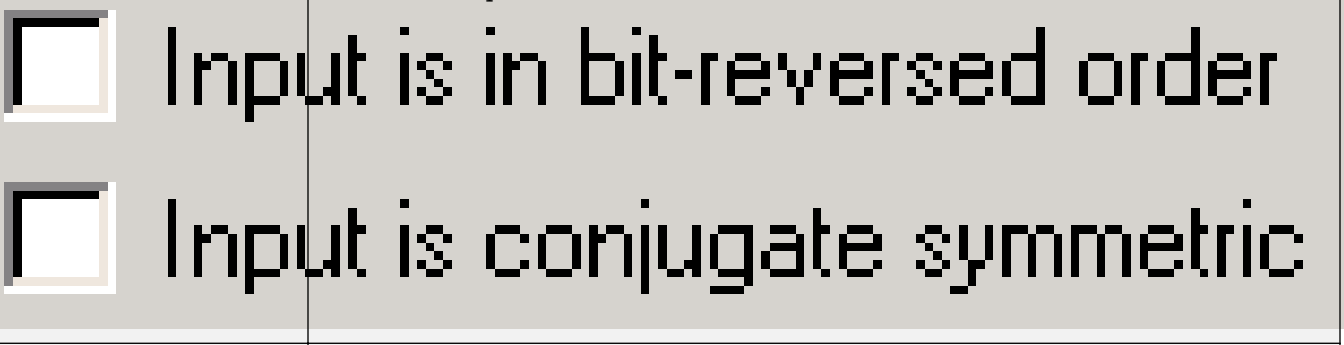
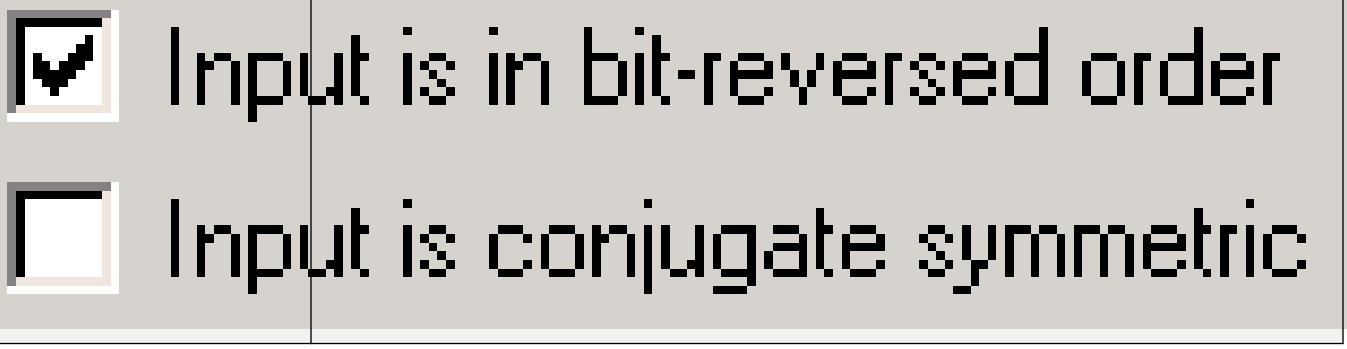
Radix-2 Implementation

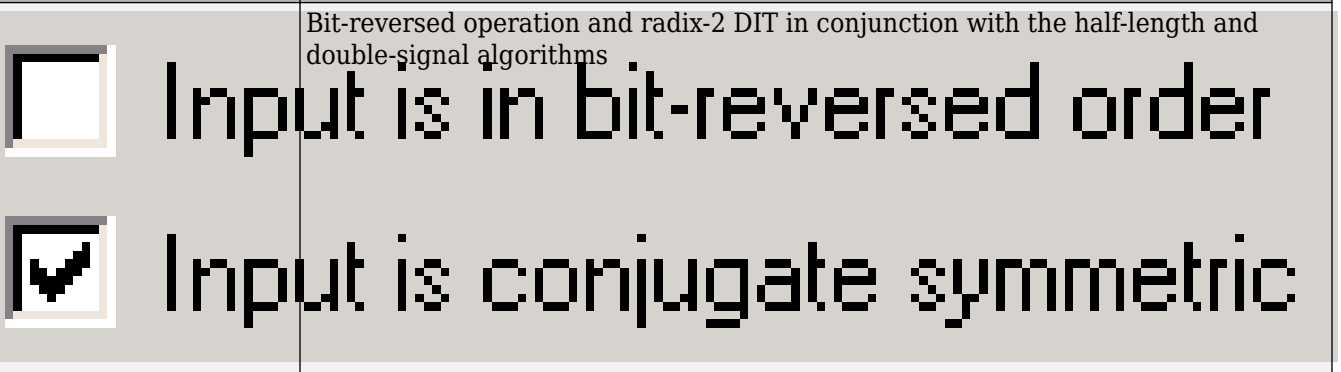
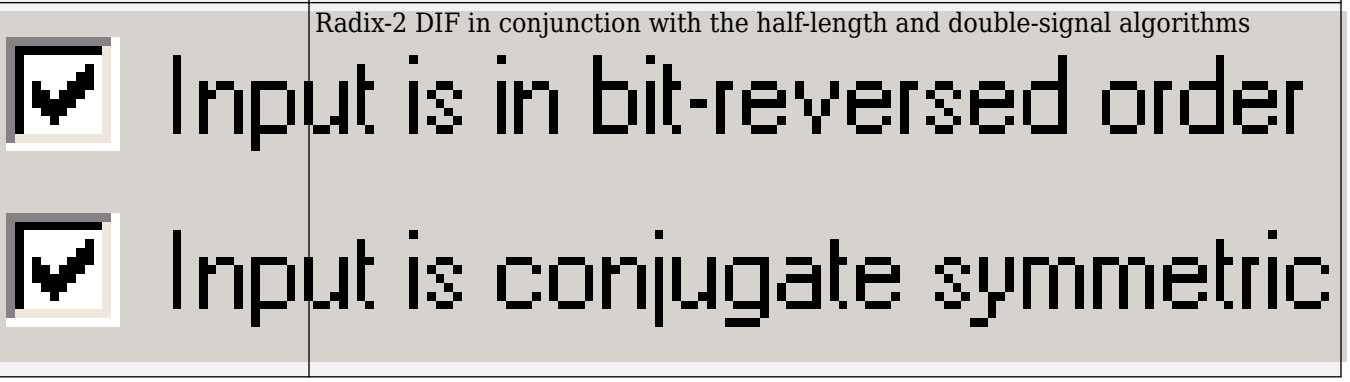
The Radix-2 implementation supports bit-reversed processing, fixed or floating-point data, and enables the block to provide portable C-code generation using “Simulink Coder”. The dimensions of the input matrix, *M* and *N*, must be powers of two. To work with other input sizes, use the Image Pad block to pad or truncate these dimensions to powers of two.

The block implements one or more of these algorithms for Radix-2 implementation.

- Butterfly operation
- Double-signal algorithm
- Half-length algorithm
- Radix-2 decimation-in-time (DIT) algorithm
- Radix-2 decimation-in-frequency (DIF) algorithm

Radix-2 Algorithms for Real and Complex Signals

Parameter Settings	Algorithms Used for FFT Computation
	Bit-reversed operation and radix-2 DIT
	Radix-2 DIF

Parameter Settings	Algorithms Used for FFT Computation
	Bit-reversed operation and radix-2 DIT in conjunction with the half-length and double-signal algorithms
	Radix-2 DIF in conjunction with the half-length and double-signal algorithms

Note The **Input is conjugate symmetric** parameter is not supported for fixed-point signals.

In certain situations, the Radix-2 algorithm computes all the possible trigonometric values of the twiddle factor:

$$e^{j\frac{2\pi k}{K}},$$

where K is the greater value of either M or N, and k is an integer in the range [`0`, K – `1`]. The block stores these values in a table and retrieves them during simulation. This table summarizes the number of table entries each for fixed-point and floating-point Radix-2 FFT algorithms:

Number of Table Entries for N-Point FFT	
Floating point	$3N/4$
Fixed point	N

Version History

Introduced before R2006a

References

[1] “FFTW Home Page.” Accessed February 23, 2022. https://www.fftw.org/.

[2] Frigo, M., and S.G. Johnson. “FFTW: An Adaptive Software Architecture for the FFT.” In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP ’98 (Cat. No.98CH36181)*, 3:1381–84. Seattle, WA, USA: IEEE, 1998. https://doi.org/10.1109/ICASSP.1998.681704.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:

- When these conditions apply, the executable generated from this block relies on prebuilt dynamic library files (`.dll` files) included with MATLAB:
  - **FFT implementation** is set to `FFTW`.
  - **Inherit FFT length from input dimensions** is cleared, and the length of each dimension of the input matrix is a power of two.

  Use the `packNGo` function to package the code generated from this block and all the relevant files in a compressed ZIP file. Using this ZIP file, you can relocate, unpack, and rebuild your project in another development environment where MATLAB is not installed.
- When the length of each dimension of the input matrix is a power of two, you can generate standalone C and C++ code from this block.

See Also

Blocks

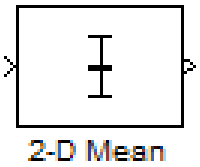
2-D FFT | 2-D DCT | 2-D IDCT | 2-D FIR Filter

Functions

`fft2` | `ifft2` | `bitrevorder`

2-D Mean

Find 2-D mean of input array



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Mean block computes the mean value of an input. It can compute the mean either along a specified dimension of the input or across the entire input. The block can also compute the running mean values in a sequence of inputs over a period of time.

Ports

Input

I — Input array
vector | matrix | *N*-D array

Input array, specified as a vector, matrix, or *N*-D array. This port is unnamed unless you select the **Enable ROI processing** parameter or select the **Running mean** parameter and set the **Reset port** parameter to a value other than `None`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Rst — Reset running mean
scalar

Reset the running mean, specified as a scalar. This port specifies the event that causes the block to reset the running mean. The sample time of the **Rst** input must be a positive integer and a multiple of the block sample time.

Dependencies

To enable this port, select the **Running mean** parameter and set the **Reset port** parameter to one of these options:

- `Rising edge`
- `Falling edge`
- `Either edge`
- `Non-zero sample`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

ROI — Region of interest
vector | matrix

Region of interest (ROI), specified as a four-element vector, *m*-by-4 matrix, or *M*-by-*N* matrix. This port accepts different input values depending on the setting of the **ROI type** parameter.

Note The block supports the **ROI** port for 2-D input images only.

Dependencies

To enable this port, select **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Label — Label matrix
matrix

Label matrix, specified as a matrix of nonnegative integers. The label matrix represents the objects in a 2-D image on a per pixel basis. The pixels labeled `0` represent the background, pixels labeled `1` make up the first object, the pixels labeled `2` make up the second object, and so on. The size of the label matrix must be the same as the size of the 2-D input image.

Dependencies

To enable this port, select **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Label Numbers — Label values of ROIs
vector

Label values of ROIs, specified as an *M*-element vector. Use this vector to specify which objects the block calculates statistics for. Each element of the vector must correspond to a labeled object in the label matrix. *M* must be less than or equal to the number of objects in the label matrix.

Dependencies

To enable this port, select **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Output

Out — Mean value of input
scalar | vector | matrix | *N*-D array

Mean value of the input, returned as a scalar, vector, matrix, or *N*-D array. The size of this output depends on the size of the input and the parameter settings of the block. This port is unnamed unless you select the **Output flag indicating if ROI is within image bounds** or **Output flag indicating if input label numbers are valid** parameter

Compute Mean Value of Input Array

The block computes the mean value along the specified dimension of the input or across the entire input. The block returns output as:

- Scalar — The input is of any size, and the **Find the mean value over** parameter is set to `Entire input`.
- Vector — The input is a matrix, and the **Find the mean value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`. If `Specified dimension` is selected, the value of the **Dimension** parameter must be either `1` or `2`.
- (*N*–1)-D array — The input is an *N*-D array, the **Find the mean value over** parameter is set to `Specified dimension`, and the value of the **Dimension** parameter is *N*.
- *N*-D array with one singleton dimension — The input is an *N*-D array, and the **Find the mean value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`. If `Specified dimension` is selected, the value of the **Dimension** parameter must be an integer less than *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - An 1-by-*N*-by-*P* array if you set the **Find the mean value over** parameter to `Entire row`.
  - An *M*-by-1-by-*P* array if you set the **Find the mean value over** parameter to `Entire column`.
  - An *M*-by-*N* matrix if you set the **Find the mean value over** parameter to `Specified dimension` and the **Dimension** parameter to `3`.

Compute Mean Value of Sequence of Inputs

Select the **Running mean** parameter. The block finds the mean value of each element of the input across the entire sequence and compiles them into a single array. The output is of the same size as the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Flag — Validation of ROI
scalar | vector

Validation of ROI, returned as a scalar or vector of logical values. If the **ROI type** parameter is set to `Rectangles` or `Lines`, the output signifies whether or not the specified ROIs lie within the input image, either completely or partially. If the **ROI type** parameter is set to `Label matrix`, the output signifies whether or not the label numbers specified in the **Label Numbers** input are present in the input label matrix. The size of this output depends on whether the **Output** parameter is set to `Individual statistics for each ROI` or `Single statistic for all ROIs`.

ROI type	`Individual statistics for each ROI`	`Single statistic for all ROIs`

`Rectangles` or `Lines`	The port returns an *M*-element vector of logical values, where *M* is the number of regions specified at the **ROI** port. When an ROI is either completely or partially inside the input image, the corresponding element of the vector is `1`. When the ROI is completely outside the input image, the corresponding element of the vector is `0`.	The port returns a logical scalar. The scalar is `0` when all the ROIs are completely outside the input image. Otherwise, the scalar is `1`.
`Label matrix`	The port returns an *M*-element vector of logical values, where *M* is the number of elements in the input specified at the **Label Numbers** port. When a specified label number is present or absent in the label matrix, the corresponding element of the vector is `1` or `0` respectively.	The port returns a logical scalar. The scalar is `0` when all elements of the vector specified at the **Label Numbers** port are absent from the label matrix. Otherwise. the scalar is `1`.

Note If an ROI is partially outside the image, the block computes the mean values for only the portion of the ROI that lies within the image bounds.

Dependencies

To enable this port, set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`. For either `Rectangles` or `Lines`, select the **Output flag indicating if ROI is within image bounds** parameter. For Label matrix, select the **Output flag indicating if input label numbers are valid** parameter.

Data Types: `Boolean`

Parameters

Main

Running mean — Running mean values

`off` (default) | `on`

Select this parameter to compute running mean values across a sequence of inputs over a period of time.

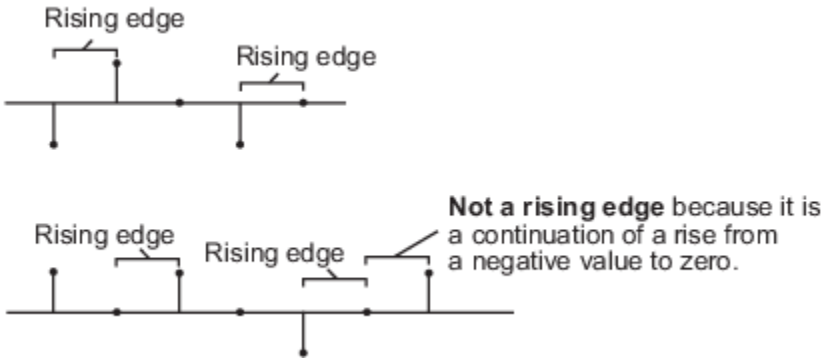
Reset port — Reset event

`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

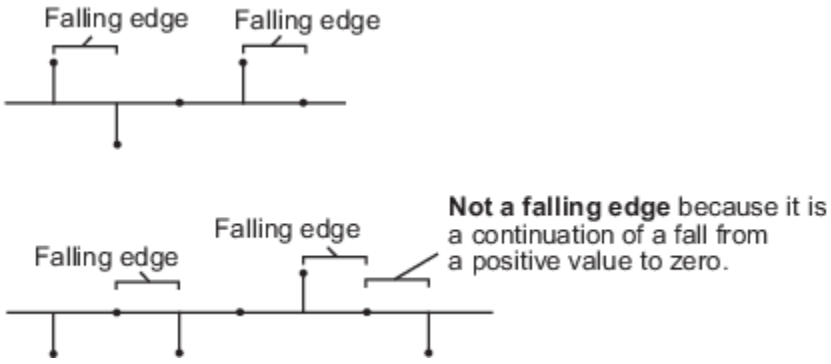
Specifies what the block detects as a reset event. The block resets the running mean when a reset event occurs at the **Rst** port. The reset sample time must be a positive integer and a multiple of the input sample time.

Specify the reset event as one of these options.

- `None` — Disable the **Rst** port.
- `Rising edge` — Trigger a reset event when the **Rst** input does one of the following.
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` — Trigger a reset event when the **Rst** input does one of the following.
  - Falls from a positive value to either a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` — Trigger a reset event when the **Rst** input is either a rising edge or a falling edge.
- `Non-zero sample` — Trigger a reset event at each sample time that corresponds to a non-zero **Rst** input.

Note When running simulations in Simulink® multitasking mode, reset signals have a one-sample latency. In this case, when the block detects a reset event, there is a one-sample delay at the **Rst** port before the block applies the reset.

Dependencies

To enable this parameter, select the **Running mean** parameter.

Find the mean value over — Dimension along which mean is computed

`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input along which the block computes the mean.

- `Entire input` — Computes mean over the entire input.
- `Each row` — Computes mean over each row.
- `Each column` — Computes mean over each column.
- `Specified dimension`— Computes mean over the dimension specified in the **Dimension** parameter.
  - If the **Dimension** parameter is set to `1`, the output is the same as when `Each column` is selected.
  - If the **Dimension** parameter is set to `2`, the output is the same as when `Each row` is selected.

Dependencies

To enable this parameter, clear the **Running mean** parameter.

Dimension — Custom dimension

`1` (default) | positive scalar

Specify the dimension (one-based value) of the input array over which the block computes the mean. The value of this parameter must not exceed the number of dimensions in the input array.

Dependencies

To enable this parameter, set the **Find the mean value over** parameter to `Specified dimension`.

Enable ROI processing — Compute mean value within ROI

`off` (default) | `on`

Select to compute the mean within the specified ROIs in the image.

Dependencies

To enable this parameter, set the **Find the mean value over** parameter to `Entire input`.

ROI type — Type of ROI

`Rectangles` (default) | `Lines` | `Label matrix` | `Binary mask`

Specify the ROI format that represents the regions in the image over which to compute the mean. The type of ROIs can be a rectangles, lines, a label matrix, or a binary mask.

ROI type	Inputs to the ROI Port	Description

`Rectangles`	• Four-element row vector [*x y width height*] • *M*-by-4 matrix: $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$, where *M* is the number of rectangular ROIs. Each row of the matrix corresponds to a different rectangle.	• *x* and *y* are the one-based coordinates of the upper left corner of the rectangle. • *width* and *height* are the width and height, in pixels, of the rectangle. The values of *width* and *height* must be greater than zero.
`Lines`	• Four-element row vector [x_1 y_1 x_2 y_2] • *M*-by-4 matrix: $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$ where *M* is the number of linear ROIs. Each row of the matrix corresponds to a different line.	• x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.
`Label matrix`	*M*-by-*N* matrix	Matrix of the same size as the input image. The matrix contains label values that represent different objects in an image. The pixels labeled `0` represent the background. The pixels labeled `1` make up one object, the pixels labeled `2` make up a second object, and so on.
`Binary mask`	*M*-by-*N* matrix	Matrix of the same size as the input image. The binary mask classifies image pixels as belonging to either the region of interest or the background. Binary mask values of `1` indicate that the corresponding image pixel belongs to the ROI. Binary mask values of `0` indicate that the corresponding image pixel is part of the background.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter.

ROI portion to process — Portion of ROI for which to compute mean

`Entire ROI` (default) | `ROI perimeter`

Specify the portion of the ROI for which the block computes the 2-D mean.

- `Entire ROI` — The block computes the mean value over the entire region of the rectangular ROI.
- `ROI perimeter` — The block computes the mean value along the perimeter of the rectangular ROI.

Dependencies

To enable this parameter, set the **ROI type** parameter to `Rectangles`.

Output — Compute individual or universal mean for ROIs

`Individual statistics for each ROI` (default) | `Single statistic for all ROIs`

Specify whether to calculate the 2-D mean individually for each ROI or across all ROIs.

- To compute separate mean values for each ROI, set this parameter to `Individual statistics for each ROI`. The block outputs a vector of mean values corresponding to different ROIs. The length of the output vector is equal to the number of ROIs.
- To compute the universal mean value across all specified ROIs, set this parameter to `Single statistic for all ROIs`. The block outputs a scalar representing the mean value across all specified ROIs.

Dependencies

To enable this parameter, set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Output flag indicating if ROI is within image bounds — Enable flag output port

`off` (default) | `on`

Select to enable the Flag output port.

Dependencies

To enable this parameter, set the **ROI type** parameter to `Rectangles` or `Lines`.

Output flag indicating if input label numbers are valid — Enable flag output port
`off` (default) | `on`

Select to enable the Flag output port.

Dependencies

To enable this parameter, set the **ROI type** parameter to `Label matrix`.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

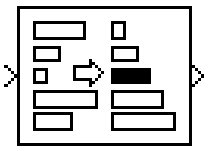
2-D Minimum | 2-D Maximum | 2-D Median

Functions

`mean`

2-D Median

2-D Median values of input array



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Median block computes the median value of an input array. The median value of an array is calculated by using these steps.

1. The values in the matrix are sorted in numeric order.
2. To find the median, of an array, *A*, with a total number of elements, *p*, where *p* is:
   - Odd — Median = *A*(`ceil`(*p* / 2))
   - Even — Median = (*A* (*p* / 2) + *A* ((*p* / 2) + 1)) / 2

Ports

Input

Port_1 — Input array
scalar | vector | matrix | *N*-D array

Specify the Input array as a vector, matrix, or *N*-D array.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Output

Port_1 — Output median values
scalar | vector | matrix | *N*-D array

The median values are returned as scalar, vector, matrix, or*N* array. The size of the output and how median is calculated depends on the setting of the **Find the median value over** parameter.

- For *M*-by-*N*,the input array, the value of the **Find the median value over** parameter determines the output form in this manner.
  - `Entire input` —The output is the median value of all the elements of the*M*-by-*N* input array as a scalar
  - `Each row` — The output is a *M*-by-1 column vector, where each element in the vector is the median value of the corresponding row in the input array.
  - `Each column` — The output is a 1-by-*N* row vector, where each element in the vector is the median value of the corresponding column in the input array.

  - `Specified dimension` — If the **Dimension** parameter is set to 1, the output is the same when you select `Each column`. If the **Dimension** parameter is set to 2, the output is the same as when you select `Each row`. If the **Dimension** parameter is set to 3, the output at each sample time is an *M*-by-*N* matrix containing the median value of each vector over the third dimension of the input
- For an *N*-D array, the value of the **Find the median value over** parameter determines the output form in this manner.
  - `Entire input` — The output is the median value of all elements of the *M*-by-*N*-by-*P* input array as scalar.
  - `Each row` — The output is a *M*-by-1-by-*P* array, where each element contains the median value of each row vector in each plane.
  - `Each column` — The output dimension is a 1-by-*N*-by-*P* array, where each element contains the median value of each column vector for each plane.
  - `Specified dimension` — The output depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output dimension is a *M*-by-*N* matrix containing the median value of each vector over the third dimension of the input

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Parameters

Main

Sort algorithm — Algorithm to sort input array
`Quick sort` (default) | `Insertion sort`

Specify the algorithm by which to sort the input array.

- `Quick sort` — This sorting algorithm uses a recursive sort method and is faster at sorting more than 32 elements.
- `Insertion sort` — This sorting algorithm uses a non-recursive method and is faster at sorting fewer than 32 elements.

When you generate code, to avoid recursive function calls, use the `Insertion sort` algorithm.

Find the median value over — Dimension in which to compute median
`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input in which the block computes the median.

- `Entire input` — Computes the median over the entire input. The output value is scalar
- `Each row` — Computes the median over each row of the input. The output is column vector
- `Each column` — Computes the median over each column of the input. The output is row vector
- `Specified dimension` — Computes the median over the dimension of the input specified in the **Dimension** parameter.
  - If **Dimension** is 1, the output is the same as when you select `Each column`
  - If **Dimension** is 2, the output is the same as when you select `Each row`

Dimension — Custom dimension
1 (default) | positive scalar

Specify the dimension (one-based value) of the input array, over which the block computes the median. The value of this parameter must not exceed the number of dimensions in the input array.

Dependencies

To enable this parameter, set the **Find the median value over** parameter to `Specified dimension`.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Option to prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Generated code relies on `memcpy` or `memset` functions (string.h) under certain conditions.

See Also

Blocks

2-D Variance | 2-D Minimum | 2-D Maximum | 2-D Mean | 2-D Standard Deviation

Functions
`median`

2-D Maximum

Compute maximum value of input or sequence of inputs



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Maximum block identifies the index of the largest element in an input array. The input can be a 1-D vector, 2-D matrix, or an *N-D*-array. The block can compute the maximum value along a specified dimension of the input or the entire input. The 2-D Maximum block can also track the maximum values in a sequence of inputs over a period of time.

Ports

Input

In — Input array
vector | matrix | *N-D*- array

Input array, specified as a vector, matrix, or *N-D*- array. This port is unnamed until you select the **Enable ROI processing** parameter or set the **Mode** parameter to `Running`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`
Complex Number Support: Yes

Rst — Reset port
scalar

Reset port, specified as a scalar. This port specifies the event that causes the block to reset the running maximum. The sample time of the **Rst** input must be a positive integer and a multiple of the input sample time.

Dependencies

To enable this port, set the **Mode** parameter to `Running` and set the **Reset port** parameter to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero sample`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

ROI — Region of interest
four-element vector | *m*-by-4 matrix | *M*-by-*N* matrix

Region of interest (ROI), specified as a four-element vector, *m*-by-4 matrix, or *M*-by-*N* matrix. The input value to this port depends on the **ROI type** parameter.

Note You can use the `ROI` port only if the input is a matrix.

Dependencies

To enable this port, set the **Find the maximum value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Label — Label matrix
matrix of nonnegative integers

Label matrix, specified as a matrix of nonnegative integers. The label matrix represents the objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on. The size of the label matrix must be the same size as the 2-D input.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Label Numbers — Label values of ROIs
M-element vector

Label values of ROIs, specified as a *M*-element vector. The value of *M* can be less than or equal to the number of objects that are labeled in the label matrix.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Output

Val — Output maximum value
scalar | vector | matrix | *N-D*-array

Output maximum value, returned as a scalar, vector, matrix, or *N-D*-array. The size of the returned maximum value depends on the size of the input, and the settings for the **Find the maximum value over** parameter.

This port is unnamed if **Mode** is set to `Running`.

Compute Maximum Value of Input Array

The **Mode** parameter is set to `Value and Index` or `Value`. In this case, you can compute the maximum value along any specified dimension of the input or the entire input. The output maximum value is a:

- Scalar if the input is of any size and **Find the maximum value over** parameter is set to `Entire input`.
- Vector if the input is a matrix and the **Find the maximum value over** parameter is set to any one of `Each row`, `Each column`, and `Specified dimension`. In this case, the **Dimension** value for `Specified dimension` can be either 1 or 2.

- (*N-1*)-D array if the input is an *N*-D array and the **Find the maximum value over** parameter is set to `Specified dimension` and the **Dimension** value is *N*.
- *N*-D array with one singleton dimension if the input is an *N*-D array and the **Find the maximum value over** parameter is set to either of `Each row`, `Each column`, and `Specified dimension`.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the maximum value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the maximum value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the maximum value over** parameter to `Specified dimension` and the **Dimension** value to 3.

Compute Maximum Value of Sequence of Inputs

When you set the **Mode** parameter to `Running`, the block computes the maximum value relative to each sample in the input sequence. The output is of the same size as the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Idx — Indices of maximum values in the input array
scalar | vector | matrix | *N-D*-array

Indices of maximum values in the input array, returned as a scalar, vector, matrix, or *N-D*-array. The size of the returned indices depends on the size of the input, and the settings for the **Find the maximum value over** parameter.

To enable this port set the **Mode** parameter to `Value and Index` or `Index`. The indices returned at the output is a:

- Scalar if the input is a N- element vector and **Find the maximum value over** parameter is set to `Entire input`.
- Vector if the input is a matrix and the **Find the maximum value over** parameter is set to any one of `Entire input`, `Each row`, `Each column`, and `Specified dimension`. In this case, the **Dimension** value for `Specified dimension` can be either 1 or 2.
- (*N-1*)-D array if the input is a *N*-D array and the **Find the maximum value over** parameter is set to `Specified dimension` and the **Dimension** value is *N*.
- *N*-D array with one singleton dimension if the input is an *N*-D array. In this case the **Find the maximum value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`. In this case, the **Dimension** value for `Specified dimension` is an integer less than *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the maximum value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the maximum value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the maximum value over** parameter to `Specified dimension` and the **Dimension** value to 3.

Note When a maximum value occurs more than once, the computed index corresponds to the first occurrence. For example, in the input vector `[3 2 1 2 3]`, the maximum value is 3 and the one-based index of the maximum value is `1`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `enumerated` | `bus`

Flag — Validation for ROI
0 | 1

Validation for ROI, returned as 0 or 1. The output value signifies if all of the ROIs specified at the input lie within the image bounds. The output value depends on the values of the **Output** and **ROI Type** parameters.

ROI Type	Output	Output from Flag port	Description
• `Rectangles` • `Lines`	`Individual statistics for each ROI`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
	`Single statistic for all ROIs`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
`Label matrix`	`Individual statistics for each ROI`	0	Label number is not in the label matrix.
		1	Label number is in the label matrix.
	`Single statistic for all ROIs`	0	None of the label numbers are in the label matrix.
		1	At least one of the label numbers is in the label matrix.

Note If the ROI is partially outside the image, the block computes the variance values only for the portion of the ROI that lies within the image bounds.

Dependencies

To enable this port, select the **Output flag indicating if ROI is within image bounds** parameter and set the value of **ROI type** parameter to `Rectangle`, `Lines`, or `Label Matrix`.

Parameters

Main Tab

Mode — Output mode
`Value and Index` (default) | `Value` | `Index` | `Running`

Specify the type of output to be returned by the block. Select the output mode as:

- `Value and Index` to return the maximum values and the corresponding indices in the given input.
- `Value` to return only the computed maximum values.
- `Index` to return only the indices of the maximum values in the given input.
- `Running` to return the maximum values in a sequence of inputs.

Index Base — Index origin
`One` (default) | `Zero`

Specify the index for the first element in the input array. Select the index origin as:

- `One` for one-based numbering. The range of the index values for the elements in an N-D input array is set to (1, N). For example, the index value for the first element in the matrix is set to (1,1).
- `Zero` for zero-based numbering. The range of the index value for the elements in an N-D input array is set to (0, N-1). For example, the index value for the first element in the matrix is set to (0,0).

Find the maximum value over — Dimension along which maximum is computed
`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input along which the block computes the maximum.

- `Entire input` — Computes maximum over the entire input.
- `Each row` — Computes maximum over each row.
- `Each column` — Computes maximum over each column.
- `Specified dimension`— Computes maximum over the dimension specified in the **Dimension** parameter.
  - If **Dimension** is 1, the output is the same as when you select `Each column`.
  - If **Dimension** is 2, the output is the same as when you select `Each row`.

Dependencies

To enable this parameter, set the **Mode** parameter to `Value and Index`, `Value`, or `Index`.

Dimension — Custom dimension
1 (default) | scalar

Specify the dimension of the input array over which the maximum is computed as a one-based value. The value of this parameter must be greater than 0 and less than or equal to the number of dimensions in the input array.

Dependencies

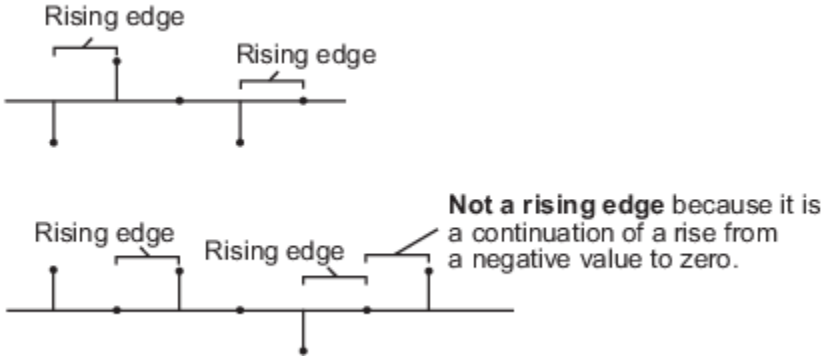
To enable this parameter, set the **Find the maximum value over** parameter to `Specified dimension`.

Reset port — Reset event
`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

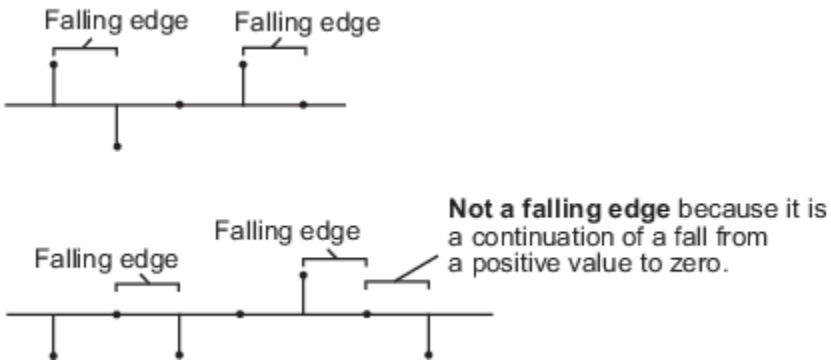
Specify when the block detects a reset event. The block resets the running mode when a reset event is detected at the optional **Rst** port. The reset sample time must be a positive integer and a multiple of the input sample time.

Specify the reset event as:

- `None` to disable the **Rst** port.
- `Rising edge` to trigger a reset event when the **Rst** input does one of the following:
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` to trigger a reset event when the **Rst** input does one of the following:
  - Falls from a positive value to either a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` to trigger a reset event when the **Rst** input is a `Rising edge` or `Falling edge`.
- `Non-zero sample` to trigger a reset event at each sample time, when the **Rst** input is not zero.

Note When running simulations in the Simulink multitasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

Dependencies

To enable this parameter, set the **Mode** parameter to `Running`.

ROI Processing

Enable ROI processing — Compute maximum value within a particular region
off (default) | on

Select to calculate maximum within a particular ROI in the image.

Note Full ROI processing is available only if you have a Computer Vision Toolbox license. If you do not have a Computer Vision Toolbox license, you can still use ROI processing, but the **ROI type** is limited to `Rectangles`.

Dependencies

To enable this parameter, set the **Find the maximum value over** parameter to `Entire input`.

ROI type — Type of ROI
`Rectangles` (default) | `Lines` | `Label matrix` | `Binary mask`

Specify the type of ROI that represents the regions in the image over which the block computes the variance. The type of ROI can be a rectangle, line, label matrix, or a binary mask.

Parameters		**Description**
ROI type	**Inputs to the ROI port**	
`Rectangles`	• Four-element row vector `[x y width height]` • *m*-by-4 matrix $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$, where *m* is the number of rectangles. Each row of the matrix corresponds to a different rectangle.	• `x` and `y` are the one-based coordinates of the upper left corner of the rectangle. • `width` and `height` are the width and height, in pixels, of the rectangle. The values of `width` and `height` must be greater than 0.

`Lines`	• Four-element row vector `[x1 y1 x2 y2]` • *m*-by-4 matrix $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$, where *m* is the number of lines. Each row of the matrix corresponds to a different line.	• x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.
`Label matrix`	*M*-by-*N* matrix	Matrix of the same size as the input image. The matrix contains label values that represent different objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on.
`Binary mask`	*M*-by-*N* matrix	Matrix of the same size as the input image. The binary mask classifies image pixels as belonging to either the region of interest or the background. The mask pixel values of 1 indicate that the image pixel belongs to the ROI. The mask pixel values of 0 indicate that the image pixel is part of the background.

Dependencies

To enable this parameter, set the **Find the maximum value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

ROI portion to process — Portion of ROI in which to calculate maximum
`Entire ROI` (default) | `ROI perimeter`

Specify the portion of the ROI from which the block has to calculate 2-D maximum. The ROI portion is either the entire ROI or the ROI perimeter.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`.

Output — Calculate maximum for individual or entire ROI
`Individual statistics for each ROI` (default) | `Single statistic for all ROIs`

Specify whether to calculate 2-D maximum individually for each ROI or for the entire ROI.

- If you select `Individual statistics for each ROI`, the block outputs a vector of maximum values. The size of the output vector is equal to the number of ROIs.
- If you select `Single statistic for all ROIs`, the block outputs a scalar value. The scalar value represents the statistical value for all the specified ROIs.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Output flag indicating if ROI is within image bounds — Enable Flag port
off (default) | on

Select to expose the **Flag** port. For a description of the **Flag** port output, see “Flag” on page 1-0 .

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
off (default) | on

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

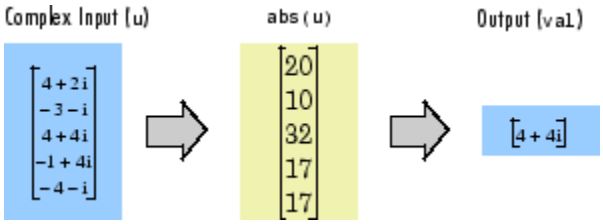
Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Maximum value of complex inputs

For complex inputs, the block computes the squared magnitude value for the elements in the input array. The squared magnitude value for a complex number $u = a + bi$ is computed as $a^2 + b^2$. The array element with maximum squared magnitude is the maximum value of the input array.



Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

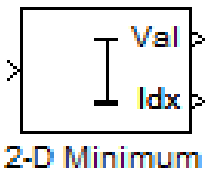
See Also

Blocks

2-D Mean | 2-D Minimum | 2-D Variance

2-D Minimum

Find minimum values in input or sequence of inputs



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Minimum block identifies the value, and optionally the position, of the smallest element in the input. The input can be a vector, a matrix, or an *N*-D array. The block identifies the minimum value either along a specified dimension of the input or across the entire input. It also tracks the minimum values in a sequence of inputs over a period of time when the **Mode** parameter is set to `Running`.

Ports

Input

In — Input array
scalar | vector | matrix | *N*-D array

Input array, specified as a vector, matrix, or *N*-D array.

Dependencies

The port is named only when you either select **Enable ROI processing** parameter or set the **Mode** parameter to `Running`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`
Complex Number Support: Yes

Rst — Reset running minimum
scalar

Reset the running minimum, specified as a scalar. This port specifies the event that causes the block to reset the running minimum. The sample time of the **Rst** input must be a positive integer and a multiple of the block input sample time.

Dependencies

To enable this port, set the **Mode** parameter to `Running` and set the **Reset port** parameter to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero sample`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

ROI — Region of interest
four-element vector | *m*-by-4 matrix | *M*-by-*N* matrix

Region of interest (ROI), specified as a four-element vector, *m*-by-4 matrix, or *M*-by-*N* matrix. This port accepts different input values depending on the setting of the **ROI type** parameter.

Note

- You can use the **ROI** port only if the input to the **In** port is a 2-D image.
- You cannot use the **ROI** port if the **Mode** parameter is set to `Running`.

Dependencies

To enable this port, set the **Find the minimum value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Label — Label matrix
matrix of nonnegative integers

Label matrix, specified as a matrix of nonnegative integers. The label matrix represents the objects in a 2-D image. The pixels labeled `0` are the background. The pixels labeled `1` make up the first object, the pixels labeled `2` make up the second object, and so on. The size of the label matrix must be same as the size of the 2-D input image.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Note You cannot enable the **Label** port if the **Mode** parameter is set to `Running`.

Data Types: `uint8` | `uint16` | `uint32`

Label Numbers — Label values of ROIs
M-element vector

Label values of ROI, specified as an *M*-element vector. This represents the object names for the corresponding numbers in the label matrix. *M* must be less than or equal to the number of objects in the label matrix.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Note You cannot enable the `Label Numbers` port if the **Mode** parameter is set to **Running**.

Data Types: `uint8` | `uint16` | `uint32`

Output

Val — Minimum values of input
scalar | vector | matrix | *N*-D array

Minimum values of the input, returned as a scalar, vector, matrix, or *N*-D array. The size of this output depends on the size of the input, and the settings of the **Mode** and **Find the minimum value over** parameters.

Note This port is unnamed if the **Mode** parameter is set to `Running`. It doesn't appear if the **Mode** parameter is set to `Index`.

Compute Minimum Value of Input Array

Set the **Mode** parameter to `Value and Index` or `Value`. The block computes the minimum value along the specified dimension of the input or across the entire input. The size of the output minimum value depends on the size of the input and the setting of the **Find the Minimum value over** parameter.

- Scalar — The input is of any size, and the **Find the minimum value over** parameter is set to `Entire input`.
- Vector — The input is a matrix, and the **Find the minimum value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`. If `Specified dimension` is selected, the value of the **Dimension** parameter must be either `1` or `2`.
- (*N*–*1*)-D array — The input is an *N*-D array, the **Find the minimum value over** parameter is set to `Specified dimension`, and the value of the **Dimension** parameter is *N*.
- *N*-D array with one singleton dimension — The input is an *N*-D array, and the **Find the minimum value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`. If `Specified dimension` is selected, the value of the **Dimension** parameter must be an integer less than *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the minimum value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the minimum value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the minimum value over** parameter to `Specified dimension` and the **Dimension** parameter to `3`.

Compute Minimum Value of Sequence of Inputs

Set the **Mode** parameter to `Running`. The block finds the minimum value of all the inputs in the given sequence and compiles them into a single array. The output is of the same size as the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Idx — Indices of minimum values in the input
scalar | vector | matrix | *N*-D array

Indices of the minimum values in the input array, returned as a scalar, vector, matrix, or *N*-D array.

To enable this port, set the **Mode** parameter to `Value and Index` or `Index`.The size of the output depends on the size of the input and the setting of the **Find the minimum value over** parameter.

- Scalar — The input is an *N*-element vector, and the **Find the minimum value over** parameter is set to `Entire input`.
- Vector — The input is a matrix, and the **Find the minimum value over** parameter is set to `Entire input`, `Each row`, `Each column`, or `Specified dimension`. If `Specified dimension` is selected, the value of the **Dimension** parameter must be either `1` or `2`.
- (*N*–*1*)-D array — The input is an *N*-D array, the **Find the minimum value over** parameter is set to `Specified dimension`, and the value of the **Dimension** parameter is *N*.
- *N*-D array with one singleton dimension — The input is an *N*-D array, and the **Find the minimum value over** parameter is set to `Each row`, `Each column`, or `Specified dimension`.If `Specified dimension` is selected, the value of the **Dimension** parameter must be an integer less than *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the minimum value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the minimum value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the minimum value over** parameter to `Specified dimension` and the **Dimension** parameter to `3`.

Note When a minimum value occurs more than once, the computed index corresponds to the first occurrence. For example, if the input vector is `[3 2 1 2 1]`, then the minimum value is `1` and the one-based index of the minimum value is `3`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Flag — ROI Validation
scalar | vector

ROI Validation, returned as a scalar or vector of logical `1`s (`true`) or `0`s (`false`). If the **ROI type** parameter is set to `Rectangles` or `Lines`, the output signifies whether or not the specified ROIs lie completely or partially within the input image. If the **ROI type** parameter is set to `Label matrix`, the output signifies whether or not the label numbers specified in the **Label Numbers** input are present in the input label matrix.

ROI type	Individual statistics for each ROI	Single statistics for all ROIs
`Rectangles`	The port returns an *m* element vector, where*m* is the number of rows in the *m*-4 matrix in the input to the **ROI** port. Each element of this vector is a `1` or `0`, indicating that the rectangular ROI is either completely or partially inside the input image (`1`) or completely outside the input image (`0`).	The port returns a scalar. The scalar is a `1` or `0`, indicating that any of the rectangles in the **ROI** input is present completely or partially inside (`1`) or completely outside (`0`) the input image.

`Lines`	The port returns a scalar. The scalar is a `1` or `0`, indicating that the input given to the **ROI** port is either completely or partially inside (`1`) the input image or completely outside (`0`) the input image.	The port returns a scalar. The scalar is a `1` or `0`, indicating that the input given to the **ROI** port is either completely or partially inside (`1`) the input image or completely outside (`0`) the input image.
`Label matrix`	The port returns an *M*-element vector, where *M* is the number of elements in the input to the **Label Numbers** port. Each element of the vector is a `1` or `0`, indicating that the associated label from the **Label Numbers** input is present in (`1`) or absent from (`0`) the Label Matrix input.	The port returns a scalar. The scalar is a `1` or `0`, indicating that any of the labels in the **Label Numbers** input are present in (`1`) or all are absent from (`0`) the Label Matrix input.

Note If the ROI is partially outside the image, the block computes the statistical values for only the portion of the ROI that lies within the image bounds.

Dependencies

To enable this port, select the **Output flag indicating if ROI is within image bounds** parameter and set the value of the **ROI type** parameter to `Rectangle`, or `Lines`, or select the **Output flag indicating if input label numbers are valid** parameter and set the value of **ROI type** parameter to`Label matrix`.

Parameters

Main

Mode — Output mode
`Value and Index` (default) | `Value` | `Index` | `Running`

Specify the output mode of the block as one of these options.

- `Value and Index` — Return both the minimum values and their corresponding indices in the given input.
- `Value` — Return only the minimum values.
- `Index` — Return only the indices of the minimum values in the given input.
- `Running` — Tracks the minimum values in a sequence of inputs.

Index Base — Index for first element
`One` (default) | `Zero`

Specify the index for the first element in the input array.

- `One` for one-based numbering. The range of index values for each dimension is 1 to *m*, where *m* is the length of that dimension. For example, the index of the first element in a matrix is (`1,1`).

- `Zero` for zero-based numbering. The range of index values for each dimension is 0 to *m*–1, where *m* is the length of that dimension. For example, the index of the first element in a matrix is (`0,0`).

Find the minimum value over — Dimension along which minimum is computed
`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input along which the block computes the minimum.

- `Entire input` — Computes minimum over the entire input.
- `Each row` — Computes minimum over each row.
- `Each column` — Computes minimum over each column.
- `Specified dimension`— Computes minimum over the dimension specified in the **Dimension** parameter.
  - If the **Dimension** parameter is set to `1`, the output is the same as when `Each column` is selected.
  - If the **Dimension** parameter is set to `2`, the output is the same as when `Each row` is selected.

Dependencies

To enable this parameter, set the **Mode** parameter to `Value and Index`, `Value`, or `Index`.

Dimension — Custom dimension
`1` (default) | positive scalar

Specify the dimension of the input array over which the block computes the minimum as a one-based value. The value of this parameter must be greater than zero and less than or equal to the number of dimensions in the input array.

Dependencies

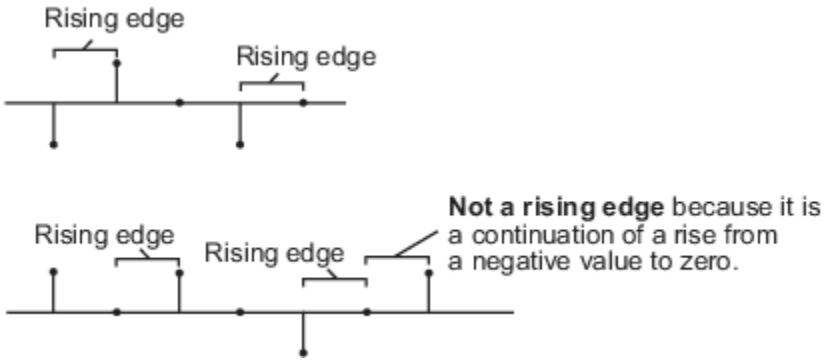
To enable this parameter, set the **Find the minimum value over** parameter to `Specified dimension`.

Reset port — Reset event
`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

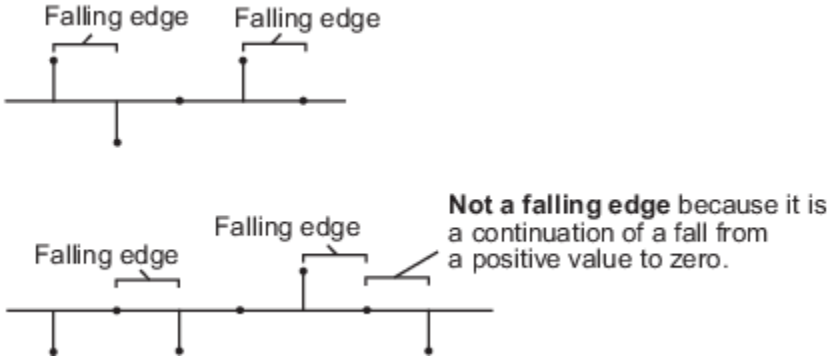
Specifies what the block detects as a reset event. The block resets the running minimum when a reset event is detected at the **Rst** port. The reset sample time must be a positive integer and a multiple of the input sample time.

Specify the reset event as one of these options.

- `None` — Disable the **Rst** port.
- `Rising edge` — Trigger a reset event when the **Rst** input does one of the following.
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` — Trigger a reset event when the **Rst** input does one of the following.
  - Falls from a positive value to either a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` — Trigger a reset event when the **Rst** input is either a `Rising edge` or a `Falling edge`.
- `Non-zero sample` — Trigger a reset event at each sample time, where the **Rst** input is not zero.

Note When running simulations in Simulink multitasking mode, reset signals have a one-sample latency. In this case, when the block detects a reset event, there is a one-sample delay at the **Rst** port rate before the block applies the reset.

Dependencies

To enable this parameter, set the **Mode** parameter to `Running`.

Enable ROI processing — Compute minimum value within ROI
`off` (default) | `on`

Select to calculate the minimum within an ROI in the image.

Note Full ROI processing is available only if you have a Computer Vision Toolbox license. If you do not have a Computer Vision Toolbox license, you can still use ROI processing, but the **ROI type** parameter is limited to `Rectangles`.

Dependencies

To enable this parameter, set the **Find the minimum value over** parameter to `Entire input`.

ROI type — Format of ROI
`Rectangles` (default) | `Lines` | `Label matrix` | `Binary mask`

Specify the ROI format that represents the regions in the image over which to compute the minimum. The type of ROI can be a rectangle, line, label matrix, or a binary mask.

ROI type	Inputs to the ROI port	Description
`Rectangles`	• Four-element row vector [*x y width height*] • *m*-by-4 matrix : $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$, where *m* is the number of rectangle ROIs. Each row of the matrix corresponds to a different rectangle.	• *x* and *y* are the one-based coordinates of the upper left corner of the rectangle. • *width* and *height* are the width and height, in pixels, of the rectangle. The values of *width* and *height* must be greater than zero.
`Lines`	• Four-element row vector [x_1 y_1 x_2 y_2] • *m*-by-4 matrix $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$, where *m* is the number of lines. Each row of the matrix corresponds to a different line.	• x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.
`Label matrix`	*M*-by-*N* matrix	Matrix of the same size as the input image. The matrix contains label values that represent different objects in an image. The pixels labeled `0` are the background. The pixels labeled `1` make up one object, the pixels labeled `2` make up a second object, and so on.

`Binary mask`	*M*-by-*N* matrix	Matrix of the same size as the input image. The binary mask classifies image pixels as belonging to either the region of interest or the background. Binary mask values of `1` indicate that the corresponding image pixel belongs to the ROI. Binary mask values of `0` indicate that the corresponding image pixel is part of the background.

Dependencies

To enable this parameter, set the **Find the minimum value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

ROI portion to process — Portion of ROI in which to calculate minimum
`Entire ROI` (default) | `ROI perimeter`

Specify the portion of the ROI for which the block calculates the 2-D minimum.

- `Entire ROI` — The block computes the minimum value over the entire region of the rectangular ROI.
- `ROI perimeter` — The block computes the minimum value along the perimeter of the rectangular ROI.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`.

Output — Calculate individual or universal minimum for ROIs
`Individual statistics for each ROI` (default) | `Single statistic for all ROIs`

Specify whether to calculate the 2-D minimum individually for each ROI or across all ROIs.

- If you select `Individual statistics for each ROI`, the block outputs a vector of minimum values, each element representing an ROI. The size of the output vector is equal to the number of ROIs.
- If you select `Single statistic for all ROIs`, the block outputs a scalar value. The scalar value is the minimum value across all specified ROIs.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Output flag indicating if ROI is within image bounds — Enable Flag port
`off` (default) | `on`

Select to enable the Flag output port.

Note The name of this parameter changes to **Output flag indicating if input label numbers are valid** when the **ROI type** parameter is set to `Label matrix`.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, or `Lines`.

Data Types

For details on the fixed-point block parameters, see "Specify Fixed-Point Attributes for Blocks" (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see "Lock the Output Data Type Setting" (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

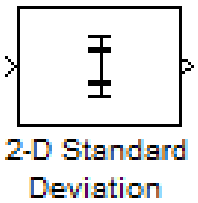
See Also

Blocks

2-D Mean | 2-D Variance | 2-D Maximum

2-D Standard Deviation

Compute standard deviation of input or sequence of inputs



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Standard Deviation block computes the standard deviation of an input array. The input can be a 1-D vector, 2-D matrix, or an *N-D*-array. The block can compute standard deviation along a specified dimension of the input or the entire input. If you select the **Running standard deviation** parameter, the block can also track the standard deviation in a sequence of inputs over a time period.

Ports

Input

In — Input array
vector | matrix | *N-D*- array

Input array, specified as a vector, matrix, or *N-D*- array. This port is unnamed until you select the **Enable ROI processing** parameter.

Data Types: `single` | `double`

Rst — Reset port
scalar

Reset port, specified as a scalar. This port specifies the event that causes the block to reset the running standard deviation. The sample time of the **Rst** input must be a positive integer and a multiple of the input sample time.

Dependencies

To enable this port, select the **Running standard deviation** parameter and set the **Reset port** parameter to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero sample`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

ROI — Region of interest
four-element vector | *m*-by-4 matrix | *M*-by-*N* matrix

Region of interest (ROI), specified as a four-element vector, *m*-by-4 matrix, or *M*-by-*N* matrix. The input value to this port depends on the **ROI type** parameter.

Note You can use the `ROI` port only if the input is a 2-D image.

Dependencies

To enable this port, set the **Find the standard deviation value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Label — Label matrix
matrix of nonnegative integers

Label matrix, specified as a matrix of nonnegative integers. The label matrix represents the objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on. The size of the label matrix must be the same size as the 2-D input.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Label Numbers — Label values of ROIs
M-element vector

Label values of ROIs, specified as a *M*-element vector. The value of *M* can be less than or equal to the number of objects that are labeled in the label matrix.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Output

Out — Computed output standard deviation
scalar | vector | matrix | *N-D*-array

Computed output standard deviation, returned as a scalar, vector, matrix, or *N-D*-array. The size of the returned output standard deviation depends on the size of the input, and the settings for the **Running standard deviation** and **Find the standard deviation value over** parameters.

This port is unnamed until you select the **Output flag indicating if ROI is within image bounds** and the **ROI type** parameter is set to `Rectangles`, `Lines`, or `Label matrix`.

If Running standard deviation Is Cleared

You can compute standard deviation along any specified dimension of the input or the entire input. The output is a:

- Scalar if the input is of any size and the **Find the standard deviation value over** parameter is set to `Entire input`.
- Vector if the input is a matrix and the **Find the standard deviation value over** parameter is set to any one of `Each row`, `Each column`, and `Specified dimension`. In this case, the **Dimension** value for `Specified dimension` can be either 1 or 2.

- Matrix if the input is a 3-D array and the **Find the standard deviation value over** parameter is set to `Specified dimension` and the **Dimension** value is 3.
- *N-D* array if the input is an *N-D* array and the **Find the standard deviation value over** parameter is set to an option other than `Entire input`.

  If you compute standard deviation along the *N*th dimension of the input, then the returned output is an *(N-1)-D* array. In this case, the **Dimension** value for `Specified dimension` is set to *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the standard deviation value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the standard deviation value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the standard deviation value over** parameter to `Specified dimension` and the **Dimension** value to 3.

If Running standard deviation Is Selected

When you select the **Running standard deviation** parameter, the block computes the standard deviation of each sample in the input with respect to all previous samples. The output is of the same size as the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Flag — Validation for ROI
0 | 1

Validation for ROI, returned as 0 or 1. The output value signifies if all of the ROIs specified at the input lie within the image bounds. The output value depends on the values of the **Output** and **ROI Type** parameters.

ROI Type	Output	Output from Flag port	Description
• `Rectangles` • `Lines`	`Individual statistics for each ROI`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
	`Single statistic for all ROIs`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
`Label matrix`	`Individual statistics for each ROI`	0	Label number is not in the label matrix.
		1	Label number is in the label matrix.
	`Single statistic for all ROIs`	0	None of the label numbers are in the label matrix.

		1	At least one of the label numbers is in the label matrix.

Note If the ROI is partially outside the image, the block computes the standard deviation values only for the portion of the ROI that lies within the image bounds.

Dependencies

To enable this port, select the **Output flag indicating if ROI is within image bounds** parameter and set the value of **ROI type** parameter to `Rectangle`, `Lines`, or `Label Matrix`.

Parameters

Main Tab

Running standard deviation — Track running standard deviation
off (default) | on

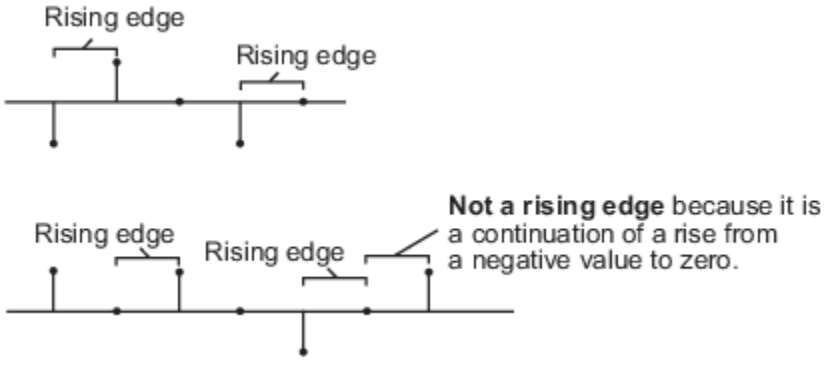
Select to track the standard deviation of successive inputs to the block. In this mode, the block treats each element as a channel.

Reset port — Reset event
`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

Specify when the block detects a reset event. The block resets the running standard deviation when a reset event is detected at the optional **Rst** port. The reset sample time must be a positive integer and a multiple of the input sample time.

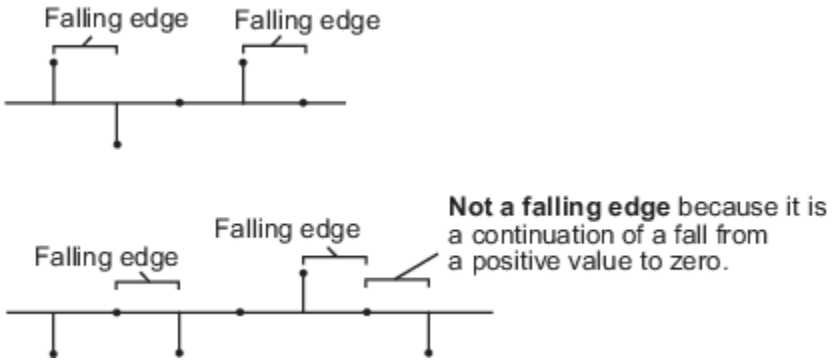
Specify the reset event as:

- `None` to disable the **Rst** port.
- `Rising edge` to trigger a reset event when the **Rst** input does one of the following:
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` to trigger a reset event when the **Rst** input does one of the following:

- Falls from a positive value to either a negative value or zero
- Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` to trigger a reset event when the **Rst** input is a `Rising edge` or `Falling edge`.
- `Non-zero sample` to trigger a reset event at each sample time, when the **Rst** input is not zero.

Note When running simulations in the Simulink multitasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

Dependencies

To enable this parameter, select the **Running standard deviation** parameter.

Find the standard deviation value over — Dimension along which standard deviation is computed
`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input along which the block computes the standard deviation.

- `Entire input` — Computes standard deviation over the entire input.
- `Each row` — Computes standard deviation over each row.
- `Each column` — Computes standard deviation over each column.
- `Specified dimension`— Computes standard deviation over the dimension specified in the **Dimension** parameter. By default, the **Dimension** parameter is set to 1 and the output is the same as when you select `Each column`.

Dependencies

To enable this parameter, clear the **Running standard deviation** parameter.

Dimension — Custom dimension
1 (default)

Specify the dimension of the input array over which the standard deviation is computed as a one-based value. The value of this parameter is set to 1..

Dependencies

To enable this parameter, set the **Find the standard deviation value over** parameter to `Specified dimension`.

ROI Processing

Enable ROI processing — Compute standard deviation within a particular region
off (default) | on

Select to calculate standard deviation within a particular ROI in the image.

Note Full ROI processing is available only if you have a Computer Vision Toolbox license. If you do not have a Computer Vision Toolbox license, you can still use ROI processing, but the **ROI type** is limited to `Rectangles`.

Dependencies

To enable this parameter, set the **Find the standard deviation value over** parameter to `Entire input`.

ROI type — Type of ROI
`Rectangles` (default) | `Lines` | `Label matrix` | `Binary mask`

Specify the type of ROI that represents the regions in the image over which the block computes the standard deviation. The type of ROI can be a rectangle, line, label matrix, or a binary mask.

Parameters		**Description**
ROI type	**Inputs to the ROI port**	
`Rectangles`	• Four-element row vector `[x y width height]` • *m*-by-4 matrix $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$, where *m* is the number of rectangles. Each row of the matrix corresponds to a different rectangle.	• `x` and `y` are the one-based coordinates of the upper left corner of the rectangle. • `width` and `height` are the width and height, in pixels, of the rectangle. The values of `width` and `height` must be greater than 0.

`Lines`	• Four-element row vector `[x1 y1 x2 y2]` • *m*-by-4 matrix $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$, where *m* is the number of lines. Each row of the matrix corresponds to a different line.	• x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.
`Label matrix`	*M*-by-*N* matrix	Matrix of the same size as the input image. The matrix contains label values that represent different objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on.
`Binary mask`	*M*-by-*N* matrix	Matrix of the same size as the input image. The binary mask classifies image pixels as belonging to either the region of interest or the background. The mask pixel values of 1 indicate that the image pixel belongs to the ROI. The mask pixel values of 0 indicate that the image pixel is part of the background.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter.

ROI portion to process — Portion of ROI in which to calculate standard deviation
`Entire ROI` (default) | `ROI perimeter`

Specify the portion of the ROI from which the block has to calculate 2-D standard deviation. The ROI portion is either the entire ROI or the ROI perimeter.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`.

Output — Calculate standard deviation for individual or entire ROI
`Individual statistics for each ROI` (default) | `Single statistic for all ROIs`

Specify whether to calculate 2-D standard deviation individually for each ROI or for the entire ROI.

- If you select `Individual statistics for each ROI`, the block outputs a vector of standard deviation values. The size of the output vector is equal to the number of ROIs.
- If you select `Single statistic for all ROIs`, the block outputs a scalar value. The scalar value represents the statistical value for all the specified ROIs.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Output flag indicating if ROI is within image bounds — Enable Flag port
off (default) | on

Select to expose the **Flag** port. For a description of the **Flag** port output, see "Flag" on page 1-0 .

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Data Types Tab

For details on the fixed-point block parameters, see "Specify Fixed-Point Attributes for Blocks" (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
off (default) | on

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see "Lock the Output Data Type Setting" (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced in R2011b

Extended Capabilities

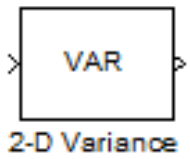
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

2-D Mean | `var` | 2-D Variance

2-D Variance

Compute variance of input or sequence of inputs



Libraries:
Computer Vision Toolbox / Statistics

Description

The 2-D Variance block computes the unbiased variance of an input array. The input can be a 1-D vector, 2-D matrix, or an *N-D* array. The block can compute variance along a specified dimension of the input or the entire input. If you select the **Running variance** parameter, the block can also track the variance in a sequence of inputs over a time period.

Ports

Input

In — Input array
vector | matrix | array

Input array, specified as a vector, matrix, or *N-D*- array. This port is unnamed until you select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Rst — Reset port
scalar

Reset port, specified as a scalar. This port specifies the event that causes the block to reset the running variance. The sample time of the **Rst** input must be a positive integer and a multiple of the input sample time.

Dependencies

To enable this port, select the **Running variance** parameter and set the **Reset port** parameter to `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero sample`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

ROI — Region of interest
vector | matrix

Region of interest (ROI), specified as a four-element vector, *m*-by-4 matrix, or *M*-by-*N* matrix. The input value to this port depends on the **ROI type** parameter.

Note You can use the `ROI` port only if the input is a 2-D image.

Dependencies

To enable this port, set the **Find the variance value over** parameter to `Entire input` and select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Label — Label matrix
matrix

Label matrix, specified as a matrix of nonnegative integers. The label matrix represents the objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on. The size of the label matrix must be the same size as the 2-D input.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Label Numbers — Label values of ROIs
vector

Label values of ROIs, specified as a *M*-element vector. The value of *M* can be less than or equal to the number of objects that are labeled in the label matrix.

Dependencies

To enable this port, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Label matrix`.

Data Types: `uint8` | `uint16` | `uint32`

Output

Out — Computed output variance
scalar | vector | matrix | *N-D*-array

Computed output variance, returned as a scalar, vector, matrix, or *N-D*-array. The size of the returned output variance depends on the size of the input, and the settings for the **Running variance** and **Find the variance value over** parameters.

This port is unnamed until you select the **Output flag indicating if ROI is within image bounds** and the **ROI type** parameter is set to `Rectangles`, `Lines`, or `Label matrix`.

If Running variance Is Cleared

You can compute variance along any specified dimension of the input or the entire input. The output is a:

- Scalar if the input is of any size and the **Find the variance value over** parameter is set to `Entire input`.
- Vector if the input is a matrix and the **Find the variance value over** parameter is set to any one of `Each row`, `Each column`, and `Specified dimension`. In this case, the **Dimension** value for `Specified dimension` can be either 1 or 2.

- Matrix if the input is a 3-D array and the **Find the variance value over** parameter is set to `Specified dimension` and the **Dimension** value is 3.
- *N-D* array if the input is an *N-D* array and the **Find the variance value over** parameter is set to an option other than `Entire input`.

  If you compute variance along the *N*th dimension of the input, then the returned output is an *(N-1)-D* array. In this case, the **Dimension** value for `Specified dimension` is set to *N*.

  Example: For a 3-D input array of size *M*-by-*N*-by-*P*, the dimension of the returned output is:

  - 1-by-*N*-by-*P* if you set the **Find the variance value over** parameter to `Entire row`.
  - *M*-by-1-by-*P* if you set the **Find the variance value over** parameter to `Entire column`.
  - *M*-by-*N* if you set the **Find the variance value over** parameter to `Specified dimension` and the **Dimension** value to 3.

If Running variance Is Selected

When you select the **Running variance** parameter, the block computes the variance of each sample in the input with respect to all previous samples. The output is of the same size as the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Flag — Validation for ROI
`0` | `1`

Validation for ROI, returned as 0 or 1. The output value signifies if all of the ROIs specified at the input lie within the image bounds. The output value depends on the values of the **Output** and **ROI Type** parameters.

ROI Type	Output	Output from Flag port	Description
• `Rectangles` • `Lines`	`Individual statistics for each ROI`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
	`Single statistic for all ROIs`	0	ROI is completely outside the input image.
		1	ROI is either completely or partially inside the input image.
`Label matrix`	`Individual statistics for each ROI`	0	Label number is not in the label matrix.
		1	Label number is in the label matrix.
	`Single statistic for all ROIs`	0	None of the label numbers are in the label matrix.
		1	At least one of the label numbers is in the label matrix.

Note If the ROI is partially outside the image, the block computes the variance values only for the portion of the ROI that lies within the image bounds.

Dependencies

To enable this port, select the **Output flag indicating if ROI is within image bounds** parameter and set the value of **ROI type** parameter to `Rectangle`, `Lines`, or `Label Matrix`.

Parameters

Main

Running variance — Track running variance
`off` (default) | `on`

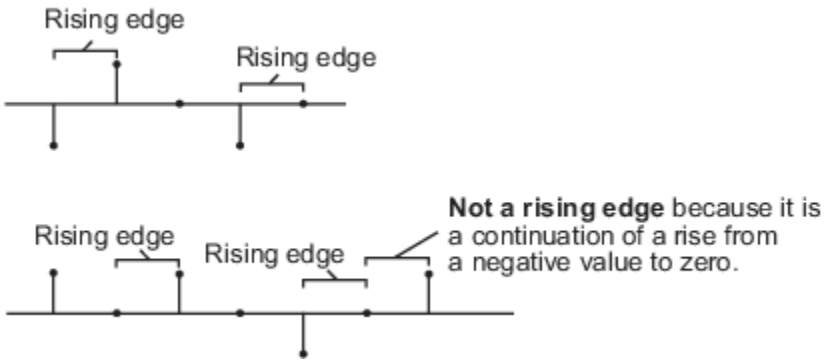
Select to track the variance of successive inputs to the block. In this mode, the block treats each element as a channel.

Reset port — Reset event
`None` (default) | `Rising edge` | `Falling edge` | `Either edge` | `Non-zero sample`

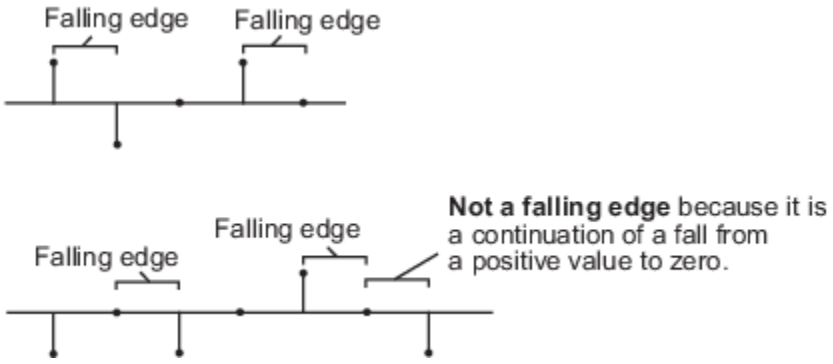
Specify when the block detects a reset event. The block resets the running variance when a reset event is detected at the optional **Rst** port. The reset sample time must be a positive integer and a multiple of the input sample time.

Specify the reset event as:

- `None` to disable the **Rst** port.
- `Rising edge` to trigger a reset event when the **Rst** input does one of the following:
  - Rises from a negative value to either a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero



- `Falling edge` to trigger a reset event when the **Rst** input does one of the following:
  - Falls from a positive value to either a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero



- `Either edge` to trigger a reset event when the **Rst** input is a `Rising edge` or `Falling edge`.
- `Non-zero sample` to trigger a reset event at each sample time, when the **Rst** input is not zero.

Note When running simulations in the Simulink multitasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset.

Dependencies

To enable this parameter, select the **Running variance** parameter.

Find the variance value over — Dimension along which variance is computed
`Entire input` (default) | `Each row` | `Each column` | `Specified dimension`

Specify the dimension of the input along which the block computes the variance.

- `Entire input` — Computes variance over the entire input.
- `Each row` — Computes variance over each row.
- `Each column` — Computes variance over each column.
- `Specified dimension`— Computes variance over the dimension specified in the **Dimension** parameter.
  - If **Dimension** is 1, the output is the same as when you select `Each column`.
  - If **Dimension** is 2, the output is the same as when you select `Each row`.

Dependencies

To enable this parameter, clear the **Running variance** parameter.

Dimension — Custom dimension
1 (default) | scalar

Specify the dimension of the input array over which the variance is computed as a one-based value. The value of this parameter must be greater than 0 and less than or equal to the number of dimensions in the input array.

Dependencies

To enable this parameter, set the **Find the variance value over** parameter to `Specified dimension`.

ROI Processing

Enable ROI processing — Compute variance within a particular region
`off` (default) | `on`

Select to calculate variance within a particular ROI in the image.

Note Full ROI processing is available only if you have a Computer Vision Toolbox license. If you do not have a Computer Vision Toolbox license, you can still use ROI processing, but the **ROI type** is limited to `Rectangles`.

Dependencies

To enable this parameter, set the **Find the variance value over** parameter to `Entire input`.

ROI type — Type of ROI
`Rectangles` (default) | `Lines` | `Label matrix` | `Binary mask`

Specify the type of ROI that represents the regions in the image over which the block computes the variance. The type of ROI can be a rectangle, line, label matrix, or a binary mask.

Parameters		**Description**
ROI type	**Inputs to the ROI port**	
`Rectangles`	• Four-element row vector `[x y width height]` • *m*-by-4 matrix $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$, where *m* is the number of rectangles. Each row of the matrix corresponds to a different rectangle.	• `x` and `y` are the one-based coordinates of the upper left corner of the rectangle. • `width` and `height` are the width and height, in pixels, of the rectangle. The values of `width` and `height` must be greater than 0.

`Lines`	• Four-element row vector `[x1 y1 x2 y2]` • *m*-by-4 matrix $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$, where *m* is the number of lines. Each row of the matrix corresponds to a different line.	• x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.
`Label matrix`	*M*-by-*N* matrix	Matrix of the same size as the input image. The matrix contains label values that represent different objects in an image. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on.
`Binary mask`	*M*-by-*N* matrix	Matrix of the same size as the input image. The binary mask classifies image pixels as belonging to either the region of interest or the background. The mask pixel values of 1 indicate that the image pixel belongs to the ROI. The mask pixel values of 0 indicate that the image pixel is part of the background.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter.

ROI portion to process — Portion of ROI in which to calculate variance
`Entire ROI` (default) | `ROI perimeter`

Specify the portion of the ROI from which the block has to calculate 2-D variance. The ROI portion is either the entire ROI or the ROI perimeter.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`.

Output — Calculate variance for individual or entire ROI
`Individual statistics for each ROI` (default) | `Single statistic for all ROIs`

Specify whether to calculate 2-D variance individually for each ROI or for the entire ROI.

- If you select `Individual statistics for each ROI`, the block outputs a vector of variance values. The size of the output vector is equal to the number of ROIs.
- If you select `Single statistic for all ROIs`, the block outputs a scalar value. The scalar value represents the statistical value for all the specified ROIs.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Output flag indicating if ROI is within image bounds — Enable Flag port
`off` (default) | `on`

Select to expose the **Flag** port. For a description of the **Flag** port output, see “Flag” on page 1-0 .

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter and set the **ROI type** parameter to `Rectangles`, `Lines`, or `Label matrix`.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

More About

Variance

The variance of an input is the square of the standard deviation of the input. Variance gives a measure of deviation of the input values from its mean value.

For a purely real or imaginary input, u, of size M-by-N, the variance is:

$$y = \sigma^2 = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\left|u_{ij}\right|^2 - \frac{\left|\sum_{i=1}^{M}\sum_{j=1}^{N}u_{ij}\right|^2}{M*N}}{M*N-1},$$

where:

- u_{ij} is the input data element at (i, j).
- M is the length of the jth column.
- N is the number of columns.

For complex inputs, the variance is:

$$\sigma^2 = \sigma_{\mathrm{Re}}{}^2 + \sigma_{\mathrm{Im}}{}^2,$$

where:

- $\sigma_{Re}{}^2$ is the variance of the real part of the complex input.
- $\sigma_{Im}{}^2$ is the variance of the imaginary part of the complex input.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

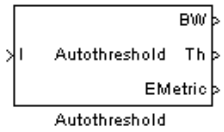
Generate C and C++ code using Simulink® Coder™.

See Also

2-D Mean | 2-D Standard Deviation | `var`

Autothreshold

Convert intensity image to binary image



Libraries:
Computer Vision Toolbox / Conversions

Description

The Autothreshold block converts an intensity image to a binary image using a threshold value. This block uses Otsu's method to determine the threshold.

Ports

Input

I — Input image
matrix

Input image, specified as an *M*-by-*N* matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Output

BW — Output binary image
matrix

Output binary image, returned as an *M*-by-*N* matrix.

Data Types: `Boolean`

Th — Threshold value
scalar

Threshold value used to convert the intensity image to a binary image, returned as a scalar.

The block calculates the threshold value using Otsu's method. The block computes this value by splitting the histogram of the input image such that the variance of each pixel group is minimized.

Dependencies

To enable this port, select the **Output threshold** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

EMetric — Effectiveness metric value
scalar

Effectiveness metric value, returned as a scalar in the range [0, 1].

If all the pixel values of the input image are the same, the value of the effectiveness metric is `0`. If the image has two unique pixels, or if the histogram of the image is symmetric, the value of the effectiveness metric is `1`.

Dependencies

To enable this port, select the **Output effectiveness metric** parameter.

Data Types: `single` | `double`

Parameters

Main

Data Types

Main

Thresholding operator — Specify thresholding operator

`>` (default) | `<=`

Select whether the block includes pixels from the intensity image with values below the threshold or those with values equal to or greater than the threshold in the output binary image.

- `>` — For pixel values of the input image greater than the threshold, the block outputs `1` in the binary image. Otherwise, the block assigns the pixel a value of `0`.
- `<=` — For pixel values of the input image less than or equal to the threshold, the block outputs `1`. Otherwise, the block assigns the pixel a value of `0`.

Output threshold — Output threshold

`off` (default) | `on`

Select **Output threshold** to return the threshold value the block uses to convert the intensity image to a binary image. Selecting this parameter enables the **Th** output port.

Output effectiveness metric — Output effectiveness metric

`off` (default) | `on`

Select **Output effectiveness metric** to return the effectiveness metric value. Selecting this parameter enables the **EMetric** output port.

Specify data range — Specify input pixel value range

`off` (default) | `on`

Select **Specify data range** to enable specification of minimum and maximum input pixel values to use to determine the threshold. Selecting this parameter enables the **Minimum value of input**, **Maximum value of input**, and **When data range is exceeded parameters**.

Minimum value of input — Minimum input pixel value

`0` (default) | scalar

Specify the minimum value of input pixels to use to determine the threshold.

Maximum value of input — Maximum input pixel value

`255` (default) | scalar

Specify the maximum value of input pixels to use to determine the threshold.

When data range is exceeded — Operation to perform when data range is exceeded

`Ignore` (default) | `Saturate` | `Warn and saturate` | `Error`

Specify the behavior of the block when the value of an input image pixel exceeds the specified data range.

- `Ignore` — The block ignores all pixels with values outside the specified data range.
- `Saturate` — The block saturates the pixels with values below the data range to the specified minimum value, and those above the data range to the specified maximum value.
- `Warn and saturate` — The block displays a warning message in the MATLAB command window and saturates the pixel values as specified for the `Saturate` option.
- `Error` — The block displays an error message and ends the simulation.

Scale threshold — Specify scale threshold

`off` (default) | `on`

Select **Scale threshold** to enable specification of a scale factor for the thresholding operation. Selecting this parameter enables the **Threshold scaling factor** parameter.

Threshold scaling factor — Scale factor for thresholding

`1` (default) | scalar

Specify the scaling factor for the thresholding operation as a scalar. The block performs thresholding by multiplying the threshold value computed by Otsu's method by the specified scale factor.

Data Types

For details on fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

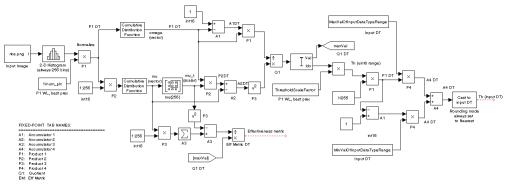
Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Fixed-Point Data Types

This diagram shows the data types used in the Autothreshold block for fixed-point signals. You can use the default fixed-point parameters if your input has a word length less than or equal to 16. In this diagram, `DT` means data type. You can set the product, accumulator, quotient, and effectiveness metric data types in the block mask.



Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

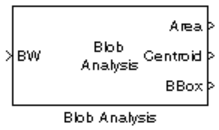
Compare To Constant | Relational Operator

Functions

`graythresh`

Blob Analysis

Statistics for labeled regions



Libraries:
Computer Vision Toolbox / Statistics

Description

The Blob Analysis block calculates statistics for labeled regions in a binary image. The block returns quantities such as the centroid, bounding box, label matrix, and blob count. The Blob Analysis block supports input and output variable-size signals. You can also use the Selector block from Simulink, to select certain blobs based on their statistics.

For information on pixel and spatial coordinate system definitions, see “Image Coordinate Systems” and “Coordinate Systems”.

Ports

Input

BW — Binary image
vector | matrix

Binary image, specified as a vector or matrix

Data Types: `Boolean`

Output

Area — Number of pixels in labeled regions
vector

Number of pixels in labeled regions, returned as a vector.

Dependencies

To enable this port, select the **Area** parameter.

Data Types: `int32`

Centroid — Centroid coordinates
M-by-2 matrix

M-by-2 matrix of *M* number of blobs.

The rows represent the coordinates of the centroid of each region, and *M* represents the number of blobs.

For example, there are two blobs where the row and column coordinates of their centroids are x_1, y_1 and x_2, y_2. The block outputs:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

at the **Centroid** port.

Dependencies

To enable this port, select the **Centroid** parameter.

Data Types: `double` | `single` | `fixed point`

BBox — Bounding box coordinates
M-by-4 matrix

Bounding box coordinates, returned as an *M*-by-4 matrix of *M* bounding boxes for blobs. Each row of the matrix defines a bounding box as a four-element vector [*x*,*y*,*width*,*height*] in pixel coordinates. The rows represent the coordinates of each bounding box, where *M* represents the number of blobs.

For example, there are two blobs, where *x* and *y* define the location of the upper-left corner of the bounding box, and *w* and *h* define the width and height of the bounding box. The block outputs

$$\begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \end{bmatrix}$$

at the **BBox** port.

Dependencies

To enable this port, select the **Bounding Box** parameter.

Parameters

Main Tab

Area — Number of pixels in labeled regions
vector

Number of pixels in labeled regions, returned as a vector.

Dependencies

To enable this port, select the **Area** parameter.

Centroid — Centroid coordinates
M-by-2 matrix

M-by-2 matrix of *M* number of blobs.

The rows represent the coordinates of the centroid of each region, and *M* represents the number of blobs.

For example, there are two blobs where the row and column coordinates of their centroids are x_1, y_1 and x_2, y_2. The block outputs:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

at the **Centroid** port.

Dependencies

To enable this port, select the **Centroid** parameter.

BBox — Bounding box coordinates
M-by-4 matrix

Bounding box coordinates, returned as an *M*-by-4 matrix of *M* bounding boxes for blobs. Each row of the matrix defines a bounding box as a four-element vector [*x*,*y*,*width*,*height*] in pixel coordinates. The rows represent the coordinates of each bounding box, where *M* represents the number of blobs.

For example, there are two blobs, where *x* and *y* define the location of the upper-left corner of the bounding box, and *w* and *h* define the width and height of the bounding box. The block outputs

$$\begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \end{bmatrix}$$

at the **BBox** port.

Dependencies

To enable this port, select the **Bounding Box** parameter.

Major axis length — Length of major axis of ellipses
off (default) | on

Enables the **Major axis length** port. Select this check box to output a vector that

- Represents the lengths of the major axes of ellipses
- Has the same normalized second central moments as the labeled regions

Minor axis length — Length of minor axis of ellipses
off (default) | on

Enables the **Minor axis length** port. Select this check box to output a vector that

- Represents the lengths of the minor axes of ellipses
- Has the same normalized second central moments as the labeled regions

Orientation — Orientation
off (default) | on

Enables the **Orientation** port.

Select this check box to output a vector that represents the angles between the major axes of the ellipses and the *x*-axis. The angle values are in radians and range between: $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ inclusive.

Eccentricity — Eccentricity
off (default) | on

Enables the **Eccentricity** port. Select this check box to output a vector that represents the eccentricities of ellipses that have the same second moments as the region.

Equivalent diameter squared — Equivalent diameter squared
off (default) | on

Enables the **Equivalent diameter squared** port. Select this check box to output a vector that represents the equivalent diameters squared.

Extent — Extent
off (default) | on

Enables the **Extent** port. Select this check box to output a vector that represents the results of dividing the areas of the blobs by the area of their bounding boxes.

Perimeter — Perimeter length
off (default) | on

Enables the **Perimeter** port. Select this check box to output an *N*-by-1 vector of the perimeter lengths, in pixels, of each blob, where *N* is the number of blobs.

Statistics output data type — Output data type
double (default) | single | specify via Data Types tab

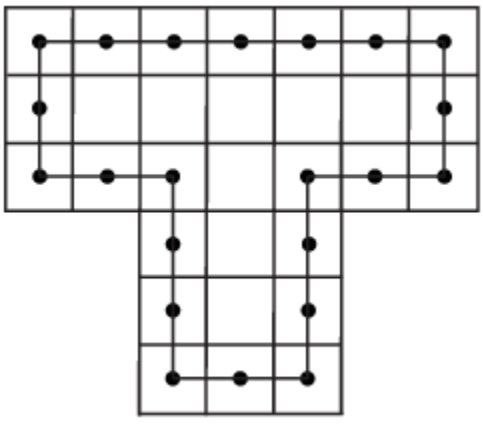
Specify the data type of the outputs as `double`, `single`, or `Specify via Data Types tab`.

Connectivity — Number of labeled regions
8 (default) | 4

Specify the number of labeled regions in each image.

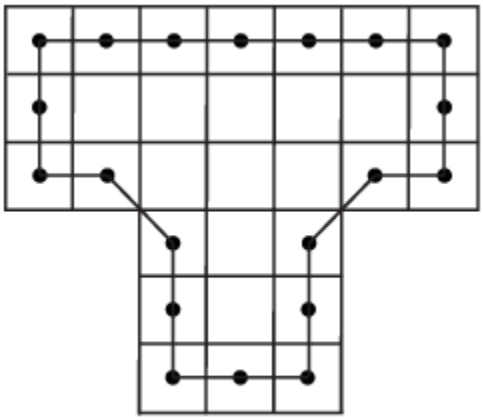
Define which pixels connect to each other. If you want to connect pixels located on the top, bottom, left, and right, select 4. If you want to connect pixels to other pixels on the top, bottom, left, right, and diagonally, select 8. For more information about this parameter, see the Label block.

The **Connectivity** parameter also affects how the block calculates the perimeter of a blob. For example, this figure illustrates how the block calculates the perimeter when you set the **Connectivity** parameter to 4.



The block calculates the distance between the center of each pixel (marked by the black dots) and estimates the perimeter to be 22.

The next figure illustrates how the block calculates the perimeter of a blob when you set the **Connectivity** parameter to 8.



The block takes a different path around the blob and estimates the perimeter to be $18 + 2\sqrt{2}$.

Output label matrix — Output label matrix
off (default) | on

Enable the **Label** port. Select this check box to output the label matrix. The pixels equal to `0` represent the background. The pixels equal to `1` represent the first object. The pixels equal to `2` represent the second object, and so on.

Blob Properties Tab

Maximum number of blobs — Maximum number of blobs
scalar

Specify the maximum number of labeled regions in each input image. The block uses this value to preallocate vectors and matrices to ensure that they are long enough to hold the statistical values. The maximum number of blobs the block outputs depends on both the value of this parameter and the size of the input image. The number of blobs the block outputs may be limited by the input image size.

Warn if maximum number of blobs is exceeded — Warn if maximum number of blobs is exceeded
on (default) | off

Select this parameter to warn if maximum number of blobs is exceeded.

Output actual number of blobs — Number of blobs
off (default) | on

Enable the **Count** port. Select this parameter to output a scalar value that represents the actual number of connected regions in each image.

Specify minimum blob area in pixels — Minimum blob area in pixels
`0` (default) | scalar

Select this parameter to specify the minimum blob area in pixels. The blob is labeled if the number of pixels meets the minimum size specified.

Tunable: Yes

Specify maximum blob area in pixels — Maximum blob area in pixels
intmax('uint32') (default) | scalar

Select this parameter to specify the maximum blob area in pixels. The blob is labeled if the number of pixels meets the maximum size specified. The maximum allowable value is the maximum of the `uint32` data type.

Tunable: Yes

Exclude blobs touching image border — Exclude blobs touching image border
off (default) | on

Specify this parameter to exclude a labeled blob that contains at least one border pixel.

Output blob statistics as a variable size signal — Output blob statistics as a variable size signal
off (default) | on

Specify this parameter to output blob statistics as a variable-size signal.

Fill empty spaces in output — Fill empty spaces in output
off (default) | on

Specify this parameter to fill empty spaces in the statistical vectors with the values you specify in the **Fill values** parameter.

Dependencies

To enable this parameter, clear the **Output blob statistics as a variable size signal** parameter.

Fill values — Scalar used to fill empty spaces in output
-1 (default) | scalar | vector

Specify a scalar value to fill all the empty spaces in the statistical vectors. If you enter a vector, it must have the same length as the number of selected statistics. The block uses each vector element to fill a different statistics vector. If the empty spaces do not affect your computation, you can clear the **Fill empty spaces in outputs** parameter.

Dependencies

To enable this parameter, clear the **Output blob statistics as a variable size signal** parameter.

Data Types Tab

To enable option on the **Data Types** tab, in the **Main** tab, set the **Statistics output data type** parameter to `Specify via Data Types tab`. For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`Boolean`[a] | `double`[b] | `fixed point`[b, c] | `integer`[b] | `single`[b]
Multidimensional Signals	`no`

Variable-Size Signals	`yes`

a Generated code will be restricted to MATLAB host computers when you set the FFT implementation parameter to FFTW, or when the transform length is not a power of two.
b This data type is supported at the output port(s).
c Fixed-point data types are not supported at the Major Axis, Minor Axis, Orientation, and Eccentricity output ports.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

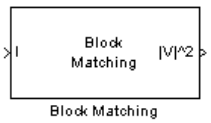
Label | `regionprops`

Topics

“Cell Counting”

Block Matching

Estimate motion between images or video frames



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Block Matching block estimates motion between two images or two video frames by comparing blocks of pixels. The block matches the block of pixels between frames by moving the block of pixels over a search region.

Ports

Input

I/I1 — Image or video frame
scalar | vector | matrix

Image or video frame, specified as a scalar, vector, or matrix. If the Estimate motion between parameter is set to Two images, the name of this port changes to **I1**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

I2 — Image or video frame
scalar | vector | matrix

Image or video frame, specified as a scalar, vector, or matrix.

Dependencies

To enable this port, set the **Estimate motion between** parameter to `Two images`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

|V|^2 — Velocity magnitudes
scalar | vector | matrix

Velocity magnitudes, returned as a scalar, vector, or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

V — Velocity components in complex form
scalar | vector | matrix

Velocity components in complex form, returned as a scalar, vector, or matrix.

Dependencies

To expose this port, set the **Velocity output** parameter to `Horizontal and vertical components in complex form`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main Tab

Estimate motion between — Estimate motion between images or frames
`Current frame and N-th frame back` (default) | `Two images`

Specify whether to estimate the motion between two images or two video frames. Select `Two images` to estimate the motion between two images. Select `Current frame and N-th frame back` to estimate the motion between two video frames that are N frames apart.

Dependencies

Set this parameter to `Current frame and N-th frame back` to expose the **N** parameter.

N — Number of frames
1 (default) | scalar

Specify a scalar value that represents the number of frames between the reference frame and the current frame.

Dependencies

To enable this parameter, set the **Estimate motion between** parameter to `Current frame and N-th frame back`.

Search method — Block match search method
`Exhaustive` (default) | `Three-step`

Specify how the block locates the block of pixels in the current frame and the next frame that best matches the block of pixels.

- `Exhaustive` — The block selects the location of the block of pixels in the next frame by moving the block over the search region one pixel at a time. This process is computationally expensive.
- `Three-step` — The block searches for the block of pixels in the next frame that best matches the block of pixels in the current frame by using a steadily decreasing step size. The block begins with a step size approximately equal to half the maximum search range. In each step, the block compares the central point of the search region to eight search points located on the boundaries of the region and moves the central point to the search point whose values is the closest to that of the central point. The block then reduces the step size by half, and begins the process again. This option is less computationally expensive, though it might not find the optimal solution.

Block matching criteria — Block matching criteria
`Mean square error (MSE)` (default) | `Mean absolute difference (MAD)`

Specify how the block measures the similarity of the block of pixels in the current frame to the block of pixels in the next frame.

If you select `Mean square error (MSE)`, the block estimates the displacement of the center pixel of the block as the (d_1, d_2) values that minimize this MSE equation:

$$MSE(d_1, d_2) = \frac{1}{N_1 \times N_2} \sum_{(n_1, n_2), \in B} \sum [s(n_1, n_2, k) - s(n_1 + d_1, n_2 + d_2, k + 1)]^2$$

In the previous equation, B is an $N_1 \times N_2$ block of pixels, and $s(x,y,k)$ denotes a pixel location at (x,y) in frame k.

If you select `Mean absolute difference (MAD)`, the Block Matching block estimates the displacement of the center pixel of the block as the (d_1, d_2) values that minimize this MAD equation:

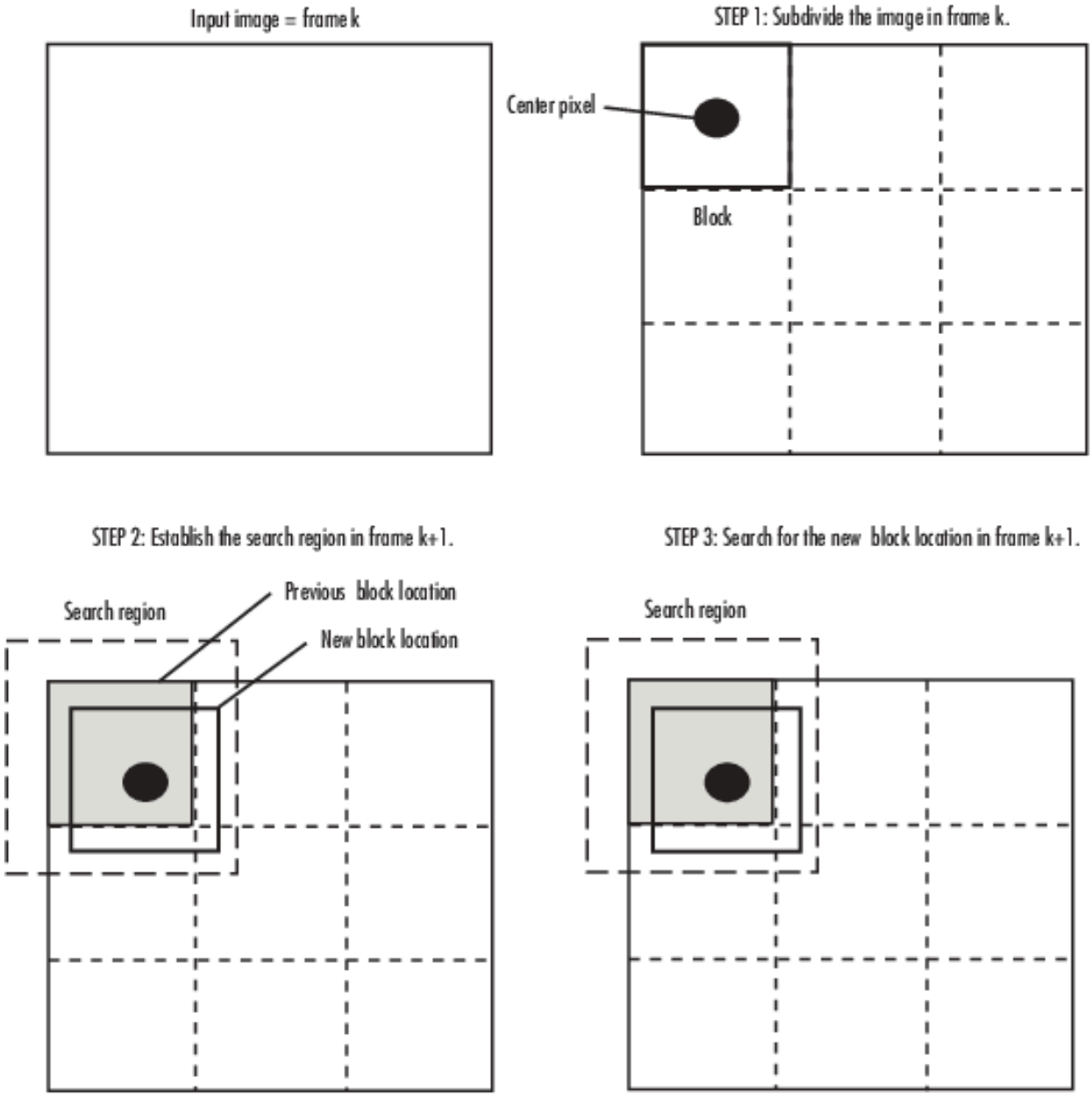
$$MAD(d_1, d_2) = \frac{1}{N_1 \times N_2} \sum_{(n_1, n_2), \in B} \sum |s(n_1, n_2, k) - s(n_1 + d_1, n_2 + d_2, k + 1)|$$

Block size [height width] — Block size
`[17 17]` (default) | two-element vector

Specify the size of the block of pixels.

Overlap size [r c] — Block overlap
`[0 0]` (default) | two-element vector

Specify the overlap (in pixels) of two subdivisions of the input image.



Maximum displacement [r c] — Maximum displacement
`[7 7]` (default) | two-element vector

Specify the maximum number of pixels any center pixel in a block of pixels might move from image to image or frame to frame. The block uses this value to determine the size of the search region.

Velocity output — Velocity output
`Magnitude-squared` (default) | `Horizontal and vertical components in complex form`

Specify the output. If you select `Magnitude-squared`, the block outputs an optical flow matrix where each element is in the form u^2+v^2. If you select `Horizontal and vertical components in complex form`, the block outputs an optical flow matrix where each element is in the form

$u + jv$. The real part of each value is the horizontal velocity component and the imaginary part of each value is the vertical velocity component.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

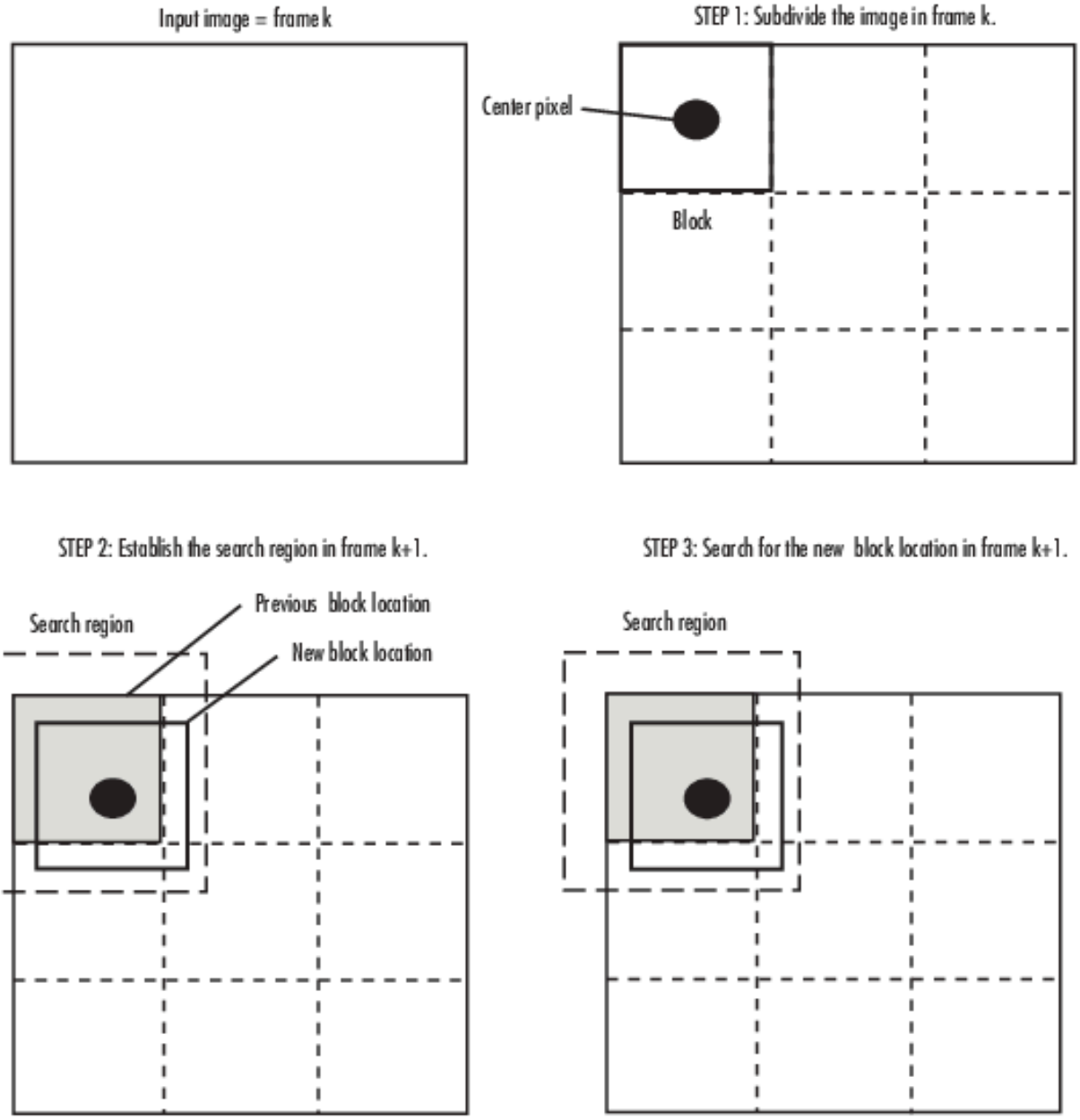
Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Algorithms

Block Matching Search

Suppose the input to the block is frame k. The block performs the following steps:

1. Subdivide the frame using the values for the **Block size [height width]** and **Overlap size [r c]** parameters. For this example, **Overlap size [r c]** equals `[0 0]`.
2. For each subdivision or block in frame k+1, a search region is established based on the value of the **Maximum displacement [r c]** parameter.
3. Search for the new block location using either the `Exhaustive` or `Three-step` search method.



Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

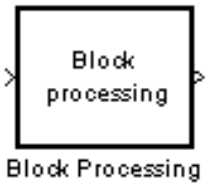
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Optical Flow | `opticalFlow` | `opticalFlowHS` | `opticalFlowLK` | `opticalFlowLKDoG`

Block Processing

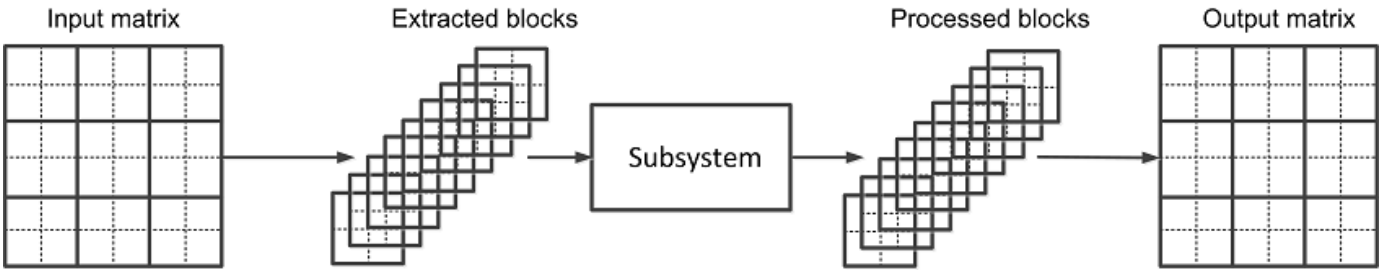
Repeat user-specified operation on blocks of input matrix



Libraries:
Computer Vision Toolbox / Utilities

Description

The Block Processing block extracts submatrices, or blocks , from an input matrix, passes each block to a subsystem that performs a user-specified operation. It then reassembles the processed blocks to form the output matrix. You can use this block to process an image block-wise, rather than processing the entire image at once. Matrix block processing proceeds from input matrix to extracted blocks, then the subsystem, and then recombines the processed blocks into an output matrix.



Ports

Input

I — Input signal
matrix

Input signal, specified as an *M*-by-*N* matrix. This port is unnamed unless you select the **Add port to supply subsystem parameters** parameter or set the **Number of inputs** parameter to 2 or more. If you select **Add port to supply subsystem parameters** parameter, the port name becomes **I**. If you set **Number of inputs** to a value greater than 1, *n*, then block has *n* input ports from **I** to **In**. The contents of the subsystem determines the data type of the input data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

P — Additional input to supply subsystem parameters
scalar | vector | matrix | array

Additional input to supply subsystem parameters, specified as a scalar, vector, matrix, or array. The contents of the subsystem determines the data type of the input data.

Dependencies

To enable this port, you must select the **Add port to supply subsystem parameters** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `string`

Output

Port_1 — Output signal
matrix

Output data, returned as a matrix. The value of the **Number of outputs** determines the number of output ports. The contents of the subsystem determine the data type of the output data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Parameters

Number of inputs — Number of inputs
`1` (default) | positive scalar

Specify the number of input ports to the Block Processing block as a positive scalar.

Add port to supply subsystem parameters — Subsystem parameter input port
`off` (default) | `on`

Select this parameter to add an additional input port to the Block Processing block for supplying subsystem parameters.

Number of outputs — Number of outputs
`1` (default) | positive scalar

Specify the number of output ports on the Block Processing block as a positive scalar.

Block size — Block size
`{[8 8]}` (default) | cell array

Specify the size of each block as a cell of two-element vectors. The cell array must contain either a single cell or one cell for each input port. If the block has one input port, enter only one vector. If the block has more than one input port, you can enter one vector to use for all inputs, or you can specify an individual vector for each input. The block uses the vectors in the order you enter them.

Overlap — Block overlap
`{[0 0]}` (default) | cell array

Specify the overlap of each block as cell of two-element vectors. The cell array must contain either a single cell or one cell for each input port. If the block has one input port, enter only one vector. If the block has more than one input port, you can enter one vector to use for all inputs, or you can specify an individual vector for each input. The block uses the vectors in the order you enter them.

Traverse order — Order of block extraction
`Row-wise` (default) | `Column-wise`

Specify the traverse order with which the Block Processing block extracts submatrices from the input matrix. If you select `Row-wise`, the block extracts submatrices by moving across the rows. If you select `Column-wise`, the block extracts submatrices by moving down the columns

Subsystem

To configure the subsystem of the Block Processing block.

- Click **Open Subsystem** to open the Block Processing subsystem.
- Drag blocks into this subsystem to define the processing operations the block performs on the submatrices.

Block Characteristics

Data Types	`Boolean`[a] | `double`[a] | `fixed point`[a] | `integer`[a] | `single`[a]
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

a The blocks inside the subsystem dictate the data types supported by this block.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

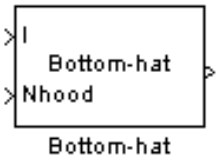
For Iterator

Functions

`blockproc`

Bottom-hat

Perform morphological bottom-hat filtering on intensity or binary images



Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Bottom-hat block performs bottom-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element. Bottom-hat filtering is the equivalent of subtracting the input image from the result of performing a morphological closing operation on the input image.

To define the structuring element that the block applies to the image, use the **Neighborhood or structuring element** parameter. Specify this element by entering a vector or matrix of `1`s and `0`s or by using the `strel` function.

This block uses two-dimensional flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of intensity values. This port is unnamed unless you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values representing a structuring element as a vector or matrix of `1`s and `0`s or by using the `strel` function.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Filtered image
matrix

The bottom-hat filtered image is returned as a matrix of intensity values. The size and data type of the filtered image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Input image type — Type of input image

`Intensity` (default) | `Binary`

Specify the type of image as one of these options:

- `Intensity` — If the input image is an intensity image, select this option.
- `Binary` — If the input image is a binary image, select this option.

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`strel('octagon',15)` (default) | vector | matrix

Specify neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object or an array of `strel` objects. When you specify this value as an array of `strel` objects, the block applies each object to the entire image in turn.

If the structuring element is decomposable into smaller elements, the block executes at higher speed due to the use of a more efficient algorithm.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter, to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Bottom-hat Transform

Image processing tasks like background equalization, feature extraction, and image enhancement can be achieved using the bottom-hat transform. This process returns an image which contains objects that are:

- Smaller in dimension than the structuring element chosen and
- Darker than the surroundings

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Erosion | Dilation | Opening | Closing | Label | Top-hat

Functions

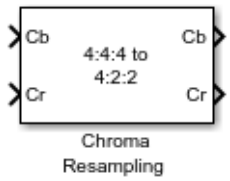
`imerode` | `imdilate` | `imopen` | `imclose` | `imbothat` | `imtophat`

Objects

`strel`

Chroma Resampling

Downsample or upsample chrominance components of images



Libraries:
Computer Vision Toolbox / Conversions

Description

The Chroma Resampling block resamples the chrominance components of an input image. Chroma resampling consists of these operations:

- Downsample — Reduces the bandwidth required for transmission or storage of a signal. You can prevent aliasing by applying a lowpass filter to bandlimit the input signal.
- Upsample — Expands the bandwidth of the signal. This block performs interpolation to approximate the missing chrominance values.

Ports

Input

Cb — Blue-difference chrominance component of image
matrix

Blue-difference chrominance component of the image, specified as an *M*-by-*D* matrix. *D* is half the width of the image.

Data Types: `single` | `double` | `uint8`

Cr — Red-difference chrominance component of image
matrix

Red-difference chrominance component of the image, specified as an *M*-by-*D* matrix. *D* is half the width of the image.

Data Types: `single` | `double` | `uint8`

Output

Cb — Blue-difference chrominance component of image
matrix

Blue-difference chrominance component of image, returned as an *M*-by-*P* matrix. *P* is half the width of the resampled image. If you downsample the input image, *P* is less than *D*. If you upsample the input image, *P* is greater than *D*.

Data Types: `single` | `double` | `int8` | `uint8`

Cr — Red-difference chrominance component of image
matrix

Red-difference chrominance component of image, returned as an *M*-by-*P* matrix. *P* is half the width of the resampled image. If you downsample the input image, *P* is less than *D*. If you upsample the input image, *P* is greater than *D*

Data Types: `single` | `double` | `int8` | `uint8`

Parameters

Resampling — Chrominance resampling method

`4:4:4 to 4:2:2` (default) | `4:4:4 to 4:2:0 (MPEG1)` | `4:4:4 to 4:2:0 (MPEG2)` | ...

Specify the resampling method to use to downsample or upsample the chrominance components of the image.

These options specify downsampling operations:

- `4:4:4 to 4:2:2`
- `4:4:4 to 4:2:0 (MPEG1)`
- `4:4:4 to 4:2:0 (MPEG2)`
- `4:4:4 to 4:1:1`
- `4:2:2 to 4:2:0 (MPEG1)`
- `4:2:2 to 4:2:0 (MPEG2)`

These options specify upsampling operations:

- `4:2:2 to 4:4:4`
- `4:2:0 (MPEG1) to 4:2:2`
- `4:2:0 (MPEG1) to 4:4:4`
- `4:2:0 (MPEG2) to 4:2:2`
- `4:2:0 (MPEG2) to 4:4:4`
- `4:1:1 to 4:4:4`

For more information, see “Chrominance Resampling Formats” on page 1-126.

Antialiasing filter — Chroma downsampling antialiasing filter

`Default` (default) | `User-defined` | `None`

Specify the lowpass filter that the block uses to prevent aliasing.

- `Default` — The block uses a built-in lowpass filter.
- `User-defined` — Specify custom filter coefficients for the block. Depending on the value of the Resampling parameter, this option enables the **Horizontal filter coefficients** parameter, the **Vertical filter coefficients** parameter, or both.
- `None` — The block does not filter the input signal.

Dependencies

To enable this parameter, set the **Resampling** parameter to one of these options:

- `4:4:4 to 4:2:2`
- `4:4:4 to 4:2:0 (MPEG1)`
- `4:4:4 to 4:2:0 (MPEG2)`
- `4:4:4 to 4:1:1`
- `4:2:2 to 4:2:0 (MPEG1)`
- `4:2:2 to 4:2:0 (MPEG2)`

.

Horizontal filter coefficients — Coefficients for horizontal lowpass filter

`[0.2 0.6 0.2]` (default) | row vector

Specify the horizontal filter coefficients as a vector of even number of elements to apply to your input signal.

Dependencies

To enable this parameter, you must:

- Set the **Resampling** parameter to one of these options:
  - `4:4:4 to 4:2:2`
  - `4:4:4 to 4:2:0 (MPEG1)`
  - `4:4:4 to 4:2:0 (MPEG2)`
  - `4:4:4 to 4:1:1`
- Set the **Antialiasing filter** parameter to `User-defined`.

Vertical filter coefficients — Coefficients for vertical lowpass filter

`[0.5 0.5]` (default) | row vector

Specify the filter coefficients as a vector of odd number of elements to apply to your input signal.

Dependencies

To enable this parameter, you must:

- Set the **Resampling** parameter to one of these options:
  - `4:4:4 to 4:2:0 (MPEG1)`
  - `4:4:4 to 4:2:0 (MPEG2)`
  - `4:2:2 to 4:2:0 (MPEG1)`
  - `4:2:2 to 4:2:0 (MPEG2)`
- Set the **Antialiasing filter** parameter to `User-defined`.

Interpolation — Chrominance upsampling interpolation method

`linear` (default) | `pixel replication`

Specify whether the block uses linear interpolation or pixel replication interpolation to approximate the missing chrominance values when upsampling an image.

- `linear` — The block uses linear interpolation to calculate the missing values.
- `pixel replication` — The block replicates the chrominance values of the neighboring pixels to assign values to the additional pixels in the upsampled image.

Dependencies

To enable this parameter, set the **Resampling** parameter to one of these options:

- `4:2:2 to 4:4:4`
- `4:2:0 (MPEG1) to 4:2:2`
- `4:2:0 (MPEG1) to 4:4:4`
- `4:2:0 (MPEG2) to 4:2:2`
- `4:2:0 (MPEG2) to 4:4:4`
- `4:1:1 to 4:4:4`

.

Input image is transposed (data order is row major) — Read data row-wise

`off` (default) | `on`

Enable this parameter to read data elements row-wise.

Block Characteristics

Data Types	`double` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Algorithms

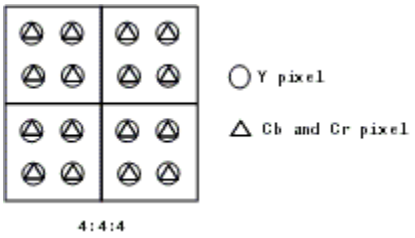
Chrominance Resampling Formats

Chrominance resampling format is represented as a three-part ratio J:a:b (for example, 4:2:2). The three-parts are represented as:

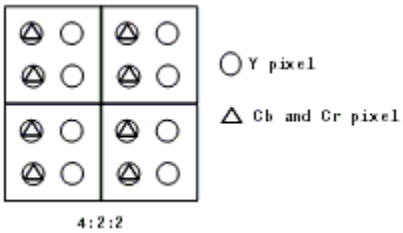
- J — Horizontal sampling reference
- a — Number of chrominance samples (Cr, Cb) in the first row of J pixels.
- b — Number of chrominance samples (Cr, Cb) in the second row of J pixels

The Chroma Resampling block supports these formats:

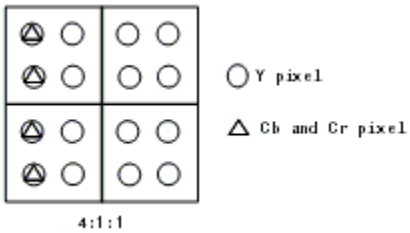
- Chrominance resampling format 4:4:4



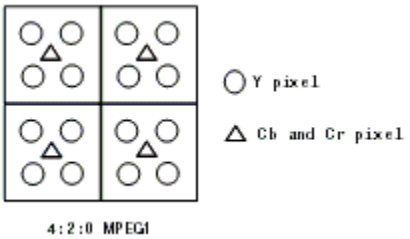
- Chrominance resampling format 4:2:2



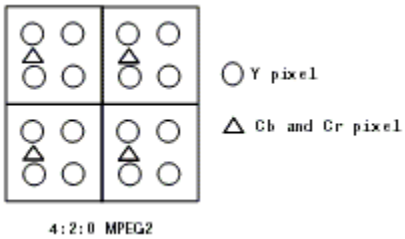
- Chrominance resampling format 4:1:1



- Chrominance resampling format 4:2:0 (MPEG1)



- Chrominance resampling format 4:2:0 (MPEG2)



Version History

Introduced before R2006a

References

[1] Haskell, Barry G., Atul Puri, and Arun N. Netravali. *Digital Video: An Introduction to MPEG-2*. Digital Multimedia Standards Series. New York: Chapman & Hall : International Thomson Pub, 1997.

[2] Wang, Yao, Jörn Ostermann, and Ya-Qin Zhang. *Video Processing and Communications*. Prentice-Hall Signal Processing Series. Upper Saddle River, N.J: Prentice Hall, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

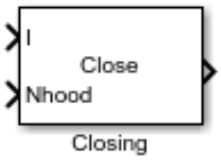
See Also

Blocks

Autothreshold | Color Space Conversion | Image Complement

Closing

Perform morphological closing on binary or intensity images



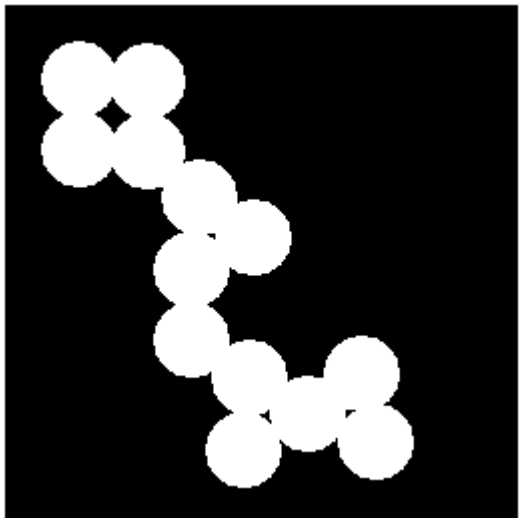
Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Closing block performs a dilation operation followed by an erosion operation on an intensity or binary image using a predefined neighborhood or structuring element.

To define the structuring element that the block applies to the image, use the **Neighborhood or structuring element** parameter. Specify this element by entering a vector or matrix of `1`s and `0`s or by using the `strel` function.

Below are an input image and the output image after the closing operation, respectively.





This block uses two-dimensional flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of intensity values. This port is unnamed unless you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values representing a structuring element as a vector or matrix of `1`s and `0`s or by using the `strel` function.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Closed image
matrix

The closed image is returned as a matrix of intensity values. The size and data type of the closed image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`strel('line',5,45)` (default) | vector | matrix

Specify neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object or an array of `strel` objects. When you specify this value as an array of `strel` objects, the block applies each object to the entire image in turn.

If the structuring element is decomposable into smaller elements, the block executes at higher speed due to the use of a more efficient algorithm.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter, to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Closing Operation

Closing smoothes the contour of an object in an image by

- Fusing narrow breaks and long thin gulfs
- Eliminating small holes and
- Filling gaps in the contour

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

[2] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Third edition. Knoxville: Gatesmark Publishing, 2020.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Erosion | Dilation | Opening | Label | Bottom-hat | Top-hat

Functions

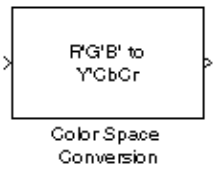
`imerode` | `imdilate` | `imopen` | `imclose`

Objects

`strel`

Color Space Conversion

Convert color space of image



Libraries:
Computer Vision Toolbox / Conversions

Description

The Color Space Conversion block converts color information between color spaces. Use the **Conversion** parameter to specify the color spaces you are converting between.

Ports

Input

Port_1 — Input image
array

Input image, specified as an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the input image.

Data Types: `single` | `double` | `uint8`

R' — Red color component of image
matrix

Red color component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **R'**, **G'**, and **B'** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `R'G'B' to Y'CbCr`
- `R'G'B' to intensity`
- `R'G'B' to HSV`
- `sR'G'B' to XYZ`
- `sR'G'B' to L*a*b*`

Data Types: `single` | `double` | `uint8`

G' — Green color component of image
matrix

Green color component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **R'**, **G'**, and **B'** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `R'G'B' to Y'CbCr`
- `R'G'B' to intensity`
- `R'G'B' to HSV`
- `sR'G'B' to XYZ`
- `sR'G'B' to L*a*b*`

Data Types: `single` | `double` | `uint8`

B' — Blue color component of image
matrix

Blue color component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **R'**, **G'**, and **B'** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `R'G'B' to Y'CbCr`
- `R'G'B' to intensity`
- `R'G'B' to HSV`
- `sR'G'B' to XYZ`
- `sR'G'B' to L*a*b*`

Data Types: `single` | `double` | `uint8`

Y' — Luma component of image
matrix

Luma component of image, specified as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `Y'CbCr to R'G'B'`.

Data Types: `single` | `double` | `uint8`

Cb — Blue-difference chroma component of image
matrix

Blue-difference chroma component of image, specified as an *M*-by-*D* matrix, where *D* is half the width of the image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `Y'CbCr to R'G'B'`.

Data Types: `single` | `double` | `uint8`

Cr — Red-difference chroma component of image
matrix

Red-difference chroma component of image, specified as an *M*-by-*D* matrix, where *D* is half the width of the image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `Y'CbCr to R'G'B'`.

Data Types: `single` | `double` | `uint8`

H — Hue component of image
matrix

Hue component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **H**, **S**, and **V** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `HSV to R'G'B'`.

Data Types: `single` | `double`

S — Saturation component of image
matrix

Saturation component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **H**, **S**, and **V** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `HSV to R'G'B'`.

Data Types: `single` | `double`

V — Brightness component of image
matrix

Brightness component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **H**, **S**, and **V** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `HSV to R'G'B'`.

Data Types: `single` | `double`

X — X component of image
matrix

X component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **X**, **Y**, and **Z** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `XYZ to sR'G'B'`.

Data Types: `single` | `double`

Y — Y component of image
matrix

Y component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **X**, **Y**, and **Z** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `XYZ to sR'G'B'`.

Data Types: `single` | `double`

Z — Z component of image
matrix

Z component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **X**, **Y**, and **Z** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `XYZ to sR'G'B'`.

Data Types: `single` | `double`

L* — Lightness component of image
matrix

Lightness component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **L***, **a***, and **b*** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `L*a*b* to sR'G'B'`.

Data Types: `single` | `double`

a* — Red-green component of image
M-by-*N* matrix

Red-green component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **L***, **a***, and **b*** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `L*a*b* to sR'G'B'`.

Data Types: `single` | `double`

b* — Yellow-blue component of image
M-by-*N* matrix

Yellow-blue component of image, specified as an *M*-by-*N* matrix.

Note Inputs to the **L***, **a***, and **b*** ports must have the same dimensions and data type as one another.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `L*a*b* to sR'G'B'`.

Data Types: `single` | `double`

Output

Port_1 — Output image
array

Output image, returned as an *M*-by-*N*-by-*P* array, where *P* is the number of color channels in the output image. For Y'CbCr output, the value range for the Y component differs from those of the Cb and Cr components.

- If the input is `uint8`, then Y'CbCr is `uint8`. `Y'` is in the range `[16, 235]`, and `Cb` and `Cr` are in the range `[16, 240]`.
- If the input is `double`, then `Y'CbCr` is `double`. `Y'` is in the range `[16/255, 235/255]`, and `Cb` and `Cr` are in the range `[16/255, 240/255]`.

Data Types: `single` | `double` | `int8` | `uint8`

R' — Red color component of image
matrix

Red color component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `Y'CbCr to R'G'B'`
- `HSV to R'G'B'`
- `XYZ to sR'G'B'`
- `L*a*b* to sR'G'B'`

.

Data Types: `single` | `double` | `int8` | `uint8`

G' — Green color component of image
matrix

Green color component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `Y'CbCr to R'G'B'`
- `HSV to R'G'B'`
- `XYZ to sR'G'B'`
- `L*a*b* to sR'G'B'`

Data Types: `single` | `double` | `int8` | `uint8`

B' — Blue color component of image
matrix

Blue color component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to one of these options:

- `Y'CbCr to R'G'B'`
- `HSV to R'G'B'`
- `XYZ to sR'G'B'`
- `L*a*b* to sR'G'B'`

.

Data Types: `single` | `double` | `int8` | `uint8`

Y' — Luma component of image
matrix

Luma component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to Y'CbCr`.

Data Types: `single` | `double` | `int8` | `uint8`

Cb — Blue-difference chroma component of image
matrix

Blue-difference chroma component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to Y'CbCr`.

Data Types: `single` | `double` | `int8` | `uint8`

Cr — Red-difference chroma component of image
matrix

Red-difference chroma component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to Y'CbCr`.

Data Types: `single` | `double` | `int8` | `uint8`

I' — Intensity image
matrix

Intensity image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to intensity`.

Data Types: `single` | `double` | `int8` | `uint8`

H — Hue component of image
matrix

Hue component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to HSV`.

Data Types: `single` | `double`

S — Saturation component of image
matrix

Saturation component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to HSV`.

Data Types: `single` | `double`

V — Brightness component of image
matrix

Brightness component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `R'G'B' to HSV`.

Data Types: `single` | `double`

X — X component of image
matrix

X component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to XYZ`.

Data Types: `single` | `double`

Y — Y component of image
matrix

Y component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to XYZ`.

Data Types: `single` | `double`

Z — Z component of image
matrix

Z component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to XYZ`.

Data Types: `single` | `double`

L* — Lightness component of image
matrix

Lightness component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to L*a*b*`.

Data Types: `single` | `double`

a* — Red-green component of image
matrix

Red-green component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to L*a*b*`.

Data Types: `single` | `double`

b* — Yellow-blue component of image
matrix

Yellow-blue component of image, returned as an *M*-by-*N* matrix.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals` and set the **Conversion** parameter to `sR'G'B' to L*a*b*`.

Data Types: `single` | `double`

Parameters

Conversion — Color space conversion

`R'G'B' to Y'CbCr` (default) | `Y'CbCr to R'G'B'` | `R'G'B' to intensity` | ...

Specify which color space to convert the image from and to using these options:

- `R'G'B' to Y'CbCr`
- `Y'CbCr to R'G'B'`
- `R'G'B' to intensity`
- `B'G'R' to intensity`
- `B'G'R' to R'G'B'`
- `R'G'B' to B'G'R'`
- `R'G'B' to HSV`
- `HSV to R'G'B'`
- `sR'G'B' to XYZ`
- `XYZ to sR'G'B'`
- `sR'G'B' to L*a*b*`
- `L*a*b* to sR'G'B'`

Note The prime notation indicates that the signal is gamma corrected.

Use conversion specified by — Color conversion standard

`Rec. 601 (SDTV)` (default) | `Rec. 709 (HDTV)`

Specify the color conversion standard for converting the image between the R'G'B' and Y'CbCr color spaces as either `Rec. 601 (SDTV)` or `Rec. 709 (HDTV)`.

Dependencies

To enable this parameter, set the **Conversion** parameter to `R'G'B' to Y'CbCr` or `Y'CbCr to R'G'B'`.

Scanning standard — Scanning standard

`1125/60/2:1` (default) | `1250/50/2:1`

Specify the scanning standard for converting the image between the R'G'B' and Y'CbCr color spaces as either `1125/60/2:1` or `1250/50/2:1`.

Dependencies

To enable this parameter, set the **Conversion** parameter to `R'G'B' to Y'CbCr` or `Y'CbCr to R'G'B'` and set the **Use conversion specified by** parameter to `Rec. 709 (HDTV)`.

White point — White point

`D50` (default) | `D55` | `D65`

Specify the reference white point for converting the image between the sR'G'B' and L*a*b* color spaces as `D50`, `D55`, or `D65`.

Dependencies

To enable this parameter, set the **Conversion** parameter to `sR'G'B' to L*a*b*` or `L*a*b* to sR'G'B'`.

Image signal — Image signal input

`One multidimensional signal` (default) | `Separate color signals`

Specify whether to combine the color channels of the image into one signal or input and output them individually.

- `One multidimensional signal` — The block accepts all color channels of the input image as a multidimensional array to a single input port, and outputs the converted image as a single matrix or array from a single output port.
- `Separate color signals` — The block accepts each color channel of the input image to a separate input port, and outputs each color channel of the converted image from a separate output port.

Block Characteristics

Data Types	`Boolean` | `double` | `integer` | `single` | `image`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Algorithms

Conversion Between R'G'B' and Y'CbCr Color Spaces

These equations define R'G'B' to Y'CbCr conversion and Y'CbCr to R'G'B' conversion, respectively:

$$\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + A \times \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = B \times \left(\begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right)$$

The values in matrices `A` and `B` are based on your choices for the **Use conversion specified by** and **Scanning standard** parameters.

Matrix	Use conversion specified by = Rec. 601 (SDTV)	Use conversion specified by = Rec. 709 (HDTV)	
		Scanning standard = 1125/60/2:1	Scanning standard = 1250/50/2:1
A	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$	$\begin{bmatrix} 0.18258588 & 0.61423059 & 0.06200706 \\ -0.10064373 & -0.33857195 & 0.43921569 \\ 0.43921569 & -0.39894216 & -0.04027352 \end{bmatrix}$	$\begin{bmatrix} 0.25678824 & 0.50412941 & 0.09790588 \\ -0.1482229 & -0.29099279 & 0.43921569 \\ 0.43921569 & -0.36778831 & -0.07142737 \end{bmatrix}$

Matrix	Use conversion specified by = Rec. 601 (SDTV)	Use conversion specified by = Rec. 709 (HDTV)	
		Scanning standard = 1125/60/2:1	Scanning standard = 1250/50/2:1
B	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 1.1643836 & 2.0172321 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.16438356 & 0 & 1.79274107 \\ 1.16438356 & -0.21324861 & -0.53290933 \\ 1.16438356 & 2.11240179 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.1643836 & 0 & 1.5960268 \\ 1.1643836 & -0.39176229 & -0.81296765 \\ 1.1643836 & 2.0172321 & 0 \end{bmatrix}$

Conversion from R'G'B' to Intensity

This equation defines conversion from the R'G'B' color space to an intensity image:

$$\text{intensity} = [0.299\ \ 0.587\ \ 0.114]\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Conversion from B'G'R' to Intensity

This equation defines conversion from the B'G'R' color space to an intensity image:

$$\text{intensity} = [0.114\ \ 0.587\ \ 0.2999]\begin{bmatrix} B' \\ G' \\ R' \end{bmatrix}$$

Conversion Between R'G'B' and HSV Color Spaces

In the equations defining the R'G'B' to HSV conversion, MAX and MIN are the highest and lowest component values, respectively, of a given R'G'B' triplet. H, S, and V vary from 0 to 1.

$$H = \begin{cases} \left(\frac{G' - B'}{MAX - MIN}\right)/6, & \text{if } R' = MAX \\ \left(2 + \frac{B' - R'}{MAX - MIN}\right)/6, & \text{if } G' = MAX \\ \left(4 + \frac{R' - G'}{MAX - MIN}\right)/6, & \text{if } B' = MAX \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

This equation defines the HSV to R'G'B' conversion, where M is the highest value among R_{tmp}, G_{tmp} and B_{tmp}:

$H_i = \lfloor 6H \rfloor$

$f = 6H - H_i$

$p = 1 - S$

$q = 1 - (f * S)$

$t = 1 - S(1 - f)$

if $H_\text{i} = 0,\ R_\text{tmp} = 1,\ G_\text{tmp} = t,\ B_\text{tmp} = p$

if $H_\text{i} = 1,\ R_\text{tmp} = q,\ G_\text{tmp} = 1,\ B_\text{tmp} = p$

if $H_\text{i} = 2,\ R_\text{tmp} = p,\ G_\text{tmp} = 1,\ B_\text{tmp} = t$

if $H_\text{i} = 3,\ R_\text{tmp} = p,\ G_\text{tmp} = q,\ B_\text{tmp} = 1$

if $H_\text{i} = 4,\ R_\text{tmp} = t,\ G_\text{tmp} = p,\ B_\text{tmp} = 1$

if $H_\text{i} = 5,\ R_\text{tmp} = 1,\ G_\text{tmp} = p,\ B_\text{tmp} = q$

$u = V/M$

$R' = u * R_\text{tmp}$

$G' = u * G_\text{tmp}$

$B' = u * B_\text{tmp}$

Conversion Between sR'G'B' and XYZ Color Spaces

The sR'G'B' to XYZ conversion is a two-step process. First, the block converts the gamma-corrected sR'G'B' values to linear sRGB values using these equations:

If R'_sRGB, G'_sRGB, and B'_sRGB are each less than or equal to 0.03928, then:

$R_\text{sRGB} = R'_\text{sRGB}/12.92$

$G_\text{sRGB} = G'_\text{sRGB}/12.92$

$B_\text{sRGB} = B'_\text{sRGB}/12.92$

Otherwise:

$$R_\text{sRGB} = \left[(R'_\text{sRGB} + 0.055) \Big/ 1.055 \right]^{2.4}$$

$$G_\text{sRGB} = \left[(G'_\text{sRGB} + 0.055) \Big/ 1.055 \right]^{2.4}$$

$$B_\text{sRGB} = \left[(B'_\text{sRGB} + 0.055) \Big/ 1.055 \right]^{2.4}$$

Then, the block converts the sRGB values to XYZ values using this equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41239079926596 & 0.35758433938388 & 0.18048078840183 \\ 0.21263900587151 & 0.71516867876776 & 0.07219231536073 \\ 0.01933081871559 & 0.11919477979463 & 0.95053215224966 \end{bmatrix} \times \begin{bmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{bmatrix}$$

The XYZ to sR'G'B' conversion is also a two-step process. First, the block converts the XYZ values to linear sRGB values using this equation:

$$\begin{bmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{bmatrix} = \begin{bmatrix} 0.41239079926596 & 0.35758433938388 & 0.18048078840183 \\ 0.21263900587151 & 0.71516867876776 & 0.07219231536073 \\ 0.01933081871559 & 0.11919477979463 & 0.95053215224966 \end{bmatrix}^{-1} \times \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Then the block applies gamma correction, using these equations, to obtain the sR'G'B' values:

If R_{sRGB}, G_{sRGB} and B_{sRGB} are each less than or equal to 0.00304, then:

$$R'_{\text{sRGB}} = 12.92R_{\text{sRGB}}$$
$$G'_{\text{sRGB}} = 12.92G_{\text{sRGB}}$$
$$B'_{\text{sRGB}} = 12.92B_{\text{sRGB}}$$

Otherwise:

$$R'_{\text{sRGB}} = 1.055R_{\text{sRGB}}^{(1.0/2.4)} - 0.055$$
$$G'_{\text{sRGB}} = 1.055G_{\text{sRGB}}^{(1.0/2.4)} - 0.055$$
$$B'_{\text{sRGB}} = 1.055B_{\text{sRGB}}^{(1.0/2.4)} - 0.055$$

Note Computer Vision Toolbox uses a D65 white point, which is specified in Recommendation ITU-R BT.709, for this conversion. In contrast, the Image Processing Toolbox™ conversion is based on ICC profiles, and it uses a D65 to D50 Bradford adaptation transformation to the D50 white point. If you are using these two products and comparing results, you must account for this difference.

Conversion Between sR'G'B' and L*a*b* Color Spaces

The Color Space Conversion block converts sR'G'B' values to L*a*b* values in two steps. First, it converts sR'G'B' to XYZ values using the equations described in “Conversion Between sR'G'B' and XYZ Color Spaces” on page 1-145. Then, it uses these equations to transform the XYZ values to L*a*b* values. Here, X_n, Y_n, and Z_n are the tristimulus values of the reference white point you specify using the **White point** parameter:

If Y/Y_{n} is greater than 0.008856, then:

$$L^* = 116(Y/Y_{\text{n}})^{1/3} - 16$$

Otherwise:

$$L^* = 903.3Y/Y_{\text{n}}$$

$$a^* = 500(f(X/X_{\text{n}}) - f(Y/Y_{\text{n}}))$$
$$b^* = 200(f(Y/Y_{\text{n}}) - f(Z/Z_{\text{n}})),$$

If t is greater than 0.008856, then:

$$f(t) = t^{1/3}$$

Otherwise:

$$f(t) = 7.787t + 16/166$$

The block converts L*a*b* values to sR'G'B' values in two steps, as well. First, the block transforms the L*a*b* values to XYZ values using these equations:

If Y/Y_n is greater than 0.008856, then:

$$X = X_n(P + a^*/500)^3$$

$$Y = Y_n P^3$$

$$Z = Z_n(P - b^*/200)^3,$$

where:

$$P = (L^* + 16)/116$$

Otherwise:

$$X = {X_n((P + {a}/{500}) - 0.138)}/{7.787}$$

$$Y = {Y_n(P - 0.138)}/{7.787}$$

$$Z = Z_n\left(\left(P - {b}/{200}\right) - 0.138\right)(7.787)$$

Then it converts XYZ to sR'G'B' values using the equations described in “Conversion Between sR'G'B' and XYZ Color Spaces” on page 1-145.

Version History

Introduced before R2006a

References

[1] Poynton, Charles A. *A Technical Introduction to Digital Video*. New York: John Wiley & Sons, 1996.

[2] International Telecommunication Union (ITU). *Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios.* Recommendation ITU-R BT.601-5. Geneva, Switzerland: ITU, recommended October 1995; superseded January 2007.

[3] International Telecommunication Union (ITU). *Parameter Values for the HDTV Standards for Production and International Programme Exchange.* Recommendation ITU-R BT.709-5. Geneva, Switzerland: ITU, recommended April 2002; superseded June 2015.

[4] Stokes, Michael, Matthew Anderson, Srinivasan Chandrasekar, and Ricardo Motta. “A Standard Default Color Space for the Internet - sRGB. World Wide Web Consortium (W3C).” November 5, 1996. https://www.w3.org/Graphics/Color/sRGB.html

[5] Berns, Roy S., Fred W. Billmeyer, Max Saltzman, and Fred W. *Billmeyer. Billmeyer and Saltzman’s Principles of Color Technology.* 3rd ed. New York: Wiley, 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Blocks
Chroma Resampling

Functions
`applycform` | `makecform` | `rgb2gray` | `ycbcr2rgb` | `rgb2ycbcr` | `hsv2rgb` | `rgb2hsv`

Compositing

Combine two images or apply mask to image



Libraries:
Computer Vision Toolbox / Text & Graphics

Description

The Compositing block combines two images, overlays one image over another, or highlights selected pixels in an image. When combining two images, the block uses this equation, in which the output image O is a linear combination of the two images I_1 and I_2:

$$O(i, j) = (1 - X) * I_1(i, j) + X * I_2(i, j) .$$

The opacity factor X scales each pixel before combining the images, where $0 \le X \le 1$.

When overlaying one image over another image, or highlighting selected pixels in an input image, the block uses a binary mask to specify which pixels to overwrite or highlight.

Note This block supports binary, intensity, and color images.

Ports

Input

Image1 — Input image one
matrix | array

Input image one, specified as an M-by-N matrix or M-by-N-by-P array. If the input is a M-by-N-by-P array, P is the number of color channels in the image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Image2 — Input image two
matrix | array

Input image two, specified as an M-by-N matrix or M-by-N-by-P array. If the input is a M-by-N-by-P array, P is the number of color channels in the image.

Dependencies

To enable this port, set the **Operation** parameter to `Blend` or `Binary mask`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Mask — Binary mask
scalar | matrix

Binary mask, specified as a scalar or matrix. This mask determines which pixels of **Image1** the block overwrites with the pixel values of **Image2** or highlights, depending on the value of the **Operation** parameter.

- `Binary mask` — The block overwrites the pixel values of **Image1**, indicated by the mask, with the corresponding pixel values of **Image2**. You can specify **Mask** as a scalar `0` or `1`, which the block uses for all pixels of **Image2**, or as binary matrix of same size as **Image2**.
- `Highlight selected pixels` — The block highlights the pixels of **Image1** indicated by the mask.

Dependencies

To enable this port:

- If overwriting pixel values in one image with pixel values in another image, set the **Operation** parameter to `Binary mask` and the **Mask source** parameter to `Input port`.
- If highlighting selected pixels in an image, set the **Operation** parameter to `Highlight selected pixels`.

Data Types: `single` | `double` | `Boolean`

Factor — Opacity factor
scalar | matrix

Opacity factor, specified as a scalar or matrix. If specified as a scalar, the value is the scale of combination for each pixel of **Image2** with each corresponding pixel of **Image1**. If specified as a matrix, its dimensions must match the dimensions of **Image2**. Each opacity factor must be in the range [0, 1].

Dependencies

To enable this port, set the **Operation** parameter to `Blend` and the **Opacity factor(s) source** parameter to `Input port`.

Data Types: `single` | `double`

Location — Location index
vector

Location index, specified as a two-element vector. The first element represents the row and the second represents the column position of the upper-left corner of **Image2**, relative to the upper-left corner of **Image1**. If the location of the upper-left corner of **Image2** is such that any of its pixels would render outside the bounds of **Image1**, the block clips **Image2** at the final row or column or **Image1**, instead.

Note When the **Image1** input has a fixed-point or boolean data type, the **Location** input must have a built-in integer data type.

Dependencies

To enable this port, set the **Location source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Port_1 — Output image
matrix

Output image, returned as an *M*-by-*N* matrix or *M*-by-*N*-by-*P* array. If the output is an *M*-by-*N*-by-*P* array, *P* is the number of color channels in the image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Main

Operation — Specify operation

`Blend` (default) | `Binary mask` | `Highlight selected pixels`

Select the image operation for the block to perform.

- `Blend` — Linearly combines the pixel values of **Image1** with the pixel values of **Image2**, with the specified opacity factor.
- `Binary mask` — Overwrites the pixel values of **Image1** with the values of **Image2** specified by the binary mask.
- `Highlight selected pixels` — Highlights the pixels of **Image1** indicated by the binary mask. The highlight operation assign the highest intensity or color values to the pixels of **Image1** corresponding to the pixels with logic 1 of the binary mask.

Opacity factor(s) source — Source of opacity factor

`Specify via dialog` (default) | `Input port`

Specify the source of the opacity factor. If you set this parameter to `Input port`, it enables the **Factor** input port.

Dependencies

To enable this parameter, set the **Operation** parameter to `Blend`.

Opacity factor(s) — Opacity factor

`0.75` (default) | scalar | matrix

Defines the scale of combination for each pixel of **Image2** with each corresponding pixel of **Image1**. When you specify the opacity factor as matrix, its dimensions must match the dimensions of **Image2**. If you specify it as a scalar, the block uses that value as the opacity factor for all pixels of **Image2**. The opacity factor must be in the range [0, 1].

Dependencies

To enable this parameter, set the **Operation** parameter to `Blend` and the **Opacity factor(s) source** parameter to `Specify via dialog`.

Location source — Source of overlay location

`Specify via dialog` (default) | `Input port`

Select the source of the overlay location. If you set this parameter to `Input port`, it enables the **Location** input port.

Location [x y] — Overlay location index

`[1 1]` (default) | two-element vector

Specify the row and column position of the upper-left corner of **Image2**, relative to the upper-left corner of **Image1**, as a two-element vector. If the location of the upper-left corner of **Image2** is such that any of its pixels would render outside the bounds of **Image1**, the block clips **Image2** at the final row or column or **Image1**, instead.

Dependencies

To enable this parameter, set the **Location source** parameter to `Specify via dialog`.

Mask source — Source of mask

`Specify via dialog` (default) | `Input port`

Select the source of the binary mask. If you set this parameter to `Input port`, it enables the **Mask** input port.

Dependencies

To enable this parameter, set the **Operation** parameter to `Binary mask`.

Mask — Binary mask

`0` (default) | `1` | matrix

Binary mask, specified as a scalar or matrix, defines which pixels of **Image1** the block overwrites with the corresponding pixel values of **Image2** or highlights. You specify `0` or `1`, which the block uses for all pixels in **Image2**, or a binary matrix of the same size of as **Image2**.

Dependencies

To enable this parameter, set the **Operation** parameter to `Binary mask` and the **Mask source** parameter to `Specify via dialog`.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`

Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

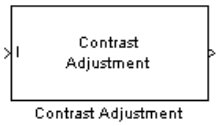
Generate C and C++ code using Simulink® Coder™.

See Also

Insert Text | Draw Markers | Draw Shapes

Contrast Adjustment

Adjust image contrast using linear scaling



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Contrast Adjustment block adjusts the contrast of an image by linearly scaling the pixel values between specified upper and lower limits. Pixel values that are above or below the specified range are saturated to the upper or lower limit value, respectively. For more information, see “Adjust Contrast of Image” on page 1-157.

Ports

Input

I — Input image
matrix | array

Input image, specified as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the input image.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Output

Port_1 — Output image
matrix | array

Output image, returned as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Parameters

Main

Adjust pixel values from — Input pixel value range definition
`Full input data range [min max]` (default) | `User-defined range` | `Range determined by saturating outlier pixels`

Specify how the block determines the pixel value range of the input image as one of these options:

- `Full input data range [min max]` — Range set to the minimum and maximum pixel values in the input image.

- `User-defined range` — Select this value to enable a **Range [low high]** parameter. Specify the **Range [low high]** parameter as a two-element vector of nonnegative values, in which the first element is the lower limit of the input pixel value range and the second element is the upper limit of the input pixel value range.
- `Range determined by saturating outlier pixels` — Range specified by saturating the outlier pixels. The block determines the range based on the **Percentage of pixels to saturate [low high]**, **Specify number of histogram bins (used to calculate the range when outliers are eliminated)**, and **Number of histogram bins** parameters. For more information, see “Determine Contrast Adjustment Range by Saturating Outlier Pixels” on page 1-158.

Range [low high] — Input pixel value range
`[0 255]` (default) | two-element vector of nonnegative numbers

Specify the input pixel value range as a two-element vector of nonnegative numbers. The first and second elements of the vector specify the lower and upper limit of the input pixel value range, respectively.

Dependencies

To enable this parameter, set the **Adjust pixel values from** parameter to `User-defined`.

Percentage of pixels to saturate [low high] — Maximum percentages of pixels to saturate
`[1 1]` (default) | two-element vector of nonnegative numbers

Specify the maximum percentages of pixels to saturate as a two-element vector of nonnegative numbers. Each element must be in the range 0 to 100. The block calculates the lower limit of the input pixel value range such that the percentage of pixels in the input image with values less than the lower limit is at most the value of the first element. Similarly, the block calculates the upper limit of the input pixel value range such that the percentage of pixels in the input image with values greater than the upper limit is at most the value of the second element.

Dependencies

To enable this parameter, set the **Adjust pixel values from** parameter to `Range determined by saturating outlier pixels`.

Specify number of histogram bins (used to calculate the range when outliers are eliminated) — Custom number of histogram bins
`on` (default) | `off`

Select this parameter to enable the **Number of histogram bins** parameter, which enables you to specify a custom number of histogram bins for the block to use to calculate the input pixel value range after outliers are eliminated.

Dependencies

To enable this parameter, set the **Adjust pixel values from** parameter to `Range determined by saturating outlier pixels`.

Number of histogram bins — Number of histogram bins
`256` (default) | positive integer

Specify the number of histogram bins to use to calculate the range for scaling the input pixel values.

Dependencies

To enable this parameter, set the **Adjust pixel values from** parameter to `Range determined by saturating outlier pixels`, and select the **Specify number of histogram bins (used to calculate the range when outliers are eliminated)** parameter.

Adjust pixel values to — Output pixel value range definition
`Full data type range` (default) | `User-defined range`

Specify how the block determines the pixel value range of the output image as one of these options:

- `Full data type range` — Specifies the output range of pixel values as from the minimum to the maximum value of the input data type.
- `User-defined range` — Select this value to enable a **Range [low high]** parameter. Specify the **Range [low high]** parameter as a two-element vector of nonnegative values, in which the first element is the lower limit of the output pixel value range and the second element is the upper limit of the output pixel value range.

Range [low high] — Output pixel value range
`[0 255]` (default) | two-element vector of nonnegative numbers

Specify the output pixel value range as a two-element vector of nonnegative numbers. The first and second elements of the vector specify the lower and upper limit of the output pixel value range, respectively.

Dependencies

To enable this parameter, set the **Adjust pixel values to** parameter to `User-defined range`.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

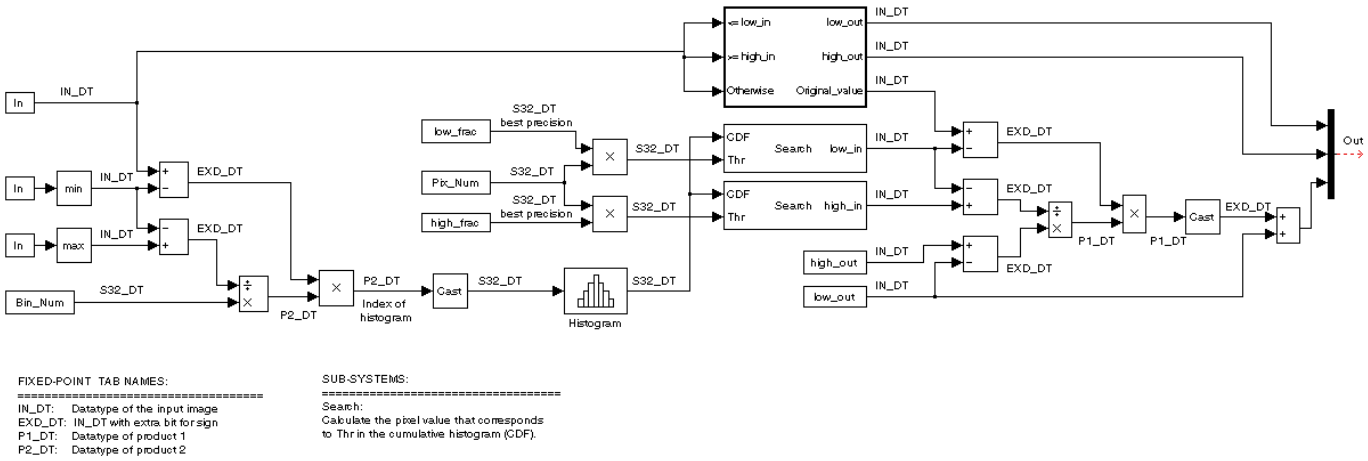
Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

The following diagram shows the data types used in the Contrast Adjustment block for fixed-point signals:



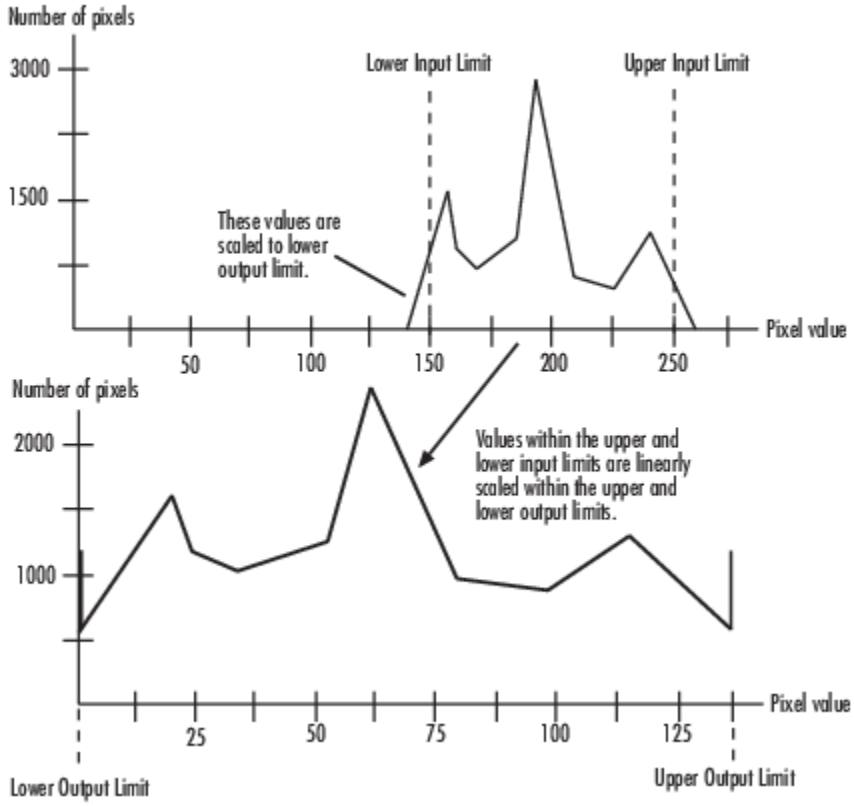
Algorithms

Adjust Contrast of Image

The contrast adjustment operation is defined by this equation.

$$O_i = \left\{ \begin{array}{c} l_{\text{out}},\ I_i \le l_{\text{in}} \\ l_{\text{out}} + (I_i - l_{\text{in}})\dfrac{h_{\text{out}} - l_{\text{out}}}{h_{\text{in}} - l_{\text{in}}},\ l_{\text{in}} < I_i < h_{\text{in}} \\ h_{\text{out}},\ I_i \ge h_{\text{in}} \end{array} \right\},$$

where I_i and O_i are the input and output image pixels, the input pixel value range is [l_{in} h_{in}] and the output pixel value range is [l_{out} h_{out}].



Determine Contrast Adjustment Range by Saturating Outlier Pixels

When you set the **Adjust pixel values from** parameter to `Range determined by saturating outlier pixels`, the block determines the input pixel value range using these steps.

1. Finds the minimum input pixel value *min_in* and maximum input pixel value *max_in*.
2. Scales the pixel values in the range [*min_in*, *max_in*] to [0, *num_bins*–1], where *num_bins* is the number of histogram bins specified by the **Number of histogram bins** parameter. Then, the block calculates the histogram of the scaled input. For more information on image histograms, see the 2-D Histogram block.
3. Determines the lower input pixel value limit such that the percentage of pixels in the input image with values less than the lower limit is at most the value of the first element of the **Percentage of pixels to saturate [low high]** parameter. Similarly, find the upper input pixel value limit such that the percentage of pixels in the input image with values greater than the upper limit is at most the value of the second element of the parameter.

Version History

Introduced in R2006b

Extended Capabilities

C/C++ Code Generation

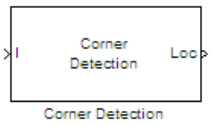
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
2-D Histogram | Histogram Equalization

Corner Detection

Calculate corner metric matrix and find corners in images



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Corner Detection block finds corners in an image by using the Harris corner detection (by Harris and Stephens), minimum eigenvalue (by Shi and Tomasi), or local intensity comparison (based on the *Accelerated Segment Test, (FAST)* method by Rosten and Drummond) method. The block finds the corners in the image based on the pixels that have the largest corner metric values.

Ports

Input

I — Input image
matrix of intensity values

Input image, specified as a matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

Loc — Corner locations
M-by-2 matrix

Corner locations, returned as an *M*-by-2 matrix of [*x y*] coordinates. *M* represents the number of corners and is less than or equal to the **Maximum number of corners** parameter.

Dependencies

To enable this port, set the **Output** parameter to one of these:

- `Corner location`
- `Corner location and metric matrix`

Data Types: `uint32`

Count — Number of detected corners
scalar

Number of detected corners, returned as a scalar.

Dependencies

To enable this port, set these block parameters:

- **Output**: `Corner location` | `Corner location and metric matrix`
- **Output variable size signal**: `off`

Data Types: `uint32`

Metric — Corner metric values
matrix

Corner metric values, returned as a matrix of intensity values. The returned matrix is the same size as the input image.

Dependencies

To enable this port, set the **Output** parameter to one of these:

- `Corner location and metric matrix`
- `Metric matrix`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main

Method — Corner detection method
`Harris corner detection (Harris & Stephens)` (default) | `Minimum eigenvalue (Shi & Tomasi)` | `Local intensity comparison (Rosten & Drummond)`

Specify the corner detection method as `Harris corner detection (Harris & Stephens)`, `Minimum eigenvalue (Shi & Tomasi)`, or `Local intensity comparison (Rosten & Drummond)`.

To get the most accurate results, use `Minimum eigenvalue (Shi & Tomasi)`. For the fastest computation, use `Local intensity comparison (Rosten & Drummond)`. For a balance between accuracy and computation speed, use `Harris corner detection (Harris & Stephens)`. For more information on each method, see the “Algorithms” on page 1-166 section.

Sensitivity factor (0<k<0.25) — Sensitivity factor
0.04 (default) | scalar

Specify the sensitivity factor, k. As k decreases, the likelihood that the algorithm can detect sharp corners increases.

Tunable: Yes

Dependencies

To enable this parameter, set the **Method** parameter to `Harris corner detection (Harris & Stephens)`.

Coefficients for separable smoothing filter — Filter coefficients
fspecial('gaussian', [1 5], 1.5) (default) | vector

Specify a vector of filter coefficients for the smoothing filter.

Dependencies

To enable this parameter, set the **Method** parameter to one of these:

- `Harris corner detection (Harris & Stephens)`
- `Minimum eigenvalue (Shi & Tomasi)`

Intensity comparison threshold — Intensity threshold
0.1 (default) | scalar

Specify the intensity threshold value used to find valid surrounding pixels.

Tunable: Yes

Dependencies

To enable this parameter, set the **Method** parameter to `Local intensity comparison (Rosten & Drummond)`.

Maximum angle to be considered a corner (in degrees) — Maximum corner angle
157.5 (default) | scalar

Specify the maximum corner angle.

Tunable: Yes

Dependencies

- To enable this parameter, set the **Method** parameter to `Local intensity comparison (Rosten & Drummond)`.
- This parameter is tunable for simulation only.

Output — Block output
`Corner location` (default) | `Corner location and metric matrix` | `Metric matrix`

Specify the block output as `Corner location`, `Corner location and metric matrix`, and `Metric matrix`.

Set this parameter to `Corner location` or `Corner location and metric matrix` to expose the **Maximum number of corners**, **Minimum metric value that indicates a corner**, and **Neighborhood size (suppress region around detected corners)** parameters.

Maximum number of corners — Maximum number of corners
200 (default) | integer

Specify the maximum number of corners you want the block to find.

Dependencies

To enable this parameter, set the **Output** parameter to one of these:

- `Corner location`
- `Corner location and metric matrix`

Minimum metric value that indicates a corner — Minimum corner metric
0.0005 (default) | scalar

Specify the minimum corner metric value.

Tunable: Yes

Dependencies

To enable this parameter, set the **Output** parameter to one of these:

- `Corner location`
- `Corner location and metric matrix`

Neighborhood size — Neighborhood size
[11 11] (default) | 2-element vector

Specify the neighborhood size as a two-element vector of positive odd integers, [*row*, *column*]. The block uses this parameter to suppress the region around a valid corner point with the largest corner metric value.

Dependencies

To enable this parameter, set the **Output** parameter to one of these:

- `Corner location`
- `Corner location and metric matrix`

Output variable size signal — Variable size signal
`on` (default) | `off`

Select this parameter to output a variable size signal.

Dependencies

To enable this parameter, set the **Output** parameter to one of these:

- `Corner location`
- `Corner location and metric matrix`

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Data type override

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

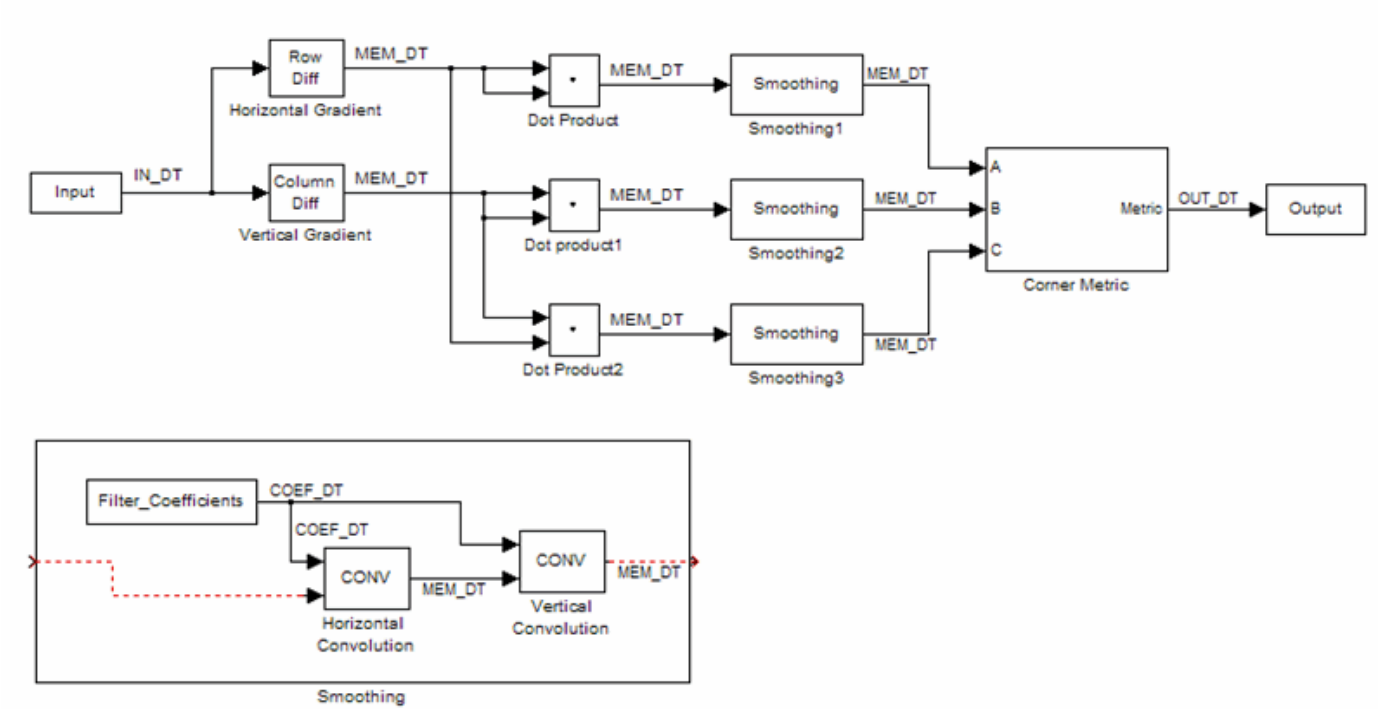
Data Types	`Boolean`[a] | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`

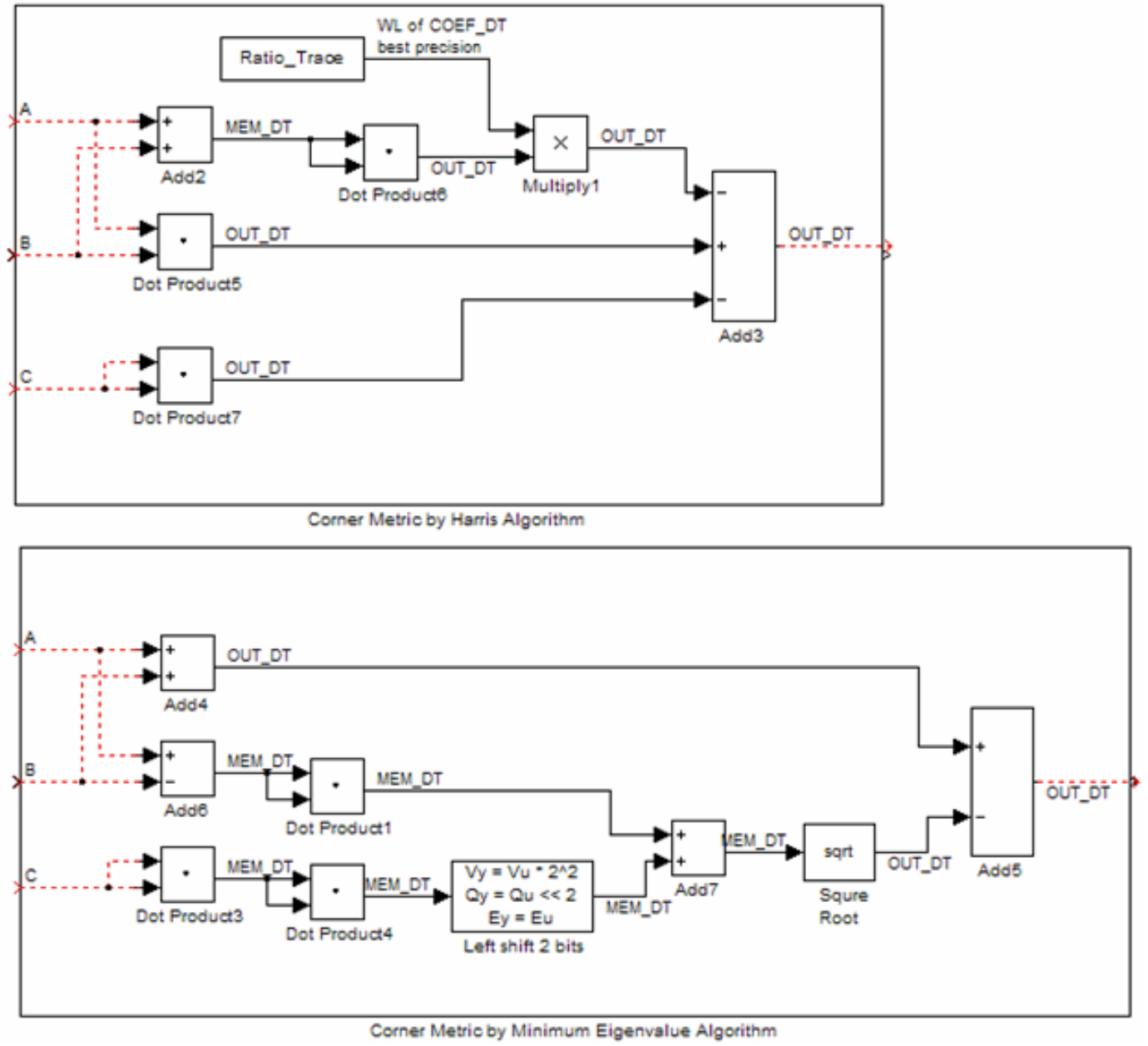
Variable-Size Signals	yes

a This data type is not supported for the Local Intensity Comparison method.

More About

These diagrams show the data types used in the Corner Detection block for fixed-point signals. These diagrams apply to the Harris corner detection and minimum eigenvalue methods only.





The following table summarizes the variables used in the previous diagrams.

Variable Name	Definition
IN_DT	Input data type
MEM_DT	Memory data type
OUT_DT	Metric output data type
COEF_DT	Coefficients data type

Algorithms

Determine final corner values

To determine the final corner values, the block follows this process:

1. Find the pixel with the largest corner metric value.
2. Verify that the metric value is greater than or equal to the value you specified for the **Minimum metric value that indicates a corner** parameter.
3. Suppress the region around the corner value by the size defined in the **Neighborhood size (suppress region around detected corners)** parameter.

The block repeats this process until it finds all the corners in the image or it finds the number of corners you specified in the **Maximum number of corners** parameter.

The corner metric values computed by the `Minimum eigenvalue` and `Local intensity comparison` methods are always nonnegative. The corner metric values computed by the `Harris corner detection` method can be negative.

Minimum Eigenvalue Method

The method is more computationally expensive than the Harris corner detection algorithm because it directly calculates the eigenvalues of the sum of the squared difference matrix, M.

The sum of the squared difference matrix, M, is defined as follows:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

The previous equation is based on the following values:

$$A = (I_x)^2 \otimes w$$

$$B = (I_y)^2 \otimes w$$

$$C = (I_x I_y)^2 \otimes w$$

where I_x and I_y are the gradients of the input image, I, in the x and y direction. The $\otimes$ symbol denotes a convolution operation.

Use the **Coefficients for separable smoothing filter** parameter to define a vector of filter coefficients. The block multiplies this vector of coefficients by its transpose to create a matrix of filter coefficients, w.

The block calculates the smaller eigenvalue of the sum of the squared difference matrix. This minimum eigenvalue corresponds to the corner metric matrix.

Harris Corner Method

The Harris corner detection method avoids the explicit computation of the eigenvalues of the sum of squared differences matrix by solving for the corner metric matrix, R:

$$R = AB - C^2 - k(A + B)^2$$

See the Minimum Eigenvalue Method section for the definitions of A, B, and C.

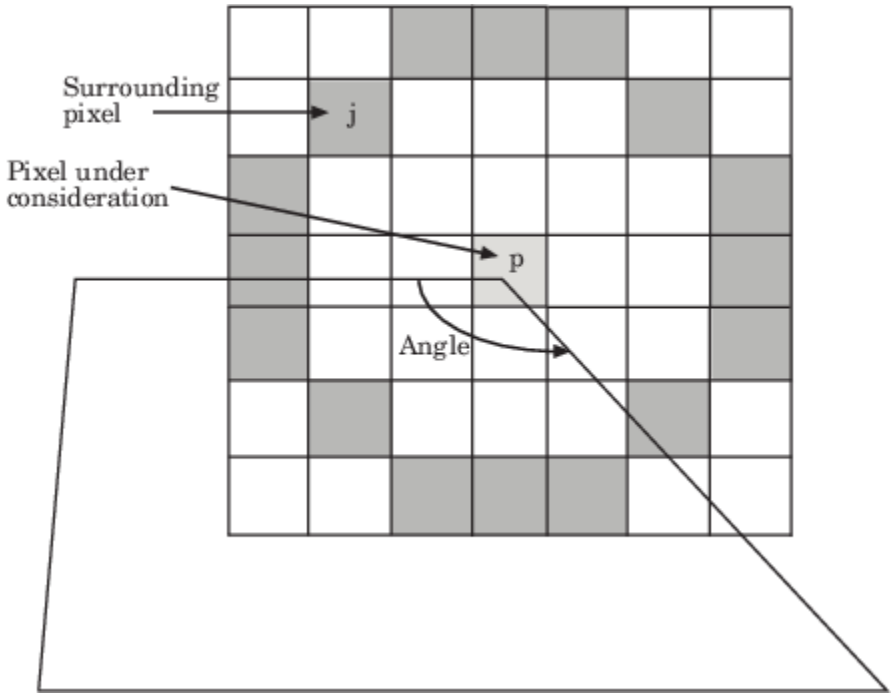
The variable *k* corresponds to the sensitivity factor. You can specify its value using the **Sensitivity factor (0<k<0.25)** parameter. As *k* decreases, the likelihood that the algorithm can detect sharp corners increases.

Use the **Coefficients for separable smoothing filter** parameter to define a vector of filter coefficients. The block multiplies this vector of coefficients by its transpose to create a matrix of filter coefficients, *w*.

Local Intensity Method

The method determines that a pixel is a possible corner if it has either, *N* contiguous valid bright surrounding pixels, or *N* contiguous dark surrounding pixels.

Suppose that *p* is the pixel under consideration and *j* is one of the pixels surrounding *p*. The locations of the other surrounding pixels are denoted by the shaded areas in the following figure. The shaded areas in this figure denote the locations of other surrounding pixels.



I_p and I_j are the intensities of pixels *p* and *j*, respectively. Pixel *j* is a valid bright surrounding pixel if $I_j - I_p \geq T$. Similarly, pixel *j* is a valid dark surrounding pixel if $I_p - I_j \geq T$. In these equations, *T* is the value you specified for the **Intensity comparison threshold** parameter.

The block repeats this process to determine whether the block has *N* contiguous valid surrounding pixels. The value of *N* is related to the value you specify for the **Maximum angle to be considered a corner (in degrees)** parameter, as shown in this table.

Number of Valid Surrounding Pixels and Angle

Number of Valid Surrounding Pixels	Angle (degrees)
15	22.5
14	45
13	67.5
12	90
11	112.5
10	135
9	157.5

After the block determines that a pixel is a possible corner, it computes its corner metric using the following equation:

$$R = \max\left(\sum_{j:I_j \geq I_p + T} \left|I_p - I_j\right| - T, \sum_{j:I_j \leq I_p - T} \left|I_p - I_j\right| - T\right)$$

Version History

Introduced in R2007b

References

[1] Harris, C. and M Stephens. “A Combined Corner and Edge Detector.” *Proceedings of the 4th Alvey Vision Conference*, 147-151. August 1988.

[2] Shi, J. and C. Tomasi. “Good Features to Track.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 593-600. June 1994.

[3] Rosten, E. and T. Drummond. “Fusing Points and Lines for High Performance Tracking.” *Proceedings of the IEEE International Conference on Computer Vision Vol. 2*, 1508-1511. October 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Blocks

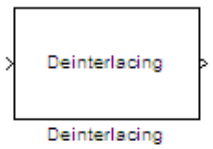
Find Local Maxima | Estimate Geometric Transformation | Warp

Functions

matchFeatures | extractFeatures | detectSURFFeatures | detectHarrisFeatures | detectFASTFeatures | detectORBFeatures | estimateGeometricTransform2D

Deinterlacing

Remove interlacing effect



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The `Deinterlacing` block removes the motion artifact known as the interlacing effect from an interlaced image or a video. An interlaced image or video frame comprises of top field and bottom field that has slightly different capture times. The odd numbered lines in an interlaced image constitute the top field and the even numbered lines constitute the bottom field.

When you display an interlaced image, the temporal difference between the top and bottom fields causes a visible artifact where the alternate lines in the interlaced image appear displaced.

This artifact is known as the interlacing effect and can be corrected through deinterlacing techniques. You can use the `Deinterlacing` block to convert an interlaced video into a deinterlaced video by using one of these methods:

- Line repetition
- Linear interpolation
- Vertical temporal median filtering

You can use this block to deinterlace both intensity and color images.

Ports

Input

Port_1 — Interlaced input
numeric matrix | 3-D numeric array | image | video

Interlaced input, specified as a numeric matrix, 3-D numeric array, image, or video.

To input an interlaced image, read the image file by using the Image From File or Image From Workspace block. The image can also be a video frame. To input an interlaced video, read the video file by using the From Multimedia File or Video From Workspace block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

Port_1 — Deinterlaced output
numeric matrix | 3-D numeric array | image | video

Deinterlaced output, returned as a numeric matrix, 3-D numeric array, image, or video. The output is of same size and datatype as that of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main Tab

Method — Deinterlacing method

`Line repetition` (default) | `Linear interpolation` | `Vertical temporal median filtering`

Use this parameter to specify the method for deinterlacing the input data.

- If you set the **Method** parameter value to `Line repetition`, the values in the bottom field of the interlaced image or video frame are replaced with the values in the preceding top field.

Line Repetition

Original Interlaced Video

	Top Field			Bottom Field			
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

	Block Input			Block Output - Deinterlaced Video			
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	A	B	C
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	G	H	I
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

- If you set the **Method** parameter value to `Line interpolation`, the values in a bottom field are computed as the average of the values in the two neighboring top fields.

Linear Interpolation

Original Interlaced Video

Top Field				Bottom Field			
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

Block Input				Block Output - Deinterlaced Video			
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	(A+G)/2	(B+H)/2	(C+I)/2
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	(G+M)/2	(H+N)/2	(I+O)/2
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

- If you set the **Method** parameter value to `Vertical temporal median filtering`, the values in a bottom field are computed as the median of the values in the bottom field and its two neighboring top fields.

Vertical Temporal Median Filtering

Original Interlaced Video

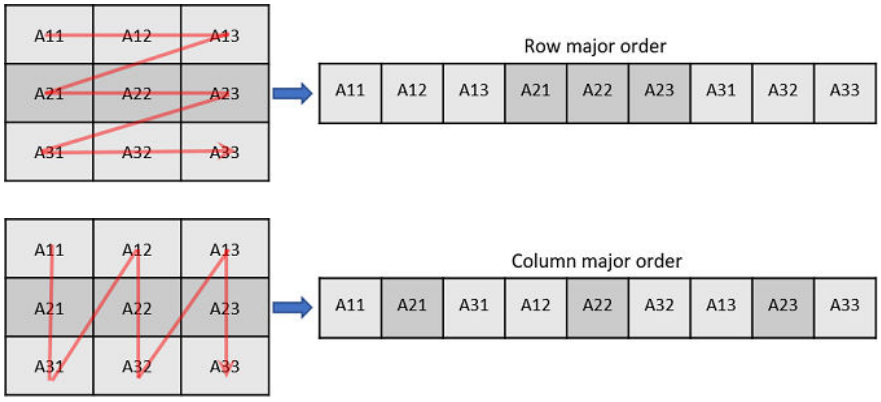
	Top Field				Bottom Field		
Row 1	A	B	C	Row 1			
Row 2				Row 2	D	E	F
Row 3	G	H	I	Row 3			
Row 4				Row 4	J	K	L
Row 5	M	N	O	Row 5			
Row 6				Row 6	P	Q	R

Block Input				Block Output - Deinterlaced Video			
Row 1	A	B	C	Row 1	A	B	C
Row 2	D	E	F	Row 2	median([A,D,G])	median([B,E,H])	median([C,F,I])
Row 3	G	H	I	Row 3	G	H	I
Row 4	J	K	L	Row 4	median([G,J,M])	median([H,K,N])	median([I,L,O])
Row 5	M	N	O	Row 5	M	N	O
Row 6	P	Q	R	Row 6	M	N	O

Input image is transposed (data order is row major) — Specify data order

off (default) | on

You can use this parameter to specify if the interlaced input is stored in row major order. When you select this parameter, the block assumes that the input buffer contains contiguous data elements of an array that are arranged sequentially row by row. Otherwise, the blocks assumes that the buffer stores data in the column major order such that the elements are arranged sequentially column by column.



Use this parameter only when

- You are developing algorithms to run on an embedded target that uses the row-major format.
- You want to limit the additional processing required to take the transpose of signals at the interfaces of the row-major and column-major systems.

When you select this check box, the first two dimensions of the Deinterlacing block's input are swapped.

Data Types Tab

To enable options on the **Data Types** tab, set the **Method** parameter in the **Main** tab to `Linear interpolation`.

Fixed-point operational parameters

Rounding mode — Rounding mode for calculating fixed-point value

`Floor` (default) | `Ceiling` | `Convergent` | `Nearest` | `Round` | `Simplest` | `Zero`

Specifies the rounding mode for the block to use when the specified data type and scaling cannot exactly represent the result of a fixed-point calculation.

See “Rounding Modes” for more information on the available options.

Saturate on integer overflow — Specify overflow mode
on (default) | off

When you select this parameter, the block saturates the result of its fixed-point operation. When you clear this parameter, the block wraps the result of its fixed-point operation.

For details on saturate and wrap, see “Overflow Handling” for fixed-point operations.

Fixed-point data types

Accumulator — Accumulator data type

`Binary point scaling` (default) | `Same as input`

Use this parameter to specify the data type and scaling of the accumulator (sum).

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits. The default value for word length is set to 12 and fraction length is set to 3. The data type format is same as that of the input data format, signed or unsigned.
- When you select `Same as input`, the word length and the fractional length of the accumulator match those of the input.

Output — Data type for output
`Same as input` (default) | `Binary point scaling`

Use this parameter to specify the data type and scaling of the output data.

- When you select `Same as input`, the word length and the fractional length of the accumulator match those of the input.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits. The default value for word length is set to 8 and fraction length is set to 0. The data type format is same as that of the input data type format, signed or unsigned.

Lock data type settings against change by the fixed-point tools — Data type override
off (default) | on

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask. For more information, see `fxptdlg`, a reference page on the Fixed-Point Tool.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

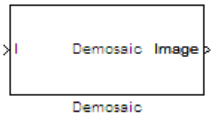
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

`vision.Deinterlacer`

Demosaic

Demosaic Bayer format images



Libraries:
Computer Vision Toolbox / Conversions

Description

The Demosaic block takes images in Bayer format and outputs RGB images. The block performs this operation using a gradient-corrected linear interpolation algorithm or a bilinear interpolation algorithm. This figure illustrates a 4-by-4 image in Bayer format with each pixel labeled R, G, or B.

B	G	B	G
G	R	G	R
B	G	B	G
G	R	G	R

Ports

Input

I — Image to demosaic
matrix

Image to demosaic, specified as a 2-D numeric matrix. The size requirements for the matrix depend on the value of the **Interpolation algorithm** parameter.

- If the value of **Interpolation algorithm** is `Bilinear`, the matrix must have three or more rows and three or more columns.
- If the value of **Interpolation algorithm** is `Gradient-corrected linear`, the matrix must have five or more rows and five or more columns.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

R — Red color channel of processed image
matrix

Red color channel of processed image, returned as a 2-D numeric matrix of the same size and data type as **I**.

Dependencies

To enable this port, set the **Output image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

G — Green color channel of processed image
matrix

Green color channel of processed image, returned as a 2-D numeric matrix of the same size and data type as **I**.

Dependencies

To enable this port, set the **Output image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

B — Blue color channel of processed image
matrix

Blue color channel of processed image, returned as a 2-D numeric matrix of the same size and data type as **I**.

Dependencies

To enable this port, set the **Output image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Image — Multidimensional processed image
matrix

Multidimensional processed image, returned as a 3-D numeric matrix of the same data type as **I**. If the input image **I** has the dimensions *M*-by-*N*, then **Image** has the dimensions *M*-by-*N*-by-*P*, where *P* is the number of color channels.

Dependencies

To enable this port, specify the **Output image signal** parameter as `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main

Interpolation algorithm — Interpolation algorithm

`Bilinear` (default) | `Gradient-corrected linear`

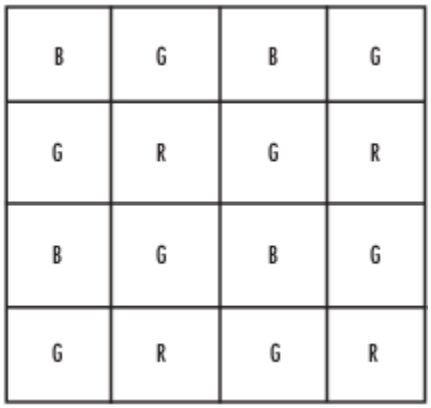
Specify the algorithm the block uses to calculate the missing color information. If you select `Bilinear`, the block spatially averages the values of the neighboring pixels of a missing color value to calculate its color information.

If you select `Gradient-corrected linear`, the block uses a Weiner approach to minimize the mean-squared error in the interpolation. This method performs well on the edges of objects in the image. Both methods use symmetric padding at the image boundaries. For more information, see the Image Pad block reference page.

Sensor alignment — R,G,B pixel alignment

`RGGB` (default) | `GRBG` | `GBRG` | `BGGR`

Specify the alignment of the input image. Select the sequence of R, G, and B pixels that correspond to the 2-by-2 block of pixels in the top-left corner of the image. Specify the sequence in left-to-right, top-to-bottom order. For example, for this image, you would select `BGGR`.



Output image signal — Dimensions of output image

`One multidimensional signal` (default) | `Separate color signals`

Specify how to output the RGB image. Given an *M*-by-*N* input image, this parameter affects the output of this block in these ways.

- `One multidimensional image` — The block has a single port, Image, that outputs an *M*-by-*N*-by-*P* RGB image, where *P* is the number of color channels.
- `Separate color signals` — The block has three ports, R, G, and B, that each output one *M*-by-*N* color channel of the RGB image.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

Version History

Introduced in R2006b

References

[1] Malvar, H.S., Li-wei He, and R. Cutler. “High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images.” In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3:iii-485–88. Montreal, Que., Canada: IEEE, 2004. https://doi.org/10.1109/ICASSP.2004.1326587.

[2] Gunturk, B.K., J. Glotzbach, Y. Altunbasak, R.W. Schafer, and R.M. Mersereau. “Demosaicking: Color Filter Array Interpolation.” *IEEE Signal Processing Magazine*, 22, no. 1 (January 2005): 44–54. https://doi.org/10.1109/MSP.2005.1407714.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

`demosaic`

Dilation

Dilate binary or intensity image by finding local maxima

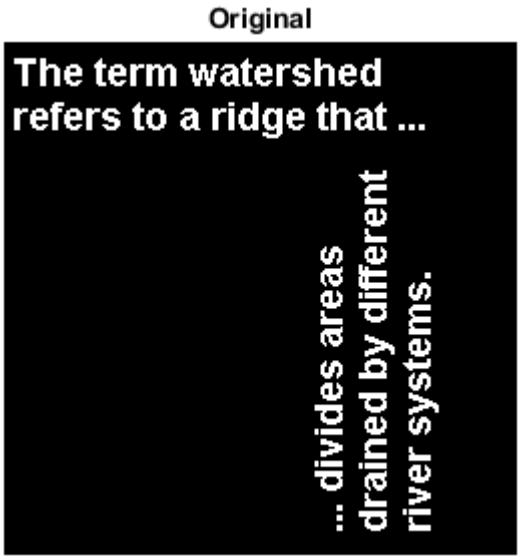


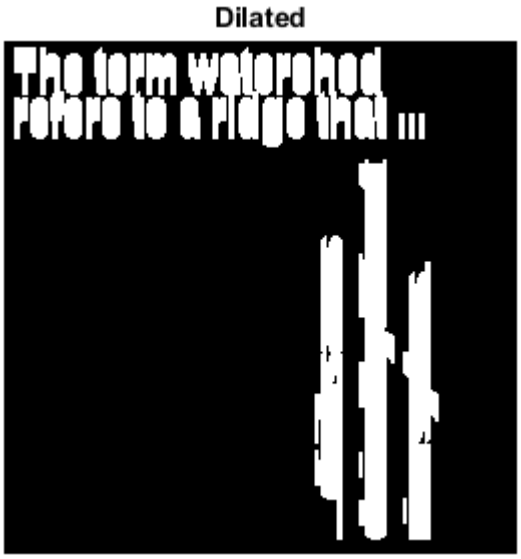
Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Dilation block performs morphological dilation on an intensity or binary image and outputs the dilated image.

To define the structuring element that the block applies to the image, use the **Neighborhood or structuring element** parameter.





This block uses two-dimensional flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of intensity values. This port is unnamed unless you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values representing a structuring element as a vector or matrix of `1`s and `0`s or by using the `strel` function.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Dilated image
matrix

The block returns the dilated image as a matrix of intensity values. The size and data type of the output image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of the neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`[1 1; 1 1]` (default) | vector | matrix

Specify a neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object, or an array of `strel` objects. When you specify an array of `strel` objects, the block applies each object to the entire image sequentially.

If the structuring element is decomposable into smaller elements, the block uses a more efficient algorithm that can execute more quickly.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter, to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Dilation and Structuring Element

Performing dilation on an image includes:

- Increasing the size of the object by expanding the image pixels and adding pixels to the boundary of the object.
- Filling small holes and broken areas
- Connecting areas separated by a space smaller than the structuring element

The direction and extent of the growth of an object in an image depends on the shape of the structuring element. Each structuring element has an origin.

- If the structuring element is symmetrical, the origin coincides with its centroid.
- If the structuring element is asymmetrical, the pixel in the structuring element that is close to its centroid and is in the top-left direction is considered as the origin.

The Dilation block rotates the structuring element about this origin and places the origin over each pixel of the image object. The block creates the dilated image from the local maxima values at each pixel.

Binary Dilation

The binary dilation of A by B, denoted as $A \oplus B$, is defined as the set operation:

$$A \oplus B = \left\{ z \middle| \left(\widehat{B} \right)_z \cap A \neq \varnothing \right\},$$

where $\widehat{B}$ is the reflection of the structuring element B. Note that some applications use a definition of dilation in which the structuring element is not reflected.

Grayscale Dilation

In the general form of grayscale dilation, the structuring element has a specified height. The grayscale dilation of $A(x, y)$ by $B(x, y)$ is defined as:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') + B(x', y') | (x', y') \in D_B\},$$

where D_B is the domain of the structuring element B and $A(x, y)$ is assumed to be $-\infty$ outside the domain of the image. To create a structuring element with nonzero height values, use the syntax `strel(nhood,height)`, where `height` specifies the height values and `nhood` corresponds to the structuring element domain D_B.

Grayscale dilation is often performed with a flat structuring element ($B(x,y) = 0$). Grayscale dilation using such a structuring element is equivalent to a local-maximum operator:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') | (x', y') \in D_B\}.$$

All of the `strel` syntaxes except for `strel(nhood,height)`, `strel("arbitrary",nhood,height)`, and `strel("ball",...)` produce flat structuring elements.

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

[2] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Third edition. Knoxville: Gatesmark Publishing, 2020.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Erosion | Opening | Closing | Label | Bottom-hat | Top-hat

Functions

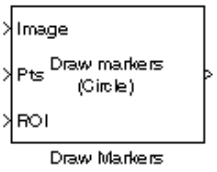
`imerode` | `imdilate` | `imopen` | `imclose`

Objects

`strel`

Draw Markers

Draw markers on image



Libraries:
Computer Vision Toolbox / Text & Graphics

Description

The Draw Markers block draws markers on an image or video frame by overwriting pixel values in the input image. This embeds the markers in the output image. The block supports five types of markers: `Circle`, `X-mark`, `Plus`, `Star`, and `Square`.

The block uses Bresenham's circle drawing algorithm to draw circles and Bresenham's line drawing algorithm to draw all other markers.

Ports

Input

Image — Input Image
matrix | array

Specify the input image as either an *M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* array of color values with *P* color planes.

Dependencies

To enable this port, set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color component of image
matrix

Specify the red color component of the input RGB image as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color component of image
matrix

Specify the green color component of the input RGB image as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color component of image
matrix

Specify the blue color component of the input RGB image as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Pts — Marker coordinates
vector | matrix

Specify the coordinates of markers as one of these options:

- Single marker — Two-element vector that specifies the *x*- and *y*- coordinates for the center of the marker.
- Multiple markers — *K*-by-2 matrix. *K* is the number of markers. Each row of the matrix specifies the *x*- and *y*- coordinates for the center of a marker.

Note The block does not support single-precision or double-precision floating-point data types for this parameter unless the input image is also floating point.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Clr — Marker color
scalar | vector | matrix

Specify the color values for the markers. Use this table to determine the type of value to input at this port.

Type of Input Image	Same Color for All Markers	Different Color for Each Marker
Intensity image	scalar value	*K*-element vector. *K* is the number of markers.
Color image	*P*-element vector. *P* is the number of color planes.	*K*-by-*P* matrix. *P* is the number of color planes and *K* is the number of markers.

Dependencies

To enable this port, use one of these options:

- The **Marker shape** parameter is set to `X-mark`, `Plus`, or `Star`, or the **Filled** parameter is cleared — Set the **Border color source** parameter to `Input port`.
- The **Marker shape** parameter is set to `Circle` or `Square` and the **Filled** parameter is selected — Set the **Fill color source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

ROI — Region of interest
vector

Specify a rectangular region of interest (ROI) as a four-element vector of form [*x y width height*]. The *x* and *y* elements are the coordinates of the top-left corner of the ROI.

Dependencies

To enable this port, set the **Draw markers in** parameter to `Specify region of interest via port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Port_1 — Output image
matrix | array

Output image, returned as an *M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*P* array of color values. *P* is the number of color planes in the output image. The output image has the same size and data type as the input image.

Dependencies

To enable this port, set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color component of output image
matrix

Red color component of the output RGB image, returned as an *M*-by-*N* matrix. The red color component of the output image has the same dimensions and data type as the red color component of the input image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color component of output image
matrix

Green color component of the output RGB image, returned as an *M*-by-*N* matrix. The green color component of the output image has the same dimensions and data type as the green color component of the input image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color component of output image
matrix

Blue color component of the output RGB image, returned as an *M*-by-*N* matrix. The blue color component of the output image has the same dimensions and data type as the blue color component of the input image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Main

Marker shape — Shape of marker

`Circle` (default) | `X-mark` | `Plus` | `Star` | `Square`

Specify the shape of the marker to draw. You can render the `Circle` and`Square` markers as outlines or as filled shapes.

Marker size — Size of marker

3 (default) | positive integer

Specify the size of the marker as a positive integer, *M*, that defines a square area, with sides of length 2*M*+1, into which the marker fits.

Filled — Filled marker

`off` (default) | `on`

Select this parameter to fill the marker with the specified intensity value or color.

Dependencies

To enable this parameter, set the **Marker shape** parameter to `Circle` or `Square`.

Border color source — Color source of marker border
`Specify via dialog` (default) | `Input port`

Specify the source of the color for the border of the marker as one of these options:

- `Specify via dialog` — Use the **Border color** parameter to specify a color for the border of the marker.
- `Input port` — Use the **Clr** input port to specify a color for the border of the marker.

Dependencies

To enable this parameter, clear the **Filled** parameter or set the **Marker shape** parameter to `X-mark`, `Plus`, or `Star`.

Border color — Color value of marker
`Black` (default) | `White` | `User-specified value`

Specify the color value for the border of the marker as `Black`, `White`, or `User-specified value`.

Dependencies

To enable this parameter, clear the **Filled** parameter and set the **Border color source** parameter to `Specify via dialog`.

Fill color source — Color source of filled marker
`Specify via dialog` (default) | `Input port`

Specify the source of the color with which to fill the marker as one of these options:

- `Specify via dialog` — Use the **Fill color** parameter to specify a color with which to fill the marker.
- `Input port` — Use the **Clr** input port to specify a color with which to fill the marker.

Dependencies

To enable this parameter, set the **Marker shape** parameter to `Circle` or `Square` and select the **Filled** parameter.

Fill color — Color value of filled marker
`Black` (default) | `White` | `User-specified value`

Specify the color value with which to fill the marker as `Black`, `White`, or `User-specified value`.

Dependencies

To enable this parameter, set the **Marker shape** parameter to `Circle` or `Square`, select the **Filled** parameter, and set the **Fill color source** parameter to `Specify via dialog`.

Opacity factor (between 0 and 1) — Opacity factor

`0.6` (default) | scalar in range [`0`, `1`] | *K*-element vector

Specify the opacity factor for the marker in the range [`0`, `1`], where `0` is completely transparent and `1` is completely opaque. Specify this parameter as one of these options:

- Scalar — Apply the specified opacity factor to all markers.
- *K*-element Vector — Apply a different opacity factor to each marker. *K* is the number of markers.

Dependencies

To enable this parameter, set the **Marker shape** parameter to `Circle` or `Square` and select the **Filled** parameter.

Color value(s) — User-specified color values

`[200 255 100]` (default) | scalar | *P*-element vector | *K*-by-*P* matrix

Specify the color values for the markers. Use this table to determine the type of value to enter for this parameter.

Type of Input Image	Same Color for All Markers	Different Color for Each Marker
Intensity image	scalar value	*K*-element vector. *K* is the number of markers.
Color image	*P*-element vector. *P* is the number of color planes.	*K*-by-*P* matrix. *P* is the number of color planes and *K* is the number of markers.

Dependencies

To enable this parameter, use one of these options:

- The **Marker shape** parameter is set to `X-mark`, `Plus`, or `Star`, or the **Filled** parameter is cleared — Set the **Border color source** parameter to `Specify via dialog` and the **Border color** parameter to `User-specified value`.
- The **Marker shape** parameter is set to `Circle` or `Square` and the **Filled** parameter is selected — Set the **Fill color source** parameter to `Specify via dialog` and the **Fill color** parameter to `User-specified value`.

Draw Markers in — Area for markers

`Entire image` (default) | `Specify region of interest via port`

Specify where to draw the markers. By default, you can draw markers in the entire image. To specify markers within a rectangular ROI, set this parameter to `Specify region of interest via port`, and specify a four-element vector of the form [*x y width height*] to the **ROI** input port.

Use antialiasing — Smoothing algorithm

`off` (default) | `on`

Select this parameter to perform a smoothing algorithm on the `Circle`, `X-mark`, or `Star` marker shapes.

Dependencies

To enable this parameter, set the **Marker shape** parameter to `Circle`, `X-mark`, or `Star`.

Image signal — Port configuration for RGB image

`One multidimensional signal` (default) | `Separate color signals`

Specify how to input and output an RGB color image.

- `One multidimensional signal` — The block accepts input using only the **Image** port and returns the output using only the **Port_1** port.
- `Separate color signals` — The block accepts and returns the color image using three separate color channels, divided across the **R**, **G**, and **B** input and output ports.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

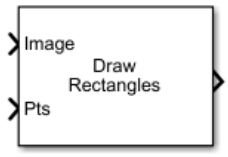
Draw Shapes

Functions

`insertMarker` | `insertShape`

Draw Shapes

Draw rectangles, lines, polygons, or circles on images



Libraries:
Computer Vision Toolbox / Text & Graphics

Description

The Draw Shapes block draws rectangles, lines, polygons, or circles on images by overwriting the pixel values. As a result, the shapes are drawn on the output image. The block uses Bresenham's line drawing algorithm to draw lines, polygons, and rectangles and Bresenham's circle drawing algorithm to draw circles. The output signal is of the same size and data type as the input signal fed to the **Image**, **R**, **G**, and **B** ports.

You can set the shape fill or border color via the input port or by manually entering the values using the parameters. Use the color parameters to specify the appearance of the rectangles, lines, polygons, or circles.

Ports

Input

Image — Input image or video stream
M-by-*N* matrix of intensity values | *M*-by-*N*-by-*3* array of color values

Input image, specified as either an *M*-by-*N* matrix of intensity values or an *M*-by-*N*-by-*3* matrix of color values with *3* color planes.

When the input is a video stream, the block processes it frame by frame.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

R — Red color channel of the image or video frame
M-by-*N* matrix

Red color channel of the image or video frame, specified as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

G — Green color channel of the image or video frame
M-by-*N* matrix

Green color channel of the input image or video frame, specified as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

B — Blue color channel of the image or video frame
M-by-*N* matrix

Blue color channel of the input image or video frame, specified as an *M*-by-*N* matrix. The **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

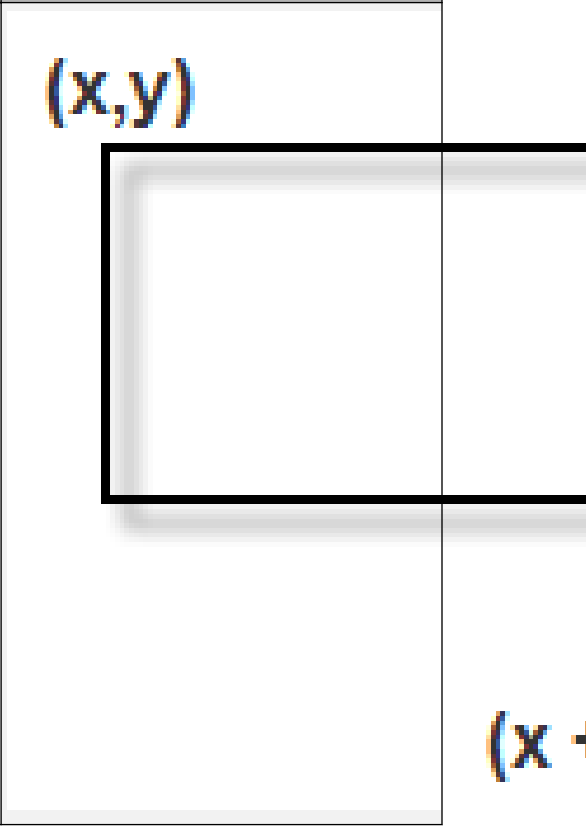
To enable this port, set the **Image signal** parameter to `Separate color signals`.

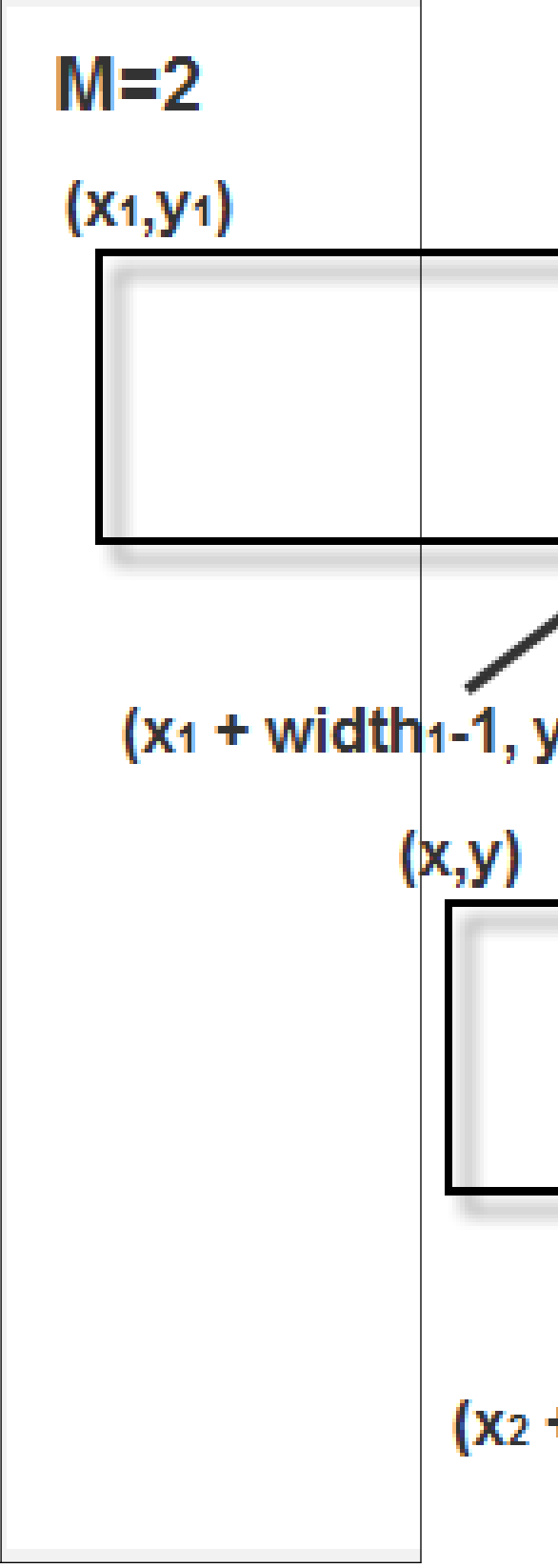
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Pts — Shape coordinates
M-by-3 matrix | *M*-by-4 matrix | *M*-by-2*L* matrix

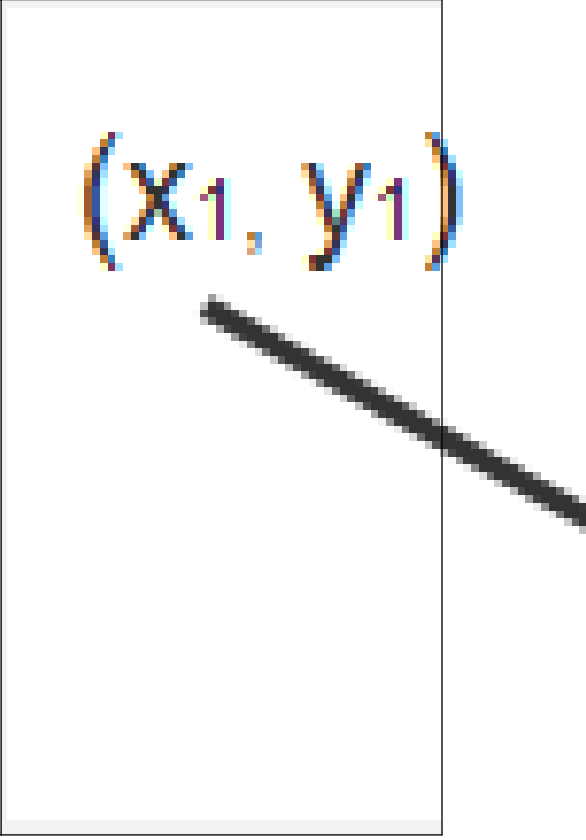
Coordinates of the input shapes to be drawn on the input. The format of the points depends on the type of shape you specify with the **Shape** parameter.

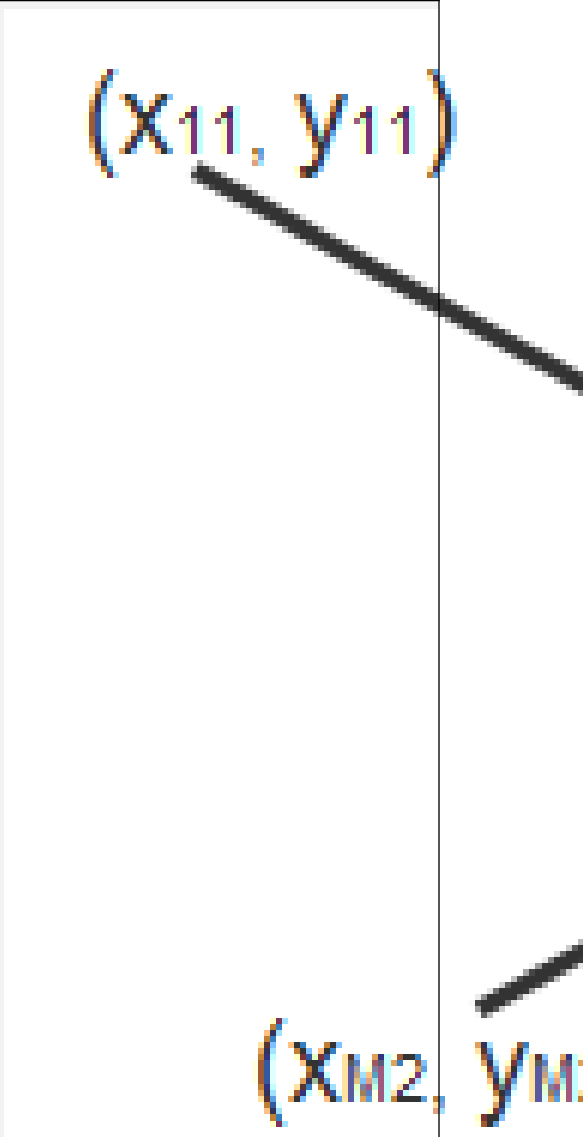
- `Rectangles` — *M*-by-4 matrix of the form [*x y width height*]. *M* is the number of rectangles.

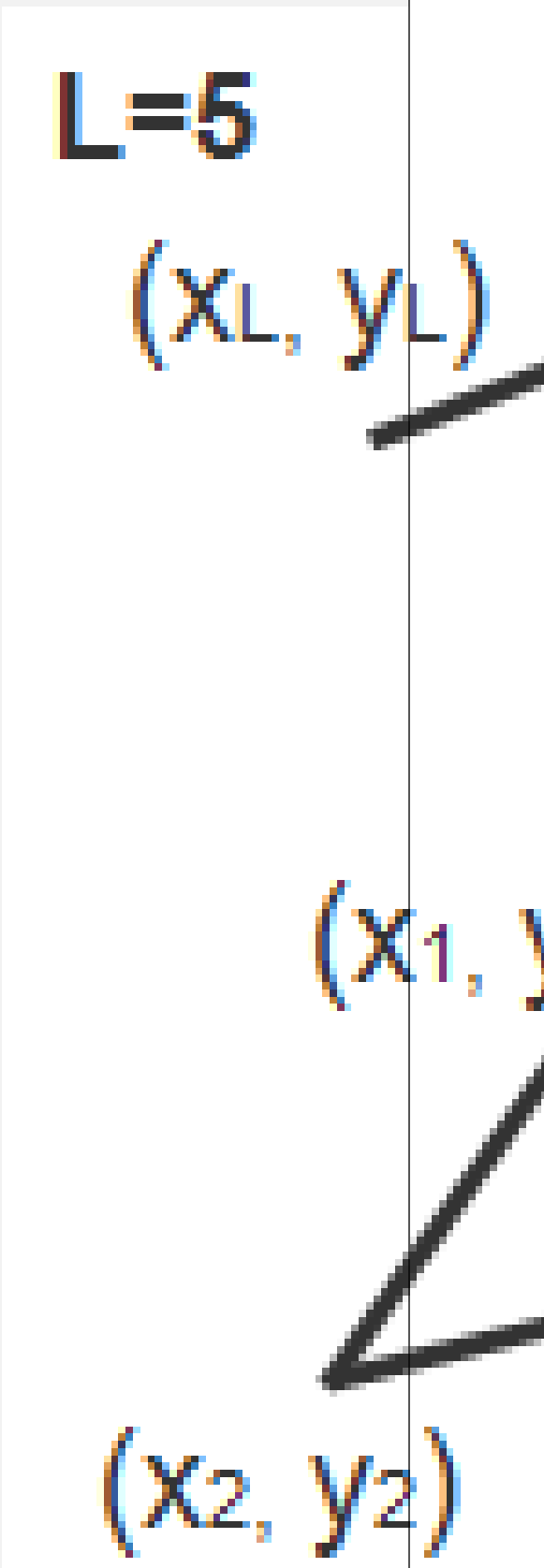
Shape	Input to the Pts Port	Drawn Shape
Single Rectangle	Four-element row vector in the format `[x y width height]`, where: • `x` and `y` are the coordinates of the upper-left corner of the rectangle. • `width` and `height` parameters are the width and height of the rectangle, in pixels. The values of `width` and `height` must be greater than 0.	

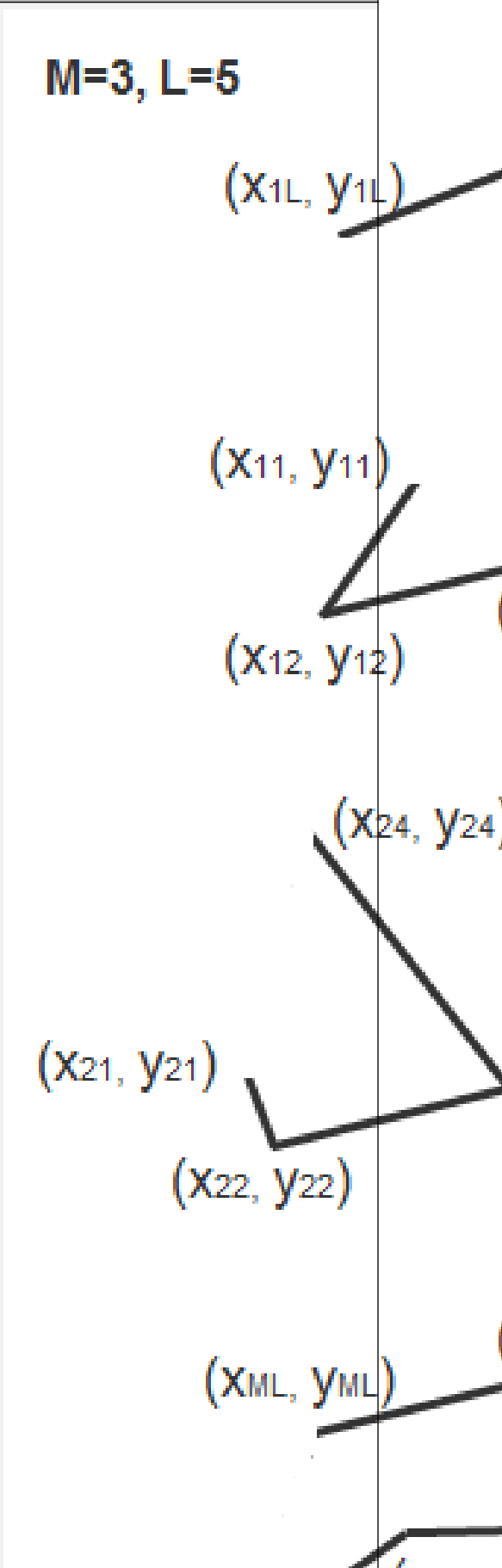
Shape	Input to the Pts Port	Drawn Shape
M Rectangles	*M*-by-4 matrix in the format $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$,where each row of the matrix corresponds to a different rectangle and is of the same form as the vector for a single rectangle.	

- `Lines` — M-by-$2L$ matrix of the form $[x_1, y_1, x_2, y_2,\ldots,x_L, y_L]$. M is the number of polylines.

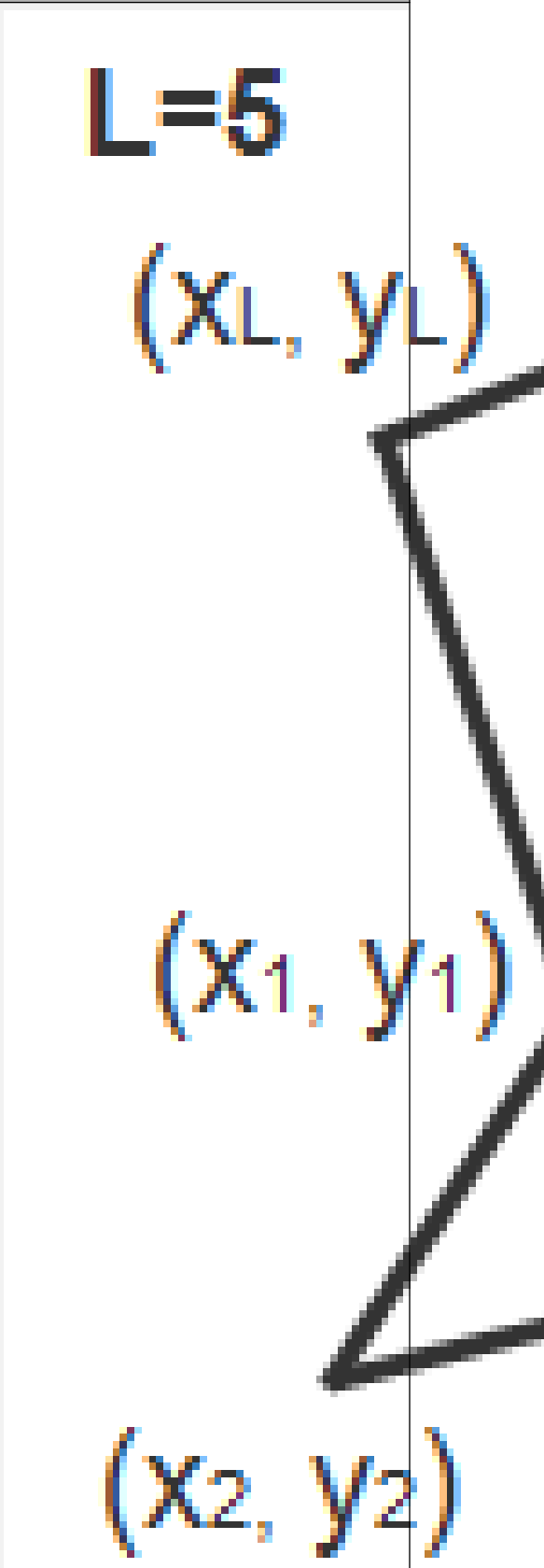
Shape	Input to the Pts Port	Drawn Shape
Single Line	Four-element row vector in the format `[x1 y1 x2 y2]`, where: • x_1 and y_1 are the coordinates of the beginning of the line. • x_2 and y_2 are the coordinates of the end of the line.	

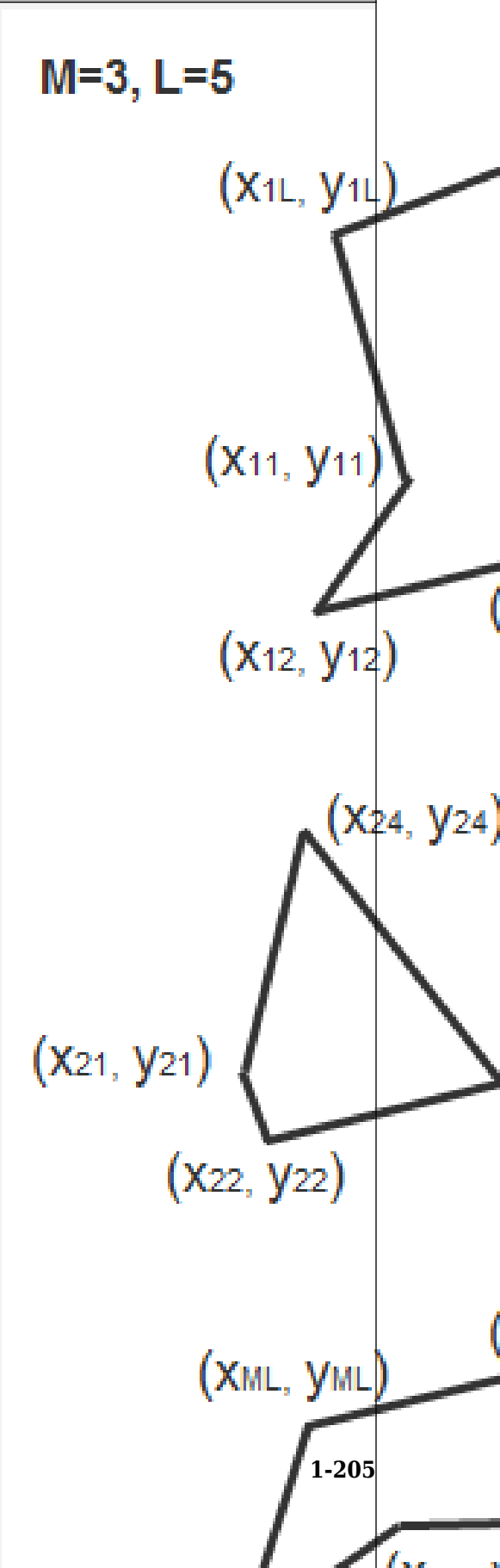
Shape	Input to the Pts Port	Drawn Shape
M Lines	*M*-by-4 matrix in the format $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ x_{21} & y_{21} & x_{22} & y_{22} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} \end{bmatrix}$,where each row of the matrix corresponds to a different line and is of the same form as the vector for a single line.	

Shape	Input to the Pts Port	Drawn Shape
Single Polyline with (L-1) Segments	Vector of size $2L$, where L is the number of vertices, in the format, `[`x_1`, `y_1`, `x_2`, `y_2`, ..., `x_L`, `y_L`]`, where • x_1 and y_1 are the coordinates of the beginning of the first line segment. • x_2 and y_2 are the coordinates of the end of the first line segment and the beginning of the second line segment. • x_L and y_L are the coordinates of the end of the $(L\text{-}1)^{th}$ line segment. The polyline always contains (L-1) number of segments because the first and last vertex points do not connect. The block produces an error message when the number of columns is less than two or not a multiple of two.	

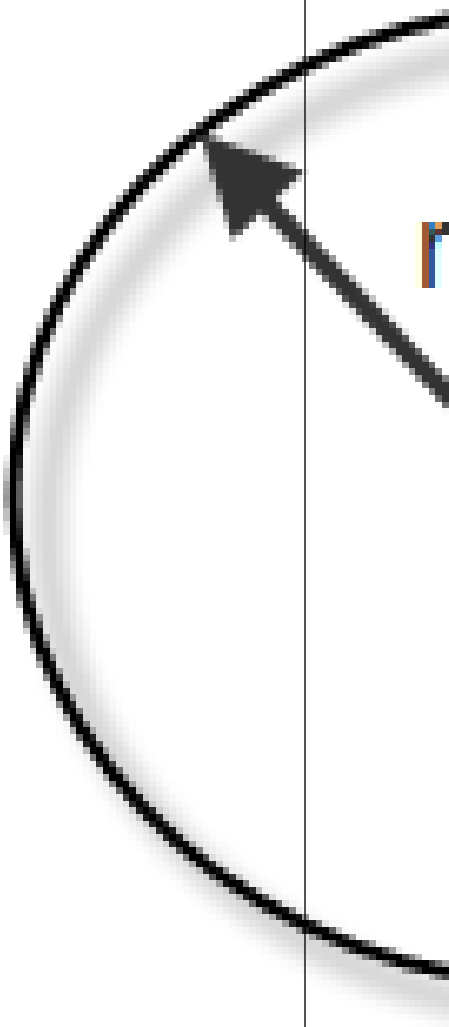
Shape	Input to the Pts Port	Drawn Shape
M Polylines with (*L*-1) Segments	*M*-by-*2L* matrix in the format $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} & y_{1L} \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} & y_{2L} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} & y_{ML} \end{bmatrix}$, where each row of the matrix corresponds to a different polyline and is of the same form as the vector for a single polyline. When one of the polylines contains less than (*L*–1) number of segments, fill the matrix by repeating the coordinates of the last vertex. The block produces an error message if the number of columns is less than two or not a multiple of two.	

- `Polygons` — M-by-$2L$ matrix of the form $[x_1, y_1, x_2, y_2,...,x_L, y_L]$. M is the number of polygons. The points are specified in consecutive order to form a polygon.

Shape	Input to the Pts Port	Drawn Shape
Single Polygon with L line segments	Row vector of size $2L$, where L is the number of vertices, in the format `[x1 y1 x2 y2 ... xL yL]`, where: • x_1 and y_1 are the coordinates of the beginning of the first line segment. • x_2 and y_2 are the coordinates of the end of the first line segment and the beginning of the second line segment. • x_L and y_L are the coordinates of the end of the $(L\text{-}1)^{th}$ line segment and the beginning of the L^{th} line segment. The block connects `[x1 y1]` to `[xL yL]` to complete the polygon. The block produces an error if the number of columns is less than two or not a multiple of two.	

Shape	Input to the Pts Port	Drawn Shape
M Polygons with the largest number of line segments in any line being *L*	*M*-by-2*L* matrix in the format $\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} & \cdots & x_{1L} & y_{1L} \\ x_{21} & y_{21} & x_{22} & y_{22} & \cdots & x_{2L} & y_{2L} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{M1} & y_{M1} & x_{M2} & y_{M2} & \cdots & x_{ML} & y_{ML} \end{bmatrix}$, where each row of the matrix corresponds to a different polygon and is of the same form as the vector for a single polygon. When one of the polygons contains less than *L* number of segments, fill the matrix by repeating the coordinates of the last vertex. The block produces an error message if the number of columns is less than two or not a multiple of two.	

- `Circles` — *M*-by-3 matrix of the form [*x y radius*]. *M* is the number of circles.

Shape	Input to the Pts Port	Drawn Shape
Single Circle	Three-element row vector in the format `[x y radius]`, where: • `x` and `y` are coordinates for the center of the circle. • `radius` is the radius of the circle, which must be greater than 0.	

Shape	Input to the Pts Port	Drawn Shape
M Circles	*M-by-3* matrix in the format $\begin{bmatrix} xctr_1 & yctr_1 & radius_1 \\ xctr_2 & yctr_2 & radius_2 \\ \vdots & \vdots & \vdots \\ xctr_M & yctr_M & radius_M \end{bmatrix}$, where each row of the matrix corresponds to a different circle and is of the same form as the vector for a single circle.	

Dependencies

- Double-precision floating point is only supported if the input to the **Image** or **R**, **G**, and **B** ports is floating point.
- Single-precision floating point is only supported if the input to the **Image** or **R**, **G**, and **B** ports is floating point.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

ROI — Region of interest
4-element vector

Region of interest, specified as a four-element vector of the form [*x y width height*]. The region defines a rectangular area to draw shapes. The first two elements represent the one-based coordinates of the upper-left corner of the area. The remaining two elements represent the width and height of the area.

Dependencies

To enable this port, set the **Draw shapes in** parameter to `Specify region of interest via port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Clr — Color of shape
3-element vector | *M*-by-*3* matrix

Color of the shape, specified as either a *3*-element vector or an *M*-by-*3* matrix. *M* is the number of shapes and three denotes the number of color planes. You can specify different color for each of the *M* shapes, or specify one color for all shapes.

Dependencies

To enable this port, set the **Border color source** or **Fill color source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Output

Port_1 — Output image
M-by-*N* matrix of intensity values | *M*-by-*N*-by-*3* array of color values

Output image, returned as an *M*-by-*N* matrix or *M*-by-*N*-by-*3* array. The output image is of the same dimension as the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `enumerated` | `bus`

Parameters

Main Tab

Shape — Shape to draw
`Rectangles` (default) | `Lines` | `Polygons` | `Circles`

One or more shapes to draw on the image, specified as one of the following:

- `Rectangles` — Draw one or more rectangles on the input image or the video frame using the specified shape coordinates.
- `Lines` — Draw one or more lines on the input image or the video frame using the specified shape coordinates.
- `Polygons` — Draw one or more polygons on the input image or the video frame using the specified shape coordinates.
- `Circles` — Draw one or more circles on the input image or the video frame using the specified shape coordinates.

Fill shapes — Fill shapes with color
off (default) | on

Select **Fill shapes** to enable the color parameters to fill shapes.

Note If you are generating the code and selecting the **Fill shapes** check box, the word length of the block input(s) cannot exceed 16 bits.

Line width — Line width for the shapes
1 (default) | positive scalar

Specify the line width for the shapes to be drawn, as a positive scalar.

Fill color source — Input source to specify fill color
`Specify via dialog` (default) | `Input port`

Specify the input source for the fill color value. Select **Fill color source** as one of the following:

- `Specify via dialog` — To specify the color value by using **Fill color** parameter.
- `Input port` — To specify the color value by using **Clr** port.

Dependencies

To enable this parameter, select **Fill shapes**.

Fill color — Color to fill the shapes
`Black` (default) | `White` | `User-specified value`

Specify the shape fill color as either `Black`, `White` or `User-specified value`. The `User-specified value` can be scalar or a 3-element vector. Use a scalar to specify an intensity value and 3-element vector to specify RGB color value.

Color Value(s) for Filling One Shape or Multiple Shapes with the Same Color	Color Value(s) for Filling Multiple Shapes with Unique Colors
Scalar intensity value	*R*-element vector, where *R* is the number of shapes.
3-element RGB vector	*R*-by-*3* matrix, where *3* is the number of color planes and *R* is the number of shapes.

For each value in the **Color Value(s)** parameter, enter a number between the minimum and maximum values that can be represented by the data type of the input image. If you enter a value outside this range, the block produces an error message.

Dependencies

To enable this parameter, select the **Fill shapes** parameter.

Border color source — Input source to specify border color
`Specify via dialog` (default) | `Input port`

Specify the input source for the border color value. Select **Border color source** as one of the following:

- `Specify via dialog` — To specify the color value by using **Border color** parameter.
- `Input port` — To specify the color value by using **Clr** port.

Border color — Outline color of the shape
`Black` (default) | `White` | `User-specified value`

Specify the border color of shape as either `Black`, `White` or `User-specified value`. The `User-specified value` can be scalar or a 3-element vector. Use a scalar to specify an intensity value and 3-element vector to specify RGB color value.

Color Value(s) for Drawing One Shape or Multiple Shapes with the Same Color	Color Value(s) for Drawing Multiple Shapes with Unique Colors
Scalar intensity value	*R*-element vector, where *R* is the number of shapes.
3-element RGB vector	*R*-by-*3* matrix, where three is the number of color planes and *R* is the number of shapes.

For each value in the **Color Value(s)** parameter, enter a number between the minimum and maximum values that can be represented by the data type of the input image. If you enter a value outside this range, the block produces an error message.

Color value(s) — Color value to draw the border or fill the shape
scalar | 3-element vector

Specify color value to draw the border or fill the shape as a scalar or 3-element vector. Use a scalar to specify intensity value and a 3-element vector for RGB color value.

Tunable: Yes

Dependencies

To enable this parameter set the **Border color** or **Fill color** parameter to `User-specified value`.

Opacity factor (between 0 and 1) — Opacity of the color
value in the range [0,1] | *R*-element vector

Specify the opacity of the color as a positive value in the range [0, 1]. `0` is transparent and `1` is opaque.

Opacity Factor value for Filling One Shape or Multiple Shapes with the Same Color	Opacity Factor value for Filling Multiple Shapes with Unique Colors
Scalar intensity value	*R*-element vector where *R* is the number of shapes

Dependencies

To enable this parameter, select the **Fill shapes** parameter.

Use antialiasing — Smoothing algorithm
off | on

Performs smoothing on the line, polygon, or circle.

Dependencies

- To enable this parameter, set the **Shape** parameter to either `Lines`, `Polygons`, or `Circles`.

Draw shapes in — Region to draw the shape
`Entire image` (default) | `Specify region of interest via port`

Specify the extent of the region to draw the specified shape. Select the value as one of the following:

- `Entire image` — To draw the shape on the entire image.
- `Specify region of interest via port` — To draw the shape within an ROI specified by the input **ROI** port.

Image signal — Number of input ports for the image signal
`One multidimensional signal` (default) | `Separate color signals`

Specify the number of input ports to use for specifying the input image. Select the value as one of the following:

- `One multidimensional signal` — To specify intensity or color image using a single port. The block enables the **Image** port for a *M*-by-*N*-by-*3* color image, or a *M*-by-*N* intensity image.
- `Separate color signals` — To specify red, green, and, blue color planes of the input color image. The block enables the **R**,**G**, and, **B** input ports. Each port accepts the *M*-by-*N* plane of an RGB color image.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Draw Markers | Insert Text

Edge Detection

Find edges of objects in images using Sobel, Prewitt, Roberts, or Canny method



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Edge Detection block finds edges of objects in an input image. The block supports four methods: Sobel, Prewitt, Roberts, and Canny. The first three methods find the edges by approximating the gradient magnitude of the image. The block convolves the input matrix with the Sobel, Prewitt, or Roberts kernel respectively, and outputs the resulting gradient components of the image. Additionally, the block outputs a binary image of the edge detections by thresholding the gradient magnitude.

The Canny method computes the gradient of input image using the derivative of the Gaussian filter. It then finds edges by looking for the local maxima of the gradient. This method uses two thresholds to detect the strong and weak edges. The weak edges are part of the output only if they are connected to strong edges. The block outputs a binary image of the edge detections for this method.

Ports

Input

I — Input image
matrix

Input image, specified as a matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Note The Canny method does not support 8-bit, 16-bit, or 32-bit signed fixed-point integers or 8-bit, 16-bit, or 32-bit unsigned integers.

Th — Threshold
scalar | vector

Specify a threshold as a scalar or a two-element vector to detect edges from the gradient. The data type of this value must match the input image data type. For the Canny edge detection method, the threshold must be a two-element vector in the range [0 1]. For the other methods, the threshold is a scalar.

Dependencies

To enable this port:

- Set the **Method** parameter to `Canny`. Alternatively, set the **Method** parameter to `Sobel`, `Prewitt`, or `Roberts`, and also set the **Output type** parameter to `Binary image` or `Binary image and gradient components`.
- Select the **User-defined threshold**.
- Set the **Threshold source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Note The Canny method does not support 8-bit, 16-bit, or 32-bit signed fixed-point integers or 8-bit, 16-bit, or 32-bit unsigned integers.

Output

Edge — Edge detections
matrix

Edge detections, returned as a binary image. The pixels with the value of `1` represent the edge pixels.

Dependencies

This port is always enabled when you set the **Method** parameter to `Canny`. To enable this port for the `Sobel`, `Prewitt`, or `Roberts` methods, set the **Output type** parameter to `Binary image` or `Binary image and gradient components`.

Data Types: `Boolean`

Gv — Gradient responses to vertical edges
matrix

Matrix of gradient responses to the vertical edges.

Dependencies

To enable this port, set these parameters:

- **Method** — `Sobel` or `Prewitt`
- **Output type** — `Gradient components` or `Binary image and gradient components`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Gh — Gradient responses to horizontal edges
matrix

Matrix of gradient responses to the horizontal edges.

Dependencies

To enable this port, set these parameters:

- **Method** — `Sobel` or `Prewitt`
- **Output type** — `Gradient components` or `Binary image and gradient components`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

G45 — Gradient responses to edges at 45 degrees
matrix

Matrix of gradient responses to edges at 45 degrees.

Dependencies

To enable this port, set these parameters:

- **Method** — `Roberts`
- **Output type** — `Gradient components` or `Binary image and gradient components`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

G135 — Gradient responses to edges at 135 degrees
matrix

Matrix of gradient responses to edges at 135 degrees.

Dependencies

To enable this port, set these parameters:

- **Method** — `Roberts`
- **Output type** — `Gradient components` or `Binary image and gradient components`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main

Method — Edge detection method
`Sobel` (default) | `Prewitt` | `Roberts` | `Canny`

Specify a method to detect edges in the input image. You can choose from `Sobel`, `Prewitt`, `Roberts` or `Canny`.

Output Type — Output image type
`Binary image` (default) | `Gradient components` | `Binary image and gradient components`

Specify the type of output image from one of these options:

- `Binary image` — The block outputs a Boolean matrix at the **Edge** port. The elements of the matrix labeled `1` represent the edge pixels.
- `Gradient components` — The block outputs the gradient components. For the `Sobel` and `Prewitt` methods, the gradient components correspond to the horizontal and vertical edge responses at the **Gh** and **Gv** ports respectively. For the `Roberts` method the gradient components correspond to 45 degree and 135 degree edge responses at the **G45** and **G135** ports respectively.

- `Binary image and gradient components` — The block outputs both the binary image and the gradient components of the image

Dependencies

To enable this parameter, set the **Method** parameter to `Sobel`, `Prewitt`, or `Roberts`.

User-defined threshold — User-defined threshold
off (default) | on

Specify a desired threshold value via dialog box or the **Th** port.

Dependencies

This parameter is always enabled when you set the **Method** parameter to `Canny`. To enable this parameter for the `Sobel`, `Prewitt`, or `Roberts` methods, set the **Output type** parameter to `Binary image` or `Binary image and gradient components`.

Threshold source — Threshold source
`Input port` (default) | `Specify via dialog`

Select the source to enter the threshold value from one of these options:

- `Input port` — Specify a threshold value using the **Th** port. The data type of this value must match the input image data type.
- `Specify via dialog` — Enter the desired threshold value in the dialog box.

Dependencies

To enable this parameter:

- Set the **Method** parameter to `Canny`. Alternatively set the **Method** parameter to `Sobel`, `Prewitt`, or `Roberts`, and also set the **Output type** parameter to `Binary image` or `Binary image and gradient components`.
- Select the **User-defined threshold**.

Threshold — Threshold
20 (default) | scalar

Specify the threshold value that is within the range of your input data. This threshold applies to Sobel, Prewitt, and Roberts edge detection methods.

Dependencies

To enable this parameter, set these block parameters:

- **Method** — `Sobel`, `Prewitt`, or `Roberts`
- **Output type** — `Binary image` or `Binary image and gradient components`
- **User-defined threshold** — `On`.
- **Threshold source** — `Specify via dialog`

Threshold, [Low High] — Threshold for Canny edge detection
[0.25 0.60] (default) | vector

Specify the threshold for the Canny edge detection as a two-element vector in the range [0 1]. The elements of the vector specifies the low and high threshold values for the weak and strong edges respectively.

Dependencies

To enable this parameter, set these block parameters:

- **Method** — `Canny`
- **User-defined threshold** — `On`.
- **Threshold source** — `Specify via dialog`

Threshold scale factor:(used to automatically calculate threshold value) — Threshold scale factor
4 (default) | scalar

Specify the threshold scale factor to automatically calculate threshold value. The block computes the automatic threshold using the mean of the gradient magnitude squared image and multiplies the value of this parameter with the automatic threshold value to determine a new threshold value.

Dependencies

To enable this parameter, set these block parameters:

- **Method** — `Sobel`, `Prewitt`, or `Roberts`
- **Output type** — `Binary image` or `Binary image and gradient components`
- **User-defined threshold** — `Off`

Edge Thinning — Edge thinning
`off` (default) | `on`

Select this parameter to reduce the thickness of the edges in your output images. This option requires additional processing time and memory resources.

Tip This block is most efficient in terms of memory usage and processing time when you clear the **Edge thinning** check box and use the **Threshold** parameter to specify a threshold value.

Dependencies

To enable this parameter, set these block parameters:

- **Method** — `Sobel`, `Prewitt`, or `Roberts`
- **Output type** — `Binary image` or `Binary image and gradient components`

Approximate percentage of weak edge and nonedge pixels (used to automatically calculate threshold values) — Approximate percentage of weak edge and nonedge pixels
70 (default) | scalar

Specify the approximate percentage of weak edge and nonedge image pixels. The block computes the automatic threshold values using this value.

Dependencies

To enable this parameter, set these block parameters:

- **Method** — `Canny`
- **User-defined threshold** — `Off`

Standard deviation of Gaussian filter — Standard deviation of Gaussian filter
1 (default) | scalar

Specify the standard deviation of Gaussian filter used for Canny edge detection algorithm.

Dependencies

To enable this parameter, set the **Method** parameter to `Canny`.

Data Types

For information about the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

References

[1] Gonzales, Rafael C. and Richard E. Woods. *Digital Image Processing. 2nd ed*. Englewood Cliffs, NJ: Prentice Hall, 2002.

[2] Pratt, William K. *Digital Image Processing, 2nd ed*. NY: John Wiley & Sons, 1991.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

When you select the **Normalized output** check box, the block input cannot be fixed point.

See Also

`edge` | `edge3`

Erosion

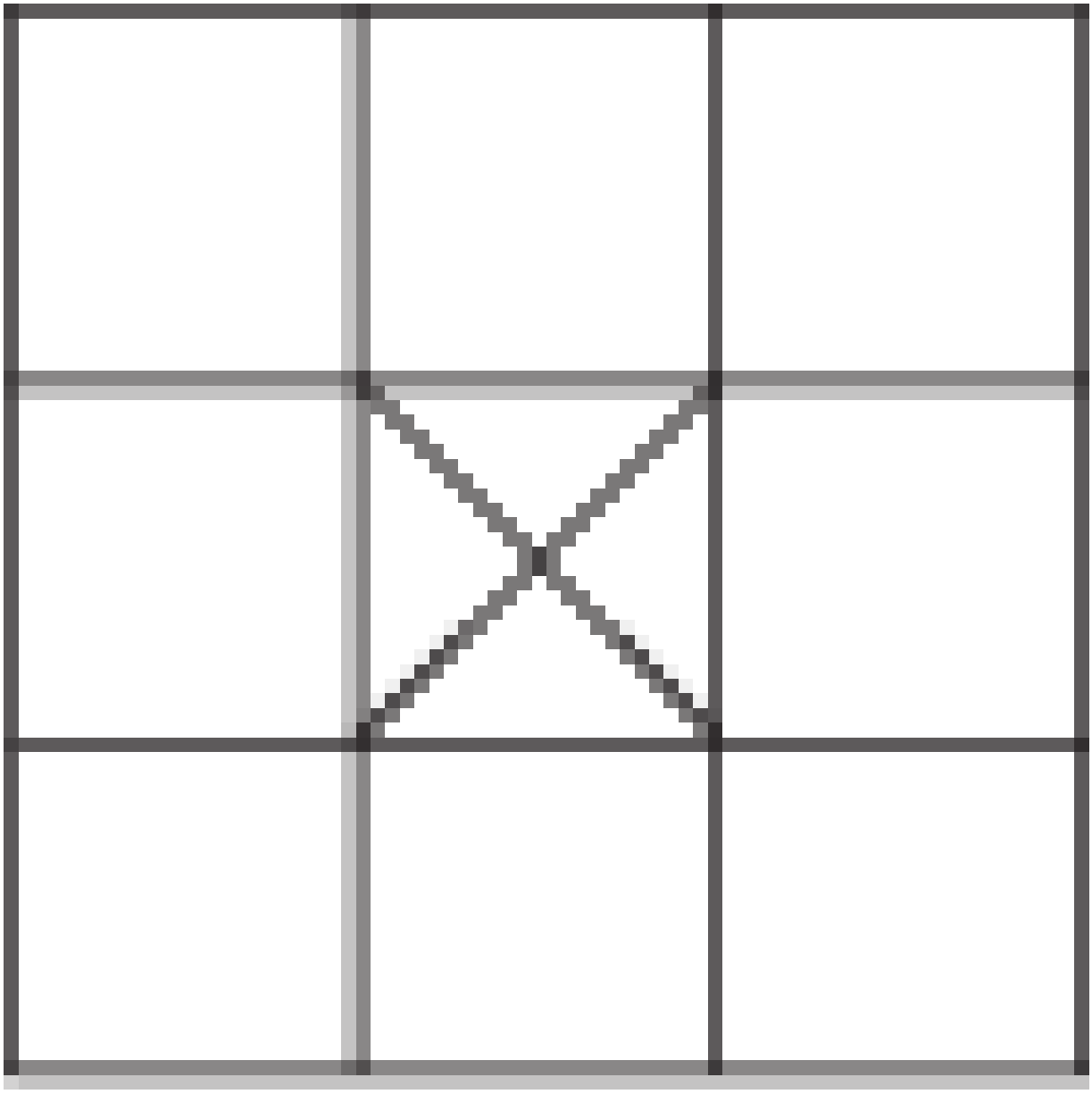
Find local minima in binary or intensity image



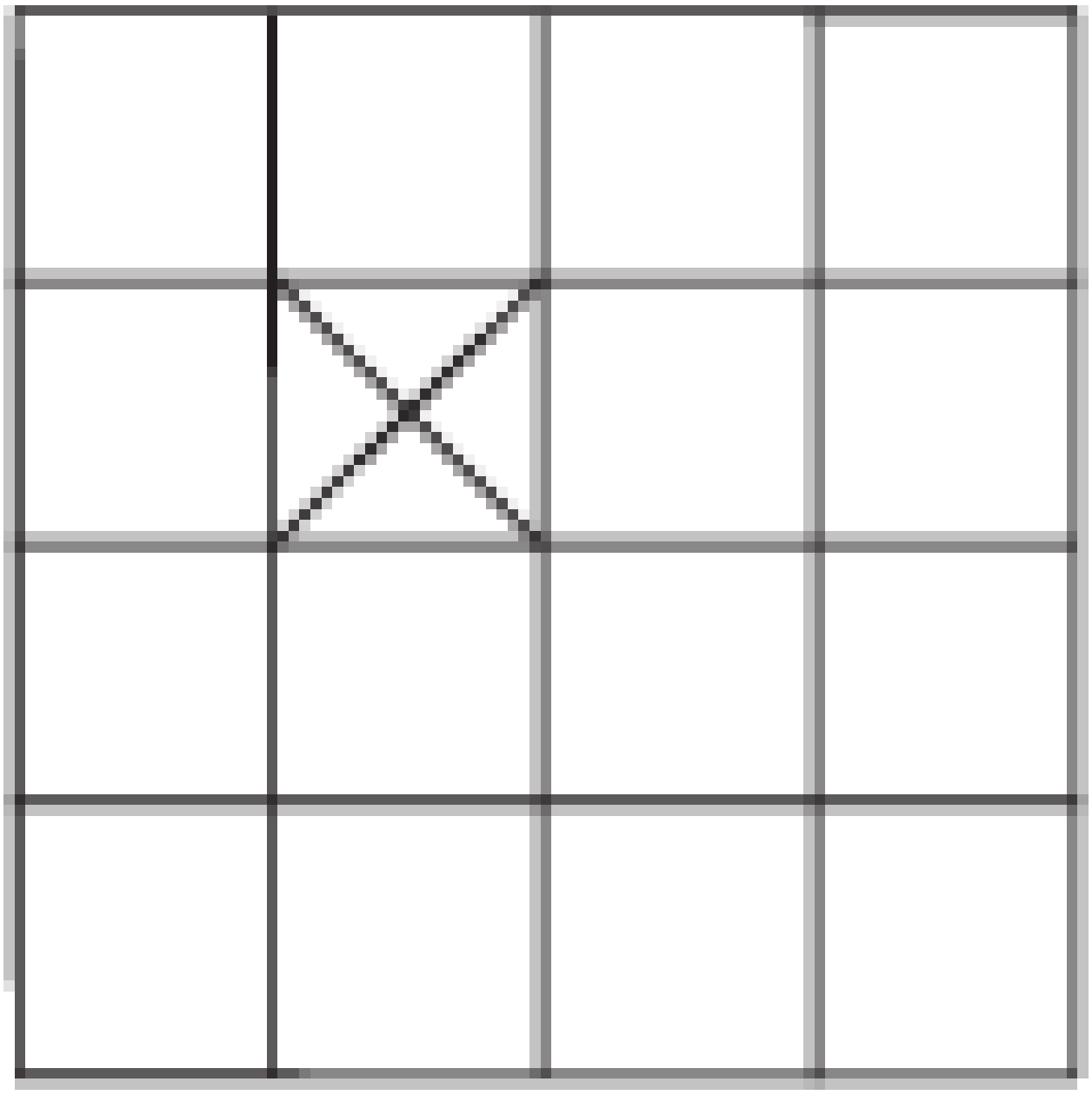
Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Erosion block slides a neighborhood or structuring element over an image, finds the local minima, and creates an output matrix from these minimum values. If the neighborhood or structuring element has a center element, the block places the minima there, as shown in this figure.



If the neighborhood or structuring element does not have an exact center, the block has a bias toward the upper-left of the central elements and places the minima there, as shown in this figure.



This block uses flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of binary or intensity values. This port is unnamed until you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values, representing a structuring element, as a vector or matrix of `1`s and `0`s.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Eroded image
matrix

Eroded image, returned as a matrix. The size and data type of the output image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of the neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`strel('square',4)` (default)

Specify the neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object, or an array of `strel` objects. When you specify an array of `strel` objects, the block applies each object to the entire image in sequence.

If the structuring element is decomposable into smaller elements, the block execution speed improves due to the more efficient algorithm.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Dilation | Opening | Closing | Label | Bottom-hat | Top-hat

Functions

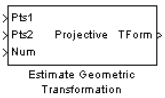
`imerode` | `imdilate` | `imopen` | `imclose`

Objects

`strel`

Estimate Geometric Transformation

Estimate geometric transformation from matching point pairs



Libraries:
Computer Vision Toolbox / Geometric Transformations

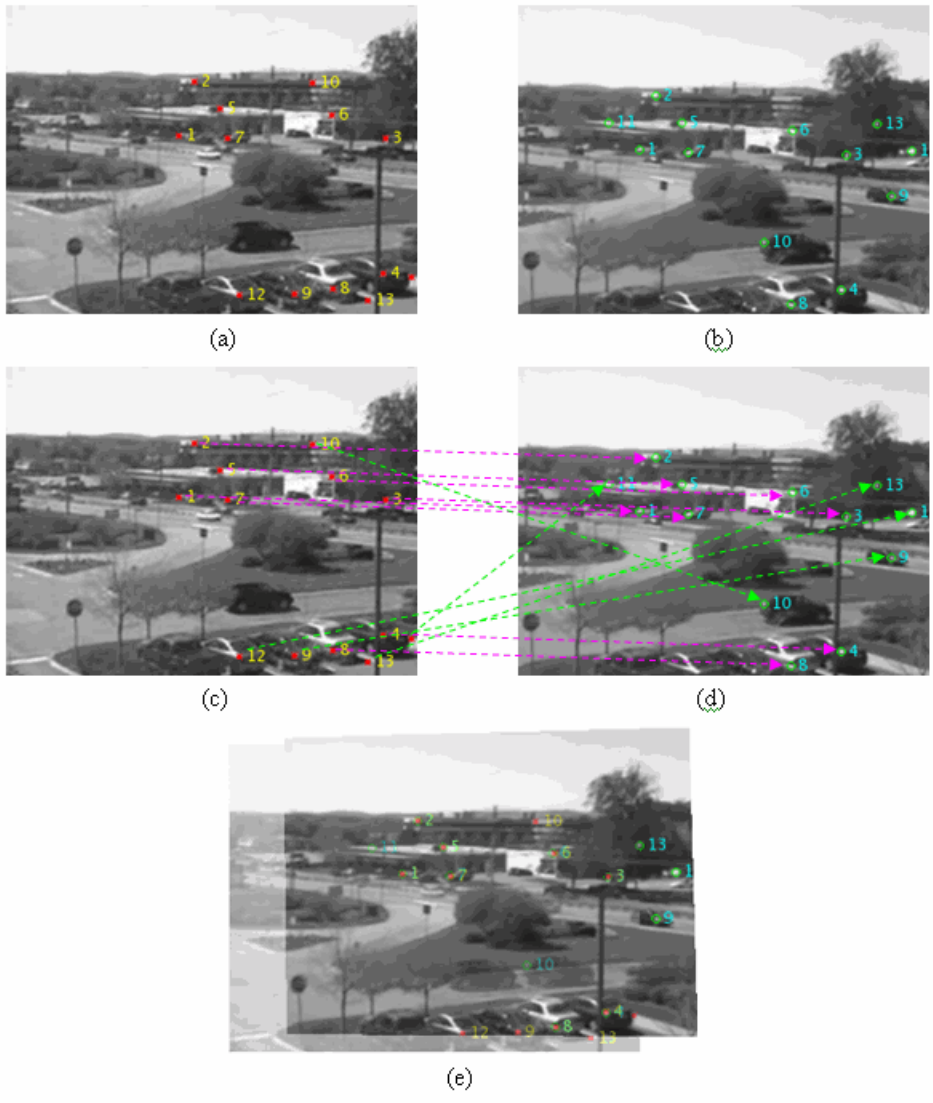
Description

Use the Estimate Geometric Transformation block to find the transformation matrix which maps the greatest number of point pairs between two images. A point pair refers to a point in the input image and its related point on the image created using the transformation matrix. You can select to use the RANdom SAmple Consensus (RANSAC) or the Least Median Squares algorithm to exclude outliers and to calculate the transformation matrix. You can also use all input points to calculate the transformation matrix.

Examples

Calculate transformation matrix from largest group of point pairs

1. Examples of input data and application of the Estimate Geometric Transformation block appear in the following figures. Figures (a) and (b) show the point pairs. The points are denoted by stars or circles, and the numbers following them show how they are paired. Some point pairs can be mapped by the same transformation matrix. Other point pairs require a different transformation matrix. One matrix exists that maps the largest number of point pairs, the block calculates and returns this matrix. The block finds the point pairs in the largest group and uses them to calculate the transformation matrix. The point pairs connected by the magenta lines are the largest group.
2. The transformation matrix can then be used to stitch the images as shown in Figure (e).



Ports

Input

Pts1 — Point coordinates
M-by-2 matrix

Point coordinates, specified as an *M*-by-2 matrix of one-based [*x* *y*] point coordinates, where *M* represents the number of points.

The block outputs the same data type for the transformation matrix as the **Pts1** and **Pts2** image points.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Pts2 — Point coordinates
M-by-2 matrix

Point coordinates, specified as an *M*-by-2 matrix of one-based [*x* *y*] point coordinates, where *M* represents the number of points.

The block outputs the same data type for the transformation matrix as the **Pts1** and **Pts2** image points.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Num — Number of valid points
scalar

Number of valid points to find in **Pts1** and **Pts2**, specified as a scalar. This port appears when you enable the **Allow variable-size signal input** parameter.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

TForm — Transformation
3-by-2 matrix | 3-by-3 matrix

Transformation, returned as either a 3-by-2 or a 3-by-3 matrix. The block outputs the same data type for the transformation matrix as the **Pts1** and **Pts2** image points.

Dependencies

- When **Pts1** and **Pts2** are single or double, the output transformation matrix will also have single or double data type.
- When **Pts1** and **Pts2** images are built-in integers, the option is available to set the transformation matrix data type to either `Single` or `Double`.

Data Types: `single` | `double`

Inlier — Points used
M-by-1 vector

Points used to calculate `TForm`, returned as an *M*-by-1 vector.

Dependencies

- The **Inlier** port appears when you enable the **Output Boolean signal indicating which point pairs are inliers** parameter.

Data Types: `Boolean`

Parameters

Transformation type — Transformation type
`Affine` (default) | `Nonreflective similarity` | `Projective`

Specify the transformation type as either `Nonreflective similarity`, `Affine`, or `Projective`. See “Transformations” on page 1-231 for a more detailed discussion.

Dependencies

You can set additional parameters depending on the transformation type:

- For `Projective` transformation, you can specify a scalar algebraic distance threshold for determining inliers.
- For `Affine` or `Nonreflective similarity` transformation, you can specify the distance threshold for determining inliers in pixels.

Find and exclude outliers — Find and exclude outliers
on (default) | off

Enable to find and exclude outliers from the input points and use only the inlier points to calculate the transformation matrix. When you turn this parameter off, all input points are used to calculate the transformation matrix.

Method — Method to find outliers
`RANdom SAmple Consensus (RANSAC)` (default) | `Least Median of Squares`

Select the method to find outliers as either `RANdom SAmple Consensus (RANSAC)` or `Least Median of Squares` See “RANSAC and Least Median Squares Algorithms” on page 1-230 for a more detailed discussion.

Dependencies

This parameter appears when you enable the **Find and exclude outliers** check box.

Algebraic distance threshold for determining inliers — Algebraic distance threshold for determining inliers
1.5 (default) | scalar

Specify a scalar threshold value for determining inliers. The threshold controls the upper limit used to find the algebraic distance in the RANSAC algorithm.

Dependencies

This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and the **Transformation type** parameter to `Projective`.

Distance threshold for determining inliers (in pixels) — Distance threshold for determining inliers (in pixels)
`1.5` (default) | scalar

Specify the upper limit distance a point can differ from the projection location of its corresponding point.

Dependencies

This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and you set the value of the **Transformation type** parameter to `Nonreflective similarity` or `Affine`.

Determine number of random samplings using — Determine number of random samplings using
`Specified value` (default) | `Desired confidence`

Select `Specified value` to enter a positive integer value for the number of random samplings. Select `Desired confidence` to set the number of random samplings as a percentage and a maximum number.

Dependencies

This parameter appears when you select the **Find and exclude outliers** check box, and you set the value of the **Method** parameter to `Random Sample Consensus (RANSAC)`.

Number of random samplings — Number of random samplings
`500` (default) | scalar

Specify the number of random samplings for the algorithm to perform.

Dependencies

This parameter appears when you set the value of the **Determine number of random samplings using** parameter to `Specified value`.

Desired confidence (in %) — Desired confidence (in %)
`99` (default) | scalar

Specify a percent desired confidence by entering a number between `0` and `100`. The value represents the probability of the algorithm to find the largest group of points that can be mapped by a transformation matrix.

Dependencies

This parameter appears when you set the **Determine number of random samplings using** parameter to `Desired confidence`.

Maximum number of random samplings — Maximum number of random samplings
`1000` (default) | integer

Specify an integer number for the maximum number of random samplings.

Dependencies

This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)` and you set the value of the **Determine number of random samplings using** parameter to `Desired confidence`.

Stop sampling earlier when a specified percentage of point pairs are determined to be inlier — Stop sampling
off (default) | on

Enable this parameter to stop random sampling when a percentage of input points have been found as inliers.

Dependencies

This parameter appears when you set the **Method** parameter to `Random Sample Consensus (RANSAC)`.

Perform additional iterative refinement of the transformation matrix — Perform additional iterative refinement
off (default) | on

Specify whether to perform refinement on the transformation matrix.

Dependencies

This parameter appears when you select **Find and exclude outliers** check box.

Output Boolean signal indicating which point pairs are inliers — Output Boolean signal
off (default) | on

Enable this parameter to output the inlier point pairs that were used to calculate the transformation matrix.

Dependencies

- This parameter appears when you select the **Find and exclude outliers** check box.
- The block will not use this parameter with signed or double, data type points.

When Pts1 and Pts2 are built-in integers, set transformation matrix date type to — Set transformation matrix date type
`Single` (default) | `Double`

Specify transformation matrix data type as `Single` or `Double` when the input points are built-in integers.

Dependencies

The block will not use this parameter with signed or double, data type points.

Allow variable-sized signal input — Allow variable-sized signal input
on (default) | off

Enable this parameter to allow variable-sized signal input.

Dependencies

Block Characteristics

Data Types	`double` | `integer`[a] | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

a Generated code will be restricted to MATLAB host computers when you set the FFT implementation parameter to FFTW, or when the transform length is not a power of two.

Troubleshooting

The success of estimating the correct geometric transformation depends heavily on the quality of the input point pairs. If you chose the RANSAC or LMS algorithm, the block will randomly select point pairs to compute the transformation matrix and will use the transformation that best fits the input points. There is a chance that all of the randomly selected point pairs may contain outliers despite repeated samplings. In this case, the output transformation matrix, `TForm`, is invalid, indicated by a matrix of zeros.

To improve your results, try the following:
Increase the percentage of inliers in the input points.

Increase the number for random samplings.
For the RANSAC method, increase the desired confidence.
For the LMS method, make sure the input points have 50% or more inliers.
Use features appropriate for the image contents
Be aware that repeated patterns, for example, windows in office building, will cause false matches when you match the features. This increases the number of outliers.
Do not use this function if the images have significant parallax. You can use the `estimateFundamentalMatrix` function instead.
Choose the minimum transformation for your problem.
If a projective transformation produces the error message, "A portion of the input image was transformed to the location at infinity. Only transformation matrices that do not transform any part of the image to infinity are supported.", it is usually caused by a transformation matrix and an image that would result in an output distortion that does not fit physical reality. If the matrix was an output of the Estimate Geometric Transformation block, then most likely it could not find enough inliers.

Algorithms

RANSAC and Least Median Squares Algorithms

The *RANSAC* algorithm relies on a distance threshold. A pair of points, p_i^a(image a, Pts1) and p_i^b (image b, Pts2) is an inlier only when the distance between p_i^b and the projection of p_i^abased on the transformation matrix falls within the specified threshold. The distance metric used in the RANSAC algorithm is as follows:

$$d = \sum_{i=1}^{Num} \min(D(p_i^b, \psi(p_i^a{:}H)), t)$$

The Least Median Squares algorithm assumes at least 50% of the point pairs can be mapped by a transformation matrix. The algorithm does not need to explicitly specify the distance threshold. Instead, it uses the median distance between all input point pairs. The distance metric used in the Least Median of Squares algorithm is as follows:

$$d = median(D(p_1^b, \psi(p_1^a{:}H)), D(p_2^b, \psi(p_2^a{:}H)), \ldots, D(p_{Num}^b, \psi(p_N^a{:}H)))$$

For both equations:

p_i^a is a point in image a (`Pts1`)

p_i^b is a point in image b (`Pts2`)

$\psi(p_i^a{:}H)$ is the projection of a point on image a based on transformation matrix H

$D(p_i^b, p_j^b)$ is the distance between two point pairs on image b

t is the threshold

$Num$is the number of points

The smaller the distance metric, the better the transformation matrix and therefore the more accurate the projection image.

Transformations

The Estimate Geometric Transformation block supports `Nonreflective similarity`, `Affine`, and `Projective` transformation types, which are described in this section.

Nonreflective similarity transformation supports translation, rotation, and isotropic scaling. It has four degrees of freedom and requires two pairs of points.



The transformation matrix is: $H = \begin{bmatrix} h_1 & -h_2 \\ h_2 & h_1 \\ h_3 & h_4 \end{bmatrix}$

The projection of a point $[x\ y]$ by $H$is: $[\widehat{x}\ \widehat{y}] = [x\ y\ 1]H$

affine transformation supports anisotropic scaling in addition to all transformations that the nonreflective similarity transformation supports. It has six degrees of freedom that can be determined from three pairs of noncollinear points.



The transformation matrix is: $H = \begin{bmatrix} h_1 & h_4 \\ h_2 & h_5 \\ h_3 & h_6 \end{bmatrix}$

The projection of a point $[x\ y]$ by $H$is: $[\widehat{x}\ \widehat{y}] = [x\ y\ 1]H$

Projective transformation supports tilting in addition to all transformations that the affine transformation supports.



The transformation matrix is : $h = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$

The projection of a point $[x\ y]$ by $H$is represented by homogeneous coordinates as:
$[\widehat{u}\ \widehat{v}\ \widehat{w}] = [x\ y\ 1]H$

Distance Measurement

For computational simplicity and efficiency, this block uses algebraic distance. The algebraic distance for a pair of points, $\begin{bmatrix} x^a & y^a \end{bmatrix}^T$ on image a, and $\begin{bmatrix} x^b & y^b \end{bmatrix}$ on image b , according to transformation H,is defined as follows;

For projective transformation:

$D(p_i^b, \psi(p_i^a : H)) = ((\widehat{u}^a - \widehat{w}^a x^b)^2 + (\widehat{v}^a - \widehat{w}^a y^b)^2)^{\frac{1}{2}}$, where $\begin{bmatrix} \widehat{u}^a & \widehat{v}^a & \widehat{w}^a \end{bmatrix} = \begin{bmatrix} x^a & y^a & 1 \end{bmatrix} H$

For Nonreflective similarity or affine transformation: $D(p_i^b, \psi(p_i^a : H)) = ((\widehat{x}^a - x^b)^2 + (\widehat{y}^a - \widehat{y}^b)^2)^{\frac{1}{2}}$,

where $\begin{bmatrix} \widehat{x}^a & \widehat{y}^a \end{bmatrix} = \begin{bmatrix} x^a & y^a & 1 \end{bmatrix} H$

Algorithm

The block performs a comparison and repeats it K number of times between successive transformation matrices. If you select the **Find and exclude outliers** option, the RANSAC and Least Median Squares (LMS) algorithms become available. These algorithms calculate and compare a distance metric. The transformation matrix that produces the smaller distance metric becomes the new transformation matrix that the next comparison uses. A final transformation matrix is resolved when either:

- K number of random samplings is performed
- The RANSAC algorithm, when enough number of inlier point pairs can be mapped, (dynamically updating K)

The Estimate Geometric Transformation algorithm follows these steps:

1. A transformation matrix H is initialized to zeros
2. Set `count = 0` (Randomly sampling).
3. While `count < K` , where `K` is total number of random samplings to perform, perform the following;

   a. Increment the count; `count = count + 1`.
   b. Randomly select pair of points from images a and b, (2 pairs for Nonreflective similarity, 3 pairs for affine, or 4 pairs for projective).
   c. Calculate a transformation matrix H, from the selected points.
   d. If Hhas a distance metric less than that of H, then replace H with H.

      (Optional for RANSAC algorithm only)

      i. Update `K` dynamically.
      ii. Exit out of sampling loop if enough number of point pairs can be mapped by H.
4. Use all point pairs in images a and b that can be mapped by H to calculate a refined transformation matrix H
5. Iterative Refinement, (Optional for RANSAC and LMS algorithms)

   a. Denote all point pairs that can be mapped by H as inliers.
   b. Use inlier point pairs to calculate a transformation matrix H.
   c. If Hhas a distance metric less than that of H, then replace H with H, otherwise exit the loop.

Number of Random Samplings

The number of random samplings can be specified by the user for the RANSAC and Least Median Squares algorithms. You can use an additional option with the RANSAC algorithm, which calculates this number based on an accuracy requirement. The **Desired Confidence** level drives the accuracy.

The calculated number of random samplings, K used with the RANSAC algorithm, is as follows:

$$K = \frac{\log(1-p)}{\log(1-q^s)}$$

where

- p is the probability of independent point pairs belonging to the largest group that can be mapped by the same transformation. The probability is dynamically calculated based on the number of inliers found versus the total number of points. As the probability increases, the number of samplings, K, decreases.
- q is the probability of finding the largest group that can be mapped by the same transformation.
- s is equal to the value 2, 3, or 4 for Nonreflective similarity, affine, and projective transformation, respectively.

Iterative Refinement of Transformation Matrix

The transformation matrix calculated from all inliers can be used to calculate a refined transformation matrix. The refined transformation matrix is then used to find a new set of inliers. This procedure can be repeated until the transformation matrix cannot be further improved. This iterative refinement is optional.

Version History

Introduced in R2008a

References

[1] R. Hartley and A. Ziserman, “Multiple View Geometry in Computer Vision,” Second edition, Cambridge University Press, 2003

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

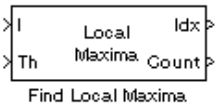
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

`fitgeotform2d`

Find Local Maxima

Find local maxima in matrices



Libraries:
Computer Vision Toolbox / Statistics

Description

The Find Local Maxima block finds the local maxima in a given input matrix. The block finds the local maxima of values within a region specified by the neighborhood size. The block compares the values in the search region to a user-specified threshold and identifies a value that is greater than or equal to the threshold as the local maximum. After finding the local maximum, the block sets all the matrix values in the neighborhood, including the maximum value, to zero. This step ensures that subsequent searches do not include this maximum. The process repeats until the block either finds all valid maxima or the specified number of local maxima.

The threshold and the neighborhood size must be chosen such that the block detects only the true local maxima and eliminates false peaks.

The input to this block can also be a Hough matrix generated from the Hough Transform block. The Find Local Maxima block outputs the one-based [*x* *y*] coordinates of the detected local maxima. The data sent to all input ports must be the same data type.

Ports

Input

I — Input
M-by-*N* matrix

Input, specified as the *M*-by-*N* matrix in which you want to detect the local maxima. If the input is a Hough matrix output from the Hough Transform block, select the **Input is Hough matrix spanning full theta range** parameter that renames this port as **Hough**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Th — Threshold for detecting local maxima
scalar

Threshold for the detecting local maxima, specified as a scalar.

Dependencies

To enable this port, set the **Source of threshold value** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Output

Idx — Coordinates of local maxima
P-by-2 matrix

Coordinates of the local maxima, returned as a *P*-by-2 matrix of one-based [*x y*] coordinates. *P* represents the number of local maxima.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32`

Count — Number of local maxima
scalar

Number of local maxima, returned as a scalar.

Dependencies

To enable this port, clear the check box for the **Output variable size signal** parameter.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32`

Parameters

Maximum number of local maxima — Maximum number of local maxima to detect

2 (default) | positive integer

Specify the maximum number of local maxima to detect in the input matrix. The value must be a positive integer greater than 0.

Neighborhood size — Size of neighborhood region

[5 7] (default) | two-element vector

Specify the size of the neighborhood region as a two-element vector in the form `[r c]`. The vector specifies the window size *r*×*c*, within which the block determines the local maxima. The window size must be greater than or equal to 1×1.

Source of threshold value — Source to enter the threshold value

`Specify via dialog` (default) | `Input port`

Specify the source for entering the threshold value. When you select the `Input port`, the block enables the **Th** port. When you select the `Specify via dialog`, the block enables the **Threshold** parameter.

Threshold — Threshold for detecting local maxima

10 (default) | scalar

Specify a scalar value that represents the threshold for detecting the local maxima.

Dependencies

To enable this parameter, set the **Source of threshold value** parameter to `Specify via dialog`.

Input is Hough matrix spanning full theta range — Whether input is a Hough matrix

off (default) | on

Select the parameter, if the input to this block is the Hough matrix output from the Hough Transform block. The Find Local Maxima block assumes that the input is a Hough matrix that is antisymmetric about the rho axis, with values of theta from -pi/2 to pi/2 radians. As a result, if the local maximum is near the boundary and its neighborhood region is outside the Hough matrix, the block detects only one local maximum. The block ignores the corresponding antisymmetric maximum.

Index output data type — Data type for coordinates of local maximums
`uint32` (default) | `double` | `single` | `uint8` | `uint16`

Specify the data type for the coordinates of local maximums returned at the output port **Idx**.

Output variable size signal — Support input signals of varying length
on (default) | off

Specify if the model supports input signals of varying length.

Count output data type — Data type for number of local maximums
`uint32` (default) | `double` | `single` | `uint8` | `uint16`

Specify the data type for the number of local maximums returned at the output port **Count**.

Dependencies

To enable this parameter, clear the **Output variable size signal** parameter.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

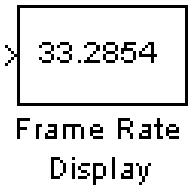
Blocks

Hough Lines | Corner Detection

Functions
houghpeaks | hough

Frame Rate Display

Calculate and display video frame rate



Libraries:
Computer Vision Toolbox / Sinks

Description

The Frame Rate Display block calculates and displays the average update rate of an input signal. For an input video signal, this block displays the video frame rate of the simulation. The calculated rate is in relation to the wall clock time. For example, if the model is updating the input signal 30 times for every second, this block displays `30` on the block mask.

This block supports images and video frames as input. Inputs can contain intensity or color values.

Ports

Input

Port_1 — Input data
matrix | array

Input data, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Update rate — Frequency of updates

`10` (default) | positive scalar

Specify this parameter to control how often the block updates the display. The block displays the average update rate across the specified number of video frames. For example, if this parameter is set to `10`, the Frame Rate Display block calculates the elapsed time to pass `10` video frames to the block, divides the elapsed time by `10`, and displays the average video frame rate.

Note If the input port of Frame Rate Display block is not connected to a signal, the block displays the base (fastest) rate of the Simulink model.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

See Also

Blocks

To Multimedia File | To Video Display | Video To Workspace | Video Viewer

From Multimedia File

Read video frames and audio samples from multimedia file



Libraries:
Computer Vision Toolbox / Sources

Description

The From Multimedia File block reads video frames, audio samples, or both from a multimedia file. The block imports data from a file, of a supported file format, into a Simulink model. For more information about supported file formats, see “Supported Platforms and File Formats” on page 1-246.

This block enables you to read WMA or WMV streams from your system or over a network connection. Similarly, the To Multimedia File block enables you to write WMA or WMV streams to your system or over a network connection. If your system does not have the codec necessary to support a file format, you must re-encode files of that format into a file format supported by the Computer Vision Toolbox.

If you have a Simulink Coder™ license, you can generate code from a model containing this block. To run a generated executable file, you may need to add precompiled shared library files to your system path. For more information, see “Simulink Shared Library Dependencies” and “Accelerating Simulink Models”.

Note This block supports code generation for a host computer that has file I/O available. You cannot use this block with Simulink Desktop Real-Time™ software because that product does not support file I/O.

Ports

Output

Image — Color image or video frame
array

Color image or video frame, returned as an *M*-by-*N*-by-*P* array. *M* and *N* are the number of rows and columns in each image. *P* is the number of color channels in each image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to RGB and **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Simulink.ImageType`

I — Grayscale image or video frame
matrix

Grayscale image or video frame, returned as an *M*-by-*N* matrix. *M* and *N* are the number of rows and columns in the grayscale image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to `Intensity`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

R — Red color component of image or video frame
matrix

Red color component of the image or video frame, returned as an *M*-by-*N* matrix. *M* and *N* are the number of rows and columns in the image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to RGB and **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

G — Green color component of image or video frame
matrix

Green color component of the image or video frame, returned as an *M*-by-*N* matrix. *M* and *N* are the number of rows and columns in the image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to RGB and **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

B — Blue color component of image or video frame
matrix

Blue color component of the image or video frame, returned as an *M*-by-*N* matrix. *M* and *N* are the number of rows and columns in the image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to `RGB` and **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Y — Luma component of image or video frame
matrix

Luma component of the image or video frame, returned as an *M*-by-*N* matrix. *M* and *N* are the number of rows and columns in the image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cb — Blue-difference chrominance component of image or video frame
matrix

Blue-difference chrominance component of the image or video frame, returned as an *M*-by-*N*/2 matrix. *M* and *N* are the number of rows and columns in the input image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cr — Red-difference chrominance component of image or video frame
matrix

Red-difference chrominance component of the image or video frame, returned as an *M*-by-*N*/2 matrix. *M* and *N* are the number of rows and columns in each input image.

Dependencies

To enable this port:

- Set the **File name** parameter to the name of a multimedia file that contains video or image data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Audio — Audio signal
vector | matrix

Audio signal, returned as an *M*-element vector or an *M*-by-*N* matrix. *M* is the number of samples in each channel and *N* is the number of channels.

Dependencies

To enable this port, set the **File name** parameter to the name of a multimedia file that contains audio data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Audio only` or `Video and audio`.

Data Types: `single` | `double` | `int16` | `uint8`

EOF — End-of-file indicator
scalar

End-of-file indicator, returned as a logical scalar for each video frame or audio sample. The end-of-file indicator is `1` when the block outputs the final video frame or audio sample from the file, and is `0` otherwise.

Dependencies

To enable this port, select the **Output end-of-file indicator** parameter.

Data Types: `Boolean`

Parameters

Main

File name — Name of multimedia file

`vipmen.avi` (default) | character vector

Specify the name of the multimedia file for the block to read. If the file is not located on your MATLAB path, select **Browse** and navigate to the file, or specify the full file path.

On Microsoft® Windows® platforms, this parameter supports URLs that point to Microsoft Media Server (MMS) streams.

Inherit sample time from file — Inherit sample time from file

`on` (default) | `off`

Specify whether the block inherits the sample time from the multimedia file or uses a specified sample time. When you select this parameter, the block inherits the sample time from the input multimedia file.

Tip Select this parameter when you do not know the intended sample rate of the multimedia file.

Desired sample time — Block sample time

`1/30` (default) | real positive scalar

Specify a sample time for the block.

Dependencies

To enable this parameter, clear the **Inherit sample time from file** parameter.

Number of times to play file — Number of times to play multimedia file

`inf` (default) | real positive integer

Specify the number of times to play the input multimedia file. The default value of `inf` specifies that the block repeats playing the multimedia file until the simulation stop time.

Output end-of-file indicator — End-of-file indicator

`off` (default) | `on`

Select this parameter to enable the **EOF** port, which indicates whether the output frame or sample is the final video frame or audio sample in the multimedia file.

Multimedia outputs — Type of media

`Video only` (default) | `Audio only` | `Video and audio`

Specify the type of media to read from a file containing both video and audio data.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains both audio and video data.

Output color format — Color format of video

`RGB` (default) | `Intensity` | `YCbCr 4:2:2`

Specify the output color format of the input video data. The block supports the `RGB`, `Intensity`, and `YCbCr 4:2:2` color formats.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains video data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.

Image signal — Output port configuration for RGB video

`One multidimensional signal` (default) | `Separate color signals` | `Simulink image signal`

Specify how the block outputs an RGB color video. If you select `One multidimensional signal`, the block outputs the video data using only the **Image** port. If you select `Separate color signals`, the block outputs the video data using three separate color channels, divided across the **R**, **G**, and **B** ports. If you select `Simulink image signal`, the block outputs a video in `Simulink.ImageType` datatype.

Dependencies

To enable this parameter:

- Set the **File name** parameter to the name of a multimedia file that contains video data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.
- Set the **Output color format** parameter to RGB.

Read range — Range of audio samples

`[1 Inf]` (default) | two-element row vector

Specify the range of audio samples to read from the input file as a two-element row vector of the form `[StartSample EndSample]`. `StartSample` is the first sample the block reads from the audio file, and `EndSample` is the final sample the block reads from the audio file. `StartSample` must be greater than or equal to `1`. `EndSample` must be greater than `StartSample`.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains audio data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Audio only`.

Samples per audio channel — Samples per audio channel

`1024` (default) | real positive integer

Specify the number of samples per audio channel.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains audio data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Audio only`.

Data Types

Video output data type — Video output data type

`single` (default) | `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `Inherit from file`

Specify the data type for the output video frames.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains video data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Video only` or `Video and audio`.

Audio output data type — Audio output data type

`int16` (default) | `double` | `single` | `uint8`

Specify the data type for the output audio samples.

Dependencies

To enable this parameter, set the **File name** parameter to the name of a multimedia file that contains audio data. If the specified file contains both audio and video data, set the **Multimedia outputs** parameter to `Audio only` or `Video and audio`.

Block Characteristics

Data Types	`double` | `integer`[a] | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

a Supports 16- and 32-bit signed and 8-bit unsigned integers.

More About

Supported Platforms and File Formats

Which supported file formats are available to you depends on the codecs installed on your system.

Platform	Supported File Formats
All Platforms	AVI (`.avi`) Motion JPEG 2000 (`.mj2`)
Microsoft Windows	**Image**: JPEG(`.jpg`) Bitmap (`.bmp`)
	Video: MPEG (`.mpeg`) MPEG-2 (`.mp2`) MPEG-1 (`.mpg`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Windows Media Video (`.wmv`, `.asf`, `.asx`) Any format supported by Microsoft DirectShow® 9.0 or higher.

Platform	Supported File Formats
	Audio: WAVE (`.wav`) Windows Media Audio File (`.wma`) Audio Interchange File Format (`.aif`, `.aiff`) Compressed Audio Interchange File Format (`.aifc`) MP3 (`.mp3`) Sun Audio (`.au`) Apple (`.snd`)
macOS	**Video**: MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (`.mov`) Any format supported by QuickTime®.
	Audio: Uncompressed AVI (`.avi`)
Linux®	Any format supported by your installed plug-ins for GStreamer 0.1 or higher, including Ogg Theora (`.ogg`).

Sample Time Calculations Used for Video and Audio Files

The sample rate that the block uses depends on the audio and video sample rate. While the block operates at a single rate in Simulink, the underlying audio and video streams can produce different rates. In some cases, the block makes a small adjustment to the video rate when the block outputs both audio and video.

Sample time = $\frac{ceil(AudioSampleRate/FPS)}{AudioSampleRate}$. *AudioSampleRate* is the sample rate for audio data, and *FPS* is the video frame rate.

When the audio sample time, $\frac{AudioSampleRate}{FPS}$, is not an integer,, the equation cannot reduce to $\frac{1}{FPS}$.

To prevent synchronization problems, when the audio stream leads the video stream by more than $\frac{1}{FPS}$, the block drops the corresponding video frame.

In summary, the block outputs one video frame at each Simulink time step. To calculate the number of audio samples to output at each time step, the block divides the audio sample rate by the video frame rate. If the audio sample rate does not divide evenly by the number of video frames per second (FPS), the block rounds the number of audio samples up to the nearest whole number. If necessary, the block periodically drops a video frame to maintain synchronization for large files.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:

- Host computer only. Excludes Simulink Desktop Real-Time code generation.
- The executable generated from this block relies on prebuilt dynamic library files (`.dll` files) included with MATLAB. Use the `packNGo` function to package the code generated from this block and all the relevant files in a compressed ZIP file. Using this ZIP file, you can relocate, unpack, and rebuild your project in another development environment where MATLAB is not installed.

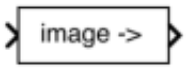
See Also

Blocks

To Multimedia File | Video From Workspace | Video To Workspace | Image From File

From Simulink Image

Unpack numeric matrix from Simulink image



Libraries:
Computer Vision Toolbox / Utilities

Description

The From Simulink Image block converts Simulink image data to matrix data.

Ports

Input

Port_1 — Input signal
`Simulink.ImageType`

Input signal to the From Simulink Image block. The input is a Simulink image data type.

Data Types: `Simulink.ImageType`

Output

Port_1 — Output signal
matrix

Output signal from the From Simulink Image block. The output is a matrix or an array, depending on the format of the input.

- *M*-by-*N* matrix — The input is a `Grayscale` image. *M* and *N* are the height and width of the image, in pixels.
- *M*-by-*N*-by-*C* matrix — The input is an RGB or BGR image. *M* and *N* are the height and width of the image, in pixels. *C* is the number of color channels.

Data Types: `integer` | `single` | `double` | `Boolean`

Parameters

Output dimensions — Dimensions of block output signal
`[480 640 3]` (default)

Specify the dimensions of the output image signal. The expression must be a three-element numeric row vector of form: `[Rows Columns Channels]`.

Programmatic Use

Block Parameter: `ImageSize`
Type: character vector
Value: character vector

Default: `[480 640 3]`

Block Characteristics

Data Types	`double` | `single` | `integer` | `Boolean` | `image`
Direct Feedthrough	`yes`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

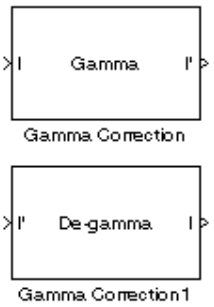
Supports C++ code generation.

See Also

To Simulink Image

Gamma Correction

Apply or remove gamma correction to or from image or video stream



Libraries:
Computer Vision Toolbox / Conversions

Description

The Gamma Correction block applies or removes gamma correction to or from an image or video stream.

Ports

Input

I — Input image
matrix | array

Input image, specified as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the input image.

Dependencies

To enable this port, set the **Operation** parameter to `Gamma`.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

I' — Gamma-corrected input image
matrix | array

Gamma-corrected input image, specified as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the input image.

Dependencies

To enable this port, set the **Operation** parameter to `De-gamma`.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Output

I' — Gamma-corrected output image
matrix | array

Gamma-corrected output image, returned as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels.

Dependencies

To enable this port, set the **Operation** parameter to `Gamma`.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

I — Gamma-removed output image
matrix | array

Gamma-removed output image, returned as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels.

Dependencies

To enable this port, set the **Operation** parameter to `De-gamma`.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Parameters

Operation — Block operation

`Gamma` (default) | `De-gamma`

Specify the gamma operation the block performs.

- `Gamma` — Apply gamma correction to the input image.
- `De-gamma` — Remove gamma correction from the input image.

Gamma — Gamma value

`2.2` (default) | scalar

Specify the gamma as a value greater than or equal to 1. A value less than 1 encodes luminance values in image and a value greater than 1 decodes luminance values in image, represented as gamma compression and gamma expansion respectively.

Linear segment — Linear segment

`on` (default) | `off`

Select this parameter to define the gamma curve as a linear segment near the origin. For more information, see “Gamma Correction Method” on page 1-253.

Break point — Break point value

`0.018` (default) | scalar

Specify a scalar break point value, which indicates the *I*-axis value at the end of the gamma curve linear segment. For more information, see “Gamma Correction Method” on page 1-253.

Dependencies

To enable this parameter, select the **Linear segment** parameter.

Block Characteristics

Data Types	`double` | `fixed point`[a] | `integer`[a] | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

a Supports word lengths less than or equal to 16 bits.

Algorithms

Gamma Correction Method

For input signals normalized between 0 and 1, gamma correction is defined by these equations. For integers and fixed-point data types, these equations are generalized by applying scaling and offset values specific to each data type.

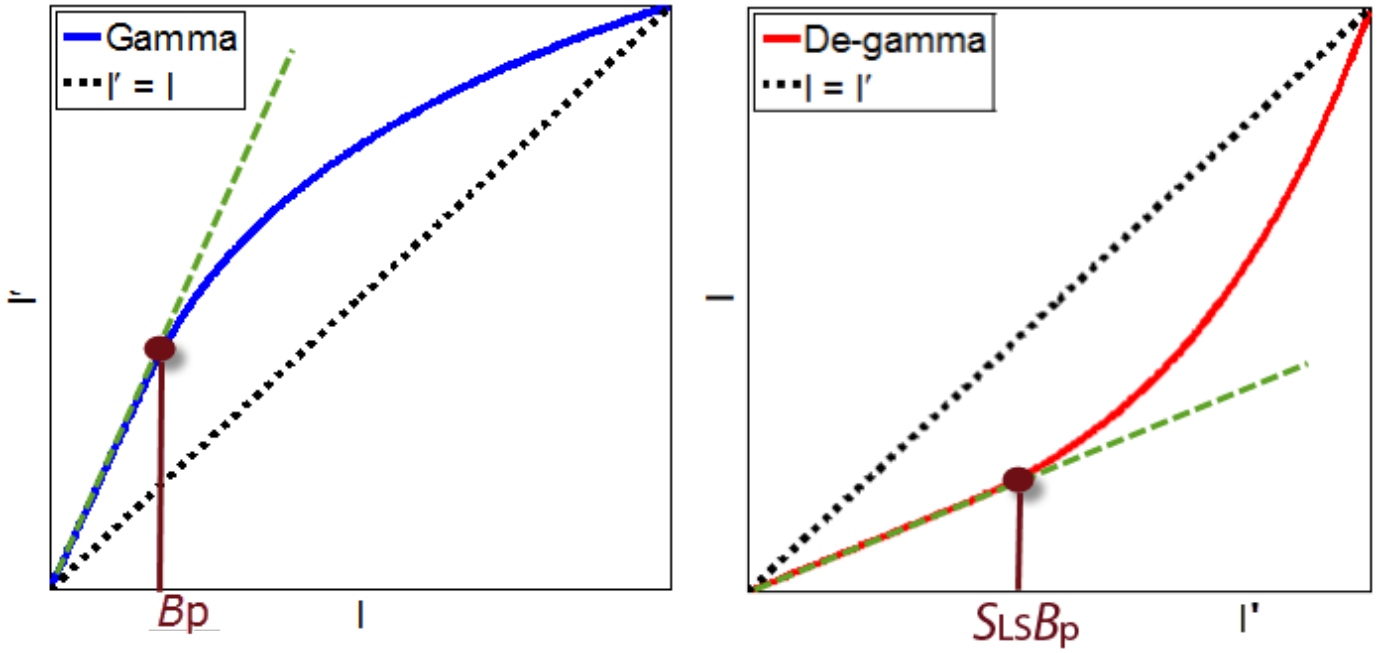
$$S_{LS} = \frac{1}{\frac{\gamma}{B_p^{(1/\gamma - 1)}} - \gamma B_p + B_p}$$

$$F_S = \frac{\gamma S_{LS}}{\left(\frac{1}{\gamma}-1\right) B_p}$$

$$C_O = F_S B_p^{\frac{1}{\gamma}} - S_{LS} B_p$$

$$I' = \begin{cases} S_{LS} I, & I \le B_p \\ F_S I^{\frac{1}{\gamma}} - C_O, & I > B_p \end{cases}$$

S_{LS} is the slope of the straight line segment. B_P is the break point of the straight line segment, which corresponds to the **Break point** parameter. F_S is the slope-matching factor, which matches the slope of the linear segment to the slope of the power function segment. C_O is the segment offset, which ensures that the linear segment and the power function segments connect. This figure shows the slope of the straight line segment S_{LS} and break point of the straight line segment B_P for gamma and de-gamma operations.



By linearizing the input normalized signal, the block removes gamma correction as defined by this equation.

$$I = \begin{cases} \dfrac{I'}{S_{LS}}, & I' \le S_{LS}B_p \\ \left(\dfrac{I' + C_O}{F_S}\right)^{\gamma}, & I' > S_{LS}B_p \end{cases}$$

Typical gamma values range from 1 to 3. Most LCD monitor gamma values range from 1.8 to 2.2. Check with the manufacturer of your hardware to obtain its gamma value. This table shows the gamma function parameters for different standards.

Standard	Slope	Break Point	Gamma
CIE L*	9.033	0.008856	3
Recommendation ITU-R BT.709-3, Parameter Values for the HDTV Standards for Production and International Programme Exchange	4.5	0.018	20/9
sRGB	12.92	0.00304	2.4

Version History

Introduced before R2006a

References

[1] Poynton, Charles A. *Digital Video and HDTV: Algorithms and Interfaces.* Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Amsterdam ; Boston: Morgan Kaufmann Publishers, 2003.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Color Space Conversion

Functions

`imadjust`

Gaussian Pyramid

Perform Gaussian pyramid decomposition



Libraries:
Computer Vision Toolbox / Transforms

Description

The Gaussian Pyramid block computes Gaussian pyramid reduction or expansion to resize an image. The image reduction process involves lowpass filtering and then downsampling the image pixels. The image expansion process involves upsampling and then lowpass filtering the image pixels. You can also use this block to build a Laplacian pyramid.

Ports

Input

I — Input image
scalar | vector | matrix | array

Input image, specified as a scalar, a vector, an *M*-by-*N* matrix, or an *M*-by-*N*-by-*C* array. *C* is the number of color channels in the input image. To specify this input as a scalar or vector, you must set the **Operation** parameter to `Expand`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Output

Port_1 — Output image
scalar | vector | matrix | array

Output image, returned as a scalar, a vector, a matrix, or an array. The dimensions of the output image are determined by the **Operation**and **Pyramid level** parameter values. For this port to return a scalar or vector, you must set the **Operation** parameter to `Reduce`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Operation — Select operation

`Reduce` (default) | `Expand`

Select the Gaussian pyramid operation as one of these methods:

- `Reduce` — The block applies a lowpass filter and then downsamples the input image.
- `Expand` — The block upsamples and then applies a lowpass filter to the input image.

Pyramid level — Image scaling factor

`1` (default) | positive integer

Specify the factor of 2 by which to upsample or downsample each dimension of the image.

For an *M*-by-*N* input image, if you set the **Operation** parameter to `Reduce` and the **Pyramid level** to `1`, the dimensions of the output image are `ceil(M/2)`-by-`ceil(N/2)`.

For an *M*-by-*N* input image, if you set the **Operation** parameter to `Expand`, the dimensions of the output image are $[(M-1)2^l+1]$-by-$[(N-1)2^l+1]$, where the **Pyramid level** parameter l must be a positive integer.

Coefficient source — Lowpass filter coefficient source

`Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]` (default) | `Specify via dialog`

Specify the lowpass filter coefficient source as `Default separable filter [1/4-a/2 1/4 a 1/4 1/4-a/2]` or `Specify via dialog`.

Coefficients for separable filter — Separable filter coefficient

`[0.0625 0.25 0.375 0.25 0.0625]` (default) | vector

Specify the separable filter coefficients as a vector.

Hidden coefficients — Hidden separable filter coefficients

`[1/4-0.375/2 1/4 0.375 1/4 1/4-0.375/2]` (default) | vector

Specify the hidden separable filter coefficients as a vector.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

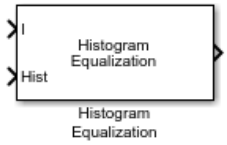
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
Resize

Histogram Equalization

Enhance contrast of images using histogram equalization



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Histogram Equalization block enhances the contrast of images. The block transforms the intensity values in an image so that the histogram of the output image approximately matches a specified histogram. By default, the block transforms the input image so that the histogram of the output image is nearly flat.

Ports

Input

I — Input image
matrix

Input image, specified as a matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Hist — Target histogram
vector of integers

Target histogram for the output image, specified as a vector of integers. Each element of the vector represents the desired count value for the corresponding bin. You must normalize the input vector such that the sum of all the bin counts equals the number of pixels in the input image.

When the data type of the input image is `single` or `double`, the **Hist** port input must be the same data type.

Dependencies

To enable this port, set these block parameters:

- **Target histogram**: `User-defined`
- **Histogram source**: `Input port`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Port_1 — Output image
matrix

Output image, returned as a matrix of intensity values. The data type of the output image is the same as the data type of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Parameters

Target histogram — Type of desired histogram

`Uniform` (default) | `User-defined`

Specify the desired histogram type.

- `Uniform` — The block transforms the input image so that the histogram of the output image is approximately flat.
- `User-defined` — The block transforms the input image so that the histogram of the output image approximately matches the specified histogram.

Number of bins — Number of bins of target histogram

`64` (default) | integer greater than one

Specify the number of bins for the target histogram. The block spaces the bins equally.

Dependencies

To enable this parameter, set the **Target histogram** parameter to `Uniform`.

Histogram source — Source of target histogram

`Specify via dialog` (default) | `Input port`

Specify the source of the target histogram as either `Specify via dialog` or `Input port`.

- `Specify via dialog` — Use the **Histogram** parameter to specify a vector of integer values representing the desired counts of the equally spaced bins of the target histogram.
- `Input port` — Use the **Hist** input port to specify the target histogram.

Dependencies

To enable this parameter, set the **Target histogram** parameter to `User-defined`.

Note Normalize the input vector, specified using the selected **Histogram source**, such that the sum of all the bin counts equals the number of pixels in the input image. The block does not display an error if you do not normalize the vector.

Histogram — Bin counts of target histogram

`ones(1,64)` (default) | vector of integers

Specify the count for each bin of the target histogram. The sum of the bin counts must equal the number of pixels in the input image.

Dependencies

To enable this parameter, set these block parameters:

- **Target histogram**: `User-defined`
- **Histogram source**: `Specify via dialog`

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

References

[1] Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing: International Edition*. 3rd ed. Upper Saddle River, NJ: Pearson, 2008.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

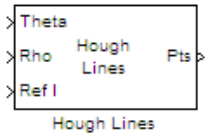
Contrast Adjustment

Functions

`histeq` | `imadjust`

Hough Lines

Find Cartesian coordinates of lines described by rho and theta pairs



Libraries:
Computer Vision Toolbox / Transforms

Description

The Hough Lines block finds the points of intersection between the boundary lines of a reference image and a line specified as a rho-theta pair. The block outputs Cartesian coordinates for the points of intersection. The boundary lines indicate the left and right vertical boundaries and the top and bottom horizontal boundaries of the reference image.

If the line specified by a rho-theta pair does not intersect two border lines in the reference image, the block outputs the values, `[0 0 0 0]` for that line. This output value enables the next block in your model to ignore the outputs for that line, as the Hough Lines block generally precedes a block that draws a point or shape at each output intersection.

Ports

Input

Theta — Theta value
scalar | vector

Theta value, specified as a scalar or *M*-element vector. *M* is the number of specified lines. Each element represents the rotation angle of the corresponding line in radians.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Rho — Rho value
scalar | vector

Rho value, specified as a scalar or *M*-element vector. *M* is the number of specified lines. Each element that represents the distance of the corresponding line from the top-left coordinate origin of the image at the angle specified by the corresponding element of **Theta**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Ref I — Reference image
matrix

Reference image, specified as an *M*-by-*N* matrix.

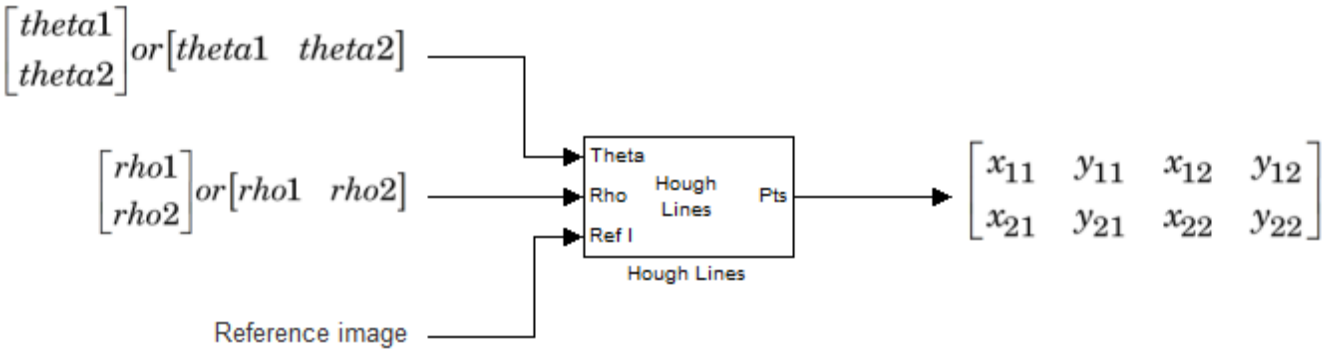
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Output

Pts — Intersection points
matrix

Intersection points, returned as an *M*-by-4 matrix of coordinate values, where *M* is the number of input lines.

For example, consider a Hough Lines block with two-element vectors as the theta and rho inputs, describing two lines.

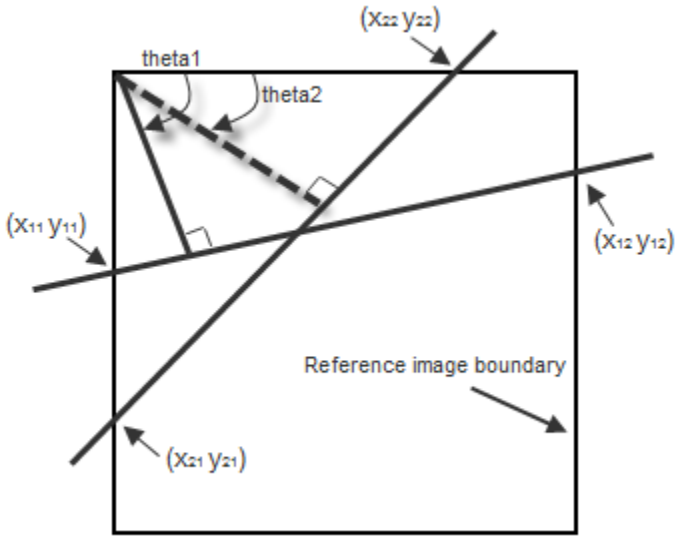


The block outputs a 2-by-4 matrix, in which each row consists of two *xy*-pairs of coordinates that describe the points at which the corresponding line intersects with the boundary lines of the reference image.

$[x_{11}\ y_{11}\ x_{12}\ y_{12};$

$x_{21}\ y_{21}\ x_{22}\ y_{22}]$

This figure shows the line described by the first rho-theta pair intersecting the boundaries of the reference image at $[x_{11}\ y_{11}\ x_{12}\ y_{12}]$ and the line described by the second pair intersecting the boundaries at $[x_{21}\ y_{21}\ x_{22}\ y_{22}]$.



Data Types: `int32`

Parameters

Sine value computation method — Sine value computation method

`Table lookup` (default) | `Trigonometric function`

You can specify one of these options.

- `Table lookup` — Before the simulation starts, the block computes and stores the trigonometric values required to calculate the intersections of the lines. In this case, the block requires additional memory.
- `Trigonometric function` — The block computes the sine and cosine values required to calculate the intersections of the lines during the simulation.

Theta resolution (radians) — Theta resolution

`pi/180` (default) | positive scalar

Specify the spacing of the theta-axis. Value must be greater than 0 and less than `pi/2`.

Dependencies

To enable this parameter, set the **Sine value computation method** parameter to `Table lookup`.

Block Characteristics

Data Types	`Boolean`[a] | `double` | `fixed point`[b, c] | `integer`[d] | `single`
Multidimensional Signals	`no`

Variable-Size Signals	yes

a This data type is only supported at the input port Ref1.
b Generated code will be restricted to MATLAB host computers when you set the FFT implementation parameter to FFTW, or when the transform length is not a power of two.
c Input ports Theta and Rho support signed fixed-point data types with word lengths less than or equal to 32, while Ref1 supports signed and unsigned fixed-point data types.
d The output port supports 32-bit signed integers only. Input ports Theta and Rho support signed integers, while Ref1 supports signed and unsigned integers.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

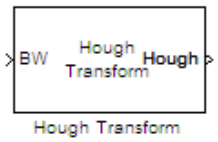
See Also

Blocks

Find Local Maxima | Hough Transform

Hough Transform

Find lines in images



Libraries:
Computer Vision Toolbox / Transforms

Description

Use the Hough Transform block to find straight lines in an image. The block outputs the Hough space matrix and, optionally, the *rho*-axis and *theta*-axis vectors. Peak values in the matrix represent potential straight lines in the input image. You can use the output of the Hough Transform block to find straight lines in the image by using the Hough Lines block. Alternatively, you can use a custom algorithm to identify potential straight lines by locating peaks in the Hough space matrix.

Ports

Input

BW — Binary image
matrix

Binary image, specified as a matrix of `Boolean` values.

Data Types: `Boolean`

Output

Hough — Parameter space matrix
matrix

Parameter space matrix, returned as a numeric matrix. The values in the parameter space matrix are in the (*rho*, *theta*) space.

Data Types: `single` | `double` | `fixed point`

Theta — Theta values
vector

Theta values, returned as a numeric vector. Each element represents the rotation angle of the corresponding line, in radians.

Dependencies

To enable this port, you must select the **Output theta and rho values** parameter.

Data Types: `single` | `double` | `fixed point`

Rho — Rho values
vector

Rho values, returned as a numeric vector. Each element represents the distance of the corresponding line from the top-left point of the image.

Dependencies

To enable this port, you must select the **Output theta and rho values** parameter.

Data Types: `single` | `double` | `fixed point`

Parameters

Main

Theta resolution (radians) — Theta resolution

`pi/180` (default) | positive scalar

Specify the spacing of the Hough transform bins along the *theta*-axis. This value must be less than `pi/2`.

Rho resolution (pixels) — Rho resolution

`1` (default) | positive scalar

Specify the spacing of the Hough transform bins along the *rho*-axis.

Output theta and rho values — Enable Theta and Rho output ports

`off` (default) | `on`

Select this parameter to enable the **Theta** and **Rho** output ports. The block outputs the *theta*-axis and *rho*-axis vectors, respectively, at these ports.

Output data type — Output data type

`double` (default) | `single` | `Specify via Fixed-point tab`

Specify the data type of your output signal. If you select `double` or `single`, all the outputs are of the specified data type. If you select `Specify via Fixed-point tab`, you can specify the word length of each output in the **Data Types** tab.

Data Types

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types
`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

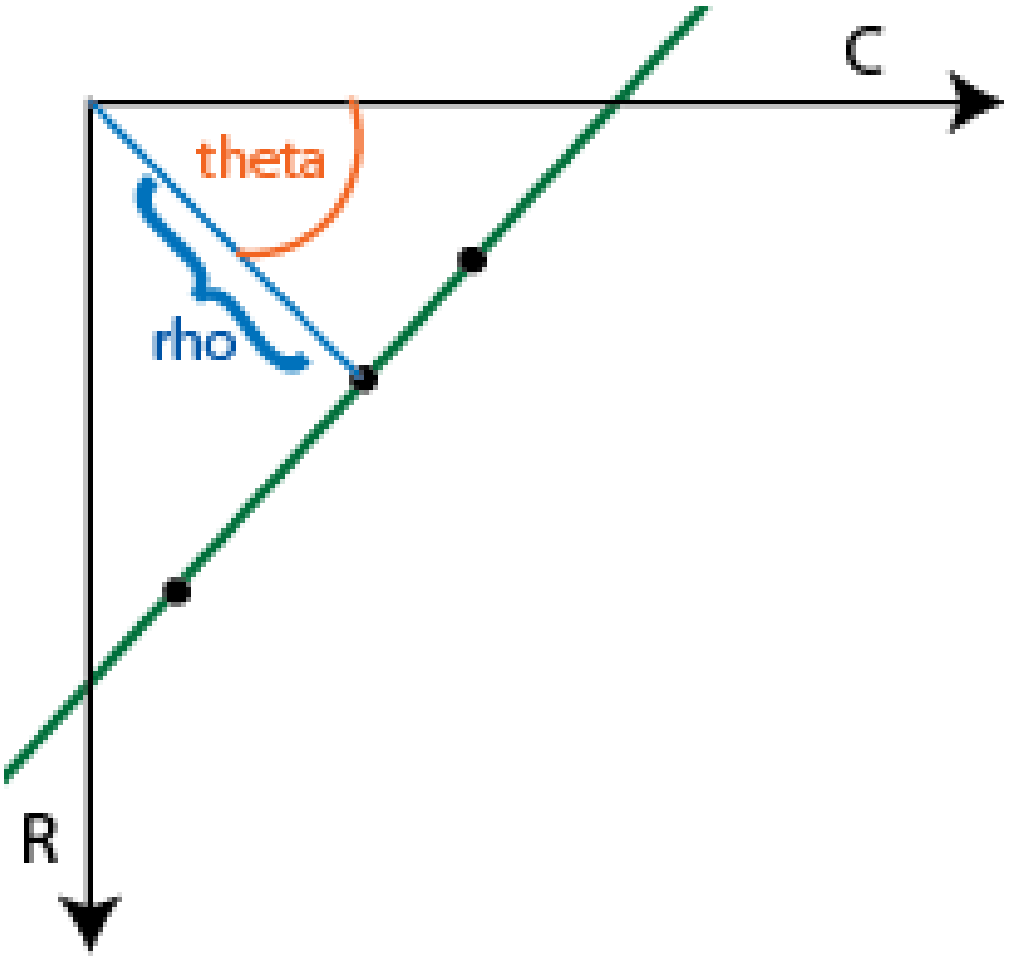
Data Types	`Boolean` | `double`[a] | `fixed point` | `integer` | `single`[a]
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

a This data type is supported at the output port(s).

Algorithms

The Hough Transform block implements the Standard Hough Transform (SHT). The SHT uses the parametric representation of a line:

$$rho = x * \cos(theta) + y * \sin(theta)$$



The block uses the upper-left corner pixel of the image as the origin, assigning it the *xy*-value of (0, 0).

The *rho* value indicates the perpendicular distance from the origin to the Hough line.

The *theta* value indicates the angle of inclination of the normal line from the *x*-axis. The range of *theta* is $-\frac{\pi}{2} \le \theta < +\frac{\pi}{2}$, with a step-size determined by the **Theta resolution (radians)** parameter. The SHT measures the angle of the line clockwise, with respect to the positive *x*-axis.

The Hough Transform block creates an accumulator matrix. The (*rho, theta*) pair represent the location of a cell in the accumulator matrix. Every valid (logical `true`) pixel of the input binary image represented by (*row, col*) produces a rho value for all theta values. The block quantizes the rho values to the nearest number in the rho vector. The rho vector depends on the size of the input image and the user-specified rho resolution. The block increments a counter (initially set to zero) in those accumulator array cells represented by the (*rho, theta*) pairs found for each pixel. This process validates that the point (*row, col*) is on the line defined by (*rho, theta*). The block repeats this process for each logical `true` pixel in the image. The Hough Transform block outputs the resulting accumulator matrix.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

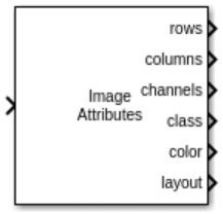
Find Local Maxima | Hough Lines | `hough` | `houghlines` | `houghpeaks`

Topics

“Detect and Track Face”

Image Attributes

Output attributes of Simulink image signal



Libraries:
Computer Vision Toolbox / Utilities

Description

The Image Attributes block outputs attributes of Simulink image signal as numeric data. Use Image Attributes block to:

- Display image attributes for debugging purposes.
- Use image attributes in control or data flow of an image processing algorithm.

Ports

Input

Port_1 — Input signal
`Simulink.ImageType`

Input signal to the Image Attributes block. The input is a Simulink image data type.

Data Types: `Simulink.ImageType`

Output

rows — Number of rows in image data
scalar

Number of rows in the image data, returned as a positive integer.

Data Types: `uint32`

columns — Number of columns in image data
scalar

Number of columns in the image data, returned as a positive integer.

Data Types: `uint32`

channels — Number of color channels or samples for each pixel in array
scalar

Number of color channels or samples for each pixel in the array, returned as 1, 3, or 4. The number of channels correspond to the number of color channels in the color format of the image data. These are the values of `Channels` for the supported color formats:

Color Format	Channels
Grayscale	1
RGB	3
BGR	3
BGRA	4

Data Types: uint32

class — Data type of underlying image data
string

Data type of the underlying image data, returned as one of these values:

- 'uint8'
- 'int8'
- 'uint16'
- 'int16'
- 'uint32'
- 'int32'
- 'single'
- 'double'
- 'boolean'

Data Types: string

color — Color format of underlying image data
string

Color format of the underlying image data, returned as 'RGB', 'Grayscale', 'BGR', or 'BGRA'. The color format determines what each color channel of a pixel in the image represents.

Data Types: string

layout — Array layout of image data
string

Memory arrangement of the matrix data in the image, returned as 'ColumnMajor' or 'RowMajor'.

Data Types: string

Block Characteristics

Data Types	integer | string | image
Direct Feedthrough	yes
Multidimensional Signals	yes
Variable-Size Signals	no

Zero-Crossing Detection	no

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Supports C++ code generation.

See Also

`Simulink.ImageType` | To Simulink Image | To Simulink Image

Image Complement

Compute the complement of pixel values in binary or intensity images



Libraries:
Computer Vision Toolbox / Conversions

Description

The Image Complement block computes the complement of a binary or intensity image.

For binary images, the block replaces pixel values equal to `0` with `1` and pixel values equal to `1` with `0`. In the output image, black and white are reversed. For intensity images, the block subtracts each pixel value from the maximum value that can be represented by the input data type and outputs the difference. In the output image, dark areas become lighter and light areas become darker.

For example, suppose the input pixel values are given by $x(i)$ and the output pixel values are given by $y(i)$. If the data type of the input is boolean, double or single, the block outputs $y(i) = 1.0-x(i)$. If the input is an 8-bit unsigned integer, the block outputs $y(i) = 255-x(i)$.

Ports

Input

Port_1 — Input image
matrix

Specify an input image as a matrix of intensity values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean`

Output

Port_1 — Output image
matrix

The complement of the binary or intensity input image is returned as a matrix of intensity values. The size and data type of the output image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean`

Block Characteristics

Data Types	`Boolean` | `double` | `integer` | `single`

Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Autothreshold | Chroma Resampling | Color Space Conversion | Demosaic | Gamma Correction | Image Data Type Conversion | Image Pad

Image Data Type Conversion

Convert and scale input image to specified output data type



Libraries:
Computer Vision Toolbox / Conversions

Description

The Image Data Type Conversion block converts the data type of the input image to the specified data type and scales the pixel values to the dynamic range of the output data. To convert between data types without scaling, use the Data Type Conversion block.

The behavior of the block changes depending on the input and output data types of the image.

- When converting between floating-point data types, the block casts the input values into the output data type and clips values below or above the range of the output data type to `0` or `1`, respectively.
- When converting to the `Boolean` data type, the block maps `0` values to `0` (`false`) and all other values to `1` (`true`).
- Between all other data types, the block casts and scales the input data type values into the dynamic range of the output data type. For double- and single-precision floating-point data types, the dynamic range is between `0` and `1`. For fixed-point data types, the dynamic range is between the minimum and maximum pixel values of that data type.

Ports

Input

I — Input image
matrix | array

Input image, specified as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the input image.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Output

Port_1 — Output image
matrix | array

Output image, returned as an *M*-by-*N* matrix or an *M*-by-*N*-by-*C* array, where *C* is the number of color channels in the output image.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

Parameters

Output data type — Output data type

`double` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `boolean` | `Fixed-point` | `Inherit via back propagation`

Specifies the data type of your output signal.

Signed — Signed output

`on` (default) | `off`

Specifies the fixed-point data type of your output signal as signed.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Word length — Word length

`16` (default) | scalar

Specifies the word length of your fixed-point output.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Fraction length — Fraction length

`0` (default) | scalar

Specifies the fraction length of your fixed-point output.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point`[a] | `integer`[a] | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

a Supports word lengths less than or equal to 16 bits.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Autothreshold | Color Space Conversion

Image To Matrix

Converts Simulink image data to matrix data



Libraries:
Computer Vision Toolbox OpenCV Interface for Simulink

Description

The Image To Matrix block converts Simulink image data to matrix data.

Ports

Input

Port_1 — Input signal
`Simulink.ImageType`

Input signal to the Image To Matrix block. The input is a Simulink image data type.

Data Types: `Simulink.ImageType`

Output

Port_1 — Output signal
matrix

Output signal from the Image To Matrix block. The output is a matrix or an array, depending on the format of the input.

- *M*-by-*N* matrix — The input is a `Grayscale` image. *M* and *N* are the height and width of the image, in pixels.
- *M*-by-*N*-by-*C* matrix — The input is an RGB or BGR image. *M* and *N* are the height and width of the image, in pixels. *C* is the number of color channels.

Data Types: `uint8` | `uint16` | `int8` | `int16` | `single` | `double`

Parameters

Rows — Number of rows in image data
`'480'` (default) | positive integer

Specify the number of rows in the image data as a positive integer.

Programmatic Use
Block Parameter: Rows
Type: `double`
Value: positive integer

Default: `'480'`

Columns — Number of columns in image data
`'640'` (default) | positive integer

Specify the number of columns in the image data as a positive integer.

Programmatic Use
Block Parameter: `Columns`
Type: `double`
Value: positive integer
Default: `'640'`

Channels — Number of color channels or samples for each pixel
3 (default) | 1

Specify the number of color channels or samples for each pixel in the array as 1 or 3. The number of channels must correspond to the number of color channels in the color format of the image data. These are the values of **Channels** for the supported color formats:

Color Format	Channels
`Grayscale`	1
`RGB`	3
`BGR`	3

Programmatic Use
Block Parameter: `Channels`
Type: `double`
Value: positive integer
Default: `'3'`

Block Characteristics

Data Types	`double` | `single` | `uint8` | `uint16` | `int8` | `int16` | `int32` | `image`
Direct Feedthrough	`Yes`
Multidimensional Signals	`Yes`
Variable-Size Signals	`No`

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

Supports C++ code generation.

See Also

Matrix To Image | From Simulink Image

Topics

“Install and Use Computer Vision Toolbox Interface for OpenCV in Simulink”

Image From File

Read image from file location



Libraries:
Computer Vision Toolbox / Sources

Description

The Image From File block reads an image from a specified file location and imports it to the Simulink workspace. The input file can contain a binary image, grayscale image, or color image.

Ports

Output

Image — Output image
matrix | 3-D array

Output image, returned as a matrix or 3-D array. If the image read is a binary or grayscale image, the output is an *M*-by-*N* matrix. If the image read is a color image, the output is an *M*-by-*N*-by-3 array.

Dependencies

To enable this port, set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color component of image
matrix

Red color component of the image, returned as an *M*-by-*N* matrix. The block returns this output if the input is a color image. If the **Image signal** parameter is set to Separate color signals and the input image is grayscale, the block returns only this port.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color component of image
matrix

Green color component of the image, returned as an *M*-by-*N* matrix. The block returns this output only if the input is a color image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color component of image
matrix

Blue color component of the image, returned as an *M*-by-*N* matrix. The block returns this output only if the input is a color image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

File name — Input image
`peppers.png` (default) | file name

Specify the name of the file that contains the image to read. If the file is not on the MATLAB path, click **Browse** and navigate to the file. For the list of supported file formats, see the `imread` function reference page in the MATLAB documentation.

Note The **File name** parameter supports URL paths.

Sample time — Sample time for block execution
`inf` (default) | `Positive scalar` | `Vector`

Specify the sample time for block execution. Vector length must be 2 which represents [Period, Offset] values where offset should be less than period and non-negative. Given the default value of inf, the block outputs a matrix for grayscale and binary. For an input grayscale or color image, the output is a 3-D or 4-D array, respectively.

The value of the added dimension is equal to floor((T_{sim}/T_s)+1).

T_{sim} is the simulation time and T_s is the sample time, for block execution. For example, if the input image is a 256-by-256 matrix, the simulation time is 10 sec, and the sample time is set to 0.5 sec, then the output is a 256-by-256-by-21 array.

Image signal — Signal returned by block
`One multidimensional signal` (default) | `Separate color signals`

Specify how the block returns the image read from the file.

- `One multidimensional signal`— The block has one output port, which returns the entire image.
- `Separate color signals` — The block has three output ports, one for each color channel, which returns the individual color channels of the image.

Output port labels — Output port labels
`R|G|B` (default) | port names

Specify the labels for the output ports. Each label can be any value. Use vertical bar character, |, as a delimiter between the names of all the output ports.

Dependencies

To enable this parameter, set the **Image signal** parameter to `Separate color signals`.

Data type

Output data type — Data type of output image
inherit from input image (default) | double | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean | fixed-point | user-defined | inherit via back propagation

Specify the data type of the output image. If the input image data type,differs from the selected value of the **Output data type** parameter, the block scales the pixel values, adds an offset to the pixel values so that they are in the dynamic range of the new data type, or both. Double- and single-precision floating-point pixel values must be in the range [0 1].

Signed — Signed fixed-point data type
`on` (default) | `off`

Select this parameter if the input image data type is signed fixed-point.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Word length — Bits in fixed-point data type
16 (default) | positive scalar

Specify the word length, in bits, of the fixed-point output data type. The value should be less than 32.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Set fraction length in output to — Scaling factor of fixed-point data type
`User-defined` (default) | `Best precision`

Specify the scaling factor of the fixed-point output as one of these options.

- `Best precision`— The output scales automatically such that the output signal has the best possible precision.
- Choose `User-defined`— Specify the output scaling in the **Fraction length** parameter.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Fractional length — Fractional bits
15 (default) | scalar

Specify the number of bits to the right of the decimal point.

Dependencies

To enable this port ,set the **Output data type** parameter to `Fixed-point`, and set the **Set fraction length in output** parameter to `User-defined`.

User-defined data type — User-defined fixed-point data type
`uint(8)` (default) | `User-defined fixed-point data type`

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the `fixdt` function.

Dependencies

To enable this parameter, set the **Output data type** parameter to `User-defined`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

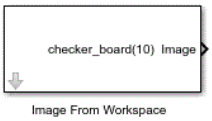
From Multimedia File | Image From Workspace | To Video Display | Video From Workspace | Video Viewer

Functions

`im2double` | `im2uint8` | `imread`

Image From Workspace

Import image from MATLAB workspace



Libraries:
Computer Vision Toolbox / Sources

Description

The Image From Workspace block imports an image from a matrix, 3-D array, MATLAB function, or variable stored in the MATLAB workspace, to the Simulink workspace.

Ports

Output

Image — Output image
matrix | 3-D array

Output image, returned as a matrix or 3-D array. If the imported image is binary or grayscale image, the output is a matrix of size *M*-by-*N*. If the imported image is a color image, the output is a 3-D array of size *M*-by-*N*-by-3.

Dependencies

To enable this port ,set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color component of image
matrix

Red color component of the image, returned as an *M*-by-*N* matrix. The block returns this output if the input is a color image. If the **Image signal** parameter is set to `Separate color signals` and the input image is binary or grayscale, the block returns only this port

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color component of image
matrix

Green color component of the image, returned as an *M*-by-*N* matrix. The block returns this output only if the input is a color image.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color component of image
matrix

Blue color component of the image, returned as an *M*-by-*N* matrix. The block returns this output only if the input is a color image.

Dependencies

To enable this port ,set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Main

Value — Image to be imported
`rand(100,100,3)` (default) | vector | matrix | 3-D array | `MATLAB workspace variable` | `MATLAB function`

Specify the image as a vector, matrix, 3-D array, function, or MATLAB workspace variable. The value parameter can be:

- Vector — The value given to the **Value** parameter can be a row vector or column vector
- 3-D numeric array — The value given to the **Value** parameter can be 3-D array, which is generated using MATLAB functions
- MATLAB workspace variable — The value given to the **Value** parameter can be a variable name, which is MATLAB workspace variable
- MATLAB function — The value given to the **Value** parameter can be a MATLAB function with specified input arguments to generate a numeric array

Sample time — Sample time for block execution
`inf` (default) | `positive scalar` | `Vector`

This parameter specifies the sample time for block execution. Vector length must be 2 which represents [Period, Offset] values where offset should be less than period and non-negative. Given the default value of inf, the block outputs a matrix for grayscale and binary. The output is 3-D or 4-D, if the input image is a gray scale or color image respectively.

The value of the added dimension is equal to $floor((T_{sim}/T_s)+1)$

T_{sim} is the simulation time and T_s is the sample time, for block execution. For example, if the input image is a 256-by-256 matrix, the simulation time is 10 sec and the sample time is set to 0.5 sec, then the output is a 256-by-256-by-21 array.

Image signal — Signal returned by block
`One multidimensional signal` (default) | `Separate color signals`

Specify how the block returns the image read from the MATLAB workspace.

- `One multidimensional signal`— The block has one output port, **Image**, that returns the entire image.
- `Separate color signals` — The block has an output port for each color channel. Each port returns a single color channel of the image.

Output port labels — Output port labels
`R|G|B` (default) | port names

Specify labels for the output ports. Each label can be any value. Use the vertical bar character, |, as a delimiter to name all the output ports.

Dependencies

To enable this parameter, set the **Image signal** parameter to `Separate color signals`.

Data type

Output data type — Data type of the output image
`Inherit from Value` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean` | `Fixed-point` | `User-defined` | `Inherit via back propagation` | ...

Specify the data type of the output image. If the data type of the imported image, specified in the **Value** parameter, differs from the value of the **Output data type** parameter, the block scales the pixel values, adds an offset to the pixel values so that they are in the dynamic range of the new data type, or does both. Double- and single-precision floating-point pixel values must be in the range of [0, 1].

Signed — Signed fixed-point data type
`on` (default) | `off`

Select this parameter if the input image data type is signed fixed-point.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Word length — Bits in fixed-point data type
16 (default) | positive scalar

Specify the word length, in bits, of the fixed-point output data type. The value should be less than 32.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Set fraction length in output to — Scaling factor of the fixed-point
`User-defined` (default) | `Best precision`

Specify the scaling factor of the fixed-point output by either of the following two methods:

- `Best precision`— The output scaling automatically such that the output signal has the best possible precision.
- `User-defined` — The output scaling in the **Fraction length** parameter.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`.

Fractional length — Fractional bits
15 (default) | `scalar`

Specify the number of bits to the right of the decimal point.

Dependencies

To enable this parameter, set the **Output data type** parameter to `Fixed-point`, and set the **Set fraction length in output** parameter to `User-defined`.

User-defined data type — User-defined fixed-point data type
`sfix(16)` (default) | `user-defined fixed-point data type`

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the `fixdt` function.

Dependencies

To enable this parameter, set the **Output data type** parameter to `User-defined`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

R2022b: `Image From Workspace` block default value changes
Behavior changed in R2022b

This release changes the default value for the Image From Workspace block to `rand(100,100,3)`. Previously, the default value was `checker_board`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

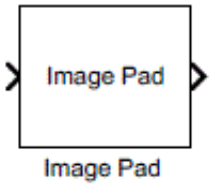
See Also

Blocks
From Multimedia File | To Video Display | Video From Workspace | Video Viewer

Functions
`im2double` | `im2uint8`

Image Pad

Pad image by adding rows, columns, or both



Libraries:
Computer Vision Toolbox / Utilities

Description

The Image Pad block expands the dimensions of an image by padding its rows, columns, or both.

Ports

Input

I — Input image
matrix | array

Input image, specified as an *M*-by-*N* matrix or an *M*-by-*N*-by-*P* array, where *P* is the number of color planes in the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

PVal — Pad value
scalar

Pad value, specified as a scalar. The block uses this value for each element of the rows or columns added to the input image.

Dependencies

To enable this port, set the **Method** parameter to `Constant` and the **Pad value source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

Port_1 — Output image
matrix | array

Output image, returned as a padded matrix or an array. The dimensions of the output image are specified by the **Specify** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Method — Padding method

`Constant` (default) | `Replicate` | `Symmetric` | `Circular`

Pad the image using one of these methods:

- `Constant` — Pad the image with a constant value. See “Pad with Constant Value” on page 1-295.
- `Replicate` — Pad the image by repeating the border values of the input image. See “Pad by Repeating Border Values” on page 1-296.
- `Symmetric` — Pad the image using a mirror image of the input data. See “Pad with Mirror Image” on page 1-297.
- `Circular` — Pad the image using a circular repetition of the elements in the input image. See “Pad Using Circular Repetition of Elements” on page 1-298.

Pad value source — Source of constant pad value

`Specify via dialog` (default) | `Input port`

Specify the padding source as `Specify via dialog` or `Input port`.

- `Specify via dialog` — Specify a pad value using the **Pad Value** parameter.
- `Input port` — Specify a pad value using the **PVal** input port.

Dependencies

To enable this parameter, set the **Method** parameter to `Constant`.

Pad value — Pad value

`0` (default) | scalar

Specify a scalar value with which to pad the image.

Dependencies

To enable this parameter, set the **Method** parameter to `Constant` and the **Pad value source** parameter to `Specify via dialog`.

Specify — Padding size specification format

`Pad size` (default) | `Output size`

Select how to specify the size of the image padding from these two options:

- `Pad size` — Specify the number of columns and rows with which to pad the image, using the **Number of added columns** and **Number of added rows** parameters, respectively.
- `Output size` — Specify the size of the padded output image using a combination of the **Output row mode**, **Number of output rows**, **Output column mode**, and **Number of output columns** parameters.

Add columns to — Column padding location

`Left` (default) | `Right` | `Both left and right` | `No padding`

Select which sides of the input image to pad with additional columns:

- `Left` — Add columns to the left side of the input image.
- `Right` — Add columns to the right side of the input image.
- `Both left and right` — Add columns to both the left and right sides of the input image.
- `No padding` — Do not add columns to the input image.

Output column mode — Output column sizing method

`User-specified` (default) | `Next power of two`

Specify the number of columns in the output image using one of these options:

- `User-specified` — Specify the number of columns in the output image using the **Number of output columns** parameter.
- `Next power of two` — Add columns to the input image until the total number of columns is a power of two.

Dependencies

To enable this parameter, set the **Specify** parameter to `Output size`.

Number of output columns — Number of columns in output image

`10` (default) | positive integer

Specify the number of columns in the output image as a positive integer M. If the value of this parameter is less than the number of columns in the input image, then the output image contains only the last M columns of the input image.

Dependencies

To enable this parameter, set the **Specify** parameter to `Output size` and the **Output column mode** parameter to `User-specified`.

Number of added columns — Number of columns added to input image

2 (default) | positive integer | two-element vector of positive integers

Specify how many columns to add to the input signal. The behavior of this parameter changes depending on whether you are padding only one side or both sides of the image.

Add columns to	Number of added columns	Description
`Left`	P	Add P columns to the left side of the input image.
`Right`	Q	Add Q columns to the right side of the input image.
`Both left and right`	$[P\ Q]$	Add P columns to the left side and Q columns to the right side of the input image.

Dependencies

To enable this parameter, set the **Specify** parameter to `Pad size` and the **Add columns** to parameter to `Left`, `Right`, or `Both left and right`.

Add rows to — Row padding location

`Top` (default) | `Bottom` | `Both top and bottom` | `No padding`

Select which sides of the input image to pad with additional rows:

- `Top` — Add rows to the top of the input image.
- `Bottom` — Add rows to the bottom of the input image.
- `Both top and bottom` — Add rows to both the top and bottom of the input image.
- `No padding` — Do not add rows to the input image.

Output row mode — Output row sizing method

`User-specified` (default) | `Next power of two`

Specify the number of rows in the output image using one of these options:

- `User-specified` — Specify the number of rows in the output image using the **Number of output rows** parameter.
- `Next power of two` — Add rows to the input image until the total number of rows is a power of two.

Dependencies

To enable this parameter, set the **Specify** parameter to `Output size`.

Number of output rows — Number of rows in output image

`12` (default) | positive integer

Specify the number of rows in the output image as a positive integer *N*. If the value of this parameter is less than the number of rows in the input image, then the output image contains only the last *N* rows of the input image.

Dependencies

To enable this parameter, set the **Specify** parameter to `Output size` and the **Output row mode** parameter to `User-specified`.

Number of added rows — Number of rows added to input image

`[2 3]` (default) | positive integer | two-element vector of positive integers

Specify how many rows to add to the input signal. The behavior of this parameter changes depending on whether you are padding only one side or both sides of the image.

Add rows to	Number of added rows	Description
`Top`	*R*	Add *R* rows to the top of the input image.
`Bottom`	*S*	Add *S* rows to the bottom of the input image.
`Both top and bottom`	[*R S*]	Add *R* rows to the top and *S* rows to the bottom of the input image.

Dependencies

To enable this parameter, set the **Specify** parameter to `Pad size` and the **Add rows** to parameter to `Top`, `Bottom`, or `Both top and bottom`.

Action when truncation occurs — Action when input signal is truncated

`None` (default) | `Warning` | `Error`

You can specify how the block behaves when the input signal is truncated as one of these options:

- `None` — Does not notify you when the signal is truncated.
- `Warning` — Displays a warning message in the MATLAB command window when the input signal is truncated.
- `Error` — Displays an error dialog box and terminates the simulation when the input signal is truncated.

Dependencies

To enable this parameter, set the **Specify** parameter to `Output size`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

More About

Pad with Constant Value

When padding an image with a constant value, every element of the rows and columns the Image Pad block adds to the input image is a specified scalar value.

If you want to pad each of your input signal with three initial values equal to `0`, and this is how your input signal is defined:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

Specify these parameters for the Image Pad block:

- **Method** — `Constant`
- **Pad value source** — `Specify via dialog`
- **Pad value** — `0`
- **Specify** — `Output size`
- **Add columns to** — `Left`
- **Output column mode** — `User-specified`
- **Number of output columns** — `6`
- **Add rows to** — `No padding`

The Image Pad block outputs this signal:

$$\begin{bmatrix} 0 & 0 & 0 & a_{00} & a_{01} & a_{02} \\ 0 & 0 & 0 & a_{10} & a_{11} & a_{12} \\ 0 & 0 & 0 & a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

Pad by Repeating Border Values

When padding an image with the **Method** parameter set to `Replicate`, the Image Pad block replicates border values of the input signal across all directions.

If you want to pad your input signal with its border values and this is how your input signal is defined:

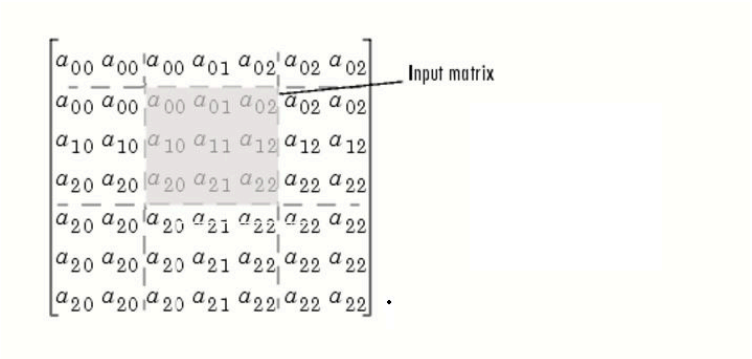
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

Specify these parameters for the Image Pad block:

- **Method** — `Replicate`
- **Specify** — `Pad size`
- **Add columns to** — `Both left and right`
- **Number of added columns** — `2`

- **Add rows to** — `Both top and bottom`
- **Number of added rows** = `[1 3]`

The Image Pad block outputs this signal:



The border values of the input signal are replicated on the top, bottom, left, and right of the input signal, so that the output is a 7-by-7 matrix. The values in the corners of this output matrix are determined by replicating the border values of the matrices on the top, bottom, left, and right sides of the original input signal.

Pad with Mirror Image

When padding an image with the **Method** parameter set to `Symmetric`, the Image Pad block flips the original input matrix and each matrix it creates, to populate across all directions of the input signal.

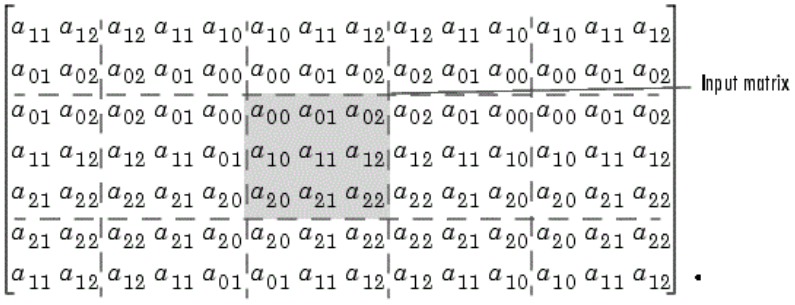
If you want to pad your input signal using its mirror image and this is how your input signal is defined:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

Specify these parameters for the Image Pad block:

- **Method** — `Symmetric`
- **Specify** — `Pad size`
- **Add columns to** — `Both left and right`
- **Number of added columns** — `[5 6]`
- **Add rows to** — `Both top and bottom`
- **Number of added rows** — `2`

The Image Pad block outputs this signal:



The block flips the original input matrix and each matrix it creates about their top, bottom, left, and right sides to populate the 7-by-13 output signal. For example, the block flips the input matrix about its right side to create the matrix directly to its right, but flips that second matrix about its bottom to create the matrix below it.

Pad Using Circular Repetition of Elements

When padding an image with the **Method** parameter set to `Circular`, the Image Pad block repeats the values of the input signal in a circular pattern across all directions.

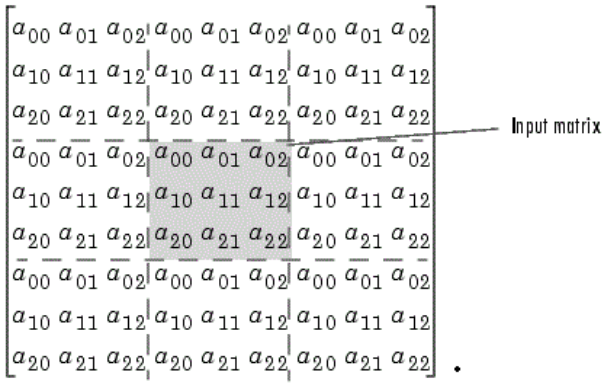
If you want to pad your input signal using a circular repetition of its values and this is how your input signal is defined:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}.$$

Specify these parameters for the Image Pad block:

- **Method** — `Circular`
- **Specify** — `Output size`
- **Add columns to** — `Both left and right`
- **Number of output columns** — `9`
- **Add rows to** — `Both top and bottom`
- **Number of output rows** — `9`

The Image Pad block outputs this signal:



The block repeats the values of the input signal in a circular pattern to populate the 9-by-9 output matrix.

Version History

Introduced in R2007a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Selector | Submatrix

Functions

`imcrop`

Insert Text

Draw text on images or video frames



Libraries:
Computer Vision Toolbox / Text & Graphics

Description

The Insert Text block draws formatted text on an input image or video stream. The block produces stylized text bitmaps using the open-source font engine Free Type 2.3.5 library. For more information about the library, see The FreeType Project website.

Note This block supports only ASCII characters.

This block enables you to insert:

- A single instance of text
- Multiple instances of the same text
- Multiple instances of text, with different text at each location

Ports

Input

Image — Input image
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, *M*-by-*N*-by-*C* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Dependencies

To enable this port, set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

R — Red color component of input image
matrix | array

Red color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array, where *T* is the number of frames in the input video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

G — Green color component of input image
matrix | array

Green color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array, where *T* is the number of frames in the input video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

B — Blue color component of input image
matrix | array

Blue color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array, where *T* is the number of frames in the input video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Select — Select the display text
positive integer

Select the display text, specified as a positive integer.

Select a display text from the cell array using this port, when the input to the **Text** parameter is a cell array of character vectors. For example, the value to the **Select** port must be in the range `[1, S]`, when the **Text** parameter is a cell array of *S* character vectors.

Dependencies

To enable this port, specify the **Text** parameter as a cell array of character vectors.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Color — Text color value
scalar | vector | matrix

Text color value, specified as a scalar, a vector or a matrix. Use this table to determine the type of input value to this port.

Input Image	Same Color for All Text	Different Color for Each Character Vector
Intensity image	scalar value	*S*-element vector. *S* is the number of character vectors in the cell array specified to **Text** parameter.
Color image	three-element vector	*S*-by-3 matrix. *S* is the number of character vectors in the cell array specified to **Text** parameter.

The data type of the input image determines the range of color values. For example, a color value for a floating-point input image data type is in the range `[0, 1]`. For an input image with a `uint8` data type, the value is in the range `[0, 255]`.

Note The input to this port must be of the same data type as the input image.

Dependencies

To enable this port, set the **Color value source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Location — Location index
vector | matrix

Location index, specified as a two-element vector of the form `[x y]` or an *S*-by-2 matrix. `[x y]` specify the coordinates of the input image to display the text. Specify it as an *S*-by-2 matrix, to display *S* character vectors at different locations in the image.

Dependencies

To enable this port, set the **Location source** parameter to `Input port`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Opacity — Opacity factor
scalar | vector

Opacity factor, specified as a scalar or an *S*-element vector.

- Scalar — Specify same opacity value to all the display text.
- *S*-element vector — Specify different opacity values to *S* character vectors specified in the **Text** parameter.

Specify opacity values in the range `[0, 1]`.

Note The input to the **Image**, **R**, **G**, **B**, and **Color** ports must be of the same data type as the input to this port.

Dependencies

To enable this port, set the **Opacity source** parameter to `Input port`.

Data Types: `single` | `double`

Variables — Text variable values

scalar | vector

Text variable values, specified as a scalar or a vector. The block replaces the format specifiers such as `%d`, `%f`, or `%s` in the **Text** parameter, with the elements specified by the **Variables** port.

Use the **Variables** port to display S character vectors or numerical values simultaneously, at S locations of the input image or video frame. The input to this port depends on the type of format specifier specified to the **Text** parameter.

Display Text	Data Type	Format Specifier	Input to the Variables Port
Character vectors	`uint8`	`'%s', '%c'`	`['Text1' 0 'Text2' 0 .... 0 'Text`S`']`
Integers	`uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`	`'%d', '%i', '%o', '%x', '%X'`	$[x_1\ x_2\ x_3 \ldots\ x_S]$
Floating point numbers	`single`, `double`	`'%f', '%e', '%E', '%g', '%G'`	$[x_1\ x_2\ x_3 \ldots\ x_S]$

For more information on format specifiers, see `sprintf`.

Dependencies

To enable this port, enter ANSI C `printf`-style format specifiers, such as `'%d'`, `'%f'`, or `'%s'` in the **Text** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Port_1 — Output image

matrix | array

Output image or video, returned as an M-by-N matrix, M-by-N-by-T array, M-by-N-by-C array, or M-by-N-by-C-by-T array. T is the number of frames in a video or image sequence, and C is the number of color channels.

Dependencies

To enable this port, set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

R — Red color component of output image

matrix | array

Red color component of the output image or video, returned as an M-by-N matrix or M-by-N-by-T array, where T is the number of frames in the output video.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

G — Green color component of output image
matrix | array

Green color component of the output image or video, returned as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array, where *T* is the number of frames in the output video.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

B — Blue color component of output image
matrix | array

Blue color component of the output image or video, returned as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array, where *T* is the number of frames in the output video.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Main

Text — Text to insert

`'Text'` (default) | character vector | cell array of character vectors

Specify the text to insert into the image, as a character vector or a cell array of character vectors. You can display either a single text or multiple texts simultaneously, using the **Text** parameter.

- Single text — You can specify **Text** parameter as either a character vector or a cell array of *S* character vectors. If you specify a cell array, select one-out-of-*S* character vectors from the cell array, by using the **Select** input port.
- Multiple texts simultaneously — You can specify ANSI C `printf`-style format specifiers in the **Text** parameter. You can then specify the texts by using the **Variable** input port. Supported format specifiers include `%d`, `%i`, `%u`, `%o`, `%x`, `%X`, `%c`, `%s`, `%f`, `%e`, `%E`, `%g`, and `%G`. For more information on format specifiers, see `sprintf`.

Use this table to determine the type of input value to the **Text** parameter.

Display	Input toText parameter	Status of Select port	Status of Variables port
A character vector	Single character vector such as `'Text'`	`Disabled`	`Disabled`

Display	Input toText parameter	Status of Select port	Status of Variables port
One-out-of-*S* character vectors	Cell array of *S* character vectors such as `{'Text1', 'Text2',...., 'TextS'}`	`Enabled`	`Disabled`
S character vectors	Format specifier such as `'%s'`	`Disabled`	`Enabled`

Color value source — Source of text color

`Specify via dialog` (default) | `Input port`

Select the text color source as either `Specify via dialog` or `Input port`. You can specify this parameter as `Specify via dialog` to specify insert text colors using the **Color value** parameter. You can specify this parameter as `Input port` to specify text colors by using the **Color** input port.

Color value — Text color value

`[0 0 0]` (default) | scalar | vector | matrix

Text color value specified as a scalar, a vector, or a matrix. Use this table to determine the type of input value to this port.

Input Image	Same Color for All Text	Different Color for Each Character Vector
Intensity image	scalar value	*S*-element vector. *S* is the number of character vectors in the cell array specified to the **Text** parameter.
Color image	three-element vector	*S*-by-*3* matrix. *S* is the number of character vectors in the cell array specified to the **Text** parameter.

The data type of the input image determines the range of color values. For example, a color value for a floating-point input image data type is in the range [`0`, `1`]. For an input image with a `uint8` data type, the value is in the range [`0`, `255`].

Dependencies

To enable this parameter, set the **Color value source** parameter to `Specify via dialog`.

Location source — Source of location indices

`Specify via dialog` (default) | `Input port`

Specify the source of location coordinates of the display text as one these options:

- `Specify via dialog` — Use the **Location [x y]** parameter to specify location coordinates of the display text.

- `Input port` — Use the **Location** input port to specify location coordinates of the display text.

Location [x y] — Text location index

`[1,1]` (default) | vector | matrix

Text location index, specified as a two-element vector `[x y]` or an *S*-by-2 matrix. `[x y]` specify the coordinates of the input image to display the text. Specify the value as an *S*-by-2 matrix, to display *S* character vectors at different locations in the image.

Dependencies

To enable this parameter, set the **Location source** parameter to `Specify via dialog`.

Opacity source — Source of opacity factor

`Specify via dialog` (default) | `Input port`

Specify opacity factor source as one of these options:

- `Specify via dialog` — Use the **Opacity** parameter to specify opacity factor.
- `Input port` — Use the **Opacity** input port to specify opacity factor.

Opacity — Opacity factor

`1` (default) | scalar | vector

Opacity factor, specified as a scalar or an *S*-element vector.

- Scalar — Specify same opacity value to the display text.
- *S*-element vector — Specify different opacity values to *S* character vectors specified to the **Text** parameter

Specify opacity value in the range `[0, 1]`.

Dependencies

To enable this parameter, set the **Opacity source** parameter to `Specify via dialog`.

Image signal — Input and output signal format

`One multidimensional signal` (default) | `Separate color signals`

Specify the format of the input and output images of the block using these options:

- `One multidimensional signal` — The block enables a single input port and single output port.
- `Separate color signals` — The block enables R, G, and B input and output signal ports for the respective color channels of the image.

Input image is transposed (data order is row major) — Mirror text and text location

`off` (default) | `on`

Enable this parameter to mirror the text string and reverse its location in the image from `[x y]` to `[y x]`.

Font

Font face — Font of text

`Roboto-Regular` (default) | `Agency FB` | `Agency FB Bold` | `Algerian` | `Ariel` | `Arial Black` | `Arial Bold` | `Arial Bold Italic` | ...

Specify the font of the inserted text. The block populates this list with the fonts installed on your system. On Windows, the block searches the system registry for font files. On UNIX, the block searches the font path of the X Server for font files.

Font size (points) — Font size

12 (default) | positive scalar

Specify the font size of the inserted text.

Anti-aliased — Smooth text edges

`on` (default) | `off`

Enable this parameter to smooth the edges of the inserted text. The smoothing process is computationally expensive.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point`[a] | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

a Supports word lengths less than or equal to 32 bits.

Version History

Introduced in R2013a

R2026a: Default value of Font face changed to `Roboto-Regular`
Behavior changed in R2026a

Starting in R2026a, the default value of the **Font face** parameter is `Roboto-Regular`. In previous releases, the default value was `LucidaSansRegular`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Draw Shapes | Draw Markers

Label

Label connected components in binary image



Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Label block finds and labels the connected components or objects in a 2-D binary image. The block ignores the background pixels in the input image and searches for connected components within remaining pixels. The block then labels all the pixels of each connected component with a unique number and computes the number of labeled objects.

Ports

Input

BW — Input binary image
vector | matrix

The input image must be a vector or matrix of binary values, each element representing a pixel of the image. The block categorizes pixels with a value of `0` as background pixels, searching for connected components within pixels with a value of `1`.

Data Types: `Boolean`

Output

Label — Labeled objects
vector | matrix

Labeled objects in the input image, returned as a vector or matrix of nonnegative integers. The elements of the output vector or matrix labeled `0` represent the background, the elements labeled `1` represent the first object, the elements labeled `2` represent the second object, and so on. The size of this output matches the size of the **BW** input.

Dependencies

To enable this port, set the **Output** parameter to either `Label matrix` or `Label matrix and number of labels`.

Data Types: `uint8` | `uint16` | `uint32`

Count — Number of labeled objects
scalar

Number of labeled objects, returned as a scalar.

Dependencies

To enable this port, set the **Output** parameter to either `Number of labels` or `Label matrix and number of labels`.

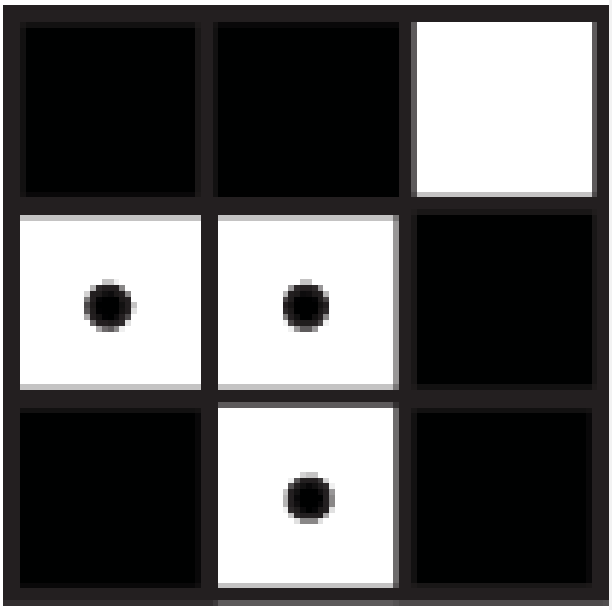
Data Types: `uint8` | `uint16` | `uint32`

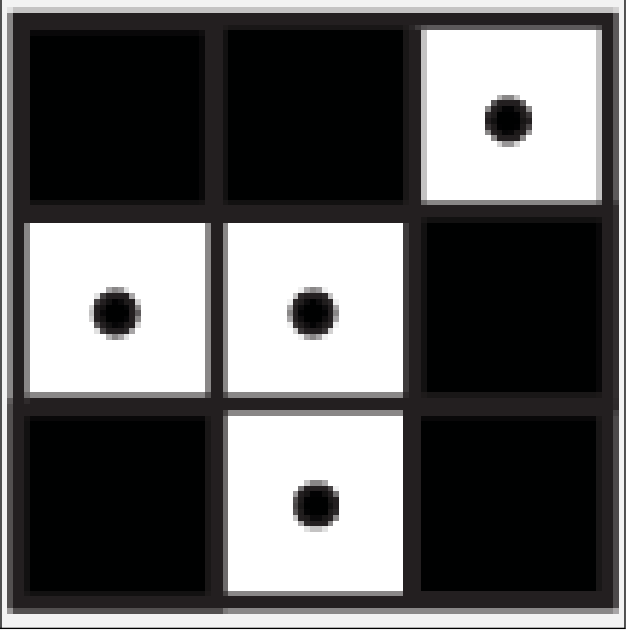
Parameters

Connectivity — Type of pixel connectivity

8 (default) | 4

Specify how the block defines which pixels are connected to each other. If you want to connect adjacent pixels in only the horizontal and vertical directions, select `4`. If you want to connect adjacent pixels in the horizontal, vertical, and diagonal directions, select `8`. These figures show how the block defines a connected component for 4-connected and 8-connected pixels. The block considers the white pixels marked by black circles to be connected to each other.

Type of Connectivity	Connected Pixels
4-connected pixels	

Type of Connectivity	Connected Pixels
8-connected pixels	

Output — Block output

`Label matrix and number of labels` (default) | `Label matrix` | `Number of labels`

Specify the block output as one of these options:

- `Label matrix` — The block outputs the labeled objects at the **Label** port.
- `Number of labels` — The block outputs the number of labeled objects at the **Count** port.
- `Label matrix and number of labels` — The block outputs both the labeled objects and number of labeled objects to the **Label** and **Count** ports, respectively.

Output data type — Data type of output

`Automatic` (default) | `uint8` | `uint16` | `uint32`

Specify the data type of the outputs at the **Label** and **Count** ports. When you specify this parameter as `Automatic`, the block calculates the maximum number of objects that can fit inside the image based on the image size and the specified connectivity. Using this calculation, it chooses the output data type with the minimum value that guarantees unique region labels.

If label exceeds data type size, mark remaining regions using — Overflow mode of output labels

`maximum value of the output data type` (default) | `zero`

Use this parameter to specify the behavior of the block when the block finds more objects than the maximum possible value the output data type can represent. The block labels the remaining objects with the value determined by one of these options:

- `maximum value of the output data type` — The block labels the remaining objects with the maximum value of the output data type.

- `zero` — The block labels the remaining objects with `0`.

Dependencies

To enable this parameter, set the **Output data type** parameter to either `uint8` or `uint16`.

Block Characteristics

Data Types	`Boolean` | `integer`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

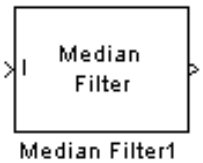
Dilation | Erosion | Opening | Closing | Blob Analysis

Functions

`bwconncomp` | `bwselect` | `regionprops` | `bwlabel` | `bwlabeln`

Median Filter

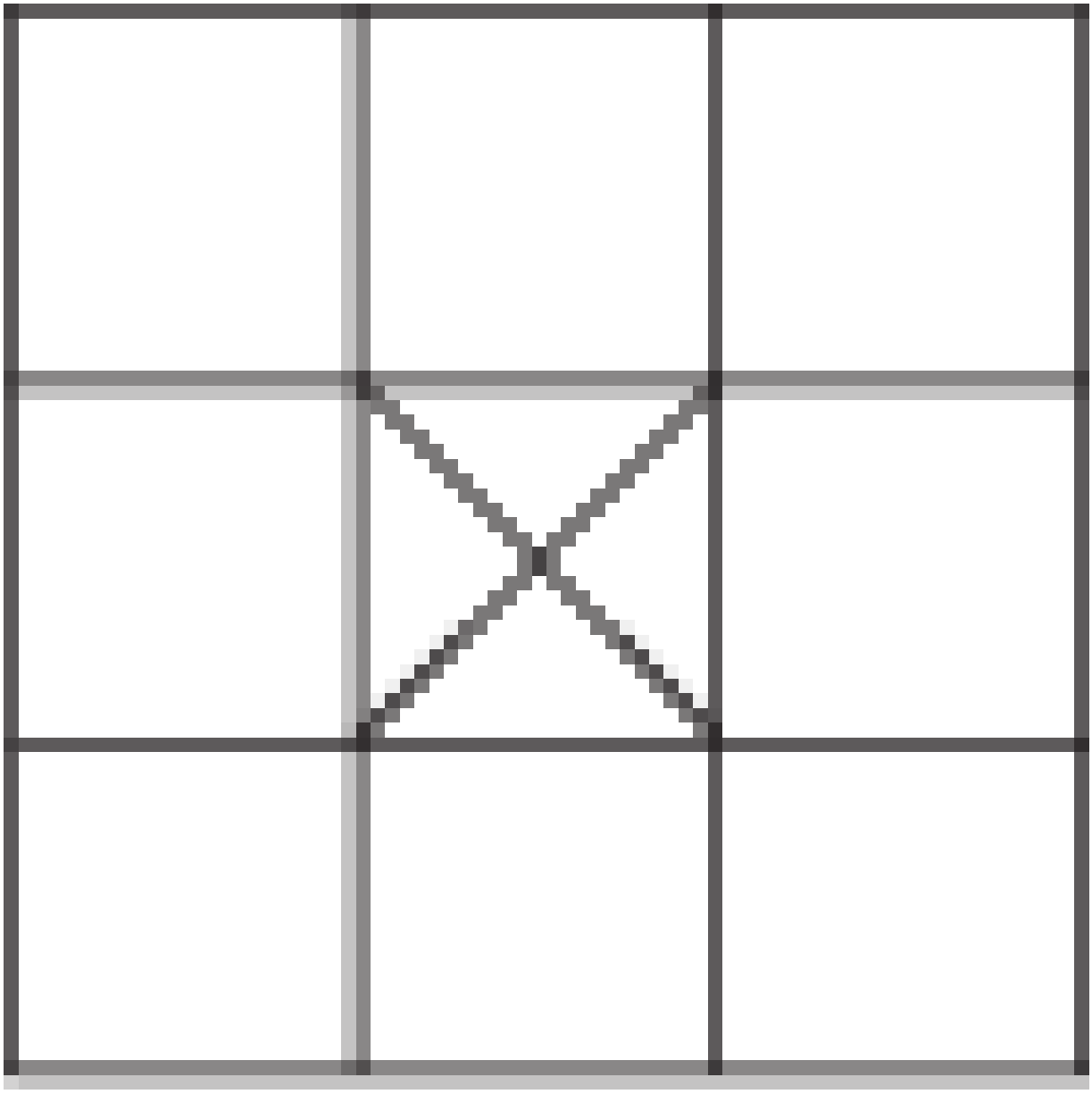
Perform 2-D median filtering



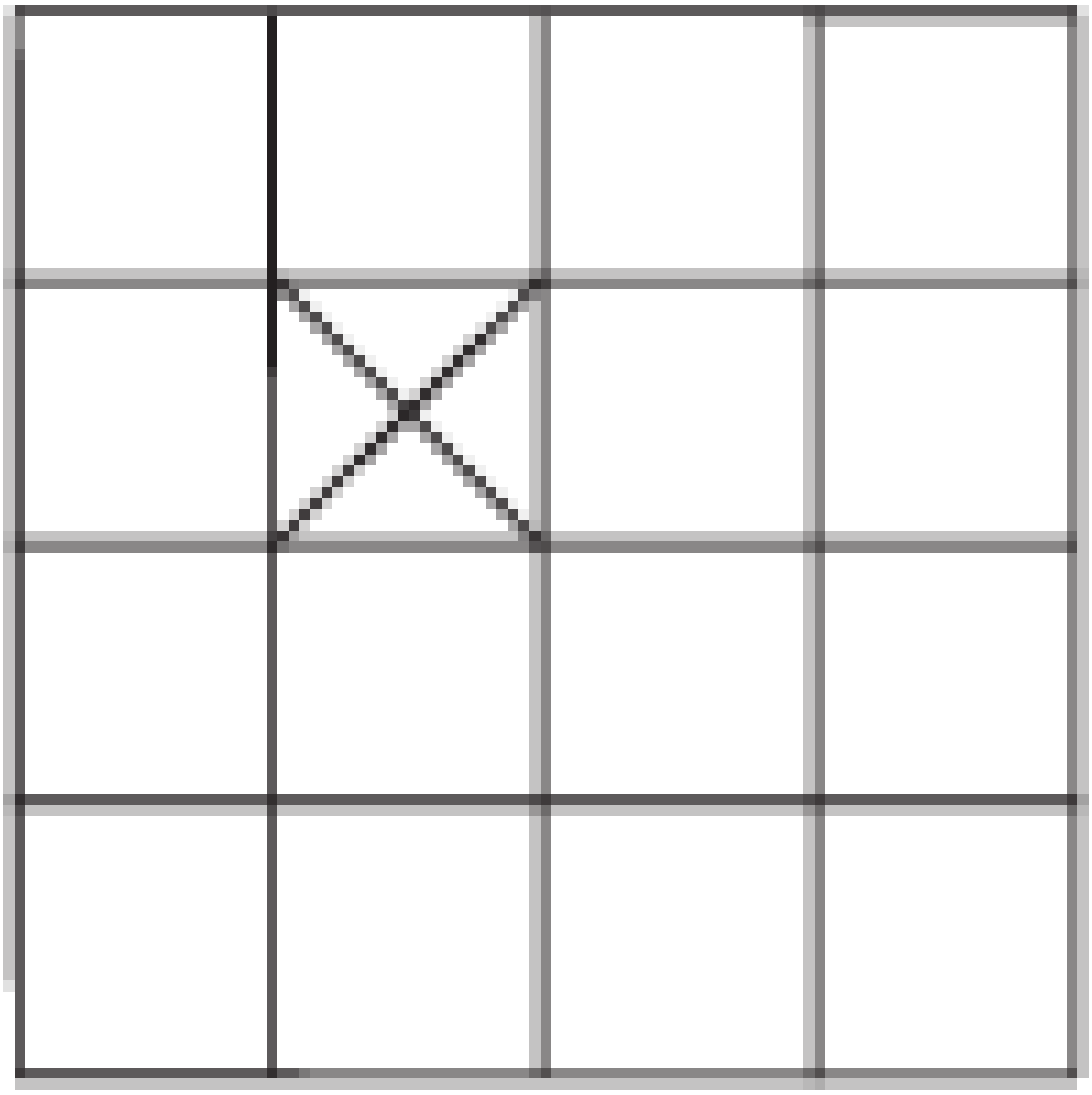
Libraries:
Computer Vision Toolbox / Analysis & Enhancement
Computer Vision Toolbox / Filtering

Description

The Median Filter block applies a median filter to each pixel of an input image. For each *M*-by-*N* neighborhood, the filter replaces the central value with the median value of the neighborhood. If the neighborhood has a center element, the block replaces the center pixel with the median value, as shown in this figure.



If the neighborhood does not have an exact center, then the block has a bias toward the upper-left corner of the neighborhood. In this figure, the median value has been placed in the top-left pixel of the four pixels that meet at the center of the neighborhood.



The block pads the edges of the input image for filtering. This causes the pixels within *M/2* pixels of the sides and *N/2* pixels of the top or bottom of the image to appear distorted.

The median value is less sensitive than the mean to extreme values. Thus, Median Filter is effective for removing salt-and-pepper noise from an image without significantly reducing the sharpness of the image.

Ports

Input

I — Input image
matrix

Input image, specified as an *M*-by-*N* matrix.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `Boolean` | `fixed point`

PVal — Pad value
scalar

Pad value, specified as a scalar. The block uses this value for each element of the rows or columns added to the input image.

Dependencies

To enable this port:

- Set the **Output size** parameter to `Same as input port I`.
- Set the **Padding options** parameter to `Constant`.
- Set the **Pad value source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

Port_1 — Output image
matrix

Output image, returned as a matrix. The dimensions of the median-filtered output image are specified by the **Output size** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Neighborhood size — Neighborhood window size

`[3 3]` (default) | positive integer | two-element vector of positive integers

Specify the window size of the neighborhood over which the block computes the median. If specified as a scalar, this value creates a square neighborhood with the specified number of rows and columns. If specified as a vector, the first element is the number of rows in the neighborhood, and the second element is the number of columns in the neighborhood.

Output size — Output sizing method

`Same as input port I` (default) | `Valid`

Specify the output sizing method as one of these options.

- `Same as input port I` — The block computes the neighborhood median for the entire image, using padding to compute the boundary pixels. The output image has the same dimensions as the input image. Selecting this option enables the **Padding options** parameter.
- `Valid` — The block computes the median where the neighborhood fits within the input image without padding. For an input image of size *M*-by-*N* and neighborhood of size *X*-by-*Y*, the output image has the dimensions (`M`–`X`+1)-by-(`N`–`Y`+1).

Padding options — Padding method

`Symmetric` (default) | `Constant` | `Replicate` | `Circular`

Pad the image using one of these methods:

- `Constant` — Pad image with a constant value. Selecting this option enables the **Pad value source** parameter. For more information, see “Pad with Constant Value” on page 1-295.
- `Replicate` — Pad the image by repeating the border values of the input image. For more information, see “Pad by Repeating Border Values” on page 1-296.
- `Symmetric` — Pad the image using a mirror image of the input data. For more information, see “Pad with Mirror Image” on page 1-297
- `Circular` — Pad the image using a circular repetition of the elements in the input image. For more information, see “Pad Using Circular Repetition of Elements” on page 1-298.

Dependencies

To enable this parameter, set the **Output size** parameter to `Same as input port I`.

Pad value source — Source of constant padding value

`Specify via dialog` (default) | `Input port`

Specify the padding source as `Specify via dialog` or `Input port`.

- `Specify via dialog` — Specify a padding value using the **Pad value** parameter.
- `Input port` — Specify a padding value using the **PVal** input port.

Dependencies

To enable this parameter, set the **Output size** parameter to `Same as input port I`, and set the **Padding options** parameter to `Constant`.

Pad value — Pad value

`0` (default) | scalar

Specify a scalar value with which to pad the image.

Dependencies

To enable this port:

- Set the **Output size** parameter to `Same as input port I`.
- Set the **Padding options** parameter to `Constant`.
- Set the **Pad value source** parameter to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

References

[1] Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing. 2nd ed.* Upper Saddle River, N.J: Prentice Hall, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

2-D Convolution | 2-D FIR Filter

Functions

`medfilt2`

Matrix To Image

Converts matrix data to Simulink image data



Libraries:
Computer Vision Toolbox OpenCV Interface for Simulink

Description

The Matrix To Image block converts matrix data to Simulink image data.

Ports

Input

Port_1 — Input signal
matrix

Input signal to the Matrix To Image block. The input is a numerical matrix.

Data Types: `uint8` | `uint16` | `int8` | `int16` | `single` | `double`

Output

Port_1 — Output signal
`Simulink.ImageType`

Output signal from the Matrix To Image block. The output is an image in `Simulink.ImageType` data type.

Data Types: `Simulink.ImageType`

Parameters

Rows — Number of rows in image data
`'480'` (default) | positive integer

Specify the number of rows in the image data as a positive integer.

Programmatic Use

Block Parameter: Rows
Type: `double`
Value: positive integer
Default: `'480'`

Columns — Number of columns in image data
`'640'` (default) | positive integer

Specify the number of columns in the image data as a positive integer.

Programmatic Use

Block Parameter: `Columns`
Type: `double`
Value: positive integer
Default: `'640'`

Channels — Number of color channels or samples for each pixel in the array
3 (default) | 1

Specify the number of color channels or samples for each pixel in the array as 1 or 3. The number of channels must correspond to the number of color channels in the color format of the image data. These are the values of **Channels** for the supported color formats:

Channels	Color Format
1	Grayscale
3	RGB
3	BGR

Programmatic Use

Block Parameter: `Columns`
Type: character vector
Value: positive integer
Default: `'3'`

Underlying Datatype — Data type of underlying image data
`'uint8'` (default) | `'int8'` | `'uint16'` | `'int16'` | `'uint32'` | `'int32'` | `'single'` | `'double'` | `'logical'`

Specify the data type of the underlying image data as one of these options:

- `'uint8'`
- `'int8'`
- `'uint16'`
- `'int16'`
- `'uint32'`
- `'int32'`
- `'single'`
- `'double'`
- `'logical'`

Programmatic Use

Block Parameter: `ClassUnderlying`
Type: character vector
Value: `'uint8'` | `'int8'` | `'uint16'` | `'int16'` | `'uint32'` | `'int32'` | `'single'` | `'double'` | `'logical'`
Default: `'uint8'`

Array Layout — Memory arrangement of matrix data in image
`'Column-major'` (default) | `'Row-major'`

Specify the memory arrangement of the matrix data in the image as `Column-major` or `Row-major`.

Programmatic Use
Block Parameter: `Layout`
Type: character vector
Value: `'Column-major'` | `'Row-major'`
Default: `'Column-major'`

Color Format — Color format of pixels in underlying image data
`'RGB'` (default) | `'Grayscale'` | `'BGR'`

Specify what each color channel of a pixel represents in the underlying image data as `Grayscale`, RGB, or BGR.

Programmatic Use
Block Parameter: `ColorFormat`
Type: character vector
Value: `'Grayscale'` | `'RGB'` | `'BGR'`
Default: `'RGB'`

Block Characteristics

Data Types	`double` | `single` | `uint8` | `uint16` | `int8` | `int16` | `int32` | `image`
Direct Feedthrough	`Yes`
Multidimensional Signals	`Yes`
Variable-Size Signals	`No`

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Supports C++ code generation.

See Also

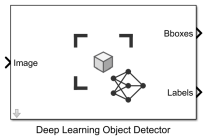
Image To Matrix | To Simulink Image

Topics

“Install and Use Computer Vision Toolbox Interface for OpenCV in Simulink”

Deep Learning Object Detector

Detect objects using trained deep learning object detector



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Deep Learning Object Detector block predicts bounding boxes, class labels, and scores for the input image by using the trained object detector specified through the block parameter. This block allows loading of a pretrained object detector into the Simulink model from a MAT file or from a MATLAB function. This block provides a graphical interface for using the detector objects in Simulink. To enable some parameters of the Deep Learning Object Detector block you must choose an object detector that supports those parameters. For example, using a `yolov2ObjectDetector` object with this block enables you to select parameters similar to the name-value arguments of the associated `detect` object function.

Ports

Input

Image — Image data
array

An *H*-by-*W*-by-*C* numeric array, where *H*, *W*, and *C* are the height, width, and number of channels of the image, respectively. Only one image per time step is allowed as input.

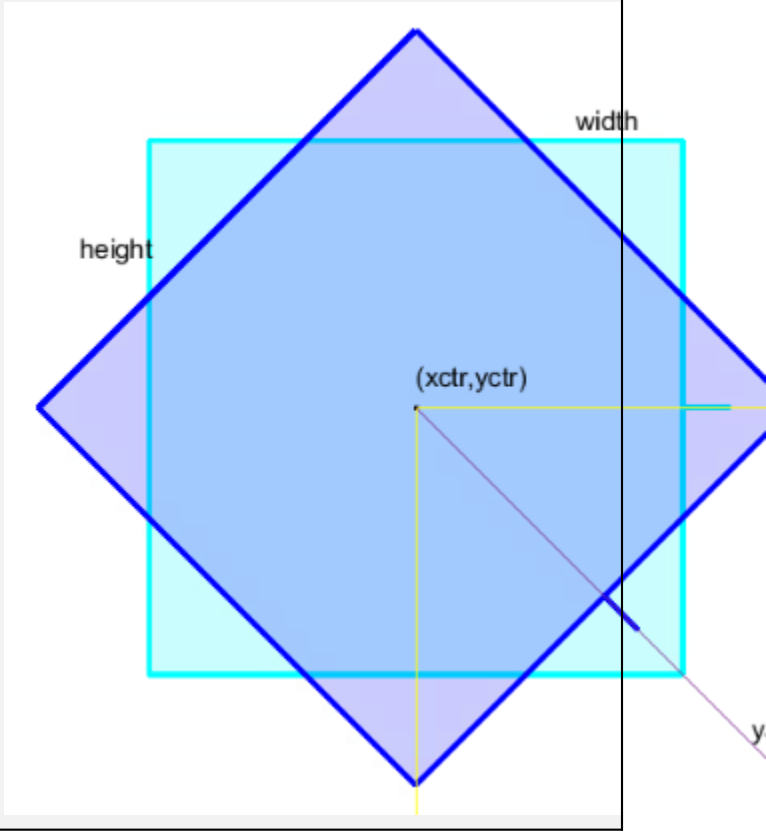
Output

Bboxes — Locations of objects detected
matrix

Locations of the objects detected within the input image, returned as an *M*-by-4 or an *M*-by-5 matrix. *M* is the number of bounding boxes detected in the image. You can put an upper bound to the size *M* by specifying the **Maximum Number of Detections** parameter.

The table describes the format of bounding boxes.

Bounding Box	*Description*
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	*Description*
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

Labels — Labels for bounding boxes
vector

Labels for the bounding boxes, returned as an *M*-by-1 enumerated vector. *M* is the number of bounding boxes detected in the image.

Scores — Detection scores
vector

Detection confidence scores for each label, returned as an *M*-by-1 vector. *M* is the number of bounding boxes detected in the image. A higher score indicates higher confidence in the detection.

Parameters

Detector — Source for trained detector object
`Detector from MAT file` (default) | `Detector from MATLAB function`

Select the source for the detector object from these options:

- `Detector from MAT file` — Import a detector object from a MAT file. For example, select a MAT file containing a `yolov2ObjectDetector` object.
- `Detector from MATLAB function` — Import a detector object from a MATLAB function. For example, specify the function `vehicleDetectorYOLOv2`, which returns a trained `yolov2ObjectDetector` object.

The imported detector must be one of these supported objects:

- `rcnnObjectDetector`
- `fastRCNNObjectDetector`
- `fasterRCNNObjectDetector`
- `rtmdetObjectDetector`
- `ssdObjectDetector`
- `yolov2ObjectDetector`
- `yolov3ObjectDetector`
- `yolov4ObjectDetector`
- `yoloxObjectDetector`

Programmatic Use
Block Parameter: `Detector`
Type: character vector, string
Values: `Detector from MAT file' | 'Detector from MATLAB function'`
Default: `Detector from MAT file'`

File path — MAT file containing detector object
`untitled.mat` (default) | MAT file name

This parameter specifies the name of the MAT file that contains the detector object to load. If the file is not on the MATLAB path, use the **Browse** button to locate the file.

Dependencies

To enable this parameter, set the **Detector** parameter to `Detector from MAT file`.

Programmatic Use
Block Parameter: `DetectorFilePath`
Type: character vector, string
Values: MAT file path or name

Default: `'untitled.mat'`

Simulate using — Specify type of simulation to run
`Auto mode` (default) | `Interpreted execution` | `Code generation`

- `Auto mode` –– This option automatically selects the simulation mode based on the model configuration and code generation compatibility. It uses `Code generation` whenever code generation is supported and switches to `Interpreted execution` when the model is incompatible for code generation.
- `Interpreted execution` –– Simulate model using the MATLAB interpreter. This option shortens startup time but has a slower simulation speed than `Code generation`. In this mode, you can debug the source code of the block.
- `Code generation` –– Simulate model using generated code.

  The first time you simulate a model that contains a Deep Learning Object Detector block, the software generates binary code or C/C++ MATLAB executable (MEX) code from the block and integrates this code with the model. The Deep Learning Object Detector block uses the same infrastructure as MATLAB Coder, which you use to generate C/C++ code from MATLAB code outside of Simulink. The generated code is reused for subsequent simulations, as long as the model does not change. This option requires additional startup time, but the speed of the subsequent simulations is comparable to `Interpreted execution`.

If your Simulink model is set up to simulate in Normal mode and if you use Deep Learning Object Detector block via code generation, individual Deep Learning Object Detector blocks can take advantage of acceleration via code generation. If the Simulink model is set up to simulate in accelerator mode or any of the other target modes, Deep Learning Object Detector block's **Simulate using** parameter does not have any effect. However, there is one exception, if the model is set up to simulate in accelerator mode and you use Deep Learning Object Detector block with interpreted execution mode, the block will run in interpreted execution mode.

Programmatic Use
Block Parameter: `SimulateUsing`
Type: character vector, string
Values: `"Auto mode"` | `"Interpreted execution"` | `"Code generation"`
Default: `"Auto mode"`

MATLAB function — MATLAB function name
`untitled` (default) | MATLAB function name

This parameter specifies the name of the MATLAB function that returns a trained object detector. For example, specify the function `vehicleDetectorYOLOv2`, which returns a trained `yolov2ObjectDetector` object, or specify a custom function.

Dependencies

To enable this parameter, set the **Detector** parameter to `Detector from MATLAB function`.

Programmatic Use
Block Parameter: `DetectorFunction`
Type: character vector, string
Values: MATLAB function name
Default: `'untitled'`

Region of interest — Search region of interest
vector of the form [*x y width height*]

Specify the search region of interest as vector of the form [*x y width height*]. The vector specifies the upper-left corner and size of a region in pixels.

Dependencies

To enable this parameter, select the **Specify region of interest** parameter.

Programmatic Use
Block Parameter: `ROI`
Type: character vector, string
Values: character vector specified as `'[x y width height]'`
Default: `'[1 1 100 100]'`

Detection threshold — Detection threshold
scalar in the range [0, 1]

Specify the detection threshold as scalar in the range [0, 1]. Detections that have scores lower than this threshold value are removed. To reduce false positives, increase this value.

Dependencies

To enable this parameter, you must use a detector that supports the **Detection threshold** parameter. For example, use a `yolov2ObjectDetector` object.

Programmatic Use
Block Parameter: `Threshold`
Type: character vector, string
Values: scalar
Default: `'0.5'`

Number of Strongest Regions — Maximum number of strongest region proposals
`2000` (default) | positive integer

Specify the maximum number of strongest region proposals as an integer. Reduce this value to speed up processing at the cost of detection accuracy. To use all region proposals, specify this parameter as `Inf`.

Dependencies

To enable this parameter, use a detector that supports the **Number of Strongest Regions** parameter. For example, use an `rcnnObjectDetector` object.

Programmatic Use
Block Parameter: `NumStrongestRegions`
Type: character vector, string
Values: integer
Default: `'2000'`

Maximum Region Size — Maximum region size
vector of the form [*height width*]

Specify the maximum region size as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest region containing the object. For example, [`50 50`] sets the size of the largest region containing the object to `50`-by-`50` pixels. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test images.

Dependencies

To enable this parameter

- Select the **Specify maximum region size** parameter.
- Use a detector that supports the **Maximum Region Size** parameter. For example, use a `yolov2ObjectDetector` object.

Programmatic Use

Block Parameter: `MaxSize`
Type: character vector, string
Values: character vector specified as `'[height width]'`
Default: `'[50 50]'`

Minimum Region Size — Minimum region size
vector of the form [*height width*]

Specify the minimum region size as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest region containing the object. For example, `[1 1]` sets the size of the smallest region containing the object to 1-by-1 pixels.

Dependencies

To enable this parameter

- Select the **Specify minimum region size** parameter.
- Use a detector that supports the **Minimum Region Size** parameter. For example, use a `yolov2ObjectDetector` object.

Programmatic Use

Block Parameter: `MinSize`
Type: character vector, string
Values: character vector specified as `'[height width]'`
Default: `'[1 1]'`

Maximum Number of Detections — Maximum number of detections
`500` (default) | positive integer

Specify the maximum number of detections as a positive integer. This value is the upper bound for the number of detections and must be greater than or equal to 2.

Programmatic Use

Block Parameter: `MaxDetections`
Type: character vector, string
Values: integer
Default: `'500'`

Version History

Introduced in R2021b

R2024b: Support to RTMDet object detector

Starting in R2024b, `Deep Learning Object Detector` block supports Real-Time Model for object Detection (RTMDet). You can import a pretrained RTMDet object detector object, `rtmdetObjectDetector` to perform object detection.

R2024b: Support for YOLOX object detector

Starting in R2024b, `Deep Learning Object Detector` block supports You Only Look Once X (YOLOX) object detection. You can import a pretrained YOLOX object detector object, `yoloxObjectDetector` to perform object detection.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:

- The **Language** parameter in the **Configuration Parameters > Code Generation** general category must be set to `C++`.
- For a list of networks and layers supported for code generation, see "Networks and Layers Supported for Code Generation" (MATLAB Coder).
- To generate code for the Math Kernel Library for Deep Neural Networks (MKL-DNN) target library with YOLOv3, YOLOv4, or RTMDet as the object detectors, set the `MaxStackSize` and `TLCOptions` model parameters to a higher value. For example, you can set the `MaxStackSize` and `TLCOptions` parameter values to '`inf`' using the following syntax.

```
>> set_param(modelName,'MaxStackSize','inf')
>> set_param(modelName,'TLCOptions','-aMaxStackVariableSize=inf')
```

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The **Language** parameter in the **Configuration Parameters > Code Generation** general category must be set to `C++`.
- For a list of networks and layers supported for CUDA® code generation, see "Supported Networks, Layers, and Classes" (GPU Coder).

See Also

Objects

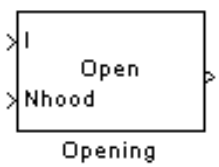
`rcnnObjectDetector` | `fastRCNNObjectDetector` | `fasterRCNNObjectDetector` | `ssdObjectDetector` | `yolov2ObjectDetector` | `yolov3ObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector`

Blocks

Classify (Deep Learning Toolbox) | Predict (Deep Learning Toolbox)

Opening

Perform morphological opening on binary or intensity images



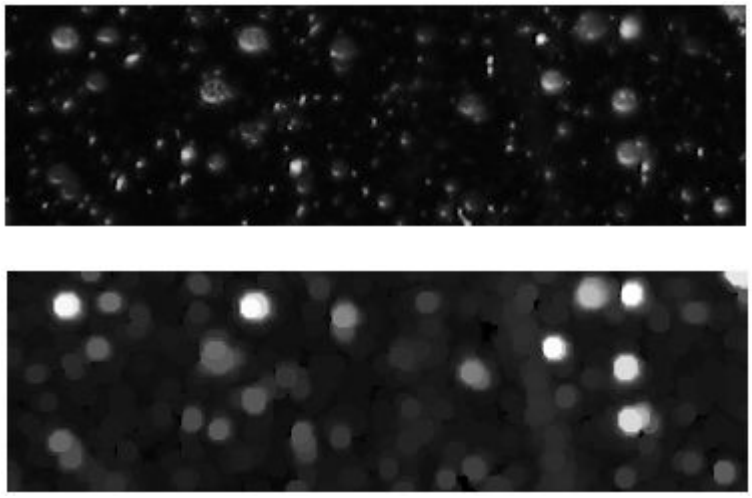
Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Opening block performs an erosion operation followed by a dilation operation on an intensity or binary image using a predefined neighborhood or structuring element.

To define the structuring element that the block applies to the image, use the **Neighborhood or structuring element** parameter. Specify this element by entering a vector or matrix of `1`s and `0`s or by using the `strel` function.

Below are an input image and the output image after the opening operation, respectively.



This block uses two-dimensional flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of intensity values. This port is unnamed unless you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values representing a structuring element as a vector or matrix of `1`s and `0`s or by using the `strel` function.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Opened image
matrix

The opened image is returned as a matrix of intensity values. The size and data type of the opened image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`strel('disk',5)` (default) | vector | matrix

Specify neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object or an array of `strel` objects. When you specify this value as an array of `strel` objects, the block applies each object to the entire image in turn.

If the structuring element is decomposable into smaller elements, the block executes at higher speed due to the use of a more efficient algorithm.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter, to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Opening Operation

Opening smoothes the contour of an object in an image by

- Breaking narrow isthmuses and
- Eliminating thin protrusions

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

[2] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Third edition. Knoxville: Gatesmark Publishing, 2020.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Erosion | Dilation | Closing | Label | Bottom-hat | Top-hat

Functions

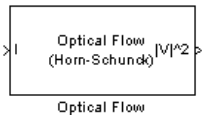
`imerode` | `imdilate` | `imopen` | `imclose`

Objects

`strel`

Optical Flow

Estimate object velocities



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Optical Flow block estimates the direction and speed of object motion between two images or between one video frame to another frame using either the Horn-Schunck or the Lucas-Kanade method.

Ports

Input

I/I1 — Image or video frame
scalar | vector | matrix

Image or video frame, specified as a scalar, vector, or matrix. If the **Compute optical flow between** parameter is set to `Two images`, the name of this port changes to **I1**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

I2 — Image
scalar | vector | matrix

Image or video frame, specified as a scalar, vector, or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Output

|V|^2 — Velocity magnitudes
scalar | vector | matrix

Velocity magnitudes, returned as a scalar, vector, or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

V — Velocity components in complex form
scalar | vector | matrix

Velocity components in complex form, specified as a scalar, vector, or matrix.

Dependencies

To enable this port, set the **Velocity output** parameter to `Horizontal and vertical components in complex form`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

Parameters

Main Tab

Method — Optical flow calculation method
`Horn-Schunck` (default) | `Lucas-Kanade`

Select the method to use to calculate the optical flow. Options include `Horn-Schunck` or `Lucas-Kanade`.

Compute optical flow between — Compute optical flow
`Current frame and N-th frame back` (default) | `Two images`

Select how to compute the optical flow. Select `Two images` to compute the optical flow between two images. Select `Current frame and N-th frame back` to compute the optical flow between two video frames that are N frames apart.

Dependencies

To enable this parameter, set the:

- **Method** parameter to `Horn-Schunck`
- **Method** parameter to `Lucas-Kanade` and the **Temporal gradient filter** to `Difference filter [-1 1]`

N — Number of frames
1 (default) | scalar

Enter a scalar value that represents the number of frames between the reference frame and the current frame.

Dependencies

To enable this parameter, set the **Compute optical flow between** parameter to `Current frame and N-th frame back`.

Smoothness factor — Smoothness factor
1 (default) | positive scalar

Specify the smoothness factor. Enter a large positive scalar value for high relative motion between the two images or video frames. Enter a small positive scalar value for low relative motion.

Dependencies

To enable this parameter, set the **Method** parameter to `Horn-Schunck`.

Stop iterative solution — Stop iterative solution
`When maximum number of iterations is reached` (default) | `When velocity difference falls below threshold` | `Whichever comes first`

Specify the method to control when the block's iterative solution process stops. If you want the process to stop when the velocity difference is below a certain threshold value, select `When velocity difference falls below threshold`. If you want the process to stop after a certain number of iterations, choose `When maximum number of iterations is reached`. You can also select `Whichever comes first`.

Dependencies

To enable this parameter, set the **Method** parameter to `Horn-Schunck`.

Maximum number of iterations — Maximum number of iterations
`10` (default) | scalar

Specify the maximum number of iterations for the block to perform.

Dependencies

To enable this parameter, set the **Method** parameter to `Horn-Schunck` and the **Stop iterative solution** parameter to `When maximum number of iterations is reached` or `Whichever comes first`.

Velocity output — Optical flow output
`Magnitude-squared` (default) | `Horizontal and vertical components in complex form`

Specify how to output an optical flow. If you select `Magnitude-squared`, the block outputs an optical flow matrix where each element is in the form $u^2 + v^2$. If you select `Horizontal and vertical components in complex form`, the block outputs the optical flow matrix where each element is in the form $u + jv$.

Temporal gradient filter — Filter used for temporal gradient
`Difference filter [-1 1]` (default) | `Derivative of Gaussian`

Specify whether the block solves for u and v using a difference filter or a derivative of a Gaussian filter.

Dependencies

To enable this parameter, set the **Method** parameter to `Lucas-Kanade`.

Number of input frames to buffer — Number of input frames to buffer for smoothing
`3` (default) | scalar

Specify the number of input frames to buffer for smoothing. Use this parameter for temporal filter characteristics such as the standard deviation and number of filter coefficients.

Dependencies

To enable this parameter, set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Standard deviation for image smoothing filter — Standard deviation for image smoothing filter
`1.5` (default) | scalar

Specify the standard deviation for the image smoothing filter.

Dependencies

To enable this parameter, set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Standard deviation for gradient smoothing filter — Standard deviation for gradient smoothing filter
`1` (default) | scalar

Specify the standard deviation for the gradient smoothing filter.

Dependencies

To enable this parameter, set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Discard normal flow estimates when constraint equation is ill-conditioned — Discard normal flow estimates
off (default) | on

Select this parameter to set the motion vector to zero when the optical flow constraint equation is ill-conditioned.

Dependencies

To enable this parameter, set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Output image corresponding to motion vectors (accounts for block delay) — Output image corresponding to motion vectors
off (default) | on

Select this parameter to output the image that corresponds to the motion vector outputted by the block.

Dependencies

To enable this parameter, set the **Temporal gradient filter** parameter to `Derivative of Gaussian`.

Threshold for noise reduction — Threshold for noise reduction
`0.0039` (default) | scalar

Specify a scalar value that determines the motion threshold between each image or video frame. The higher the number, the less small movements impact the optical flow calculation.

Dependencies

To enable this parameter, set the **Method** parameter to `Lucas-Kanade`.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`double` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Algorithms

Optical Flow Equation

To compute the optical flow between two images, you must solve this optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

.

- I_x, I_y, and I_t are the spatiotemporal image brightness derivatives.
- u is the horizontal optical flow.
- v is the vertical optical flow.

Horn-Schunck Method

By assuming that the optical flow is smooth across the entire image, the Horn-Schunck method estimates a velocity field, $[u\ v]^T$, that minimizes this equation:

$$E = \iint (I_x u + I_y v + I_t)^2 dx dy + \alpha \iint \left\{ \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right\} dx dy$$

.

In this equation, $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ are the spatial derivatives of the optical velocity component, u, and α scales the global smoothness term. The Horn-Schunck method minimizes the previous equation to obtain the velocity field, $[u\ v]$, for each pixel in the image. This method is given by the following equations:

$$u_{x,y}^{k+1} = \bar{u}_{x,y}^{k} - \frac{I_x [I_x \bar{u}^k_{x,y} + I_y \bar{v}^k_{x,y} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v_{x,y}^{k+1} = \bar{v}_{x,y}^{k} - \frac{I_y [I_x \bar{u}^k_{x,y} + I_y \bar{v}^k_{x,y} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

.

In these equations, $\left[u_{x,y}^k\ v_{x,y}^k\right]$ is the velocity estimate for the pixel at (x,y), and $\left[\bar{u}_{x,y}^k\ \bar{v}_{x,y}^k\right]$ is the neighborhood average of $\left[u_{x,y}^k\ v_{x,y}^k\right]$. For $k = 0$, the initial velocity is 0.

To solve u and v using the Horn-Schunck method:

1. Compute I_x and I_y by using the Sobel convolution kernel, $[-1\ -2\ -1;\ 0\ 0\ 0;\ 1\ 2\ 1]$, and its transposed form for each pixel in the first image.
2. Compute I_t between images 1 and 2 using the $[-1\ 1]$ kernel.
3. Assume the previous velocity to be 0, and compute the average velocity for each pixel using $[0\ 1\ 0;\ 1\ 0\ 1;\ 0\ 1\ 0]$ as a convolution kernel.
4. Iteratively solve for u and v.

Lucas-Kanade Method

To solve the optical flow constraint equation for u and v, the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity in each section. Then it performs a weighted, least-square fit of the optical flow constraint equation to a constant model for $[u\ v]^T$ in each section Ω. The method achieves this fit by minimizing this equation:

$$\sum_{x \in \Omega} W^2[I_x u + I_y v + I_t]^2$$

W is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

.

Lucas-Kanade Difference Filter

When you set the **Temporal gradient filter** to `Difference filter [-1 1]`, u and v are solved as follows:

1. Compute I_x and I_y using the kernel $[-1\ 8\ 0\ -8\ 1]/12$ and its transposed form.

   If you are working with fixed-point data types, the kernel values are signed fixed-point values with a word length equal to 16 and a fraction length equal to 15.
2. Compute I_t between images 1 and 2 by using the $[-1\ 1]$ kernel.
3. Smooth the gradient components, I_x, I_y, and I_t, by using a separable and isotropic 5-by-5 element kernel whose effective 1-D coefficients are $[1\ 4\ 6\ 4\ 1]/16$. If you are working with fixed-point data types, the kernel values are unsigned fixed-point values with a word length equal to 8 and a fraction length equal to 7.
4. Solve the 2-by-2 linear equations for each pixel using the following method:

   - If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

     then the eigenvalues of A are $\lambda_i = \frac{a+c}{2} \pm \frac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

     In the fixed-point diagrams, $P = \frac{a+c}{2}, Q = \frac{\sqrt{4b^2 + (a-c)^2}}{2}$

- The eigenvalues are compared to the threshold, τ, that corresponds to the value you enter for the threshold for noise reduction. The results fall into one of the following cases.

  Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

  A is nonsingular, and the system of equations is solved using Cramer's rule.

  Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

  A is singular (noninvertible), and the gradient flow is normalized to calculate u and v.

  Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

  The optical flow, u and v, is 0.

Derivative of Gaussian

If you set the temporal gradient filter to `Derivative of Gaussian`, u and v are solved using these steps.

1. Compute I_x and I_y.
   a. Use a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics, such as the standard deviation and number of filter coefficients, by using the **Number of frames to buffer for temporal smoothing** parameter.
   b. Use a Gaussian filter and the derivative of a Gaussian filter to smooth the image by using spatial filtering. Specify the standard deviation and length of the image smoothing filter by using the **Standard deviation for image smoothing filter** parameter.
2. Compute I_t between images 1 and 2.
   a. Use the derivative of a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics, such as the standard deviation and number of filter coefficients, by using the **Number of frames to buffer for temporal smoothing** parameter.
   b. Use the filter described in step 1b to perform spatial filtering on the output of the temporal filter.
3. Smooth the gradient components, I_x, I_y, and I_t, by using a gradient smoothing filter. Use the **Standard deviation for gradient smoothing filter** parameter to specify the standard deviation and the number of filter coefficients for the gradient smoothing filter.
4. Solve the 2-by-2 linear equations for each pixel using this method:

   - If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

     then the eigenvalues of A are $\lambda_i = \frac{a+c}{2} \pm \frac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

   - When the block finds the eigenvalues, it compares them to the threshold, τ, that corresponds to the value you enter for the **Threshold for noise reduction** parameter. The results fall into one of the following cases.

     Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

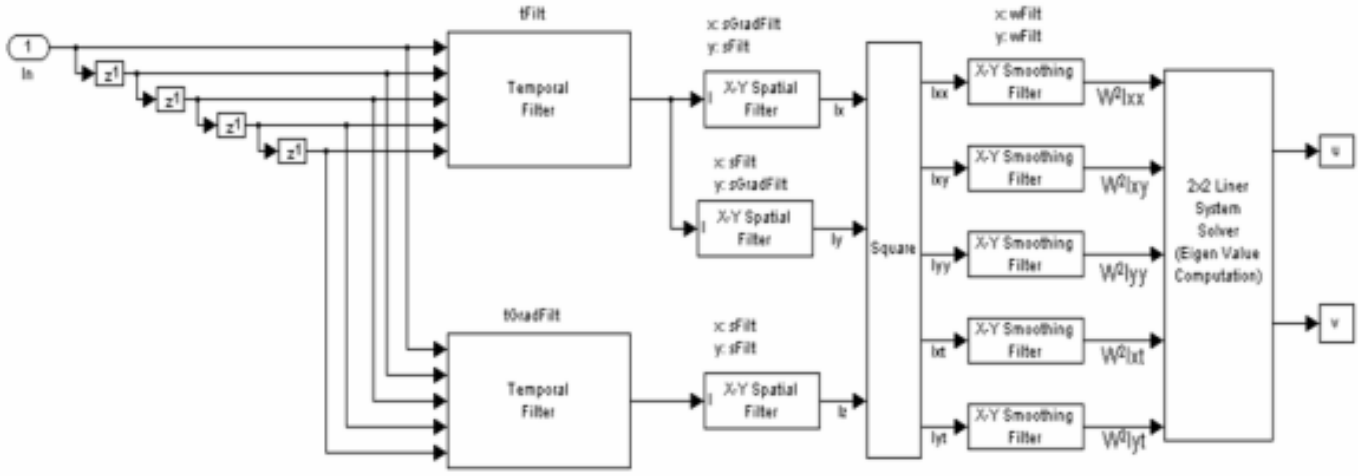
A is nonsingular, so the block solves the system of equations by using Cramer's rule.

Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

A is singular (noninvertible), so the block normalizes the gradient flow to calculate u and v.

Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

the optical flow, u and v, is 0.



tFilt = Coefficients of Gaussian Filter
tGradFilt = Coefficients of the Derivative of a Gaussian Filter

sFilt = Coefficients of Gaussian Filter
sGradFilt = Coefficients of the Derivative of a Gaussian Filter

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

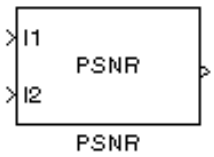
Blocks

Block Matching | Gaussian Pyramid

Objects
`opticalFlow` | `opticalFlowRAFT` | `opticalFlowHS` | `opticalFlowLK` | `opticalFlowLKDoG`

PSNR

Compute peak signal-to-noise ratio (PSNR) between images



Libraries:
Computer Vision Toolbox / Statistics

Description

The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

The mean-square error (MSE) and the peak signal-to-noise ratio (PSNR) are used to compare image compression quality. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error.

To compute the PSNR, the block first calculates the mean-squared error using the following equation:

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M*N}$$

In the previous equation, M and N are the number of rows and columns in the input images. Then the block computes the PSNR using the following equation:

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right)$$

In the previous equation, R is the maximum fluctuation in the input image data type. For example, if the input image has a double-precision floating-point data type, then R is 1. If it has an 8-bit unsigned integer data type, R is 255, etc.

Computing PSNR for Color Images

Different approaches exist for computing the PSNR of a color image. Because the human eye is most sensitive to luma information, you can compute the PSNR for color images by converting the image to a color space that separates the intensity (luma) channel, such as YCbCr. The Y (luma), in YCbCr represents a weighted average of R, G, and B. G is given the most weight, again because the human eye perceives it most easily. Compute the PSNR only on the luma channel.

Ports

Input

I1 — Input image
scalar | vector | matrix

Input image, specified as scalar, vector, or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

I2 — Input image
scalar | vector | matrix

Input image, specified as scalar, vector, or matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Output

Output — Output
scalar

Peak signal-to-noise ratio between images, returned as a scalar.

Dependencies

If the input is a fixed-point or integer data type, the block output is double-precision floating point. Otherwise, the block input and output are the same data type.

Data Types: `double`

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Generates code only for `double` or `single` data types.

Fixed-Point Conversion

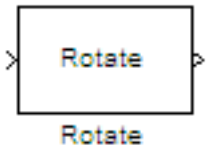
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

`psnr`

Rotate

Rotate image by specified angle



Libraries:
Computer Vision Toolbox / Geometric Transformations

Description

Use the Rotate block to rotate a grayscale or color image by an angle specified in radians.

Ports

Input

Image — Image to rotate
matrix | array

Image to rotate, specified as a 2-D numeric matrix for a grayscale image or 3-D numeric array for a color image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Angle — Rotation angle
scalar

Rotation angle, specified as a numeric scalar, in radians.

If the input to the **Image** port is of data type `single`, `double`, or `fixed point`, then the input to the **Angle** port must be of the same data type.

If the input to the **Image** port is of a signed or unsigned integer data type, then the input to the **Angle** port must be of a signed integer data type.

Dependencies

To enable this port, set the **Rotation angle source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `fixed point`

Output

Output — Rotated image
matrix | array

Rotated image, returned as 2-D numeric matrix if the input image is a grayscale image and a 3-D numeric array if the input image is a color image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `fixed point`

Parameters

Main

Output size — Size of rotated image

`Expanded to fit rotated input image` (default) | `Same as input image`

Specify the size of the rotated image as one of these options.

- `Expanded to fit rotated input image`- The block outputs a matrix or array expanded to contain all values.
- `Same as input image`- The block outputs a matrix or array the same size as the input image, which contains the middle part of the rotated image. This can result in the block cropping the edges of the rotated image.

Use the **Background fill value** parameter to specify the color or intensity value of pixels outside the rotated image.

Rotation angle source — Rotation angle source

`Specify via dialog` (default) | `Input port`

Specify the source of the rotation angle as one of these options.

- `Specify via dialog` - Specify the rotation angle using the **Angle (radians)** parameter.
- `Input port` - Specify the rotation angle using the **Angle** port. The block uses the input to this port as the rotation angle at each time step.

Angle (radians) — Rotation angle

`pi/6` (default) | numeric scalar

Specify the rotation angle in radians.

When the rotation angle is a multiple of `pi/2`, the block uses a more efficient algorithm. If the specified value for the **Angle (radians)** parameter is within `1e-5` radians of a multiple of `pi/2`, the block rounds the angle value to that multiple of `pi/2` before performing the rotation.

Dependencies

To enable this parameter, set the **Rotation angle source** parameter to `Specify via dialog`.

Maximum angle (enter pi radians to accommodate all positive and negative angles) — Maximum angle of rotation

`pi` (default) | nonnegative scalar less than or equal to `pi`

Specify the maximum angle by which to rotate the input image as a scalar value in the range [`0`, `pi`], in radians. The block determines which angle, between `0` and the specified maximum angle, requires the largest output size and sets the dimensions of the output port accordingly.

Dependencies

To enable this parameter, specify these parameter values.

- **Output size** — `Expanded to fit rotated input image`
- **Rotation angle source** — `Input port`

Display rotated image in — Position of output image

`Top-left corner` (default) | `Center`

Specify the position of the rotated image in the output image. If you select `Top-left corner`, the block positions the rotated image so that it has corners in contact with the top and left sides of the output image. If you select `Center`, the output image displays the rotated image at its center.

Dependencies

To enable this parameter, set the **Rotation angle source** parameter to `Input port`.

Sine value computation method — Sine value computation method

`Table lookup` (default) | `Trigonometric function`

Specify the sine value computation method. If you select `Trigonometric function`, the block computes the sine and cosine values required to calculate the rotation of the image during the simulation.

If you select `Table lookup`, the block computes and stores the trigonometric values required to calculate the rotation of the image before the simulation starts. For this method, the block requires extra memory.

Background fill value — Background fill value

numeric scalar | three-element numeric vector

Specify a value for the pixels that are outside the image.

If the input image is grayscale, the value of **Background fill value** must be a numeric scalar. The default value is `0` for grayscale images. If the input image is a color image, the value of **Background fill value** must be a three-element numeric vector. The default value is `[0 0 0]` for color images.

Interpolation method — Interpolation method

`Bilinear` (default) | `Nearest neighbor` | `Bicubic`

Specify which interpolation method the block uses to rotate the image from these options.

- `Nearest neighbor` - The block uses the value of one nearby pixel for the new pixel value.
- `Bilinear` - The new pixel value is the weighted average of the 4 nearest pixel values.
- `Bicubic` - The new pixel value is the weighted average of the 16 nearest pixel values.

The number of pixels the block considers affects the complexity of the computation. Therefore, the `Nearest neighbor` interpolation is the most computationally efficient. However, because the accuracy of the method is proportional to the number of pixels considered, the `Bicubic` method is the most accurate.

Data Types

For details on general fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Angle values — Fixed-point parameter for angle values

`Same word length as input` (default) | `Specify word length` | `Binary point scaling`

Choose how to specify the word length and the fraction length of the angle values.

Dependencies

To enable this parameter, set the **Rotation angle source** parameter to `Specify via dialog`.

Lock data type settings against change by the fixed-point tools — Prevent fixed-point tools from overriding data types

`off` (default) | `on`

Select this parameter to prevent the fixed-point tools from overriding the data types you specify in this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Algorithms

The Rotate block uses the three-pass shear rotation algorithm to compute its values, which differs from the algorithm used by the `imrotate` function in the Image Processing Toolbox.

Version History

Introduced before R2006a

References

[1] Wolberg, George. *Digital Image Warping*. IEEE Computer Society Press Monograph. Los Alamitos, Calif: IEEE Computer Society Press, 1992.

Extended Capabilities

C/C++ Code Generation

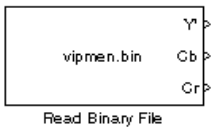
Generate C and C++ code using Simulink® Coder™.

See Also

`imrotate` | Resize | Translate | Shear

Read Binary File

Read video data from binary file



Libraries:
Computer Vision Toolbox / Sinks

Description

The Read Binary File block reads the video data from a binary file and imports it into a Simulink model.

The user-specified block parameters describe the format of the video data. These parameters, along with the raw binary file, create the video data for the Simulink model.

Note This block supports code generation for only those platforms that have file I/O. The block does not support code generation using Simulink Desktop Real-Time or Simulink Real-Time™.

Ports

Output

Y' — Luminance component
3-D array

Luminance component of the video read from the binary file, specified as a 3-D array. The label of this port can be changed using the **Component1** parameter when the **Video format** parameter is set to `Custom`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cb — Blue difference component
3-D array

Blue difference component of the video read from the binary file, specified as a 3-D array. The label of this port can be changed using the **Component2** parameter when the **Video format** parameter is set to `Custom`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cr — Red difference component
3-D array

Red difference component of the video read from the binary file, specified as a 3-D array. The label of this port can be changed using the **Component3** parameter when the **Video format** parameter is set to `Custom`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

A — Alpha component
3-D array

Alpha component of the video read from the binary file, specified as a 3-D array. The label of this port can be changed using the **Component4** parameter when the **Video format** parameter is set to `Custom`.

Dependencies

To enable this port, use one of these options.

- Set the **Video format** parameter to `Four character codes` and set the **Four character code** parameter to `AYUV` or `IF09`.
- Set the **Video format** parameter to `Custom` and the **Number of output components** parameter to `4`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

T — Transparency component
3-D array

Transparency component of the video read from the binary file, specified as a 3-D array.

Dependencies

To enable this port, set the **Video format** parameter to `Four character codes` and set the **Four character code** parameter to `Y41T` or `Y42T`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

EOF — Last video frame indicator
scalar

The port outputs a `1` when the component ports output the last video frame. For every frame before that, it outputs a `0`.

Dependencies

To enable this port, select the **Output end-of-file indicator** parameter.

Data Types: `Boolean`

Parameters

File name — Name of binary file
`vipmen.bin` (default) | file name

Specify the name of the binary source file to read video data from.

Video format — Format of video data
`Four character codes` (default) | `Custom`

Specify the format of the video data as one of these options.

- Four character codes (FOURCC) — For more information about these codes, see `https://en.wikipedia.org/wiki/FourCC`.

- Custom — Read the video data from a binary file of a custom format.

Four character code — Format of binary file
`I420` (default) | `AYUV` | `CLJR` | `cyuv` | `GREY` | `IF09` | `IMC1` | `IMC2` | `IMC3` | `IMC4` | `IUYV` | `IY41` | `IYU1` | `IYU2` | `IYUV` | `NV12` | `NV21` | `UYNV` | `UYNY` | `V210` | `Y411` | `Y41P` | `Y41T` | `Y422` | `Y42T` | `Y8` | `Y800` | `YUNV` | `YUY2` | `YUYV` | `YV12` | `YV16` | `YVU9` | `YVYU` | ...

Specify the four character code of the binary file codec. The value of this parameter also sets the number of output ports. For information on which four character codes enable which output ports, see the table below.

Output Ports for Four Character Codes

Four Character Codes	Output Ports
I420	Y'|Cb|Cr
AYUV	Y'|Cb|Cr|A
CLJR	Y'|Cb|Cr
cyuv	Y'|Cb|Cr
GREY	Y'
IF09	Y'|Cb|Cr|A
IMC1	Y'|Cb|Cr
IMC2	Y'|Cb|Cr
IMC3	Y'|Cb|Cr
IMC4	Y'|Cb|Cr
IUYV	Y'|Cb|Cr
IY41	Y'|Cb|Cr
IYU1	Y'|Cb|Cr
IYU2	Y'|Cb|Cr
IYUV	Y'|Cb|Cr
NV12	Y'|Cb|Cr
NV21	Y'|Cb|Cr
UYNV	Y'|Cb|Cr
UYNY	Y'|Cb|Cr
V210	Y'|Cb|Cr
Y411	Y'|Cb|Cr
Y41P	Y'|Cb|Cr
Y41T	Y'|Cb|Cr|T
Y422	Y'|Cb|Cr
Y42T	Y'|Cb|Cr|T
Y8	Y'
Y800	Y'
YUNV	Y'|Cb|Cr
YUY2	Y'|Cb|Cr
YUYV	Y'|Cb|Cr
YV12	Y'|Cb|Cr
YV16	Y'|Cb|Cr
YVU9	Y'|Cb|Cr
YVYU	Y'|Cb|Cr

Dependencies

To enable this parameter, set the **Video format** parameter to `Four character codes`.

FrameSize Rows — Number of rows in output array
`120` (default) | positive integer

Specify the number of rows in the output array as a positive integer. The value of this parameter must match the dimensions of the data inside the file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Four character codes`.

FrameSize Cols — Number of columns in output array
`160` (default) | positive integer

Specify the number of columns in the output array as a positive integer. The value of this parameter must match the dimensions of the data inside the file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Four character codes`.

Line ordering — Output array fill direction
`Top line first` (default) | `Bottom line first`

If you select `Top line first`, the block first fills the first row of the output array with the corresponding contents of the binary file. It then fills the other rows in increasing order. When you select `Bottom line first`, the block first fills the last row of the output array with the corresponding contents of the binary file. It then fills the other rows in decreasing order.

Number of times to play file — Number of times to play the file
`1` (default) | positive integer | `inf`

Specify the number of times to repeat the binary file as a positive integer or `inf`. For any positive integer value `n` of this parameter, the block plays the binary file `n` times. For a value of `inf`, the block plays the file indefinitely until the end of simulation.

Output end-of-file indicator — Indicate when block output reaches last frame
`off` (default) | `on`

This parameter indicates when the block output reaches the last video frame in the binary file. Selecting this parameter enables the **EOF** port. The output from the **EOF** port is `1` when the block output reaches the last video frame. For all other frames, it is `0`.

Sample time — Sample period of output
`1/30` (default) | positive scalar | `-1`

Specify the sample period of the output. Sample time indicates when, during simulation the block produces output. The value of this parameter must be a positive scalar less than the simulation time or `-1`. It cannot be `0`. When set to `-1`, the block inherits the sample time from the simulink model.

Bit stream format — Specifies if data is planar or packed
`Planar` (default) | `Packed`

Specify the data format of the binary file as planar or packed. If the data is packed, use the **FrameSize Rows** and **FrameSize Cols** parameters to define the size of the output array.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Number of output components — Number of components in binary file
3 (default) | 1 | 2 | 4

Specify the number of output components from the binary file. This parameter sets the number of output ports.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Component1 — Name for luminance component
`Y'` (default) | component name

Specify the name for the luminance component of the output array. The luminance component is the first component from the input binary file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of output components** parameter to 1 or greater.

Component2 — Name for blue difference component
`Cb` (default) | component name

Specify the name for the blue difference component of the output array. The blue difference component is the second component from the input binary file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of output components** parameter to 2 or greater.

Component3 — Name for red difference component
`Cr` (default) | component name

Specify the name for the red difference component of the output array using this parameter. The red difference component is the third component from the input binary file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of output components** parameter to 3 or greater.

Component4 — Name for alpha component
`Alpha` (default) | component name

Specify the name for the alpha component of the output array. The alpha component is the fourth component from the input binary file.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of output components** parameter to 4.

Bits1 — Size of luminance component
8 (default) | 16 | 24 | 32

Specify the size of the luminance component in the output array. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 1 or greater.

Bits2 — Size of blue difference component
8 (default) | 16 | 24 | 32

Specify the size of the blue difference component in the output array. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 2 or greater.

Bits3 — Size of red difference component
8 (default) | 16 | 24 | 32

Specify the size of the red difference component in the output array. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 3 or greater.

Bits4 — Size of alpha component
8 (default) | 16 | 24 | 32

Specify the size of the alpha component in the output array. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 4.

Rows1 — Number of rows for luminance component
`120` (default) | positive integer

Specify the number of rows for the luminance component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 1 or greater.

Rows2 — Number of rows for blue difference component
`60` (default) | positive integer

Specify the number of rows for the blue difference component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 2 or greater.

Rows3 — Number of rows for red difference component
`60` (default) | positive integer

Specify the number of rows for the red difference component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 3 or greater.

Rows4 — Number of rows for alpha component
`288` (default) | positive integer

Specify the number of rows for the alpha component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output ports** parameter to 4.

Cols1 — Number of columns for luminance component
`160` (default) | positive integer

Specify the number of columns for the luminance component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 1 or greater.

Cols2 — Number of columns for blue difference component
`80` (default) | positive integer

Specify the number of columns for the blue difference component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 2 or greater.

Cols3 — Number of columns for red difference component
`80` (default) | positive integer

Specify the number of columns for the luminance component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 3 or greater.

Cols4 — Number of columns for the alpha component
`352` (default) | positive integer

Specify the number of columns for the alpha component in the output array. The value of this parameter must be a positive integer.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of output components** parameter to 4.

Component order in binary file — Arrangement of components in binary file
`[1 2 3]` (default) | vector of valid component identifiers

Specify the arrangement of components in the binary file. Identify the components by the assigned numerals, from 1 to 4, in desired order. You must include all the enabled components when setting this parameter.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Interlaced video — Read video data from file as interlaced
`off` (default) | `on`

Select this parameter when the binary file contains interlaced video data.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Input file has signed data — File has signed integers
`off` (default) | `on`

Select this parameter when the binary file contains signed integers.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Byte order in binary file — Byte order of binary file
`Little endian` (default) | `Big endian`

Specify the byte order of the binary file as either little endian or big endian.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Block Characteristics

Data Types	`integer`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

Extended Capabilities

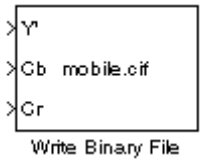
C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
From Multimedia File | Write Binary File

Write Binary File

Write binary video data to file



Libraries:
Computer Vision Toolbox / Sinks

Description

The Write Binary File block takes the video data from a Simulink model and exports it to a binary file.

This block produces a raw binary file with no header information. It has no encoded information providing the data type, frame rate, or dimensionality. The video data for this block appears in row-major format.

Note This block supports code generation only for platforms that have file I/O available. The block does not support code generation using Simulink Desktop Real-Time or Simulink Real-Time.

Ports

Input

Y' — Luminance component of input video stream
scalar | vector | matrix

Luminance component of the input video stream, specified as a scalar, vector, or matrix. The label of this port can be changed using the **Component1** parameter when the **Video format** parameter is set to `Custom`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cb — Blue difference component of the input video stream
scalar | vector | matrix

Blue difference component of the input video stream, specified as a scalar, vector, or matrix. The label of this port can be changed using the **Component2** parameter when the **Video format** parameter is set to `Custom`.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Cr — Red difference component of input video stream
scalar | vector | matrix

Red difference component of the input video stream, specified as a scalar, vector, or matrix. The label of this port can be changed using the **Component3** parameter when the **Video format** parameter is set to `Custom`.

Data Types: int8 | int16 | int32 | uint8 | uint16 | uint32

A — Alpha component of input video stream
scalar | vector | matrix

Alpha component of the input video stream, specified as a scalar, vector, or matrix. The label of this port can be changed using the **Component4** parameter when the **Video format** parameter is set to Custom.

Dependencies

To enable this port, use one of these options.

- Set the **Video format** parameter to Four character codes, and set the **Four character code** parameter to AYUV or IF09.
- Set the **Video format** parameter to Custom, and the **Number of inputs** parameter to 4. In this case, the name of this port is **Alpha**.

Data Types: int8 | int16 | int32 | uint8 | uint16 | uint32

T — Transparency component of input video stream
scalar | vector | matrix

Transparency component of the input video stream, specified as a scalar, vector, or matrix.

Dependencies

To enable this port, set the **Video format** parameter to Four character codes, and set the **Four character code** parameter to Y41T or Y42T.

Data Types: int8 | int16 | int32 | uint8 | uint16 | uint32

Parameters

File name — Name of the binary file
output.bin (default) | file name

Specify the name of the output binary file, with a .bin extension, for the video data. Click **Save As** to specify the location for the file.

Video format — Format of the video data
Four character codes (default) | Custom

Specify the format of the video data as one of these options.

- Four character codes (FOURCC) — For more information about these codes, see https://en.wikipedia.org/wiki/FourCC.
- Custom — Read the video data from a binary file of a custom format.

Four character code — Format of binary file
I420 (default) | AYUV | CLJR | cyuv | GREY | IF09 | IMC1 | IMC2 | IMC3 | IMC4 | IUYV | IY41 | IYU1 | IYU2 | IYUV | NV12 | NV21 | UYNV | UYNY | V210 | Y411 | Y41P | Y41T | Y422 | Y42T | Y8 | Y800 | YUNV | YUY2 | YUYV | YV12 | YV16 | YVU9 | YVYU | ...

Select the four character code of the desired codec for the binary file. The value of this parameter also determines the number of input ports on the block. For information on which four character codes enable which input ports, see the table below.

Input Ports for Four Character Codes

Four Character Codes	Input Ports
I420	Y'|Cb|Cr
AYUV	Y'|Cb|Cr|A
CLJR	Y'|Cb|Cr
cyuv	Y'|Cb|Cr
GREY	Y'
IF09	Y'|Cb|Cr|A
IMC1	Y'|Cb|Cr
IMC2	Y'|Cb|Cr
IMC3	Y'|Cb|Cr
IMC4	Y'|Cb|Cr
IUYV	Y'|Cb|Cr
IY41	Y'|Cb|Cr
IYU1	Y'|Cb|Cr
IYU2	Y'|Cb|Cr
IYUV	Y'|Cb|Cr
NV12	Y'|Cb|Cr
NV21	Y'|Cb|Cr
UYNV	Y'|Cb|Cr
UYNY	Y'|Cb|Cr
V210	Y'|Cb|Cr
Y411	Y'|Cb|Cr
Y41P	Y'|Cb|Cr
Y41T	Y'|Cb|Cr|T
Y422	Y'|Cb|Cr
Y42T	Y'|Cb|Cr|T
Y8	Y'
Y800	Y'
YUNV	Y'|Cb|Cr
YUY2	Y'|Cb|Cr
YUYV	Y'|Cb|Cr
YV12	Y'|Cb|Cr
YV16	Y'|Cb|Cr
YVU9	Y'|Cb|Cr
YVYU	Y'|Cb|Cr

Dependencies

To enable this parameter, set the **Video Format** parameter to `Four character codes`.

Line ordering — Binary file fill direction
`Top line first` (default) | `Bottom line first`

If you select `Top line first`, the block begins filling the binary file with the first row of the video frame. It then fills the file with the other rows in increasing order. If you select `Bottom line first`, the block begins filling the binary file with the last row of the video frame. It then fills the file with the other rows in decreasing order.

Bit stream format — Data format of binary file
`Planar` (default) | `Packed`

Specify the data format of the binary file as `Planar` or `Packed`.

Dependencies

To enable this parameter, set the **Video Format** parameter to **Custom**.

Number of inputs — Number of components in input video stream
3 (default) | 1 | 2 | 4

Specify the number of components in the input video stream. This parameter sets the number of input ports.

Dependencies

To enable this parameter, set the **Video Format** parameter to `Custom`.

Inherit size of components from input data type — All components have same number of bits
`on` (default) | `off`

Select the **Inherit size of components from input data type** parameter to indicate that all of the components have the same number of bits as the input data type. If you clear this parameter, you can specify the number of bits for each enabled component using the associated **Bits** parameter.

Dependencies

To enable this parameter, set the **Video Format** parameter to `Custom`.

Component1 — Name of luminance component input port
`Y'` (default) | component name

Specify the name of the input port for the luminance component of the input video stream.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of inputs** parameter to `1` or greater.

Component2 — Name of blue difference component
`Cb` (default) | component name

Specify the name of the input port for the blue difference component of the input video stream.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of inputs** parameter to 2 or greater.

Component3 — Name of red difference component
`Cr` (default) | component name

Specify the name of the input port for the red difference component of the input video stream.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of inputs** parameter to 3 or greater.

Component4 — Name of alpha component
`Alpha` (default) | component name

Specify the name of the input port for the alpha component of the input video stream.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, set the **Number of inputs** parameter to 4.

Bits1 — Size of luminance component
8 (default) | `0` | 16 | 24 | 32

Specify the size of the luminance component in the input video stream. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of inputs** parameter to 1 or greater, and select the **Inherit size of components from input data type** parameter.

Bits2 — Size of blue difference component
8 (default) | `0` | 16 | 24 | 32

Specify the size of the blue difference component in the input video stream. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of inputs** parameter to 2 or greater, and select the **Inherit size of components from input data type** parameter.

Bits3 — Size of red difference component
8 (default) | `0` | 16 | 24 | 32

Specify the size of the red difference component in the input video stream. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of inputs** parameter to 3 or greater, and select the **Inherit size of components from input data type** parameter.

Bits4 — Size of alpha component
`8` (default) | `0` | `16` | `24` | `32`

Specify the size of the alpha component in the input video stream. The value of this parameter determines the interpretation of the video stream data as 8-bit, 16-bit, 24-bit, or 32-bit.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`, and set the **Number of inputs** parameter to 4, and select the **Inherit size of components from input data type** parameter.

Component order in binary file — Arrangement of components in binary file
`[1 2 3]` (default) | vector of valid component identifiers

Specify the arrangement of the components in the binary file. Identify the components by the assigned numerals, from 1 to 4, in the desired order. You must include all enabled components when setting this parameter.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Interlaced video — Read video data as interlaced
`off` (default) | `on`

Select this parameter when the video stream contains interlaced video data.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Write signed data to output file — Write signed data to output binary file
`off` (default) | `on`

Select this parameter when the input data is signed.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Byte order in binary file — Byte order in output binary file
`Little endian` (default) | `Big endian`

Specify the byte order in the output binary file as either little endian or big endian.

Dependencies

To enable this parameter, set the **Video format** parameter to `Custom`.

Block Characteristics

Data Types	`integer`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

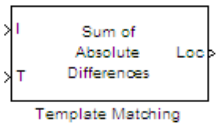
See Also

Blocks

Read Binary File | To Multimedia File

Template Matching

Locate a template in an image



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Template Matching block finds the best match of a template within an input image. The block computes match metric values by shifting a template over a region of interest or the entire image, and then finds the best match location.

The block outputs either the match metric values or the one-based (*x*,*y*) coordinates of the best template match. Optionally the block can output an *N*-by-*N* matrix of the match metric values centered around the best match location.

Ports

Input

I — Image
image

Input image to use with the template. The block does not pad the input data. Therefore, it can only compute values for the match metrics between the input image and the template, where the template is positioned such that it falls entirely on the input image. A set of all such positions of the template is termed as the *valid* region of the input image. The size of the valid region is the difference between the sizes of the input and template images plus one.

$$size_{valid}=size_{input}-size_{template}+1 \quad (1\text{-}1)$$

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

T — Template
M-by-*N* matrix

Template, specified as an *M*-by-*N* matrix

The Template Matching block does not pad the input data. Therefore, it can only compute values for the match metrics between the input image and the template, where the template is positioned such that it falls entirely on the input image. A set of all such positions of the template is termed as the *valid* region of the input image. The size of the valid region is the difference between the sizes of the input and template images plus one.

$$size_{valid}=size_{input}-size_{template}+1 \quad (1\text{-}2)$$

Data Types: `single` | `double` | `uint8` | `Boolean` | `fixed point`

ROI — Region of interest
four-element vector

Region of interest vector in the format (*x*,*y*,*width*,*height*), where (*x*,*y*) are one-based coordinates for the upper-left corner of the region. The block outputs the best match location index relative to the top left corner of the input image.

Data Types: `single` | `double` | `uint8` | `Boolean` | `fixed point`

Output

Metric — Match metric values
matrix

Matrix of match metric values. The matrix of the match metrics always implements single-step exhaustive window iteration. Therefore, the block computes the metric values at every pixel.

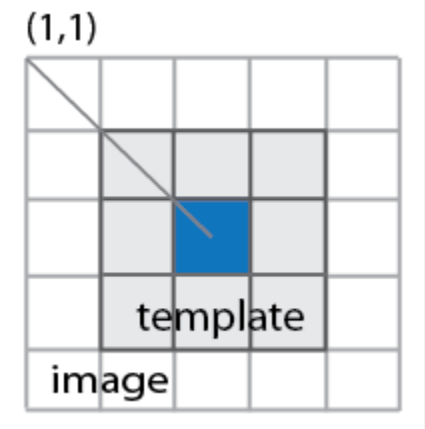
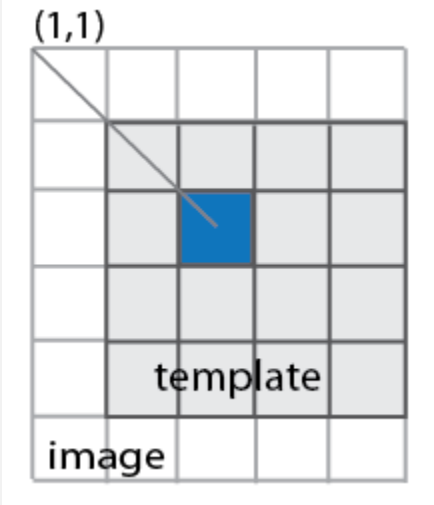
Dependencies

When you set the **Output** parameter to `Metric matrix`, the block outputs the valid image size.

Data Types: `single` | `double` | `uint8` | `Boolean` | `fixed point`

Loc — Best match locations
two-element vector

Best template match located at the one-based (*x*,*y*) coordinates. When in the ROI processing mode, the block treats the image around the ROI as an extension of the ROI subregion. Therefore, it computes the best match locations true to the actual boundaries of the ROI. The block outputs the best match coordinates, relative to the top-left corner of the image. The one-based [x y] coordinates of the location correspond to the center of the template. The following table shows how the block outputs the center coordinates for odd and even templates:

Odd number of pixels in template	Even number of pixels in template
	

Data Types: `uint32`

NMetric — Metric values in neighborhood of best match
N-by-*N* matrix

Metric values in the neighborhood of the best match, returned as an *N*-by-*N* matrix. *N*, specified in the mask, must be an odd number. When you select **Best match location** to return the matrix of metrics in a neighborhood around the best match, an exhaustive loop computes all the metric values for the *N*-by-*N* neighborhood. This output is particularly useful for performing template matching with subpixel accuracy.

Data Types: `single` | `double` | `uint8` | `Boolean` | `fixed point`

NValid — Valid neighborhood
off | on

Enable to track the valid neighborhood region. The neighborhood matrix of metric values is valid inside of the ROI. The block sets the **NValid** output as follows:

- `1` — The neighborhood containing the best match is completely inside the region of interest.
- `0` — The neighborhood containing the best match is completely or partially outside of the region of interest.

Dependencies

This port appears when you enable the **Output NxN matrix of metric values around best match** parameter.

Data Types: `Boolean`

ROIValid — Valid region of interest
off | on

Enable to track the valid region of interest. If the ROI lies partially outside the valid image, the block only processes the intersection of the ROI and the valid image. The block sets the **ROIValid** output as follows:

- `1` — The ROI lies completely inside the valid part of the input image.
- `0` — The ROI lies completely or partially outside of the valid part of the input image.

Dependencies

This port appears when you enable the **Output flag indicating if ROI is valid** parameter.

Data Types: `Boolean`

Parameters

Main Tab

Match metric — Match metric
`Sum of absolute differences` (default) | `Sum of squared differences` | `Maximum absolute difference`

Select match metric as one of:

- `Sum of absolute differences` (SAD)
- `Sum of squared differences` (SSD)
- `Maximum absolute difference` (MaxAD)

The block computes the match metric at each step of the iteration. Choose the match metric that best suits your application. The block calculates the global optimum for the best metric value. It uses the valid subregion of the input image intersected by the ROI, if provided.

Output — Match metric
`Match metric` (default) | `Best match location`

Select the type of output as one of:

- `Match metric` — Output the match metric matrix. This option adds the **Metric** output port to the block.
- `Best match location` — Output the [*x*,*y*] coordinates for the location of the best match. This option adds the **Loc** output port to the block. It also makes the **Search method**, **Output NxN matrix of metric values around best match**, and **Enable ROI processing** parameters available.

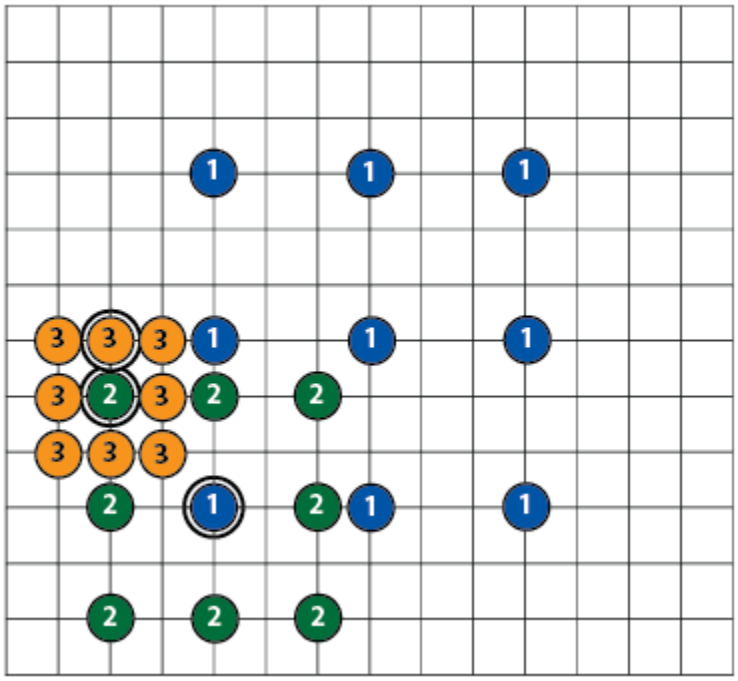
  The **Output** parameter on the Data Types pane appears when you set the **Output** parameter on the **Main tab** to `Metric matrix` or if you set it to `Best match location` and you enable the **Output NxN matrix of metric values around best match** parameter.

Search method — Search method
`Exhaustive` (default) | `Three-step`

Specify the search method as either `Exhaustive` or `Three-step`. The `Exhaustive` search method is computationally intensive because it searches at every pixel location of the image. However, this method provides a more precise result.

The `Three-step` search method is a fast search that uses a neighborhood approach versus a search at every pixel. The search starts with a step size equal to or slightly greater than half of the maximum search range and then employs the following steps:

1. The block compares nine search points in each step. There is a central point and eight search points located on the search area boundary.
2. The block decrements the step size by one, after each step, ending the search with a step size of one pixel.
3. At each new step, the block moves the search center to the best matching point resulting from the previous step. The number one blue circles in the figure below represent a search with a starting step size of three. The number two green circles represent the next search, with step size of two, centered around the best match found from the previous search. Finally, the number three orange circles represent the final search, with step size of one, centered around the previous best match.



Dependencies

This parameter appears when you set the **Output** parameter to `Best match location`.

Output NxN matrix of metric values around best match — Output *N*-by-*N* matrix
on (default) | off

Enable to add the **NMetric** and **NValid** ports.

Dependencies

This parameter appears when you set the **Output** parameter to `Best match location`.

N — Size of output matrix
3 (default) | integer

Size of output matrix. This value determines the size of the *N*-by-*N* output matrix centered around the best match location index. *N* must be an odd number.

Dependencies

This parameter appears when you enable the **Output NxN matrix of metric values around best match** parameter.

Enable ROI processing — Enable ROI processing
on (default) | off

Enable ROI processing.

Dependencies

- This parameter appears when you set the **Output** parameter to `Best match location`.

- When you enable this parameter, the block adds the **ROI** input port, and the **Output flag indicating if ROI is valid** parameter appears.

Output flag indicating if ROI is valid — Valid ROI
off (default) | on

Enable to add the **ROIValid** port.

Dependencies

This parameter appears when you enable the **Enable ROI processing** parameter.

Data Types Tab

For details on the fixed-point block parameters, see “Specify Fixed-Point Attributes for Blocks” (DSP System Toolbox).

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

Algorithms

Match Metrics

The match metrics use a difference equation with general form:

$$d_p(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

l_n^p denotes the metric space (R^n, d_p) for R^n $n > 1$.

Sum of Absolute Differences (SAD)

The SAD metric is also known as the *Taxicab* or *Manhattan Distance* metric. It sums the absolute values of the differences between pixels in the original image and the corresponding pixels in the template image. This metric is the l^1 norm of the difference image. The lowest SAD score estimates the best position of template within the search image. The general SAD distance metric becomes:

$$d_1(I_j, T) = \sum_{i=1}^{n} \left| I_{i,\,j} - T_i \right|$$

Sum of Squared Differences (SSD)

The SSD metric is also known as the Euclidean Distance metric. It sums the square of the absolute differences between pixels in the original image and the corresponding pixels in the template image. This metric is the square of the l^2 norm of the difference image. The general SSD distance metric becomes:

$$d_2(I_j, T) = \sum_{i=1}^{n} \left| I_{i,j} - T_i \right|^2$$

Maximum Absolute Difference (MaxAD)

The MaxAD metric is also known as the Uniform Distance metric. It sums the maximum of absolute values of the differences between pixels in the original image and the corresponding pixels in the template image. This distance metric provides the l^∞ norm of the difference image. The general MaxAD distance metric becomes:

$$d_\infty(I_j, T) = \lim_{x \to \infty} \sum_{i=1}^{n} \left| I_{i,j} - T_i \right|^p$$

which simplifies to:

$$d_\infty(I_j, T) = \max_{i}^{n} \left| I_{i,j} - T_i \right|^p$$

Version History

Introduced in R2009b

References

[1] Koga T., et. Al. Motion-compensated interframe coding for video conferencing. In National Telecommunications Conference. Nov. 1981, G5.3.1–5, New Orleans, LA.

[2] Zakai M., “General distance criteria” *IEEE Transaction on Information Theory*, pp. 94–95, January 1964.

[3] Yu, J., J. Amores, N. Sebe, Q. Tian, "A New Study on Distance Metrics as Similarity Measurement" IEEE International Conference on Multimedia and Expo, 2006.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

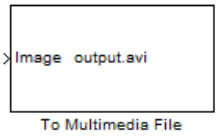
Block Matching

Topics

“Video Stabilization”

To Multimedia File

Write video frames and audio samples to multimedia file



Libraries:
Computer Vision Toolbox / Sinks

Description

The To Multimedia File block writes video frames, audio samples, or both to a multimedia file (`.avi`, `.wav`, `.mj2`, `.mp4`, or `.m4v` format).

You can compress the video frames or audio samples by selecting a compression algorithm. You can also control the type of video or audio in addition to other related properties the multimedia file receives.

If you have a Simulink Coder license, you can generate a code from a model containing this block. To run a generated executable file, you may need to add precompiled shared library files to your system path. See “Simulink Shared Library Dependencies”, and “Accelerating Simulink Models” for details.

Note This block supports code generation for platforms that have file I/O available. You cannot use this block with Simulink Desktop Real-Time software, because that product does not support file I/O.

This block performs best on platforms with Version 11 or later of Windows Media® Player software. This block supports only uncompressed RGB24 AVI files on Linux and Mac platforms.

Ports

Input

Image — Input image frame
matrix | array

Specify the input image frame as a matrix or an array. To write an intensity image frame to a multimedia file, specify it as an *M*-by-*N* matrix. For a color image frame, specify it as an *M*-by-*N*-by-3 array.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI`, `MJ-2000`, `WMV`, or `MPEG4`.
- If the **File type** parameter value is not `MJ-2000`, set the **Write** parameter to `Video and audio` or `Video only`.
- If the **File type** parameter value is `AVI` or `MPEG4`, set the **File color format** parameter to `RGB`.
- Set the **Image signal** parameter to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

R — Red color component of image
matrix

Specify the red color component of the input image frame as an *M*-by-*N* matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI`, `MJ-2000`, `WMV`, or `MPEG4`.
- If the **File type** parameter value is not `MJ-2000`, set the **Write** parameter to `Video and audio` or `Video only`.
- If the **File type** parameter value is `AVI` or `MPEG4`, set the **File color format** parameter to `RGB`.
- Set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

G — Green color component of image
matrix

Specify the green color component of the input image frame as an *M*-by-*N* matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI`, `MJ-2000`, `WMV`, or `MPEG4`.
- If the **File type** parameter value is not `MJ-2000`, set the **Write** parameter to `Video and audio` or `Video only`.
- If the **File type** parameter value is `AVI` or `MPEG4`, set the **File color format** parameter to `RGB`.
- Set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

B — Blue color component of image
matrix

Specify the blue color component of the input image frame as an *M*-by-*N* matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI`, `MJ-2000`, `WMV`, or `MPEG4`.
- If the **File type** parameter value is not `MJ-2000`, set the **Write** parameter to `Video and audio` or `Video only`.
- If the **File type** parameter value is `AVI` or `MPEG4`, set the **File color format** parameter to `RGB`.
- Set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Y — Luma component of image
matrix

Specify the luma component of the input image frame as an *M*-by-*N* matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI` or `MPEG4`.
- Set the **Write** parameter to `Video and audio` or `Video only`.
- Set the **File color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Cb — Blue-difference chrominance component of image
matrix

Specify the blue-difference chrominance component of the input image as an *M*-by-*N*/2 matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI` or `MPEG4`.
- Set the **Write** parameter to `Video and audio` or `Video only`.
- Set the **File color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Cr — Red-difference chrominance component of image
matrix

Specify the red-difference chrominance component of the input image as an *M*-by-*N*/2 matrix.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI` or `MPEG4`.
- Set the **Write** parameter to `Video and audio` or `Video only`.
- Set the **File color format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Audio — Input audio signal
vector | matrix

Specify the input audio signal as a vector or a matrix. When specified as a matrix, the block treats each column as an independent channel.

Dependencies

To enable this port:

- Set the **File type** parameter to `AVI`, `WAV`, `WMA`, `MPEG4`, `FLAC`, or `OGG`.
- If the **File type** parameter value is `AVI`, `WMV`, or `MPEG4` set the **Write** parameter to `Video and audio` or `Audio only`.

Data Types: `single` | `double` | `int16` | `int32` | `uint8`

Parameters

File name — Name of multimedia file
`output.avi` (default) | character vector

Specify the name for the multimedia file the block writes. The block appends a file extension automatically based on the specified **File type** parameter. If the specified file name, including the file extension, already exists, the block overwrites the data in the existing file with the new input data.

By default, the block saves the file in the current working directory. To specify a different file location, click the **Save As...** button.

File type — Type of multimedia file
`AVI` (default) | `WAV` | `MJ2000` | `WMA` | `WMV` | `MPEG4` | `FLAC` | `OGG`

Specify the type of multimedia file to write. The block supports the `AVI`, `WAV`, `MJ2000`, `WMA`, `WMV`, `MPEG4`, `FLAC`, and `OGG` file formats.

Write — Type of media
`Video only` (default) | `Audio only` | `Video and audio`

Specify the type of media to write. Depending on the value of the **File type** parameter, you can select `Video only`, `Audio only`, or `Video and audio`.

Type of file	Supported type of media
`AVI`	• `Video only` • `Audio only` • `Video and audio`
`WMV`	• `Video only` • `Video and audio`
`MPEG4`	• `Video only` • `Audio only`

Dependencies

To enable this parameter, set the **File type** parameter to `AVI`, `WMV`, or `MPEG4`.

Video compressor — Video compression algorithm
`None (uncompressed)` (default) | `DV Video Encoder` | `MJPEG Compressor` | `Lossy` | `Lossless`

Specify a video compression algorithm to reduce the size of the output file. The available options depend on the value of the **File type** parameter.

- `AVI` — Select either the `DV Video Encoder` or `MJPEG Compressor` algorithm. If you do not want to reduce the size of the file, select `None (uncompressed)`.

- `MJ2000` — Select either the `Lossy` or `Lossless` algorithm.

Note The options for this parameter are the video compression algorithms installed on your system. For information about a specific video compressor, see the documentation for that compressor.

Dependencies

To enable this parameter, set the **File type** parameter to `MJ2000` or `AVI`. For the `AVI` file type, set the **Write** parameter to `Video only` or `Video and audio`.

Compression Factor (>1) — Compression factor
`10` (default) | integer greater than `1`

Specify the compression factor as an integer greater than `1`.

Dependencies

To enable this parameter, set the **File type** parameter to `MJ2000` and the **Video compressor** parameter to `Lossy`.

Video Quality (0-100) — Quality of video
`75` (default) | integer in range [0, 100]

Specify the quality of the video as an integer in the range [0, 100].

Dependencies

To enable this parameter, set the **File type** parameter to `MPEG4` and the **Write** parameter to `Video only`.

File color format — Color format of output video
`RGB` (default) | `YCbCr 4:2:2`

Specify the color format of the video for the output file. The block supports the `RGB` and `YCbCr 4:2:2` color formats.

Dependencies

To enable this parameter, set the **File type** parameter to `AVI` or `MPEG4`.

Image signal — Input port configuration for RGB video
`One multidimensional signal` (default) | `Separate color signals`

Specify how the block accepts an RGB color video input. If you select `One multidimensional signal`, the block accepts input using only the **Image** port. If you select `Separate color signals`, the block accepts an input video using three separate color channels, divided across the **R**, **G**, and **B** ports.

Dependencies

To enable this parameter:

- Set the **File type** parameter to `AVI`, `MJ-2000`, `WMV`, or `MPEG4`.
- Set the **Write** parameter to `Video and audio` or `Video only`.

- If the **File type** parameter value is `AVI` or `MPEG4`, set the **File color format** parameter to `RGB`.

Audio compressor — Audio compression algorithm
`None (uncompressed)` (default) | `CCITT A-Law` | `CCITT u-Law` | `GSM 6.10` | `IMA ADPCM` | `Microsoft ADPCM` | `PCM`

Specify an audio compression algorithm to reduce the size of the output file. The block supports the `CCITT A-Law`, `CCITT u-Law`, `GSM 6.10`, `IMA ADPCM`, `Microsoft ADPCM`, and `PCM` algorithms. If you do not want to reduce the size of the file, select `None (uncompressed)`.

Note The options for this parameter are the audio compression algorithms installed on your system. For information about a specific audio compressor, see the documentation for that compressor.

Dependencies

To enable this parameter, use either of these options:

- Set the **File type** parameter to `AVI` and set the **Write** parameter to `Video and audio` or `Audio only`.
- Set the **File type** parameter to `WAV`.

Audio data type — Audio data type
`Determine from input data type` (default) | `8-bit integer` | `16-bit integer` | `24-bit integer` | `32-bit integer` | `32-bit float` | `64-bit float`

Specify the audio data type for the output file.

Dependencies

To enable this parameter select, use either of these options:

- Set the **File type** parameter to `WAV` and the **Audio compressor** parameter to `None (uncompressed)`.
- Set the **File type** parameter to `FLAC`. This file type supports only the `8-bit integer`, `16-bit integer`, and `24-bit integer` data types.

Block Characteristics

Data Types	`double` | `integer`[a] | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

a Supports 16- and 32-bit signed and 8-bit unsigned integers.

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Usage notes and limitations:

- Host computer only. Excludes Simulink Desktop Real-Time code generation.
- The executable generated from this block relies on prebuilt dynamic library files (`.dll` files) included with MATLAB. Use the `packNGo` function to package the code generated from this block and all the relevant files in a compressed ZIP file. Using this ZIP file, you can relocate, unpack, and rebuild your project in another development environment where MATLAB is not installed. For more details, see “Get Started with Code Generation, Deployment, GPU, and OpenCV Support”.

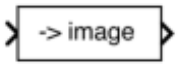
See Also

Blocks

From Multimedia File | Video From Workspace | Image From File | Video To Workspace

To Simulink Image

Pack numeric matrix into a Simulink image



Libraries:
Computer Vision Toolbox / Utilities

Description

The To Simulink Image block converts matrix data to Simulink image data.

Ports

Input

Port_1 — Input signal
matrix

Input signal to the To Simulink Image block. The input is a numerical matrix.

Data Types: `integer` | `single` | `double` | `Boolean`

Output

Port_1 — Output signal
`Simulink.ImageType`

Output signal from the To Simulink Image block. The output is an image of `Simulink.ImageType` data type.

Data Types: `Simulink.ImageType`

Parameters

Output data type — Data type of output signal values
`Inherit: auto` (default) | `Simulink.ImageType(480,640,3)`

Specify the output data type of the image signal.

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the data type attributes. For more information, see “Specify Data Types Using Data Type Assistant” (Simulink).

You can specify any of these options:

- Inherited data type
- Simulink image data type — Use the constructor for `Simulink.ImageType` object and specify the properties to describe the image. By default, the data type uses `Simulink.ImageType(480,640,3)` expression.

- Custom data type — Use a MATLAB expression that represents the rows, columns, and channels of the image respectively.

Programmatic Use
Block Parameter: `OutDataTypeStr`
Type: character vector
Value: `Inherit: auto` | `Simulink.ImageType(480,640,3)` | `<data type expression>`
Default: `Inherit: auto`

Block Characteristics

Data Types	`double` | `single` | `integer` | `Boolean` | `image`
Direct Feedthrough	`yes`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

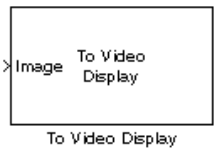
Supports C++ code generation.

See Also

From Simulink Image

To Video Display

Display images or video frames



Libraries:
Computer Vision Toolbox / Sinks

Description

The To Video Display block displays RGB or YCbCr formatted images and video frames. The display created by the block is lightweight and high performance, capable of displaying high definition video at high frame rates.

Ports

Input

Image — Input image
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, or *M*-by-*N*-by-*C*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video. If the input is a 4-D array, the third dimension is the number of color channels, and the fourth dimension is the number of frames in the video.

Dependencies

To enable this port, set the **Input Color Format** parameter to RGB, and set **Image signal** to `One multidimensional signal`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

R — Red color component of image
matrix | array

Red color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Input Color Format** parameter to RGB, and set **Image signal** to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

G — Green color component of image
matrix | array

Green color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Input Color Format** parameter to RGB, and set **Image signal** to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

B — Blue color component of image
matrix | array

Blue color component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Note Inputs to the **R**, **G**, and **B** ports must have the same dimensions and data type.

Dependencies

To enable this port, set the **Input Color Format** parameter to RGB, and set **Image signal** to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Y — Luma component of image
matrix | array

Luma component of the input image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Dependencies

To enable this port, set the **Input Color Format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Cb — Blue-difference chroma component of image
matrix | array

Blue-difference chroma component of the input image or video, specified as an *M*-by-*N/2* matrix or *M*-by-*N/2*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Dependencies

To enable this port, set the **Input Color Format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Cr — Red-difference chroma component of image
matrix | array

Red-difference chroma component of the input image or video, specified as an *M*-by-*N/2* matrix or *M*-by-*N/2*-by-*T* array. If the input is a 3-D array, the third dimension is the number of frames in the video.

Dependencies

To enable this port, set the **Input Color Format** parameter to `YCbCr 4:2:2`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

View

Window size — Display window size
`Normal` (default) | `Full-screen (Esc to exit)` | `True size (1:1)`

Display the video stream in various window sizes:

- `Normal` — Modify the display size at the start of the simulation.
- `Full-screen (Esc to exit)` — Display the video stream in a full-screen window. To exit the full-screen view, press **Esc**.
- `True size (1:1)` — Display the input image at a one-to-one pixel ratio, at the start of the simulation.

Open at Start of Simulation — Open window at start of simulation
`on` (default) | `off`

To set the **Video Display** window to open automatically and display video frames when the simulation begins, select **Open at Start of Simulation**. If you clear this parameter, the block is not included during runtime, and the simulation runs in “Accelerator Mode” (Simulink).

Settings

Input Color Format — Input color format
`RGB` (default) | `YCbCr 4:2:2`

Specify the color format of the input image:

- `RGB` — Specify the input colorspace as RGB. You can select whether to input the image as a single signal or as individual color signals using the **Image signal** parameter.
- `YCbCr 4:2:2`— Specify the input colorspace as YCbCr. You must input the image as separate signals to the three input ports, **Y**, **Cb** and **Cr**.

Image signal — Configure input port
`One multidimensional signal` (default) | `Separate color signals`

Specify whether the block accepts an RGB color input as a single signal or as separate signals for each color channel..

- `One multidimensional signal` — The block accepts RGB input as a single array through one port.

- `Separate color signals` — The block accepts RGB input as three separate matrices or arrays, one for each color channel, to the **R**, **G**, and **B** ports.

Dependencies

To enable this parameter, set the **Input Color Format** parameter to `RGB`.

Block Characteristics

Data Types	`Boolean` | `double` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Note This block supports code generation for platforms that have file I/O available. You cannot use this block with Simulink Desktop Real-Time software, because that product does not support file I/O.

The generated code for this block relies on prebuilt library files. You can run this code outside the MATLAB environment, or redeploy it, but be sure to account for these extra library files when doing so. The `packNGo` function creates a single ZIP file containing all of the pieces required to run or rebuild this code. See `packNGo` for more information.

To know the code generation process of Simulink blocks, see “Simulink Coder”. To run an executable file that was generated from a model containing this block, you may need to add precompiled shared library files to your system path. See “Simulink Shared Library Dependencies” and “Accelerating Simulink Models” for details.

Usage notes and limitations:

- Host computer only. Excludes Simulink Desktop Real-Time code generation.
- The executable generated from this block relies on prebuilt dynamic library files (`.dll` files) included with MATLAB. Use the `packNGo` function to package the code generated from this block and all the relevant files in a compressed ZIP file. Using this ZIP file, you can relocate, unpack, and rebuild your project in another development environment where MATLAB is not installed. For more details, see “Get Started with Code Generation, Deployment, GPU, and OpenCV Support”.

See Also

Blocks

Frame Rate Display | From Multimedia File | To Multimedia File | Video To Workspace | Video Viewer

Top-hat

Perform morphological top-hat filtering on intensity or binary images



Libraries:
Computer Vision Toolbox / Morphological Operations

Description

The Top-hat block performs top-hat filtering on an intensity or binary image using a predefined neighborhood or structuring element. Top-hat filtering is the equivalent of subtracting the result of performing a morphological opening operation on the input image from the input image itself.

To define the structuring element that the block applies to the image, use the **Neighborhood or structuring element** parameter. Specify this element by entering a vector or matrix of `1`s and `0`s or by using the `strel` function.

This block uses two-dimensional flat structuring elements only.

Ports

Input

I — Input image
matrix

Specify an input image as a matrix of intensity values. This port is unnamed unless you set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Nhood — Neighborhood values
vector | matrix

Specify neighborhood values representing a structuring element as a vector or matrix of `1`s and `0`s or by using the `strel` function.

Dependencies

To enable this port, set the **Neighborhood or structuring element source** parameter to `Input port`.

Data Types: `Boolean`

Output

Port_1 — Filtered image
matrix

The top-hat filtered image is returned as a matrix of intensity values. The size and data type of the filtered image are the same as those of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Input image type — Type of input image

`Intensity` (default) | `Binary`

Specify the type of image as one of these options:

- `Intensity` — If the input image is an intensity image, select this option.
- `Binary` — If the input image is a binary image, select this option.

Neighborhood or structuring element source — Source of neighborhood or structuring element

`Specify via dialog` (default) | `Input port`

Specify the source of neighborhood or structuring element as one of these options:

- `Specify via dialog` — Use the **Neighborhood or structuring element** parameter to specify a neighborhood or structuring element.
- `Input port` — Use the **Nhood** input port to specify neighborhood values representing a structuring element.

Neighborhood or structuring element — Neighborhood or structuring element

`strel('square',4)` (default) | vector | matrix

Specify neighborhood or structuring element as a vector or matrix of `1`s and `0`s, a `strel` object or an array of `strel` objects. When you specify this value as an array of `strel` objects, the block applies each object to the entire image in turn.

If the structuring element is decomposable into smaller elements, the block executes at higher speed due to the use of a more efficient algorithm.

Dependencies

To enable this parameter, set the **Neighborhood or structuring element source** parameter, to `Specify via dialog`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`

More About

Top-hat Transform

Image processing tasks like background equalization, feature extraction, and image enhancement can be achieved using the top-hat transform. This process returns an image which contains objects that are:

- Smaller in dimension than the structuring element chosen and
- Brighter than the surroundings

Version History

Introduced before R2006a

References

[1] Soille, Pierre. *Morphological Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-662-05088-0.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Erosion | Dilation | Opening | Closing | Label | Bottom-hat

Functions

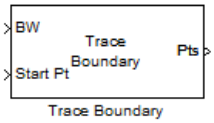
`imerode` | `imdilate` | `imopen` | `imclose` | `imbothat` | `imtophat`

Objects

`strel`

Trace Boundary

Trace object boundaries in binary images



Libraries:
Computer Vision Toolbox / Analysis & Enhancement

Description

The Trace Boundary block traces object boundaries in binary images, where nonzero pixels represent objects and zero valued pixels represent the background.

Ports

Input

BW — Input binary image
matrix

Input binary image, specified as an *M*-by-*N* matrix.

Data Types: `Boolean`

Start Pt — Coordinate of boundary starting point
two-element vector

Coordinate of the boundary starting point, specified as a two-element vector in the form [*X Y*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Pts — Coordinates of boundary points
M-by-2 matrix

Coordinates of the boundary points, returned as an *M*-by-2 matrix where each row represents a coordinate of a boundary pixel in the form [*X Y*]. *M* is the number of traced boundary pixels in the output image. Value of *M* must be less than or equal to the value specified by **Maximum number of boundary pixels** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Connectivity — Type of pixel connectivity

8 (default) | 4

Select these options to specify how the block defines which pixels are connected to each other.

- 8 — Connect adjacent pixels in the horizontal, vertical, and diagonal directions.
- 4 — Connect adjacent pixels in only the horizontal and vertical directions.

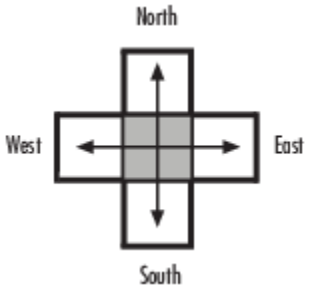
For more information, see “Connectivity” on page 1-0 .

Initial search direction — Type of initial search direction

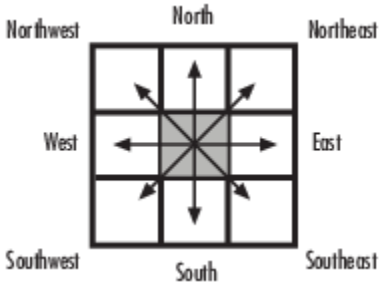
`North` (default) | `East` | `South` | `West` | `Northeast` | ...

Specify the type of initial search direction to find the boundary pixel connected to the starting pixel specified at **Start Pt** input port. These options are dependent on the specified **Connectivity** parameter.

For **Connectivity** parameter set as 4, the block enables `North`, `East`, `South`, and `West` search directions. This figure illustrates the four possible initial search directions with respect to the starting pixel.



For **Connectivity** parameter set as 8, the block enables additional `Northeast`, `Southeast`, `Southwest`, and `Northwest` search directions. This figure illustrates the eight possible initial search directions with respect to the starting pixel.



Trace direction — Boundary trace direction

`Clockwise` (default) | `Counterclockwise`

Select the direction to trace the boundary as `Clockwise` or `Counterclockwise`.

Maximum number of boundary pixels — Maximum number of boundary pixels

500 (default) | scalar

Specify the maximum number of boundary pixels for the starting point as a scalar. The number of rows at **Pts** output port must be less than or equal to this value.

Block Characteristics

Data Types	`Boolean`
Multidimensional Signals	`no`
Variable-Size Signals	`no`

Version History

Introduced in R2011b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

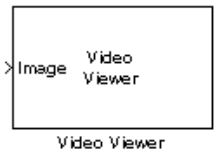
Edge Detection | Label

Functions

`bwboundaries`

Video Viewer

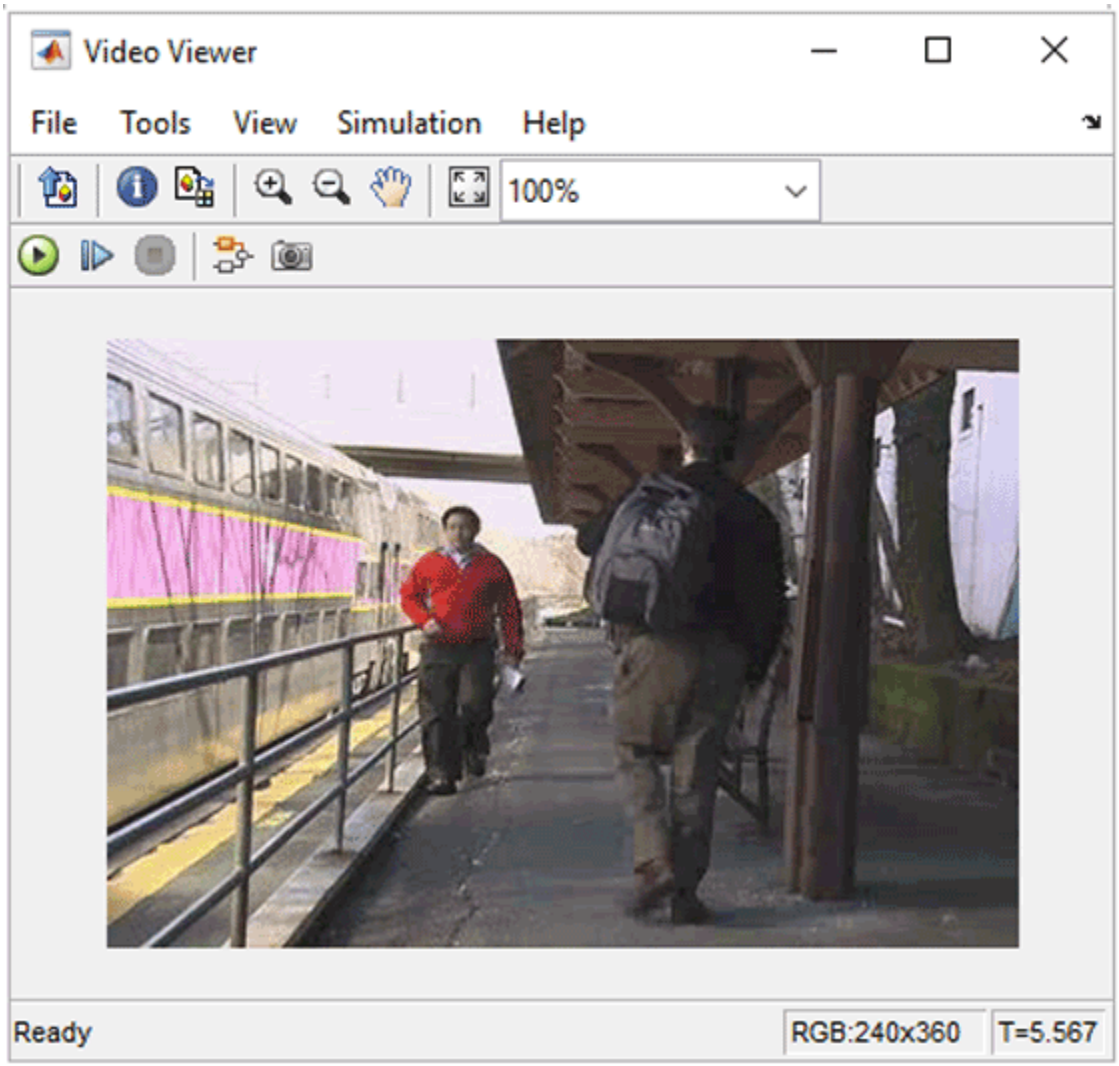
Display images or video frames



Libraries:
Computer Vision Toolbox / Sinks

Description

The Video Viewer block displays images or video frames. This block provides simulation controls to play, pause, and step while streaming a video or image sequence. This block also provides tools to analyze the pixels of a region of interest.



Features of Video Viewer

- Playback control — Control video playback using the Run, Step Forward, and Stop buttons.
- Export to **Image Viewer** — To open the displayed image in a separate window using the **Image Viewer** app, select **File > Export to Image Viewer**. Alternatively, select the Export to Image Viewer button in the toolbar. For more information, see “Get Started with Image Viewer App”.
- Pixel region analysis— To open the displayed image in a separate window using the Pixel Region tool, select **Tools > Pixel Region**. Alternatively, select the Inspect pixel values button in the toolbar. For more information about Pixel Region tool, see `impixelregion`.

Ports

Input

Image — Input image or video
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Dependencies

To enable this port, select the **One Multi-Dimensional Signal** parameter. This parameter is enabled by default.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `Simulink.ImageType`

R — Red color component of image or video
matrix | array

Red color component of the image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. *T* is the number of frames in a video or image sequence.

Note Inputs to the **R**, **G**, and **B** ports must all have the same dimensions and data type.

Dependencies

To enable this port, select the **Separate Color Signals** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

G — Green color component of image or video
matrix | array

Green color component of the image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. *T* is the number of frames in a video or image sequence.

Note Inputs to the **R**, **G**, and **B** ports must all have the same dimensions and data type.

Dependencies

To enable this port, select the **Separate Color Signals** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

B — Blue color component of image or video
matrix | array

Blue color component of the image or video, specified as an *M*-by-*N* matrix or *M*-by-*N*-by-*T* array. *T* is the number of frames in a video or image sequence.

Note Inputs to the **R**, **G**, and **B** ports must all have the same dimensions and data type.

Dependencies

To enable this port, select the **Separate Color Signals** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

File

To access these parameters, select the **File** menu in the Video Viewer window

Open at Start of Simulation — Open video viewer window at start of simulation
`on` (default) | `off`

Select this parameter to automatically open the Video Viewer window at the start of the simulation. Clear this parameter to prevent the Video Viewer window from opening at the start of the simulation.

Separate Color Signals — Enable separate R, G, and B color channels as input
`off` (default) | `on`

Select this parameter to enable the **R**, **G**, and **B** input ports for separate color channels. If you select this parameter, the **One Multi-Dimensional Signal** parameter clears automatically.

One Multi-Dimensional Signal — Enable one multi-dimensional signal as input
`on` (default) | `off`

Select this parameter to enable the **Image** input port for one multi-dimensional signal. If you select this parameter, the **Separate Color Signals** parameter clears automatically.

File > Configuration > Edit > Core

The Video Viewer **Configuration** parameters enable you to change the behavior and appearance of the graphic user interface (GUI).

- To open the Configuration dialog box and edit parameters, select **File** > **Configuration** > **Edit**.

The **Core** tab in the Configuration dialog box lists the general settings of the GUI.

- In the **Core** tab, select `General UI` item and click the **Properties** button to open the General UI Properties dialog box.

Display the full path — Display full path of source data in the title bar
`off` (default) | `on`

Select this parameter to display the model name and full path to the source data in the title bar of the Video Viewer window. If you clear this parameter, the window displays a shortened name.

File > Configuration > Edit > Tools

The Video Viewer **Configuration** parameters enable you to change the behavior and appearance of the graphic user interface (GUI).

- To open the Configuration dialog box and edit parameters, select **File** > **Configuration** > **Edit**.

The **Tools** tab in the Configuration dialog box lists the tools associated with the Video Viewer block.

- In the **Tools** tab, select the `Image Viewer` item and click the **Properties** button to open the Image Viewer Properties dialog box.

Open new Image Viewer window for each export — Open new Image Viewer window for each exported video frame
`on` (default) | `off`

Select this parameter to open a new **Image Viewer** window for each exported video frame or image in an image sequence.

Tools > Colormap

The **Colormap** parameters enable you to display an intensity image in various colormap formats and control the output intensity range. To access these parameters, select **Tools** > **Colormap** or press **C**.

Note The **Colormap** parameters are available for intensity images only.

Colormap — Set colormap

`gray(256)` (default) | `parula(256)` | `jet(256)` | `hot(256)` | `bone(256)` | `cool(256)` | `copper(256)`

This parameter specifies the colormap in which to display the input intensity image. This block supports the `gray(256)`, `parula(256)`, `jet(256)`, `hot(256)`, `bone(256)`, and `copper(256)` colormaps.

Specify range of displayed pixel values [-Inf to Inf] — Specify range of intensity value to display
`off` (default) | `on`

Select this parameter to enable the **Min** and **Max** parameters, through which you can specify the display range of pixel intensities.

Min — Minimum intensity value
`0` (default) | numeric scalar

If a pixel has an intensity value less than or equal to the value of this parameter, **Video Viewer** displays it as the first color in the selected colormap.

Note When specifying a value for **Min**, remember that the data type of the input image or video determines the range of intensity values representing that image or video.

Dependencies

To enable this parameter, select the **Specify range of displayed pixel values [-Inf to Inf]** parameter.

Max — Maximum intensity value
255 (default) | numeric scalar

If a pixel has an intensity value higher than or equal to the value of this parameter, **Video Viewer** displays it as the last color in the selected colormap.

Note When specifying a value for **Max**, remember that the data type of the input image or video determines the range of intensity values representing that image or video.

Dependencies

To enable this parameter, select the **Specify range of displayed pixel values [-Inf to Inf]** parameter.

Simulation

To access this parameter, select the **Simulation** menu in the Video Viewer window.

Drop Frames to Improve Performance — Drop frame rate of input video signal
`on` (default) | `off`

Select this parameter to reduce the frame rate of the input video signal to improve performance of the simulation. If you unselect this parameter, the block displays all the frames in the input video signal during simulation.

Block Characteristics

Data Types	`Boolean` | `double` | `integer` | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`
Zero-Crossing Detection	`no`

Version History

Introduced before R2006a

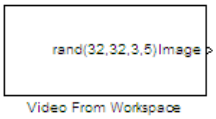
See Also

Blocks

From Multimedia File | To Multimedia File | To Video Display | Video To Workspace | Video From Workspace | Image From Workspace | Image From File

Video From Workspace

Import video from MATLAB workspace



Libraries:
Computer Vision Toolbox / Sources

Description

The Video From Workspace block imports a video from the MATLAB workspace. The output of the block depends on the format of the video in the workspace.

- If the video is represented by an *M*-by-*N*-by-*T* array, the block outputs a grayscale video. *M* and *N* are the number of rows and columns in each video frame, and *T* is the number of frames in the video.
- If the video is represented by an *M*-by-*N*-by-*C*-by-*T* array, the block outputs a color video. *M* and *N* are the number of rows and columns in each video frame, *C* is the number of color channels, and *T* is the number of frames in the video.

Note

- If you generate code from a model that contains this block, it takes long to compile the code as the Simulink Coder puts all of the video data into the `.c` file. To shorten code generation time, convert your video data into a format supported by the From Multimedia File or Read Binary File block.
- The Video From Workspace block also supports `fi` objects. For more information on `fi` objects, see `fi`.

Ports

Output

Image — Output image or video
M-by-*N*-by-*T* array | *M*-by-*N*-by-*C*-by-*T* array

Output image or video, returned as an *M*-by-*N*-by-*T* array or *M*-by-*N*-by-*C*-by-*T* array. If the output is a 3-D array, the third dimension is the number of frames in the video. If the output is a 4-D array, the third dimension is the number of color channels, and the fourth dimension is the number of frames in the video.

Note If an RGB image of size, *M*-by-*N*-by-3 array is imported from the MATLAB workspace, the block considers the third dimension to be the number of frames and processes the image as a grayscale video with 3 identical frames.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color component of input video
3-D array

Red color component of the input video, returned as a 3-D array. The third dimension is the number of frames in the input video.

Note The port label can be changed by using the **Output port labels** parameter.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color component of input video
3-D array

Green color component of the input video, returned as a 3-D array. The third dimension is the number of frames in the input video.

Note The port label can be changed by using the **Output port labels** parameter.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color component of input video
3-D array

Blue color component of the input video, returned as a 3-D array. The third dimension is the number of frames in the input video.

Note The port label can be changed by using the **Output port labels** parameter.

Dependencies

To enable this port, set the **Image signal** parameter to `Separate color signals`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Signal — Video to read
`rand(32,32,3,5)` (default) | *M*-by-*N*-by-*T* array | *M*-by-*N*-by-*C*-by-*T* array | MATLAB workspace variable

Specify the video to read from the workspace. The value for this parameter can be a 3-D numeric array, 4-D numeric array, or a MATLAB workspace variable.

To import an AVI file from the MATLAB workspace, use this process.

1. Read the file by using the `VideoReader` function.

   ```
   mov = VideoReader('filename.avi')
   ```

2. Specify the workspace variable, in this case *mov* as the value for the **Signal** parameter. If the AVI file has an associated colormap, it must satisfy these conditions.

   - The colormap must be empty or have 256 values.
   - The data must represent a grayscale image.
   - The data type of the pixel values must be `uint8`.

Sample time — Sample period of the output video
1 (default) | `-1` | positive scalar

This parameter specifies the sample period of the output video as a positive scalar or `-1`. When set to `-1`, the block inherits the sample time from the simulink model.

Form output after final value by — Form of block output after final video frame
`Setting to zero` (default) | `Holding final value` | `Cyclic repetition`

Specify the block output for the duration of the simulation after the block generates the final video frame.

- `Setting to zero` — Generate zero-valued outputs.
- `Holding final value` — Repeat the final frame.
- `Cyclic repetition` — Repeat the video from the beginning.

Image signal — Output mode of video signal
`One multidimensional signal` (default) | `Separate color signals`

Specify the format in which the block returns a color video.

- `One multidimensional signal` — Return all color channels of the video in a single 4-D array. The third dimension is the number of color channels in the video.
- `Separate color signals` — Return each color channel of the video as a separate 3-D array. This setting enables a number of output ports on the block equal to the number of color channels in the video. Each port returns a single color channel.

Output port labels — Output port names
`R|G|B` (default) | text labels separated by vertical bars

Name the output ports. Use the vertical bar character, |, as a delimiter.

Example:: Red|Green|Blue

Dependencies

To enable this parameter, set the **Image signal** parameter to `Separate color signals`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Video To Workspace | Image From Workspace | Video Viewer

Video To Workspace

Export image or video to MATLAB workspace



Libraries:
Computer Vision Toolbox / Sinks

Description

The Video To Workspace block exports an image or video to the MATLAB workspace. It accepts images or videos as a matrix, 3-D array, 4-D array and converts them to a 3-D or 4-D array. The input size determines the output size.

- *M*-by-*N*, the block exports it as an *M*-by-*N*-by-2 array. *M* and *N* are the number of rows and columns, respectively, in an image frame and 2 is the number of frames in the image.
- *M*-by-*N*-by-3, the block exports it as a *M*-by-*N*-by-3-by-2 array. *M* and *N* are the number of rows and columns in a image or video frame, 3 is the number of color channels, and 2 is the number of frames in the video.
- *M*-by-*N*-by-3-by-*T*, the block exports it as a *M*-by-*N*-by-3-by-*T* array. *M* and *N* are the number of rows and columns in a video frame, 3 represents the number of color channels, and *T* is the number of frames in the video.

Note

- Simulink Coder does not generate code for this block.
- The block supports grayscale and color images, which are exported to the MATLAB workspace with an extra dimension that represents the number of frames. An image is always exported with a frame value of 2.

Ports

Input

Image — Input video
matrix | 3-D array | 4-D array

The input image or video, specified as a matrix, 3-D array, or 4-D array. The type of signal determines the dimensions of the input. Inputs are defined in terms of *M*, *N*, *C*, and *T*. *M* and *N* are the number of rows and columns in the signal respectively. *C* is the number of color channels in the signal, and *T* is the number of frames.

Input - Output

Signal Type	Input Dimensions	Output Dimensions
Grayscale image	*M*-by-*N*	*M*-by-*N*-by-2
Color image	*M*-by-*N*-by-3	*M*-by-*N*-by-3-by-2
Grayscale video	*M*-by-*N*-by-*T*	*M*-by-*N*-by-*T*
Color video	*M*-by-*N*-by-3-by-*T*	*M*-by-*N*-by-3-by-*T*

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

R — Red color channel
scalar | vector | matrix

Red color channel of the input image or video, specified as a scalar, vector, or matrix.

Note The port label can be changed using the `Input port labels` parameter.

Dependencies

To enable this port, set the **Number of inputs** parameter to a value greater than or equal to 2.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

G — Green color channel
scalar | vector | matrix

Green color channel of the input image or video, specified as a scalar, vector, or matrix.

Note The port label can be changed using the `Input port labels` parameter.

Dependencies

To enable this port, set the **Number of inputs** parameter to a value greater than or equal to 2.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

B — Blue color channel
scalar | vector | matrix

Blue color component of the input image or video, specified as a scalar, vector, or matrix.

Note The port label can be changed using the `Input port labels` parameter.

Dependencies

To enable this port, set the **Number of inputs** parameter to a value equal to 3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Variable name — MATLAB workspace variable name
`vout` (default) | valid MATLAB variable name

Specify the MATLAB workspace variable name for the exported image or video.

Number of inputs — Number of inputs ports
`1` (default) | positive integer

Specify the number of inputs to the block. To separate the color channels in a RBG image or video, specify the number of channels. If it is a grayscale video, enter 1. If it is a color (R, G, B) video, enter 3.

Limit data points to last — Maximum number of frames to export
`10` (default) | positive integer

Specify the maximum number of frames to export to the MATLAB workspace.

Decimation — Downsample video signal
`1` (default) | positive integer

Enter the decimation factor by which to downsample the image or video.

Log fixed-point data as a fi object — Create fixed-point numeric object
`off` (default) | `on`

Select **Log fixed-point data as a fi object** to create a fi object in the MATLAB workspace when the input signal has a fixed point data type. For more information on `fi` objects, see `fi`.

Input port labels — Input port names
`R|G|B` (default)

Name the input ports using the vertical bar character as a delimiter.

Dependencies

To enable this parameter, set the **Number of inputs** parameter to a value greater than `1`.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

Version History

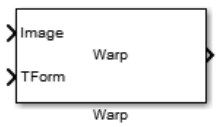
Introduced before R2006a

See Also

Blocks
Image From Workspace | Video From Workspace | Write Binary File

Warp

Apply projective or affine transformation



Libraries:
Computer Vision Toolbox / Geometric Transformations

Description

The Warp block transforms an image by applying projective on page 1-414 or affine on page 1-414 transformation. You can transform the entire image or a region of the image by defining a rectangular region of interest (ROI).

Ports

Input

Image — Input image
matrix | 3-D numeric array

Input image, specified as one of these values:

- matrix — For intensity images of size *M*-by-*N*.
- 3-D numeric array — For true color images of size *M*-by-*N*-by-3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

TForm — Transformation matrix
3-by-2 matrix | 3-by-3 matrix

Transformation matrix, specified as any one of these values:

- 3-by-2 matrix — For affine transformation.
- 3-by-3 matrix — For projective transformation.

For more information about the transformations, see “Algorithms” on page 1-414.

Dependencies

To enable this input port, set the **Transformation matrix source** parameter value to `Input port`.

Data Types: `double` | `single`

ROI — Region of interest
4-element vector

Region of interest, specified as a 4-element vector of form [x_s y_s *width height*]. x_s and y_s are the x and y coordinates of the top left corner of the ROI, respectively.

If you specify the `ROI` input, the `Warp` block applies transformation only to the specified region and returns the transformed region at the output.

Dependencies

To enable this input port, select the **Enable ROI input port** parameter.

Output

Image — Transformed image
matrix | 3-D numeric array

Transformed image, returned as one of these values:

- matrix — If input is an intensity image of size *P*-by-*Q1*.
- 3-D numeric array — If input is a true color image of size *P*-by-*Q*-by-3.

The data type of the output transformed image is same as that of the input image. The size of the output transformed image is either same as the input image or equal to the value set for the **Output image position vector [x y width height]** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Err_roi — Indicator for transformed ROI outside image region
`0` | `1`

Indicator for transformed ROI outside image region, returned as `0` or `1`.

Data Types: `Boolean`

Parameters

Transformation matrix for source — Source for passing transformation matrix

`Input port` (default) | `Custom`

Source for passing transformation matrix, specified as either `Input port` or `Custom`. If you select `Custom`, you can enter the transformation matrix coefficients by using the **Transformation matrix** parameter that appears with this selection.

Transformation matrix — Value for transformation matrix

3-by-2 matrix | 3-by-3 matrix

Value for transformation matrix, specified as a 3-by-2 matrix for affine transformation or 3-by-3 matrix for projective transformation.

Dependencies

To enable this parameter, set the **Transformation matrix for source** parameter value to `Custom`.

Interpolation method — Method for interpolating transformed pixel values

`Bilinear` (default) | `Nearest neighbor` | `Bicubic`

Method for interpolating transformed pixel values, specified as `Nearest neighbor`, `Bilinear`, or `Bicubic`.

If you select `Nearest neighbor`, the block uses the value of an nearby pixel for the new pixel value. If you select `Bilinear`, the new pixel value is the weighted average of the four nearest pixel values. If you select `Bicubic`, the new pixel value is the weighted average of the sixteen nearest pixel values.

The number of pixels the block considers affects the complexity of the computation. Therefore, the `Nearest neighbor` interpolation is the most computationally efficient. However, because the accuracy of the method is proportional to the number of pixels considered, the `Bicubic` method is the most accurate. For information about the interpolation methods, see the “More About” on page 1-412 section.

Background fill value — Intensity value for background pixels in the transformed image

`0` (default) | scalar | 3-element vector

Intensity value for background pixels in the transformed image, specified as one of these values:

- scalar — If input image is a gray scale image.
- 3-element vector — If input image is a true color image. The vector is of the form [*r g b*] specifying the red (*r*), green (*g*), and blue (*b*) color channel values for the background pixels.

The default fill value is `0` and sets the background color to black.

Output image position source — Source for passing output image size

`Same as input image` (default) | `Custom`

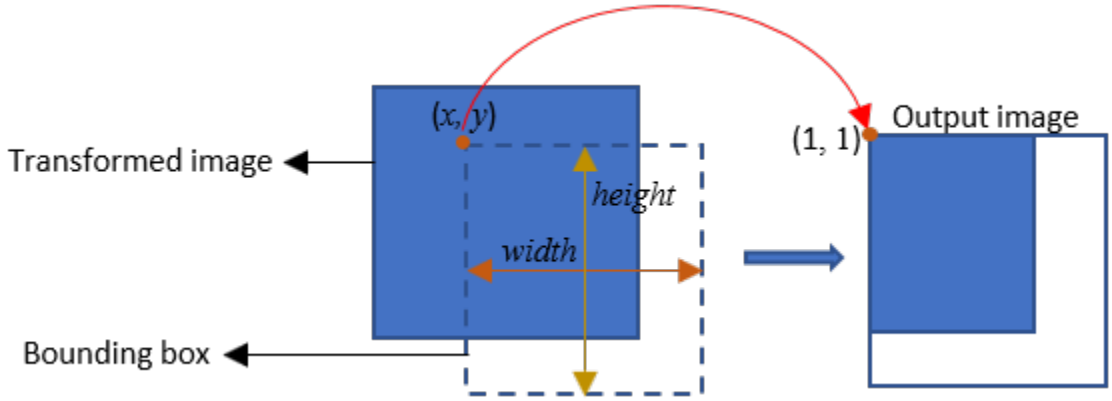
Source for passing a value for the output image size, specified as either `Same as input image` or `Custom`.

- If you select `Same as input image`, the output transformed image is of same size as that of the input image.
- If you select `Custom`, you must specify a bounding box to output only the image region that lies within the bounding box. This selection enables the **Output image position vector [x y width height]** parameter that you can use for specifying the bounding box value.

Output image position vector [x y width height] — Size of the output image

[1 1 512 512] (default) | 4-element vector

Size of the output image, specified as a four element vector of form [*x y width height*]. When you specify this parameter, the `Warp` block creates a bounding box of specified width and height values. The size of the output image is set to the size of the bounding box and will contain the transformed image region that lies within the bounding box. If the size of the output image is greater than the size of the transformed image region within the bounding box, the intensity value of the extraneous pixels in the output image are set to the value specified for **Background fill value** parameter.



x and *y* are the spatial coordinates that define top-left corner position of the bounding box with respect to the input image.

Dependencies

To enable this parameter, set the **Output image position source** parameter value to `Custom`.

Enable ROI input port — Input ROI

off (default) | on

Select this parameter to enable the `ROI` input port and specify the ROI to be transformed.

Enable output port indicating if any part of ROI is outside input image — Output if transformed ROI is outside the specified image size

off (default) | on

Select this parameter to enable the `Err_roi` output port.

Dependencies

To enable this parameter, select the **Enable ROI input port** parameter.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. For more information, see “Choosing a Simulation Mode” (Simulink).

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`yes`

More About

Nearest Neighbor Interpolation

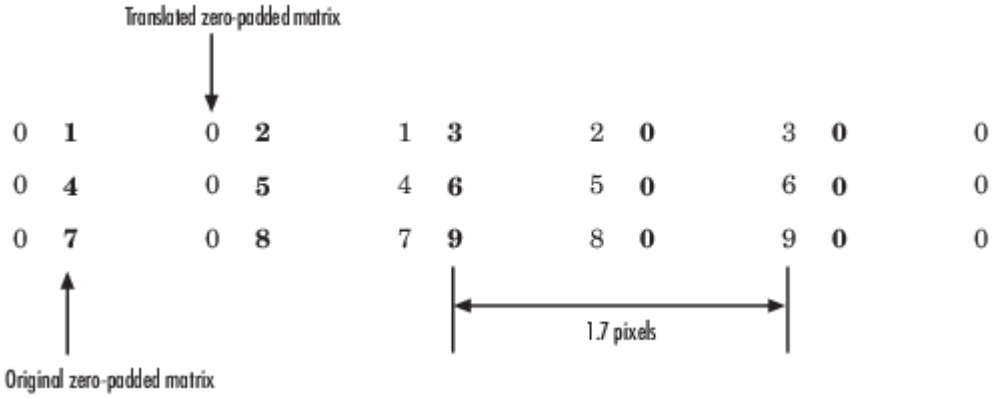
For nearest neighbor interpolation, the block uses nearby, translated, pixel values for the output pixel values.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image 1.7 pixels in the positive horizontal direction using nearest neighbor interpolation. These steps illustrate the nearest neighbor interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 1.7 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the translated value nearest to it resulting in this matrix:

0 0 1 2 3
0 0 4 5 6
0 0 7 8 9

Note Despite specifying a value of 1.7 pixels to translate the image, this method translates the image by 2 pixels. Nearest neighbor interpolation is computationally efficient, but not as accurate as the bilinear and bicubic interpolation methods.

Bilinear Interpolation

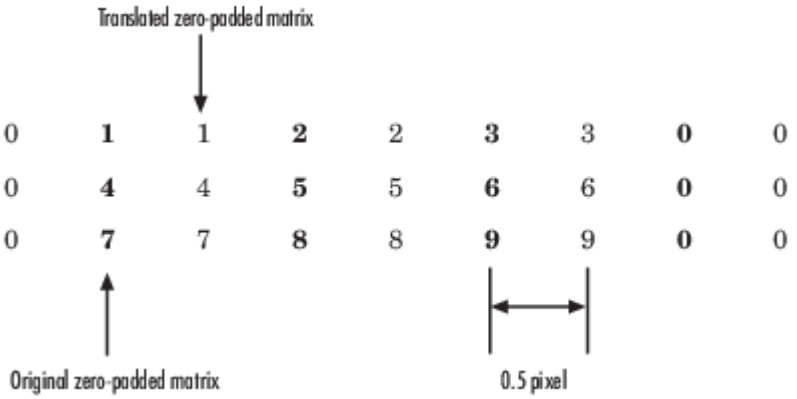
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bilinear interpolation. These steps illustrate the bilinear interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the translated pixel values on either side of the input pixel. The result is this matrix which has one column more than the input matrix.

0.5 1.5 2.5 1.5
2 4.5 5.5 3
3.5 7.5 8.5 4.5

Bicubic Interpolation

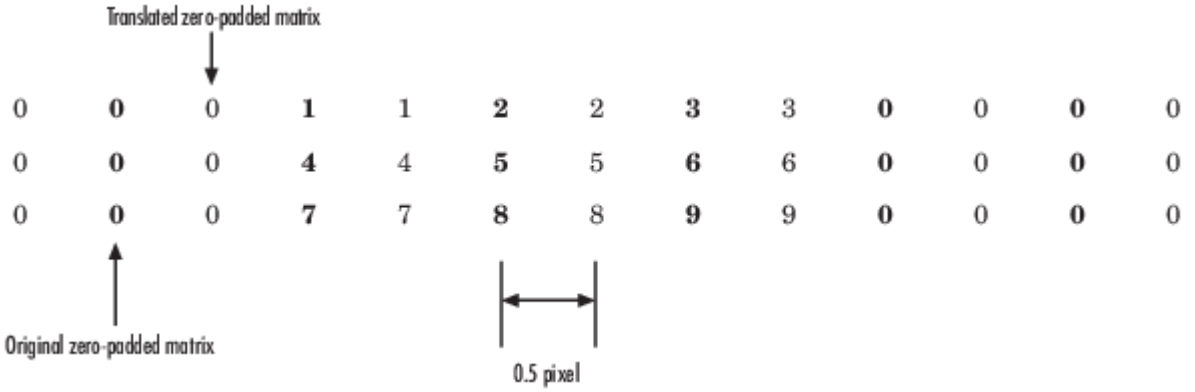
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bicubic interpolation. These steps illustrate the bicubic interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it by 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the two translated values on either side. The result is this matrix which has one column more than the input matrix:

0.375	1.5	3	1.625
1.875	4.875	6.375	3.125
3.375	8.25	9.75	4.625

Algorithms

Affine Transformation

The affine transformation matrix is a 3-by-2 matrix of form

$$H = \begin{bmatrix} h_1 & h_4 \\ h_2 & h_5 \\ h_3 & h_6 \end{bmatrix}$$

where h_1, h_2,...,h_6 are transformation coefficients.

The value of the pixel located at $[x, y]$ in the input image determines the value of the pixel located at $[\widehat{x}, \widehat{y}]$ in the output transformed image. The relationship between the input and output point locations is defined by:

$$\begin{cases} \widehat{x} = xh_1 + yh_2 + h_3 \\ \widehat{y} = xh_4 + yh_5 + h_6 \end{cases}$$

Projective Transformation

The projective transformation matrix is a 3-by-3 matrix of form

$$H = \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & h_9 \end{bmatrix}$$

where h_1, h_2,...,h_9 are transformation coefficients.

The value of the pixel located at $[x, y]$ in the input image determines the value of the pixel located at $[\widehat{x}, \widehat{y}]$ in the output transformed image.

The relationship between the input and output point locations is defined by:

$$\begin{cases} \widehat{x} = \dfrac{xh_1 + yh_2 + h_3}{xh_7 + yh_8 + h_9} \\ \widehat{y} = \dfrac{xh_4 + yh_5 + h_6}{xh_7 + yh_8 + h_9} \end{cases}$$

The values of pixels in the output point locations are calculated using any of these interpolation methods: Nearest neighbor, bilinear, and bicubic interpolation.

Version History

Introduced in R2015b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Functions

`imwarp`

Blocks

Estimate Geometric Transformation | Resize | Rotate | Shear | Translate

ToOpenCV

Converts Simulink data types to OpenCV data types



Libraries:
Computer Vision Toolbox Interface for OpenCV in Simulink

Description

The ToOpenCV block converts the Simulink data types to Simulink OpenCV data types.

Ports

Inputs

Port_1 — Input signal
scalar | vector | matrix | simulink bus | simulink image

Input signal to the ToOpenCV block. The input can be of different data types. Each of the input data types determines the output data type of the block.

Data Types: `uint8` | `uint16` | `int8` | `int16` | `int32` | `single` | `double` | `Simulink.ImageType` | `CV_DMATCH_BUS` | `CV_RECT_BUS` | `CV_ROTATEDRECT_BUS` | `CV_TERMCRITERIA_BUS` | `CV_SIZE_BUS` | `CV_RANGE_BUS` | `CV_POINT2I_BUS` | `CV_POINT2F_BUS` | `CV_POINT2D_BUS` | `CV_POINT3I_BUS` | `CV_POINT3F_BUS` | `CV_POINT3D_BUS`

Output

Port_1 — Output signal
scalar | vector | matrix | simulink bus | simulink image

Output signal from the ToOpenCV block. The output signal supports OpenCV types. For more information, see Mapping Input and Output Data Types on page 1-417.

Data Types: `OpenCV types`

Parameters

DataType — Data type of the Input signal of ToOpenCV block
`input data type` (default)

Specifies the input data type of the ToOpenCV block. The table on page 1-417 lists the one-to-one mapping of the input versus output data types of the ToOpenCV block.

Programmatic Use

Block Parameter: `DataType`
Type: character vector
Value: input data type
Default: `'Point2i'`

Block Characteristics

Data Types	
Direct Feedthrough	`Yes`
Multidimensional Signals	`Yes`
Variable-Size Signals	`No`

More About

Mapping of Input and Output Data Types

The table lists the one-to-one mapping of the input versus output data types of the ToOpenCV block.

DataType Parameter	ToOpenCV Block Output Signal Data Type	ToOpenCV Block Input Signal Data Type
Mat or Mat-Image	cv_Mat	uint8 array
		uint16 array
		int8 array
		int16 array
		int32 array
		single array
		double array
		`Simulink.ImageType`
`Simulink.ImageType`	cv_Mat	`Simulink.ImageType`
DMatch	cv_DMatch	CV_DMATCH_BUS
vector<DMatch>	vector_cv_DMatch	CV_DMATCH_BUS (1d array)
vector<vector<DMatch>>	vector_vector_cv_DMatch	CV_DMATCH_BUS (2d array)
Rect	cv_Rect	CV_RECT_BUS
vector<Rect>	vector_cv_Rect	CV_RECT_BUS (1d array)
RotatedRect	cv_RotatedRect	CV_ROTATEDRECT_BUS
vector<RotatedRect>	vector_cv_RotatedRect	CV_ROTATEDRECT_BUS (1d array)
TermCriteria	cv_TermCriteria	CV_TERMCRITERIA_BUS
Scalar	cv_Scalar	double array
Size	cv_Size	CV_SIZE_BUS
Range	cv_Range	CV_RANGE_BUS
Point	cv_Point	CV_POINT2I_BUS
Point2f	cv_Point2f	CV_POINT2F_BUS
Point2d	cv_Point2d	CV_POINT2D_BUS
Point3i	cv_Point3i	CV_POINT3I_BUS

DataType Parameter	ToOpenCV Block Output Signal Data Type	ToOpenCV Block Input Signal Data Type
Point3f	cv_Point3f	CV_POINT3F_BUS
Point3d	cv_Point3d	CV_POINT3D_BUS
vector<Point>	vector_cv_Point	CV_POINT2I_BUS (1d array)
vector<Point2f>	vector_cv_Point2f	CV_POINT2F_BUS (1d array)
vector<Point2d>	vector_cv_Point2d	CV_POINT2D_BUS (1d array)
vector<Point3i>	vector_cv_Point3i	CV_POINT3I_BUS (1d array)
vector<Point3f>	vector_cv_Point3f	CV_POINT3F_BUS (1d array)
vector<Point3d>	vector_cv_Point3d	CV_POINT3D_BUS (1d array)
vector<vector<Point>>	vector_vector_cv_Point	CV_POINT2I_BUS (2d array)
vector<vector<Point2f>>	vector_vector_cv_Point2f	CV_POINT2F_BUS (2d array)
vector<vector<Point2d>>	vector_vector_cv_Point2d	CV_POINT2D_BUS (2d array)
vector<vector<Point3i>>	vector_vector_cv_Point3i	CV_POINT3I_BUS (2d array)
vector<vector<Point3f>>	vector_vector_cv_Point3f	CV_POINT3F_BUS (2d array)
vector<vector<Point3d>>	vector_vector_cv_Point3d	CV_POINT3D_BUS (2d array)

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Supports C++ code generation.

See Also

FromOpenCV | Matrix To Image | Image To Matrix

Topics

“Install and Use Computer Vision Toolbox Interface for OpenCV in Simulink”

FromOpenCV

Converts OpenCV data types to Simulink data types



Libraries:
Computer Vision Toolbox Interface for OpenCV in Simulink

Description

The FromOpenCV block converts the Simulink OpenCV data types to Simulink data types.

Ports

Inputs

Port_1 — First input signal
scalar | vector | matrix | simulink bus | simulink image

Input signal to the FromOpenCV block. The input can be of different data types. Each of the input data types determines the output data type of the block. For more information, see Mapping Input and Output Data Types. on page 1-420

Data Types: `OpenCV types`

Output

Port_1 — Output signal
scalar | vector | matrix | simulink bus | simulink image

Output signal from the FromOpenCV block.

Data Types: `Converted Simulink types`

Parameters

DataType — Data type of the Input signal of FromOpenCV block
`input data type` (default)

Specifies the input data type of the FromOpenCV block. The table on page 1-420 lists the one-to-one mapping of the input versus output data types of the FromOpenCV block.

Programmatic Use

Block Parameter: `DataType`
Type: character vector
Value: input data type
Default: `'Point2i'`

Block Characteristics

Data Types	
Direct Feedthrough	`Yes`
Multidimensional Signals	`Yes`
Variable-Size Signals	`No`

More About

Mapping of Input and Output Data Types

The table lists the one-to-one mapping of the input versus output data types of the FromOpenCV block.

DataType Parameter	FromOpenCV Block Input Signal Data Type	FromOpenCV Block Output Signal Data Type
Mat or Mat-Image	cv_Mat	uint8 array
		uint16 array
		int8 array
		int16 array
		int32 array
		single array
		double array
		`Simulink.ImageType`
`Simulink.ImageType`	cv_Mat	`Simulink.ImageType`
DMatch	cv_DMatch	CV_DMATCH_BUS
vector<DMatch>	vector_cv_DMatch	CV_DMATCH_BUS (1d array)
vector<vector<DMatch>>	vector_vector_cv_DMatch	CV_DMATCH_BUS (2d array)
Rect	cv_Rect	CV_RECT_BUS
vector<Rect>	vector_cv_Rect	CV_RECT_BUS (1d array)
RotatedRect	cv_RotatedRect	CV_ROTATEDRECT_BUS
vector<RotatedRect>	vector_cv_RotatedRect	CV_ROTATEDRECT_BUS (1d array)
TermCriteria	cv_TermCriteria	CV_TERMCRITERIA_BUS
Scalar	cv_Scalar	double array
Size	cv_Size	CV_SIZE_BUS
Range	cv_Range	CV_RANGE_BUS
Point	cv_Point	CV_POINT2I_BUS
Point2f	cv_Point2f	CV_POINT2F_BUS
Point2d	cv_Point2d	CV_POINT2D_BUS

DataType Parameter	FromOpenCV Block Input Signal Data Type	FromOpenCV Block Output Signal Data Type
Point3i	cv_Point3i	CV_POINT3I_BUS
Point3f	cv_Point3f	CV_POINT3F_BUS
Point3d	cv_Point3d	CV_POINT3D_BUS
vector<Point>	vector_cv_Point	CV_POINT2I_BUS (1d array)
vector<Point2f>	vector_cv_Point2f	CV_POINT2F_BUS (1d array)
vector<Point2d>	vector_cv_Point2d	CV_POINT2D_BUS (1d array)
vector<Point3i>	vector_cv_Point3i	CV_POINT3I_BUS (1d array)
vector<Point3f>	vector_cv_Point3f	CV_POINT3F_BUS (1d array)
vector<Point3d>	vector_cv_Point3d	CV_POINT3D_BUS (1d array)
vector<vector<Point>>	vector_vector_cv_Point	CV_POINT2I_BUS (2d array)
vector<vector<Point2f>>	vector_vector_cv_Point2f	CV_POINT2F_BUS (2d array)
vector<vector<Point2d>>	vector_vector_cv_Point2d	CV_POINT2D_BUS (2d array)
vector<vector<Point3i>>	vector_vector_cv_Point3i	CV_POINT3I_BUS (2d array)
vector<vector<Point3f>>	vector_vector_cv_Point3f	CV_POINT3F_BUS (2d array)
vector<vector<Point3d>>	vector_vector_cv_Point3d	CV_POINT3D_BUS (2d array)

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Supports C++ code generation.

See Also

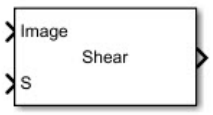
ToOpenCV | Matrix To Image | Image To Matrix

Topics

“Install and Use Computer Vision Toolbox Interface for OpenCV in Simulink”

Shear

Shift rows or columns of an image or a video frame by linearly varying offset



Libraries:
Computer Vision Toolbox / Geometric Transformations

Description

The Shear block transforms an input image or a video frame by shifting the rows or columns. The block transforms the input image using either horizontal or vertical shear.

- Horizontal shear linearly shifts the rows in the image by gradually increasing the offset from left to right or right to left.
- Vertical shear linearly shifts the columns in the image by gradually increasing the offset from top to bottom or bottom to top.

Ports

Input

Image — Input image or video
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, *M*-by-*N*-by-*C* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

S — Shear value
vector

Shear value, specified as a two-element vector of the form, `[first last]`.

- To horizontally shear the image, the block shifts the pixels of the first and last rows using the `first` and `last` values, respectively, specified to the **S** port.
- To vertically shear the image, the block shifts the pixels of the first and last columns using the `first` and `last` values, respectively,specified to the **S** port.

Dependencies

To enable this port, set the **Shear values source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Output — Transformed image or video
matrix | array

Transformed image or video, returned as a *P*-by-*Q* matrix, *P*-by-*Q*-by-*T* array, *P*-by-*Q*-by-*C* array, or *P*-by-*Q*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

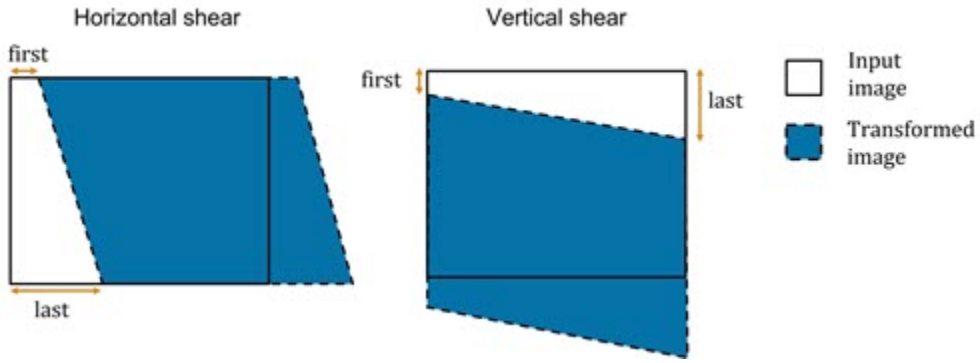
Parameters

Shear direction — Direction of transformation

`Horizontal` (default) | `Vertical`

Specify the shear direction for the input image, as one of these options:

- `Horizontal` — Shifts the rows of the input image with a linearly varying offset. Specify the offset limits of the first and the last row using the **Row/column shear values [first last]** parameter.
- `Vertical` — Shifts the columns of the input image with a linearly varying offset. Specify the offset limits of the first and the last column using the **Row/column shear values [first last]** parameter.



Output size after shear — Transformed image size

`Full` (default) | `Same as input image`

Specify the size of the transformed image as, `Full` or `Same as input image`.

- `Full` — The block outputs the entire sheared image.
- `Same as input image` — The block outputs a matrix of the same size as the input image. In this case, the output image contains the top-left portion of the sheared image, as limited by the input image dimensions.

Shear values source — Source of shear value offset limits

`Specify via dialog` (default) | `Input port`

Select the source of shear value offset limits from these options:

- `Specify via dialog` — Specify the shear value offset limits using the **Row/column shear values [first last]** parameter.
- `Input port` — Specify the shear value offset limits using the **S** port.

Row/column shear values [first last] — Shear value offset limits

`[0 3]` (default) | two-element vector

Specify the shear value offset limits as a two-element vector of the form [`first last`]. The `first` and `last` values represent the number of pixels to shift the first and last rows or columns of the input image, respectively.

For example, if you set the **Shear direction** parameter to `Horizontal` and specify the **Row/column shear values [first last]** parameter value as `[50 150]`, the block shifts the top row of the input image 50 pixels to the right and the bottom row 150 pixels to the right. To move either row to the left, specify a negative value for the corresponding element of **Row/column shear values [first last]** parameter.

Similarly, if you set the **Shear direction** parameter to `Vertical` and specify the **Row/column shear values [first last]** parameter value as `[10 50]`, the block shifts the furthest left column 10 pixels down and the furthest right column 50 pixels down. To shift either column up, specify a negative value for the corresponding element of **Row/column shear values [first last]** parameter.

Dependencies

To enable this parameter, set the **Shear values source** parameter to `Specify via dialog`.

Maximum shear value — Maximum row or column shear

`20` (default) | scalar

Specify the maximum number of pixels to shear any row or column of the input image. The shear block uses this parameter to determine the size of the sheared image. If any input to the **S** port is greater than the absolute value of the **Maximum shear value** parameter, the shear block reduces the input value to the maximum shear value.

Dependencies

To enable this parameter, set the **Shear values source** parameter to `Input port`.

Background fill value — Intensity value of the background in transformed image

`0` (default) | scalar | three-element vector

Specify the intensity value of the background of the transformed image as a scalar intensity value or a three-element vector representing an RGB triplet.

Interpolation method — Method for interpolating transformed pixel values

`Bilinear` (default) | `Nearest neighbor` | `Bicubic`

Specify the method by which to interpolate the pixel values of the sheared image, as one of these options:

- `Nearest neighbor` —The Shear block uses the value of 1 nearest pixel for the new pixel value.
- `Bilinear` — The new pixel value is the weighted average of the 4 nearest pixel values.
- `Bicubic` — The new pixel value is the weighted average of the 16 nearest pixel values.

The number of pixels the block considers for interpolation affects the complexity and accuracy of the computation. The `Nearest neighbor` interpolation, which uses the fewest pixels, is the most

computationally efficient and least accurate while the `Bicubic` interpolation is the least computationally efficient and the most accurate. For more information about the interpolation methods and kernels, see the “More About” on page 1-425 section and the “Create and Compare Resizing Interpolation Kernels” example.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

More About

Nearest Neighbor Interpolation

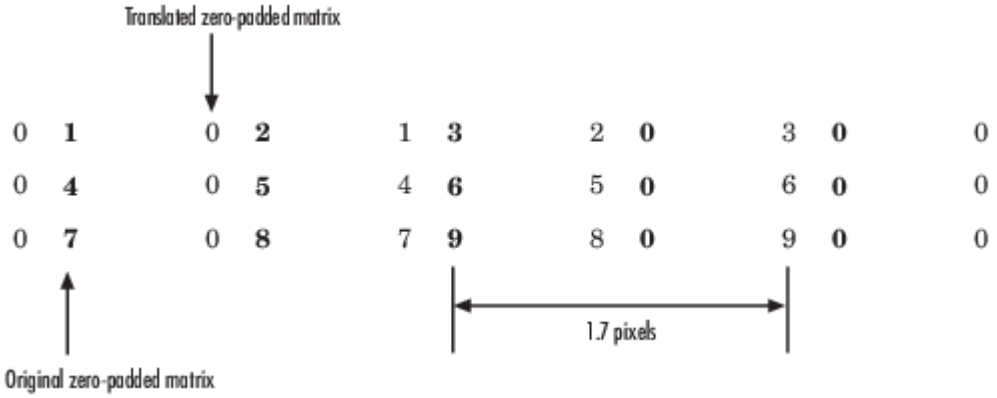
For nearest neighbor interpolation, the block uses nearby, translated, pixel values for the output pixel values.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image 1.7 pixels in the positive horizontal direction using nearest neighbor interpolation. These steps illustrate the nearest neighbor interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 1.7 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the translated value nearest to it resulting in this matrix:

0 0 1 2 3
0 0 4 5 6
0 0 7 8 9

Note Despite specifying a value of 1.7 pixels to translate the image, this method translates the image by 2 pixels. Nearest neighbor interpolation is computationally efficient, but not as accurate as the bilinear and bicubic interpolation methods.

Bilinear Interpolation

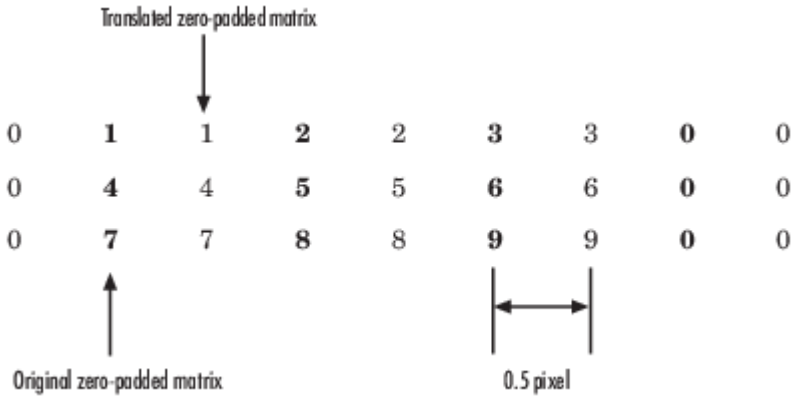
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bilinear interpolation. These steps illustrate the bilinear interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the translated pixel values on either side of the input pixel. The result is this matrix which has one column more than the input matrix.

0.5 1.5 2.5 1.5
2 4.5 5.5 3
3.5 7.5 8.5 4.5

Bicubic Interpolation

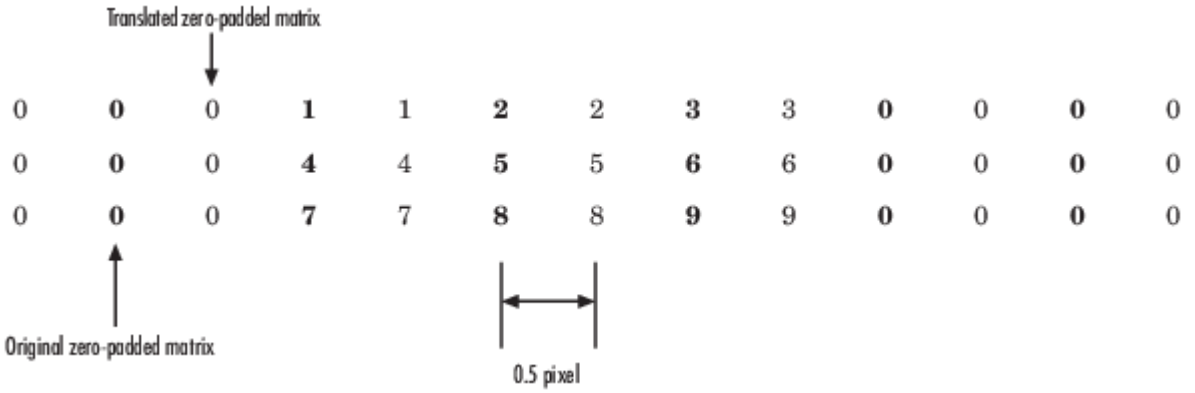
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bicubic interpolation. These steps illustrate the bicubic interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it by 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the two translated values on either side. The result is this matrix which has one column more than the input matrix:

0.375	1.5	3	1.625
1.875	4.875	6.375	3.125
3.375	8.25	9.75	4.625

Algorithms

Affine Transformation

Affine image transformation includes operations such as scaling, translation, shear, and rotation. The affine transformation matrix is a 3-by-3 matrix of the form $\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}$.

Consider an input image with r rows and c columns. $[r, c]$ are pixel coordinates in the input image and $[r', c']$ are pixel coordinates in the transformed image. The relationship between the input and output pixel coordinates is defined by:

$$[r'\ c'\ 1] = [r\ c\ 1] * \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}$$

$$r' = ra_{11} + ca_{21} + a_{31}$$
$$c' = ra_{12} + ca_{22} + a_{32}.$$

Shear transformation is an affine transformation that transforms the input image in the horizontal direction or vertical direction or both.

To obtain a horizontal shear transformation of an image, use the transformation matrix with a horizontal shear value, s_h.

$$[r' \; c' \; 1] = [r \; c \; 1] * \begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To obtain a vertical shear transformation of an image, use the transformation matrix with a vertical shear value, s_v.

$$[r' \; c' \; 1] = [r \; c \; 1] * \begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Version History

Introduced before R2006a

References

[1] Wolberg, George. *Digital Image Warping*. IEEE Computer Society Press Monograph. Los Alamitos, Calif: IEEE Computer Society Press, 1990.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Functions

`imwarp`

Blocks

Warp | Rotate | Translate | Resize

Resize

Enlarge or shrink entire image or region of interest within image



Libraries:
Computer Vision Toolbox / Geometric Transformations

Description

The `Resize` block enlarges or shrinks the input image or a region of interest (ROI) within the image. The block resizes along the row dimension followed by the column dimension, or vice-versa.

Ports

Input

Image — Input image or video
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, *M*-by-*N*-by-*C* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

ROI — Region of interest
vector

Region of interest, specified as `[x y width height]`, where `[x y]` are the coordinates for the upper-left corner of the region. `width` and `height` specify the width and the height of the region, respectively.

If you specify an input value to the `ROI` port, the `Resize` block applies the transformation only to a specified region in the input image.

Dependencies

To enable this port, select the **Enable ROI processing** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Out — Resized output image or video
matrix | array

Resized output image or video, returned as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, *M*-by-*N*-by-*C* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

The data type of the resized output image is the same as that of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Flag — ROI is outside image region
`0` | `1`

ROI is outside the image region, returned as `0` or `1`.

This table describes the output of the **Flag** port.

Flag Port Output	Description
`0`	ROI is completely inside the input image.
`1`	ROI is completely or partially outside the input image.

Dependencies

To enable this port, select these parameters:

- **Enable ROI processing**
- **Output flag indicating if ROI is within image bounds**

Data Types: `Boolean`

Parameters

Specify — Aspects of image to resize
`Output size as a percentage of input size` (default) | `Number of output columns and preserve aspect ratio` | `Number of output rows and preserve aspect ratio` | `Number of output rows and columns`

Specify aspects of the input image to consider for the transformation as one of these options:

- `Output size as a percentage of input size` — Transforms the input image dimensions such that the output image dimensions are a specified percentage of the input image size. Use the **Resize factor in percentage** parameter to specify a percentage factor for the entire image or independent percentage factors for the rows and columns of the input image.
- `Number of output columns and preserve aspect ratio` — Transforms the input image column dimension to a specified number of output columns. The block calculates the number of output rows such that the output image has the same aspect ratio as the input image. Use the **Number of output columns** parameter to specify the number of output columns.
- `Number of output rows and preserve aspect ratio` — Transforms the input image row dimension to a specified number of output rows. The block calculates the number of output columns such that the output image has the same aspect ratio as the input image. Use the **Number of output rows** parameter to specify the number of output rows.
- `Number of output rows and columns` — Transforms the output image size to a specified number of rows and columns. Use the **Number of output rows and columns** parameter to specify the number of output rows and columns.

> **Note** Specifying the number of output rows and columns can change the aspect ratio of the output image.

Resize factor in percentage — Percentage factor for input image transformation
`[200 150]` (default) | scalar | two-element vector

Specify the percentage resize factor to apply to the input image as one of these options:

- Scalar — Applies the same resize factor to the rows and the columns of the input image.
- Two-element vector value — The first element of the vector specifies the percentage resize factor to apply to the rows of the image, and the second element specifies the percentage resize factor to apply to the columns.

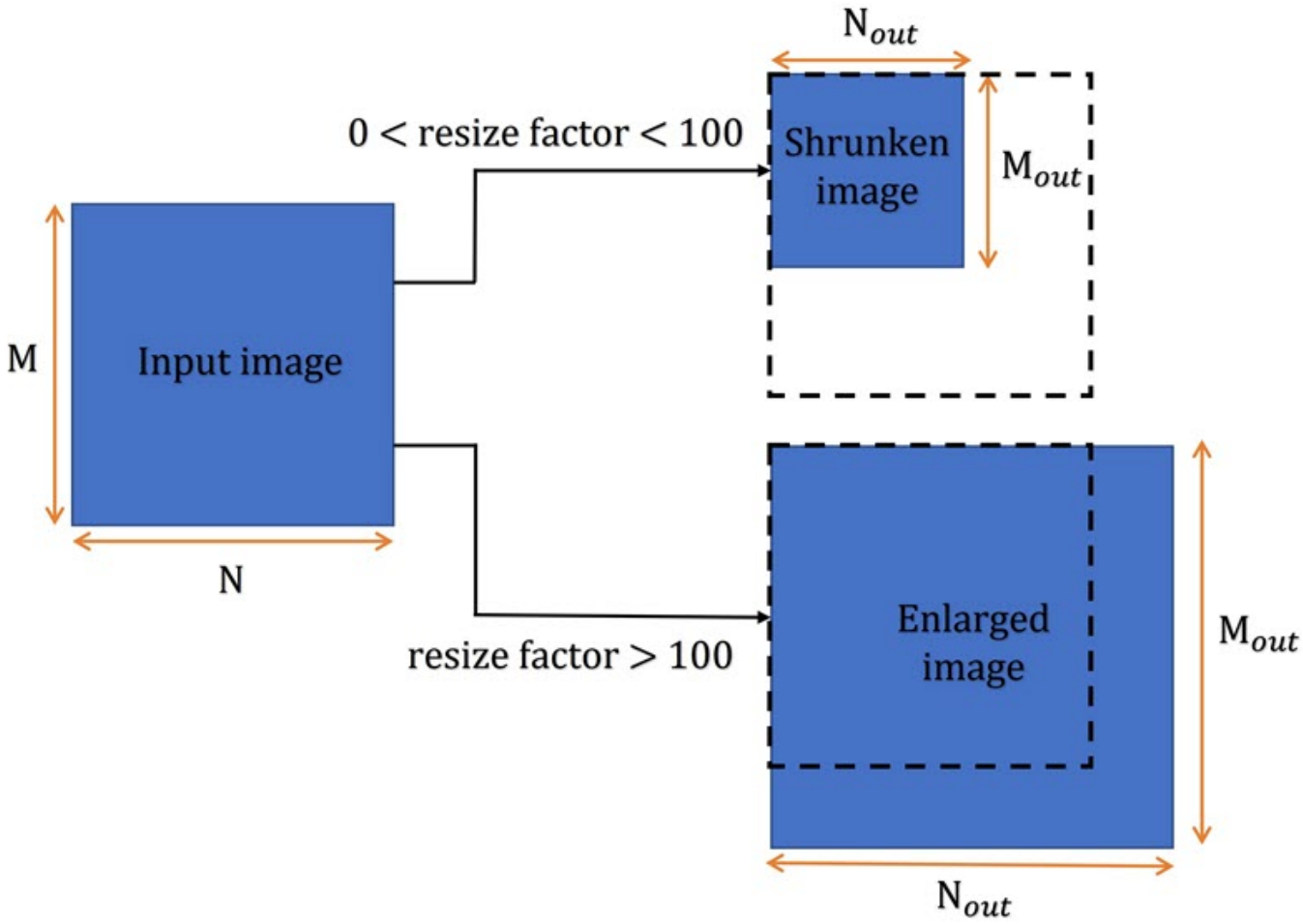
For a scalar **Resize factor in percentage** value, *resize factor*, the block calculates the dimensions $[M_{out}\ N_{out}]$ of the output image using these equations:

$$M_{out} = \frac{M * resize\ factor}{100}$$

$$N_{out} = \frac{N * resize\ factor}{100}$$

To resize the input image, specify a resize factor value greater than zero. This table describes the effect of various resize factor values on the input image.

Resize Factor Percentage	Effect on Image
`0` < *resize factor* < `100`	The Resize block shrinks the image.
resize factor = `100`	Image size is unchanged.
resize factor > `100`	The Resize block enlarges the image.



Dependencies

To enable this parameter, set the **Specify** parameter value to `Output size as a percentage of input size`.

Number of output columns — Number of columns in output image
25 (default) | scalar

Specify the number of columns in the output image.

Dependencies

To enable this parameter, set the **Specify** parameter value to `Number of output columns and preserve aspect ratio`.

Number of output rows — Number of rows in output image
25 (default) | scalar

Specify the number of rows in the output image.

Dependencies

To enable this parameter, set the **Specify** parameter value to `Number of output rows and preserve aspect ratio`.

Number of output rows and columns — Number of rows and columns in output image
`[25 35]` (default) | two-element vector

Specify the number of rows and columns in the output image, specified as a two-element vector. The first element specifies the number of rows in the output image, and the second element is the number of columns.

Dependencies

To enable this parameter, set the **Specify** parameter value to `Number of output rows and columns`.

Interpolation method — Interpolation method
`Bilinear` (default) | `Nearest neighbor` | `Bicubic` | `Lanczos2` | `Lanczos3`

Specify the method for interpolating the pixel values of the resized image as one of these options:

- `Nearest neighbor` — The Resize block uses the value of 1 nearest pixel for the new pixel value.
- `Bilinear` — The new pixel value is the weighted average of the 4 nearest pixel values.
- `Bicubic` — The new pixel value is the weighted average of the 16 nearest pixel values.
- `Lanczos2` — The new pixel value is the weighted average of the 16 nearest pixel values, calculated using the Lanczos-2 kernel.
- `Lanczos3` — The new pixel value is the weighted average of the 36 nearest pixel values, calculated using the Lanczos-3 kernel.

The number of pixels the block considers for interpolation affects the complexity and accuracy of the computation. The `Nearest neighbor` interpolation, which uses the fewest pixels, is the most computationally efficient and least accurate while the interpolation with `Lanczos3` kernel is the least computationally efficient and the most accurate. For more information about the interpolation methods and kernels, see the “More About” on page 1-434 section and the “Create and Compare Resizing Interpolation Kernels” example.

Perform antialiasing when resize factor is between 0 and 100 — Prevent aliasing when shrinking image
`off` (default) | `on`

Select this parameter to perform low-pass filtering on the input image before shrinking it. Selecting this parameter can prevent aliasing in the output image when the value of the **Resize factor in percentage** parameter is between `0` and `100`.

Enable ROI processing — Enable ROI input
`off` (default) | `on`

Select this parameter to enable the **ROI** input port, for specifying a region of interest to resize in the input image.

Dependencies

To enable this parameter:

- Set the **Specify** parameter to `Number of output rows and columns`.
- Set the **Interpolation method** to one of these options:

- `Nearest neighbor`
- `Bilinear`
- `Bicubic`

- Clear the **Perform antialiasing when resize factor is between 0 and 100** parameter.

Output flag indicating if ROI is within image bounds — Enable flag indicating if specified ROI is outside input image region
`off` (default) | `on`

Select this parameter to enable the **Flag** port which indicates if the specified ROI value is completely or partially outside the range of the input image dimensions.

Dependencies

To enable this parameter, select the **Enable ROI processing** parameter.

Block Characteristics

Data Types	`Boolean` | `double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

More About

Nearest Neighbor Interpolation

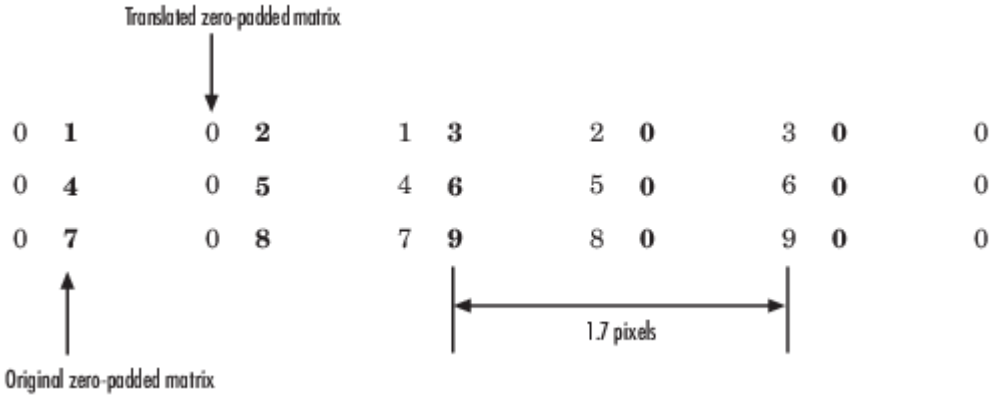
For nearest neighbor interpolation, the block uses nearby, translated, pixel values for the output pixel values.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image 1.7 pixels in the positive horizontal direction using nearest neighbor interpolation. These steps illustrate the nearest neighbor interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 1.7 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the translated value nearest to it resulting in this matrix:

0 0 1 2 3
0 0 4 5 6
0 0 7 8 9

Note Despite specifying a value of 1.7 pixels to translate the image, this method translates the image by 2 pixels. Nearest neighbor interpolation is computationally efficient, but not as accurate as the bilinear and bicubic interpolation methods.

Bilinear Interpolation

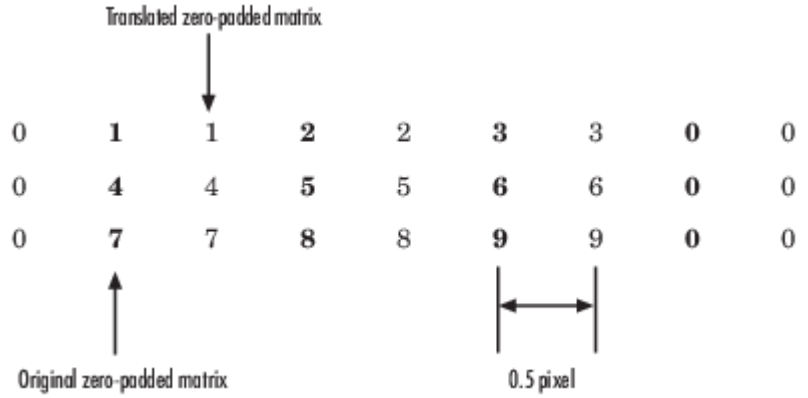
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bilinear interpolation. These steps illustrate the bilinear interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the translated pixel values on either side of the input pixel. The result is this matrix which has one column more than the input matrix.

```
0.5 1.5 2.5 1.5
 2   4.5 5.5  3
3.5 7.5 8.5 4.5
```

Bicubic Interpolation

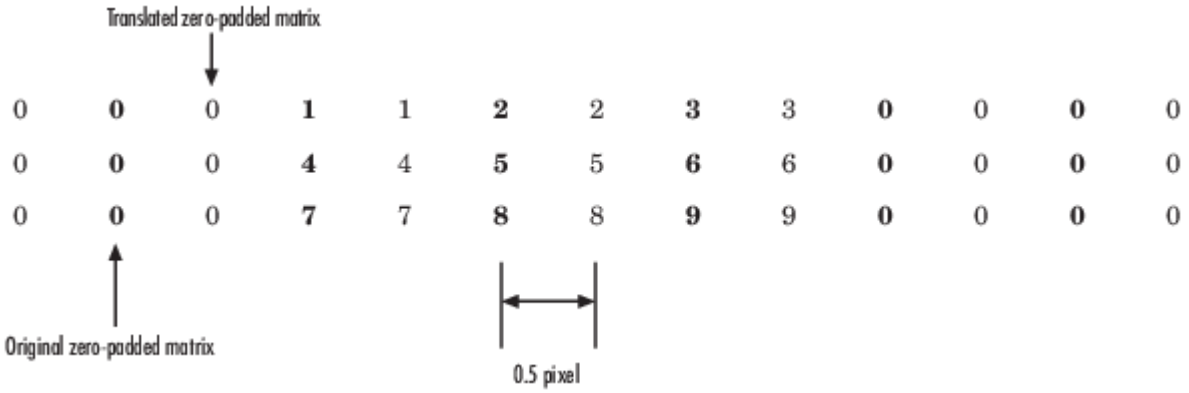
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

```
1 2 3
4 5 6
7 8 9
```

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bicubic interpolation. These steps illustrate the bicubic interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it by 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the two translated values on either side. The result is this matrix which has one column more than the input matrix:

```
0.375   1.5     3    1.625
1.875 4.875 6.375 3.125
3.375   8.25   9.75  4.625
```

Version History

Introduced before R2006a

References

[1] Ward, Joseph, and David R. Cok. “Resampling Algorithms for Image Resizing and Rotation.” In *Proceedings of the SPIE Digital Image Processing Applications*, edited by Ying-Wei Lin and Ram Srinivasan, 260–69. Los Angeles, CA, United States, 1989.

[2] Wolberg, George. *Digital Image Warping*. IEEE Computer Society Press Monograph. Los Alamitos, Calif: IEEE Computer Society Press, 1990.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Functions

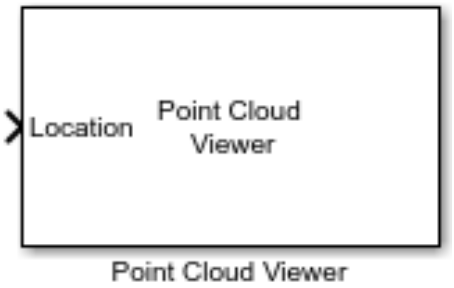
`imresize`

Blocks

Translate | Shear | Rotate | Warp

Point Cloud Viewer

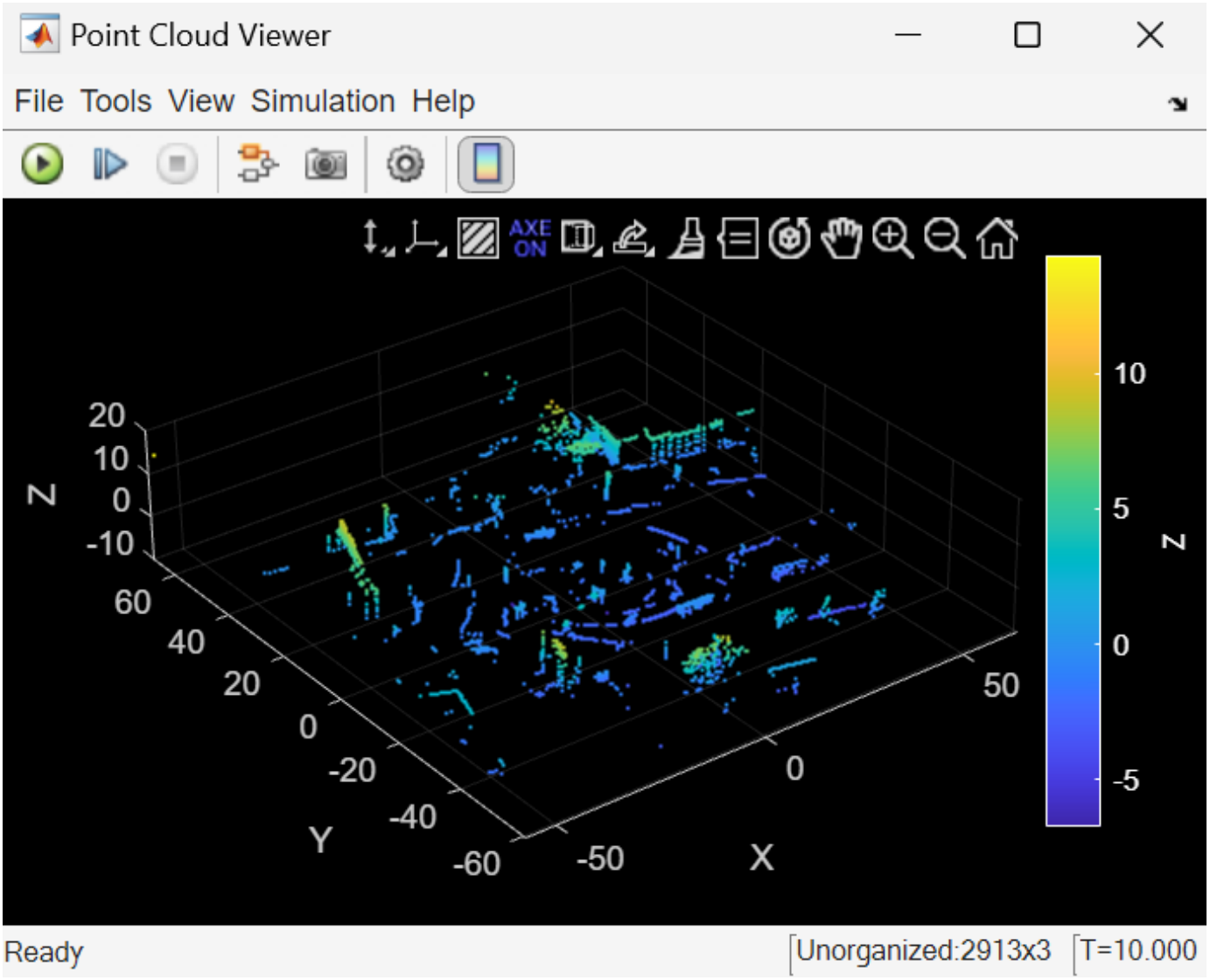
Visualize streaming point cloud data sequence



Libraries:
Computer Vision Toolbox / Sinks

Description

The Point Cloud Viewer block creates a Point Cloud Viewer window to display a sequence of 3-D point cloud data that streams from a device such as a Microsoft® Kinect® or a lidar sensor. The Point Cloud Viewer block also provides simulation controls in the Point Cloud Viewer window to play, pause, and step through the streaming point cloud data sequence.



Features of Point Cloud Viewer

- Playback control — Control streaming playback of a point cloud sequence using the **Run**, **Step Forward**, and **Stop** buttons.
- Set axis limits — Limit the display of the 3-D point cloud data by setting the *x*-, *y*-, and *z*-axis limits.
- Set background color — Set the background color of the viewer window against which to display of point cloud data.
- Vertical axis direction control — Set the vertical axis direction of the point cloud data to upward-positive or downward-positive.
- View plane control — Set the view plane to change the viewing angle of the point cloud data.
- Analyze point cloud data — Inspect point cloud data using the **Data Tips** option on the axes toolbar of the Point Cloud Viewer window.
- View colorbar — View the colorbar in the Point Cloud Viewer window using the **Colorbar** button on the simulation toolbar.

Ports

Input

Location — Positions of points in 3-D coordinate space
matrix | array

Positions of the points in 3-D coordinate space, specified as:

- An *M*-by-3 matrix or *M*-by-*N*-by-3 array for a 3-D point cloud.
- An *M*-by-3-by-*T* or *M*-by-*N*-by-3-by-*T* array for a sequence of 3-D point clouds, where *T* is the total number of 3-D point clouds in the point cloud sequence.

Each array specifies the *x*-, *y*-, and *z*- coordinates of a points in the point cloud, but the format differs between unorganized point cloud and organized point clouds.

- For unorganized point clouds, you must specify **Location** as an *M*-by-3 or *M*-by-3-by-*T* array, where *M* is the total number of points, and each row provides the *xyz*-coordinates for a point.
- For organized point clouds, you must specify **Location** as an *M*-by-*N*-by-3 or an *M*-by-*N*-by-3-by-*T* array, where *M*×*N* is the total number of points, and the third dimension of the array provides the *x*-, *y*-, and *z*-coordinates for the corresponding point. Projective cameras, such as a Kinect or a lidar sensor, store points as an organized point cloud. An organized point cloud is laid out as a 2-D or a 3-D array of points that resembles an image- or video-like structure.

Data Types: `single` | `double` | `uint8` | `uint16`

Color — Color of point cloud
`[]` (default) | matrix | array

Point cloud color, specified as:

- An *M*-by-3 matrix or *M*-by-*N*-by-3 array for a 3-D point cloud.
- An *M*-by-3-by-*T* or *M*-by-*N*-by-3-by-*T* array for a sequence of 3-D point clouds, where *T* is the total number of 3-D point clouds in a point cloud sequence.

For an unorganized point cloud, specified as an *M*-by-3 matrix or *M*-by-3-by-*T* array, each row specifies the RGB color triplet for a point in the point cloud data. For an organized point cloud, specified as an *M*-by-*N*-by-3 or *M*-by-*N*-by-3-by-*T* array, each channel of the third dimension specifies the RGB color triplet for the corresponding point in the point cloud data. You can specify the same color for all points or a different color for each point.

- For the `single` or `double` data type, you must specify the **Color** RGB value in the range [0, 1].
- For the `uint8` data type, you must specify the **Color** RGB value in the range [0, 255].
- For the `uint16` data type, you must specify the **Color** RGB value in the range [0, 65535].

Note Input to the **Color** port must have the same dimensions and data type as the input to the **Location** port.

Dependencies

To enable this port, select the **File > Location and Color Port** parameter.

Data Types: `single` | `double` | `uint8` | `uint16`

Intensity — Grayscale intensities
`[]` (default) | vector | matrix | array

Grayscale intensities at each point, specified as:

- An *M*-by-1 vector for an unorganized point cloud or *M*-by-*N* matrix for an organized point cloud.
- An *M*-by-1-by-*T* array for a sequence of unorganized point clouds or *M*-by-*N*-by-*T* array for a sequence of organized point clouds. *T* is the total number of 3-D point clouds in a point cloud sequence.

Note Input to the **Intensity** port must have the same data type as input to the **Location** port.

Dependencies

To enable this port, select the **File > Location and Intensity Port** parameter.

Data Types: `single` | `double` | `uint8` | `uint16`

Parameters

File

Open at Start of Simulation — Open Point Cloud Viewer window at start of simulation
`on` (default) | `off`

Select this parameter to display the Point Cloud Viewer window at the start of the simulation. Clear this parameter to prevent the Point Cloud Viewer window from opening at the start of the simulation.

Location Port — Enable point location as input
`on` (default) | `off`

Select this parameter to enable the **Location** port for inputting the 3-D point cloud coordinates. If you select this parameter, the **Location and Color Port** and **Location and Intensity Port** parameters clear automatically.

Location and Color Port — Enable point location and color as input
`off` (default) | `on`

Select this parameter to enable both the **Location** and **Color** input ports. If you select this parameter, the **Location Port** and **Location and Intensity Port** parameters clear automatically.

Location and Intensity Port — Enable point location and intensity as input
`off` (default) | `on`

Select this parameter to enable both the **Location** and **Intensity** input ports. If you select this parameter, the **Location Port** and **Location and Color Port** parameters clear automatically.

File > Configuration > Edit > Core

The Point Cloud Viewer Configuration dialog box enables you to change the behavior and appearance of the graphic user interface (GUI).

- To open the Configuration dialog box and edit parameters, select **File** > **Configuration** > **Edit**.

The **Core** tab in the Configuration dialog box lists the general settings of the GUI.

- In the **Core** tab, select `General UI` and click **Properties** to open the General UI Properties dialog box.

Display the full path — Display full path of source data in title bar
`off` (default) | `on`

Select this parameter to display the model name and full path to the source data in the title bar of the Point Cloud Viewer window. If you clear this parameter, the window displays a shortened name.

Tools > Point Cloud Information

The Point Cloud Information dialog box displays information about the input point cloud sequence. This information includes **Source type**, **Source name**, **Frame size**, **Source data type**, and **Display data type**.

View

Bring All Point Cloud Viewer Windows Forward — Bring all Point Cloud Viewer windows to front of display
`off` (default) | `on`

Select this parameter to bring all the Point Cloud Viewer windows forward, in front of all other open windows behind.

Simulation Toolbar — Show simulation toolbar
`on` (default) | `off`

Select this parameter to show the simulation toolbar on the Point Cloud Viewer window.



To perform a relevant operation using the simulation toolbar, select the corresponding feature as described in the table.

Feature	Description
Run	Start the simulation, and view the point cloud sequence.
Step Forward	Start the simulation, view one frame of the point cloud sequence, and pause.
Stop	Stop the simulation.
Highlight Simulink Block	Highlight the Point Cloud Viewer block with a blue background in the Simulink model.

Feature	Description
Snapshot (Freeze Display)	Freeze Point Cloud Viewer window to enable you take a snapshot of the window using the **Print** or **Print Preview** option in **File** menu. **Note** Unselect this option to return the Point Cloud Viewer window to an active state from a frozen state.
Settings	Enable the **Settings** pane on the Point Cloud Viewer window.
Colorbar	View the color bar in the Point Cloud Viewer window. By default, the Point Cloud Viewer window does not display the color bar. To view the color bar, click **Colorbar** on the simulation toolbar. To hide the color bar, click **Colorbar** again. **Dependencies** To enable this parameter, select the **File** > **Location Port** or **File** > **Location and Intensity Port** parameter.

Status Bar — Show status bar
`on` (default) | `off`

Select this parameter to show the status bar on the Point Cloud Viewer window. The status bar, at the bottom of the Point Cloud Viewer window, shows this information about the point cloud sequence.

- Dimension
- Organization
- Status of the window during simulation
- Timestamp of the current frame

Highlight Simulink Block — Highlight Point Cloud Viewer block
`off` (default) | `on`

Highlight the Point Cloud Viewer block in blue in the Simulink model.

Simulation

To access this parameter, select the **Simulation** menu in the Point Cloud Viewer window.

Run — View entire point cloud sequence
`off` (default) | `on`

Select this parameter to start the simulation and view the point cloud sequence continuously from the first to the last frame. You can also perform this action by clicking **Run** on the simulation toolbar.

Continue — Continue viewing the point cloud sequence
`off` (default) | `on`

Select this parameter to continue viewing the point cloud sequence till the last frame. To enable the **Continue** parameter, start the simulation by clicking **Step Forward** on simulation toolbar.

Stop — Stop viewing the point cloud sequence
`off` (default) | `on`

Select this parameter to stop the display of the point cloud sequence. You can also perform this action by clicking **Stop** on the simulation toolbar.

Drop Frames to Improve Performance — Reduce frame rate of input point cloud sequence
`on` (default) | `off`

Select this parameter to reduce the frame rate of the input streaming point cloud sequence to improve the performance of the simulation. If you clear this parameter, the block displays every frame in the input point cloud sequence signal during the simulation.

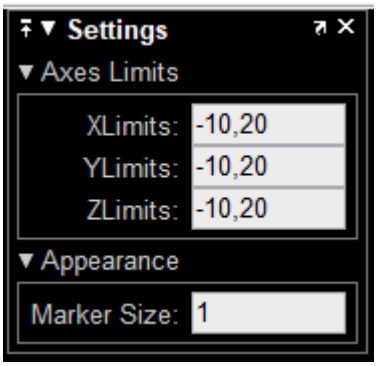
Block Characteristics

Data Types	`Boolean` | `double` | `integer` | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`yes`
Zero-Crossing Detection	`no`

More About

Settings Pane

To display the **Settings** pane on the point cloud viewer window, click Settings on the simulation toolbar.



You can use the **Settings** pane to:

- Set **Axes Limits**. The Point Cloud Viewer window does not display point cloud data outside these limits. By default, the block sets the limits for each axis based on the extent of the input point cloud data along that axis.
  - **XLimits** — Range of *x*-axis coordinates, specified in the format [`minimum limit`,`maximum limit`].
  - **YLimits** — Range of *y*-axis coordinates, specified in the format [`minimum limit`,`maximum limit`].

  - **ZLimits** — Range of *z*-axis coordinates, specified in the format [`minimum limit`,`maximum limit`].
- Set the **Appearance** of the point cloud data points in the Point Cloud Viewer window by adjusting the approximate diameter of the point marker using **Marker Size**. MATLAB graphics defines the unit of measure as points. The default **Marker Size** value is `1`. A marker size larger than `6` can reduce rendering performance.

Axes Toolbar Operations

Keyboard shortcuts are enabled only for the rotate (default) mode. To view the axes toolbar operations, point to the figure and select one of these options. Click the Restore View icon to reset the display.



Feature	Description
Vertical Axis	Select an *X*-, *Y*-, or *Z*-direction to set the direction of that axis of the point cloud as the positive vertical direction. Use the keyboard arrow keys to move the point cloud with respect to the direction selected.
View Plane	Select the viewing angle of the point cloud figure as the *XY*, *YX*, *XZ*, *ZX*, *YZ*, or *ZY* plane.
Background Color	Display a color chart from which you can select a background color.
Axes On/Off	Turn the display axes on or off.
Projection	Select perspective or orthogonal view.
Export	• — Save the content as a tightly cropped image or PDF. • — Copy the content as an image. • — Copy the content as a vector graphic.
Brush	Enable data brushing mode.
Data Tips	View the point data values of a point by selecting it in the point cloud figure. The Data Tips display shows the *x*, *y*, and *z* values of the points. Point clouds for depth images and lidar contain these additional data properties:

Point Cloud Data Source	Data Value Properties
Depth image (RGB-D sensor)	**Color**, **Row**, and **Column**
Lidar	**Intensity**, **Range**, **Azimuth** angle, **Elevation** angle, **Row**, and **Column**

<table>
<tr><th>Feature</th><th>Description</th></tr>
<tr><td>Rotate 3D</td><td>Drag to rotate the point cloud figure. You can also right-click any point on the point cloud figure and select Rotate Around a Point to specify that point as the center of rotation for the figure. To return to axes rotation, right-click again and select Rotate Around Axes Center.

While in Rotate 3D mode, you can also select the color source for the point cloud by right-clicking anywhere in the point cloud figure and selecting Color Source, and then selecting an option from the list. The figure applies the active colormap to the point cloud based on the values in the selected source. For a point cloud of only position data, the Color Source options are X, Y, Z, and User Specified. Point clouds for depth image and lidar data include these additional color source options:

<table>
<tr><th>Point Cloud Data Source</th><th>Color Source Options</th></tr>
<tr><td>Depth image (RGB-D sensor)</td><td>Color, Row, and Column</td></tr>
<tr><td>Lidar</td><td>Intensity, Range, Azimuth angle, Elevation angle, Row, and Column</td></tr>
</table>
</td></tr>
<tr><td>Pan, Zoom in, and Zoom out</td><td>Maneuver the point cloud in the figure, or zoom the point cloud figure in or out.</td></tr>
<tr><td>Restore View</td><td>Restore the original view of the axes.</td></tr>
</table>

Version History

Introduced in R2023a

R2025a: Visualize point cloud data using intensity values

The block now supports an **Intensity** input port to visualize streaming point cloud data using intensity values. To enable this port, select the **File > Location and Intensity Port** parameter.

R2025a: View color bar for point cloud data

You can now view the color bar by selecting the **Colorbar** button on the simulation toolbar. This button is available when you select the **File > Location Port** or **File > Location and Intensity Port** parameter.

R2025a: View point cloud across full extent of axes

The block now automatically adjusts the default axes limits based on the input point cloud data. Automatic adjustment enables you to view the point cloud across full extent of the axes.

See Also

`pcplayer`

Translate

Translate image or video frame



Libraries:
Computer Vision Toolbox / Geometric Transformations

Description

The Translate block shifts an input image or video frame by the number of pixels in the vertical and horizontal directions specified by a two-element offset vector.

Ports

Input

Image — Input image or video
matrix | array

Input image or video, specified as an *M*-by-*N* matrix, *M*-by-*N*-by-*T* array, *M*-by-*N*-by-*C* array, or *M*-by-*N*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Offset — Translation value
vector

Translation value, specified as a two-element vector of the form `[vertical horizontal]`. The `vertical` value in the vector is the number of pixels to shift the image up or down in the vertical direction. The `horizontal` element is the number of pixels to shift the image left or right in the horizontal direction. The axes origin is the top-left corner of the input image.

- To translate the image downward, specify positive pixel values to the `vertical` element.
- To translate the image upward, specify negative pixel values to the `vertical` element.
- To translate the image to the right, specify positive pixel values to the `horizontal` element.
- To translate the image to the left, specify negative pixel values to the `horizontal` element.

Dependencies

To enable this port, set the **Offset source** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

Output — Translated image or video
matrix | array

Translated image or video, returned as a *P*-by-*Q* matrix, *P*-by-*Q*-by-*T* array, *P*-by-*Q*-by-*C* array, or *P*-by-*Q*-by-*C*-by-*T* array. *T* is the number of frames in a video or image sequence, and *C* is the number of color channels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Output size after translation — Translated image size

`Full` (default) | `Same as input image`

Specify the size of the translated image as `Full` or `Same as input image`.

- `Full` — The block outputs a matrix that contains the entire translated image.
- `Same as input image` — The block outputs a matrix of the same size as the input image, which contains a portion of the translated image.

Offset source — Source of translation value

`Specify via dialog` (default) | `Input port`

Specify the source of translation value as one of these options:

- `Specify via dialog` — Specify the translation value using the **Offset** parameter.
- `Input port` — Specify the translation value using the **Offset** port.

Offset — Translation value

`[1.5 2.3]` (default) | two-element vector

Specify the translation value as a two-element vector of the form [`vertical horizontal`]. The vertical element in the vector is the number of pixels to shift the image up or down in the vertical direction. The horizontal element is the number of pixels to shift the image left or right in the horizontal direction. The axes origin is the top-left corner of the input image.

- To translate the image downward, specify positive pixel values to the `vertical` element.
- To translate the image upward, specify negative pixel values to the `vertical` element.
- To translate the image to the right, specify positive pixel values to the `horizontal` element.
- To translate the image to the left, specify negative pixel values to the `horizontal` element.

Dependencies

To enable this parameter, set the **Offset source** parameter to `Specify via dialog`.

Maximum offset — Translation value limits

`[8 10]` (default) | two-element vector

Specify the maximum number of pixels to translate the input image in the vertical and horizontal directions, as a vector of the form [`vertical horizontal`]. The translate block uses this parameter to determine the size of the translated image when the **Output size after translation** parameter is set to `Full` and the **Offset source** parameter is set to `Input port`. If either input value to the

Offset port is greater than the corresponding value of the **Maximum offset** parameter, the translate block reduces that input translation value to the corresponding maximum translation value.

Dependencies

To enable this parameter, set the **Offset source** parameter to `Input port`.

Background fill value — Intensity or color of background in translated image

`0` (default) | scalar | RGB triplet

Specify the intensity or color of the background of the translated image as a scalar intensity value or an RGB triplet.

Interpolation method — Method for interpolating translated pixel values

`Bilinear` (default) | `Nearest neighbor` | `Bicubic`

Specify the method by which to interpolate the pixel values of the translated image as one of these options:

- `Nearest neighbor` —The Translate block uses the value of the nearest pixel for the interpolated pixel.
- `Bilinear` — The interpolated pixel value is the weighted average of the 4 nearest pixel values.
- `Bicubic` — The interpolated pixel value is the weighted average of the 16 nearest pixel values.

The number of pixels the block considers for interpolation affects the complexity and accuracy of the computation. The `Nearest neighbor` interpolation, which uses the fewest pixels, is the most computationally efficient and least accurate, while the `Bicubic` interpolation is the least computationally efficient and the most accurate. For more information about interpolation methods and kernels, see the “More About” on page 1-449 section and the “Create and Compare Resizing Interpolation Kernels” example.

Block Characteristics

Data Types	`double` | `fixed point` | `integer` | `single`
Multidimensional Signals	`yes`
Variable-Size Signals	`no`

More About

Nearest Neighbor Interpolation

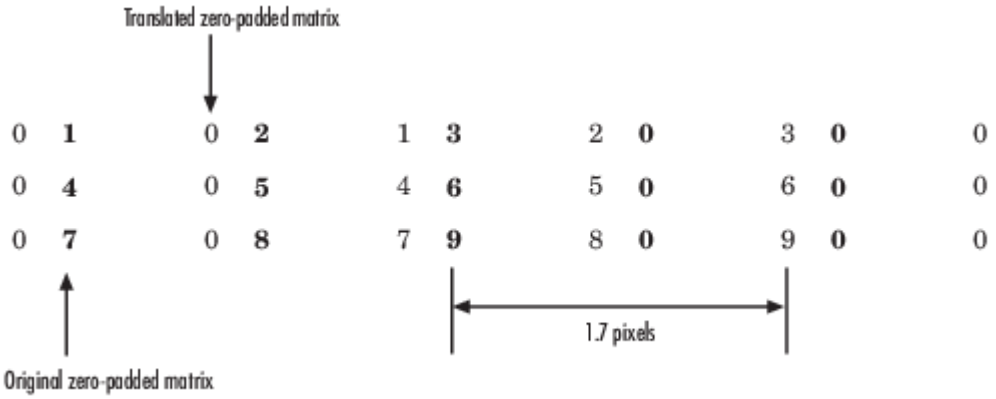
For nearest neighbor interpolation, the block uses nearby, translated, pixel values for the output pixel values.

For example, suppose this matrix, represents your input image,

```
1 2 3
4 5 6
7 8 9
```

and you want to translate this image 1.7 pixels in the positive horizontal direction using nearest neighbor interpolation. These steps illustrate the nearest neighbor interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 1.7 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the translated value nearest to it resulting in this matrix:

0 0 1 2 3
0 0 4 5 6
0 0 7 8 9

Note Despite specifying a value of 1.7 pixels to translate the image, this method translates the image by 2 pixels. Nearest neighbor interpolation is computationally efficient, but not as accurate as the bilinear and bicubic interpolation methods.

Bilinear Interpolation

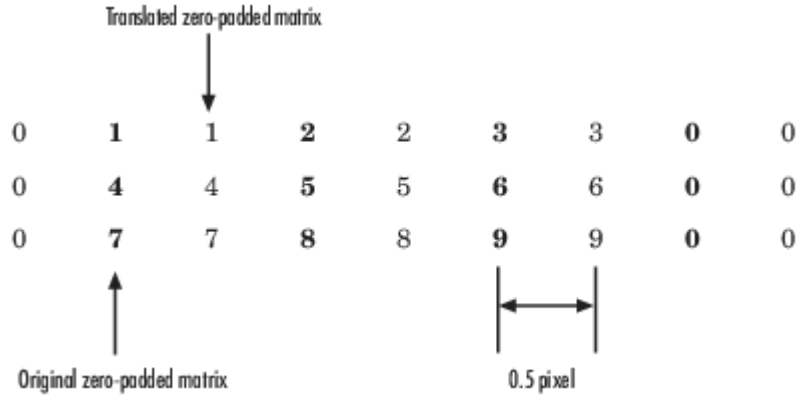
For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bilinear interpolation. These steps illustrate the bilinear interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the translated pixel values on either side of the input pixel. The result is this matrix which has one column more than the input matrix.

0.5 1.5 2.5 1.5
2 4.5 5.5 3
3.5 7.5 8.5 4.5

Bicubic Interpolation

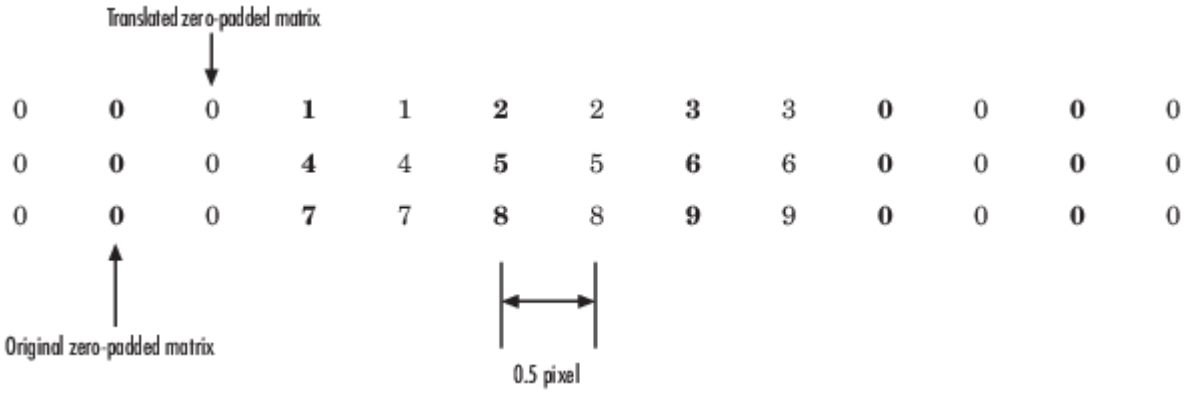
For bicubic interpolation, the block uses the weighted average of four translated pixel values for each output pixel value.

For example, suppose this matrix, represents your input image,

1 2 3
4 5 6
7 8 9

and you want to translate this image by 0.5 pixels in the positive horizontal direction using bicubic interpolation. These steps illustrate the bicubic interpolation algorithm followed by the block:

1 Zero pad the input matrix and translate it by 0.5 pixels to the right.



2 Create the output matrix by replacing each input pixel value with the weighted average of the two translated values on either side. The result is this matrix which has one column more than the input matrix:

```
0.375   1.5     3    1.625
1.875 4.875 6.375 3.125
3.375   8.25   9.75  4.625
```

Version History

Introduced before R2006a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Rotate | Shear | Resize

Objects

se3

SE(3) homogeneous transformation

Description

The `se3` object represents an SE(3) transformation as a 3-D homogeneous transformation matrix consisting of a translation and rotation for a right-handed Cartesian coordinate system.

For more information, see the “3-D Homogeneous Transformation Matrix” on page 2-8 section.

This object acts like a numerical matrix enabling you to compose poses using multiplication and division.

Creation

Syntax

```
transformation = se3
transformation = se3(rotation)
transformation = se3(rotation,translation)
transformation = se3(transformation)

transformation = se3(euler,"eul")
transformation = se3(euler,"eul",sequence)
transformation = se3(quat,"quat")
transformation = se3(quaternion)
transformation = se3(axang,"axang")
transformation = se3(angle,axis)
transformation = se3(___,translation)

transformation = se3(translation,"trvec")
transformation = se3(pose,"xyzquat")
```

Description

Rotation Matrices, Translation Vectors, and Transformation Matrices

`transformation = se3` creates an SE(3) transformation representing an identity rotation with no translation.

$$transformation = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`transformation = se3(rotation)` creates an SE(3) transformation representing a pure rotation defined by the orthonormal rotation `rotation` with no translation. The rotation matrix is represented by the elements in the top left of the `transformation` matrix.

$$rotation = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{11} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$transformation = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`transformation = se3(rotation,translation)` creates an SE(3) transformation representing a rotation defined by the orthonormal rotation `rotation` and the translation `translation`. The function applies the rotation matrix first, then translation vector to create the transformation.

$$rotation = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{11} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, translation = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

$$transformation = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`transformation = se3(transformation)` creates an SE(3) transformation representing a translation and rotation as defined by the homogeneous transformation `transformation`.

Other 3-D Rotation Representations

`transformation = se3(euler,"eul")` creates an SE(3) transformation from the rotations defined by the Euler angles `euler`.

`transformation = se3(euler,"eul",sequence)` specifies the sequence of the Euler angle rotations `sequence`. For example, the sequence `"ZYX"` rotates the *z*-axis, then the *y*-axis and *x*-axis.

`transformation = se3(quat,"quat")` creates an SE(3) transformation from the rotations defined by the numeric quaternions `quat`.

`transformation = se3(quaternion)` creates an SE(3) transformation from the rotations defined by the quaternion `quaternion`.

`transformation = se3(axang,"axang")` creates an SE(3) transformation from the rotations defined by the axis-angle rotation `axang`.

`transformation = se3(angle,axis)` creates an SE(3) transformation from the rotations `angle` about the rotation axis `axis`.

`transformation = se3(___,translation)` creates an SE(3) transformation from the translation vector `translation` along with any other type of rotation input arguments.

Other Translations and Transformation Representations

`transformation = se3(translation,"trvec")` creates an SE(3) transformation from the translation vector `translation`.

`transformation = se3(pose,"xyzquat")` creates an SE(3) transformation from the 3-D compact pose `pose`.

Note If any inputs contain more than one rotation, translation, or transformation, then the output `transformation` is an *N*-element array of `se3` objects corresponding to each of the *N* input rotations, translations, or transformations.

Input Arguments

`rotation` — Orthonormal rotation

3-by-3 matrix | 3-by-3-by-*N* matrix | `so3` object | *N*-element array of `so3` objects

Orthonormal rotation, specified as a 3-by-3 matrix, a 3-by-3-by-*N* array, a scalar `so3` object, or an *N*-element array of `so3` objects. *N* is the total number of rotations.

If `rotation` contains more than one rotation and you also specify `translation` at construction, the number of translations in `translation` must be one or equal to the number of rotations in `rotation`. The resulting number of transformation objects is equal to the value of the `translation` or `rotation` argument, whichever is larger.

Example: `eye(3)`

Data Types: `single` | `double`

`translation` — Translation

three-element row vector | *N*-by-3 matrix

Translation, specified as a three-element row vector or an *N*-by-3 array. *N* is the total number of translations and each translation is of the form [*x y z*].

If `translation` contains more than one translation, the number of rotations in `rotation` must be one or equal to the number of translations in `translation`. The resulting number of created transformation objects is equal to the value of the `translation` or `rotation` argument, whichever is larger.

Example: `[1 4 3]`

Data Types: `single` | `double`

`transformation` — Homogeneous transformation

4-by-4 matrix | 4-by-4-by-*N* array | `se3` object | *N*-element array of `se3` objects

Homogeneous transformation, specified as a 4-by-4 matrix, a 4-by-4-by-*N* array, a scalar `se3` object, or an *N*-element array of `se3` objects. *N* is the total number of transformations specified.

If `transformation` is an array, the resulting number of created `se3` objects is equal to *N*.

Example: `eye(4)`

Data Types: `single` | `double`

`quaternion` — Quaternion

`quaternion` object | *N*-element array of `quaternion` objects

Quaternion, specified as a scalar `quaternion` object or as an *N*-element array of `quaternion` objects. *N* is the total number of specified quaternions.

If `quaternion` is an *N*-element array, the resulting number of created `se3` objects is equal to *N*.

Example: `quaternion(1,0.2,0.4,0.2)`

`euler` — Euler angles

N-by-3 matrix

Euler angles, specified as an *N*-by-3 matrix, in radians. Each row represents one set of Euler angles with the axis-rotation sequence defined by the `sequence` argument. The default axis-rotation sequence is *ZYX*.

If `euler` is an *N*-by-3 matrix, the resulting number of created `se3` objects is equal to *N*.

Example: `[pi/2 pi pi/4]`

Data Types: `single` | `double`

`sequence` — Axis-rotation sequence

`"ZYX"` (default) | `"ZYZ"` | `"ZXY"` | `"ZXZ"` | `"YXY"` | `"YZX"` | `"YXZ"` | `"YZY"` | `"XYX"` | `"XYZ"` | `"XZX"` | `"XZY"`

Axis-rotation sequence for the Euler angles, specified as one of these string scalars:

- `"ZYX"` (default)
- `"ZYZ"`
- `"ZXY"`
- `"ZXZ"`
- `"YXY"`
- `"YZX"`
- `"YXZ"`
- `"YZY"`
- `"XYX"`
- `"XYZ"`
- `"XZX"`
- `"XZY"`

These are orthonormal rotation matrices for rotations of ϕ, ψ, and θ about the *x*-, *y*-, and *z*-axis, respectively:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When constructing the rotation matrix from this sequence, each character indicates the corresponding axis. For example, if the sequence is `"XYZ"`, then the `se3` object constructs the rotation matrix *R* by multiplying the rotation about *x*-axis with the rotation about the *y*-axis, and then multiplying that product with the rotation about the *z*-axis:

$$R = R_x(\phi) \cdot R_y(\psi) \cdot R_z(\theta)$$

Example: `se3([pi/2 pi/3 pi/4],"eul","ZYZ")` rotates a point by `pi/4` radians about the *z*-axis, then rotates the point by `pi/3` radians about the *y*-axis, and then rotates the point by `pi/2`

radians about the *z*-axis. This is equivalent to `se3(pi/2,"rotz") * se3(pi/3,"roty") * se3(pi/4,"rotz")`

Data Types: `string` | `char`

`quat` — Numeric quaternion
N-by-4 matrix

Numeric quaternion, specified as an *N*-by-4 matrix. *N* is the number of specified quaternions. Each row represents one quaternion of the form [*qw qx qy qz*], where *qw* is a scalar number.

If `quat` is an *N*-by-4 matrix, the resulting number of created `se3` objects is equal to *N*.

Note The `se3` object normalizes the input quaternions before converting the quaternions to a rotation matrix.

Example: `[0.7071 0.7071 0 0]`

Data Types: `single` | `double`

`axang` — Axis-angle rotation
N-by-4 matrix

Axis-angle rotation, specified as an *N*-by-4 matrix in the form [*x y z theta*]. *N* is the total number of axis-angle rotations. *x*, *y*, and *z* are vector components from the *x*-, *y*-, and *z*-axis, respectively. The vector defines the axis to rotate by the angle *theta*, in radians.

If `axang` is an *N*-by-4 matrix, the resulting number of created `se3` objects is equal to *N*.

Example: `[.2 .15 .25 pi/4]` rotates the axis, defined as `0.2` in the *x*-axis, `0.15` along the *y*-axis, and `0.25` along the *z*-axis, by `pi/4` radians.

Data Types: `single` | `double`

`angle` — Single-axis-angle rotation
N-by-*M* matrix

Single-axis-angle rotation, specified as an *N*-by-*M* matrix. Each element of the matrix is an angle, in radians, about the axis specified using the `axis` argument, and the `se3` object creates an `se3` object for each angle.

If `angle` is an *N*-by-*M* matrix, the resulting number of created `se3` objects is equal to *N*.

The rotation angle is counterclockwise positive when you look along the specified axis toward the origin.

Data Types: `single` | `double`

`axis` — Axis to rotate
`"rotx"` | `"roty"` | `"rotz"`

Axis to rotate, specified as one of these options:

- `"rotx"` — Rotate about the *x*-axis:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

- `"roty"` — Rotate about the *y*-axis:

$$R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}$$

- `"rotz"` — Rotate about the *z*-axis:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Use the `angle` argument to specify how much to rotate about the specified axis.

Example: `Rx = se3(phi,"rotx");`

Example: `Ry = se3(psi,"roty");`

Example: `Rz = se3(theta,"rotz");`

Data Types: `string` | `char`

pose — 3-D compact pose
N-by-7 matrix

3-D compact pose, specified as an *N*-by-7 matrix, where *N* is the total number of compact poses. Each row is a pose, comprised of a xyz position and quaternion, in the form [*x y z qw qx qy qz*]. *x*, *y*, and *z* are the positions in the *x*-, *y*-, and *z*-axes, respectively. *qw*, *qx*, *qy*, and *qz* together are the quaternion rotation.

If `pose` is an *N*-by-7 matrix, the resulting number of created `se3` objects is equal to *N*.

Data Types: `single` | `double`

Object Functions

Mathematical Operations

mtimes, *	Transformation or rotation multiplication
mrdivide, /	Transformation or rotation right division
rdivide, ./	Element-wise transformation or rotation right division
times, .*	Element-wise transformation or rotation multiplication

Utilities

interp	Interpolate between transformations
dist	Calculate distance between transformations
normalize	Normalize transformation or rotation matrix
transform	Apply rigid body transformation to points

Numerical Conversions

axang	Convert transformation or rotation into axis-angle rotations
eul	Convert transformation or rotation into Euler angles
rotm	Extract rotation matrix
quat	Convert transformation or rotation to numeric quaternion
trvec	Extract translation vector
tform	Extract homogeneous transformation
xyzquat	Convert transformation or rotation to compact 3-D pose representation

Object Conversions

so3	SO(3) rotation
quaternion	Create quaternion array
rigidtform3d	3-D rigid geometric transformation

Examples

Create SE(3) Transformation Using Euler Angles and Translation

Define an Euler-angle rotation of `[pi/2 0 pi/7]` with a `"XYZ"` rotation sequence, and a *xyz* translation of `[6 4 1]`.

```
angles = [pi/2 0 pi/7];
trvec = [6 4 1];
```

Create an SE(3) transformation using the Euler angles and the translation.

```
TF = se3(angles,"eul","XYZ",trvec)
```

```
TF = se3
    0.9010   -0.4339         0    6.0000
    0.0000    0.0000   -1.0000    4.0000
    0.4339    0.9010    0.0000    1.0000
         0         0         0    1.0000
```

Algorithms

3-D Homogeneous Transformation Matrix

3-D homogeneous transformation matrices consist of both an SO(3) rotation and an *xyz*-translation.

To read more about SO(3) rotations, see the 3-D Orthonormal Rotation Matrix on page 2-15 section of the `so3` object.

The translation is along the *x*-, *y*-, and *z*-axes as a three-element column vector:

$$t = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

The SO(3) rotation matrix R is applied to the translation vector t to create the homogeneous translation matrix T. The rotation matrix is present in the upper-left of the transformation matrix as 3-by-3 submatrix, and the translation vector is present as a three-element vector in the last column.

$$T = \begin{bmatrix} R & t \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} I_3 & t \\ 0_{1x3} & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0_{1x3} & 1 \end{bmatrix}$$

Version History

Introduced in R2026a

R2023a: New methods and syntaxes

`se3` supports new methods and syntaxes for converting to and from other transformations and rotations.

The new methods are:

- `axang`
- `eul`
- `so3`
- `quat`
- `quaternion`
- `xyzquat`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`so3` | `quaternion`

so3

SO(3) rotation

Description

The `so3` object represents an SO(3) rotation in 3-D in a right-handed Cartesian coordinate system.

The SO(3) rotation is a 3-by-3 orthonormal rotation matrix. For example, these are orthonormal rotation matrices for rotations of ϕ, ψ, and θ about the x-, y-, and z-axis, respectively:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For more information, see the 3-D Orthonormal Rotation Matrix section.

This object acts like a numerical matrix, enabling you to compose rotations using multiplication and division.

Creation

Syntax

```
rotation = so3
rotation = so3(rotation)
rotation = so3(quaternion)
rotation = so3(transformation)

rotation = so3(euler,"eul")
rotation = so3(euler,"eul",sequence)
rotation = so3(quat,"quat")
rotation = so3(axang,"axang")
rotation = so3(angle,axis)
```

Description

3-D Rotation Representations

`rotation = so3` creates an SO(3) rotation representing an identity rotation with no translation.

$$rotation = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

`rotation = so3(rotation)` creates an SO(3) rotation representing a pure rotation defined by the orthonormal rotation `rotation`.

$$rotation = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{11} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

`rotation = so3(quaternion)` creates an SO(3) rotation from the rotations defined by the quaternion `quaternion`.

`rotation = so3(transformation)` creates an SO(3) rotation from the SE(3) transformation `transformation`.

Other Numeric 3-D Rotation Representations

`rotation = so3(euler,"eul")` creates an SO(3) rotation from the rotations defined by the Euler angles `euler`.

`rotation = so3(euler,"eul",sequence)` specifies the sequence of the Euler angle rotations `sequence`. For example, the sequence `"ZYX"` rotates the *z*-axis, then the *y*-axis and *x*-axis.

`rotation = so3(quat,"quat")` creates an SO(3) rotation from the rotations defined by the numeric quaternions `quat`.

`rotation = so3(axang,"axang")` creates an SO(3) rotation from the rotations defined by the axis-angle rotation `axang`.

`rotation = so3(angle,axis)` creates an SO(3) rotation from the rotations `angle` about the rotation axis `axis`.

Note If any inputs contain more than one rotation, then the output `rotation` is an *N*-element array of `so3` objects corresponding to each of the *N* input rotations.

Input Arguments

`rotation` — Orthonormal rotation

3-by-3 matrix | 3-by-3-by-*N* matrix | `so3` object | *N*-element array of `so3` objects

Orthonormal rotation, specified as a 3-by-3 matrix, a 3-by-3-by*N* array, a scalar `so3` object, or an *N*-element array of `so3` objects. *N* is the total number of rotations.

If `rotation` is an array, the resulting number of created `so3` objects in the output array is equal to *N*.

Example: `eye(3)`

`transformation` — Homogeneous transformation

4-by-4 matrix | 4-by-4-by-*N* array | `se3` object | *N*-element array of `se3` objects

Homogeneous transformation, specified as a 4-by-4 matrix, a 4-by-4-by-*N* array, a scalar `se3` object, or an *N*-element array of `se3` objects. *N* is the total number of transformations specified.

If `transformation` is an array, the resulting number of created `so3` objects is equal to *N*.

Example: `eye(4)`

Data Types: `single` | `double`

`quaternion` — Quaternion
`quaternion` object | *N*-element array of `quaternion` objects

Quaternion, specified as a scalar `quaternion` object or as an *N*-element array of `quaternion` objects. *N* is the total number of specified quaternions.

If `quaternion` is an *N*-element array, the resulting number of created `so3` objects is equal to *N*.

Example: `quaternion(1,0.2,0.4,0.2)`

`euler` — Euler angles
N-by-3 matrix

Euler angles, specified as an *N*-by-3 matrix, in radians. Each row represents one set of Euler angles with the axis-rotation sequence defined by the `sequence` argument. The default axis-rotation sequence is *ZYX*.

If `euler` is an *N*-by-3 matrix, the resulting number of created `so3` objects is equal to *N*.

Example: `[pi/2 pi pi/4]`

Data Types: `single` | `double`

`sequence` — Axis-rotation sequence
`"ZYX"` (default) | `"ZYZ"` | `"ZXY"` | `"ZXZ"` | `"YXY"` | `"YZX"` | `"YXZ"` | `"YZY"` | `"XYX"` | `"XYZ"` | `"XZX"` | `"XZY"`

Axis-rotation sequence for the Euler angles, specified as one of these string scalars:

- `"ZYX"` (default)
- `"ZYZ"`
- `"ZXY"`
- `"ZXZ"`
- `"YXY"`
- `"YZX"`
- `"YXZ"`
- `"YZY"`
- `"XYX"`
- `"XYZ"`
- `"XZX"`
- `"XZY"`

These are orthonormal rotation matrices for rotations of ϕ, ψ, and θ about the *x*-, *y*-, and *z*-axis, respectively:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When constructing the rotation matrix from this sequence, each character indicates the corresponding axis. For example, if the sequence is `"XYZ"`, then the `so3` object constructs the

rotation matrix R by multiplying the rotation about x-axis with the rotation about the y-axis, and then multiplying that product with the rotation about the z-axis:

$$R = R_x(\phi) \cdot R_y(\psi) \cdot R_z(\theta)$$

Example: `so3([pi/2 pi/3 pi/4],"eul","ZYZ")` rotates a point by `pi/4` radians about the z-axis, then rotates the point by `pi/3` radians about the y-axis, and then rotates the point by `pi/2` radians about the z-axis. This is equivalent to `so3(pi/2,"rotz") * so3(pi/3,"roty") * so3(pi/4,"rotz")`

Data Types: `string` | `char`

`quat` — Numeric quaternion
N-by-4 matrix

Numeric quaternion, specified as an *N*-by-4 matrix. *N* is the number of specified quaternions. Each row represents one quaternion of the form [*qw qx qy qz*], where *qw* is a scalar number.

If `quat` is an *N*-by-4 matrix, the resulting number of created `so3` objects is equal to *N*.

Note The `so3` object normalizes the input quaternions before converting the quaternions to a rotation matrix.

Example: `[0.7071 0.7071 0 0]`

Data Types: `single` | `double`

`axang` — Axis-angle rotation
N-by-4 matrix

Axis-angle rotation, specified as an *N*-by-4 matrix in the form [*x y z theta*]. *N* is the total number of axis-angle rotations. *x*, *y*, and *z* are vector components from the *x*-, *y*-, and *z*-axis, respectively. The vector defines the axis to rotate by the angle *theta*, in radians.

If `axang` is an *N*-by-4 matrix, the resulting number of created `so3` objects is equal to *N*.

Example: `[.2 .15 .25 pi/4]` rotates the axis, defined as `0.2` in the *x*-axis, `0.15` along the *y*-axis, and `0.25` along the *z*-axis, by `pi/4` radians.

Data Types: `single` | `double`

`angle` — Single-axis-angle rotation
N-by-*M* matrix

Single-axis-angle rotation, specified as an *N*-by-*M* matrix. Each element of the matrix is an angle, in radians, about the axis specified using the `axis` argument, and the `so3` object creates an `so3` object for each angle.

If `angle` is an *N*-by-*M* matrix, the resulting number of created `so3` objects is equal to *N*.

The rotation angle is counterclockwise positive when you look along the specified axis toward the origin.

Data Types: `single` | `double`

`axis` — Axis to rotate
`"rotx"` | `"roty"` | `"rotz"`

Axis to rotate, specified as one of these options:

- `"rotx"` — Rotate about the x-axis:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

- `"roty"` — Rotate about the y-axis:

$$R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}$$

- `"rotz"` — Rotate about the z-axis:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Use the `angle` argument to specify how much to rotate about the specified axis.

Example: `Rx = so3(phi,"rotx");`

Example: `Ry = so3(psi,"roty");`

Example: `Rz = so3(theta,"rotz");`

Data Types: `string` | `char`

Object Functions

Mathematical Operations

mtimes, *	Transformation or rotation multiplication
mrdivide, /	Transformation or rotation right division
rdivide, ./	Element-wise transformation or rotation right division
times, .*	Element-wise transformation or rotation multiplication

Utilities

interp	Interpolate between transformations
dist	Calculate distance between transformations
normalize	Normalize transformation or rotation matrix
transform	Apply rigid body transformation to points

Numerical Conversions

axang	Convert transformation or rotation into axis-angle rotations
eul	Convert transformation or rotation into Euler angles
rotm	Extract rotation matrix
quat	Convert transformation or rotation to numeric quaternion
trvec	Extract translation vector

tform	Extract homogeneous transformation
xyzquat	Convert transformation or rotation to compact 3-D pose representation

Object Conversions

se3	SE(3) homogeneous transformation
quaternion	Create quaternion array
rigidtform3d	3-D rigid geometric transformation

Examples

Convert SO(3) Rotation to Euler Angles

Create SO(3) rotation defined by a Euler angles.

```
eul1 = [pi/4 pi/3 pi/8]
```

```
eul1 = 1×3

    0.7854    1.0472    0.3927
```

```
R = so3(eul1,"eul")
```

```
R = so3
    0.3536   -0.4189    0.8364
    0.3536    0.8876    0.2952
   -0.8660    0.1913    0.4619
```

Get the Euler angles from the transformation.

```
eul2 = eul(R)
```

```
eul2 = 1×3

    0.7854    1.0472    0.3927
```

Algorithms

3-D Orthonormal Rotation Matrix

SO(3) rotation matrices are 3-by-3 orthonormal matrices that represent any rotation in 3-D Euclidean space. SO(3) rotations have many special properties. For example, SO(3) rotation matrices are in the 3-D special orthogonal group, so the product of two SO(3) rotation matrices is an SO(3) rotation matrix. This enables you to compose rotations from multiple rotations. For example, these are orthonormal rotation matrices for rotations of ϕ, ψ, and θ about the x-, y-, and z-axis, respectively:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Depending on the order in which you multiply these *x*-, *y*-, and *z*-axis rotations, you can construct compound matrices that represent any rotation in 3-D Euclidean space.

These are other properties of SO(3) rotations:

- Each column is both orthogonal and a unit vector, meaning that none of the columns are multiples of each other.
- The determinant of the matrix is positive 1.
- The inverse of the matrix is the same as the transpose of the matrix: $R^{-1} = R^T$.

Version History

Introduced in R2026a

R2023a: New methods and syntaxes

`so3` supports new methods and syntaxes for converting to and from other transformations and rotations.

The new methods are:

- `axang`
- `eul`
- `quat`
- `quaternion`
- `tform`
- `trvec`
- `xyzquat`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`se3` | `quaternion`

groundingDinoObjectDetector

Detect and localize objects using Grounding DINO object detector

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for Grounding DINO Object Detection add-on.

The `groundingDinoObjectDetector` object creates a Grounding DINO vision-language object detector for zero-shot object detection. This detector identifies and localizes arbitrary objects in an image using natural language descriptions (text prompts). Use the `groundingDinoObjectDetector` object to create a pretrained Grounding DINO object detector with either a Swin-Tiny or Swin-Base backbone network. You can then use the `detect` function of the `groundingDinoObjectDetector` object to detect objects in an unknown image. This feature also requires the Deep Learning Toolbox™ license.

Creation

Syntax

```
detector = groundingDinoObjectDetector
detector = groundingDinoObjectDetector(name)
detector = groundingDinoObjectDetector( ___ ,Name=Value)
```

Description

`detector = groundingDinoObjectDetector` creates a pretrained Grounding DINO object detector by using Swin-Tiny as the backbone network.

`detector = groundingDinoObjectDetector(name)` creates a pretrained Grounding DINO object detector using the backbone network specified by `name`.

`detector = groundingDinoObjectDetector( ___ ,Name=Value)` sets writable properties using one or more name-value arguments. Use this syntax to set the `ClassNames` and `ClassDescriptions` properties of the object detector.

You can use the `ClassDescriptions` property to provide natural language queries for detection and the `ClassNames` property to assign output labels for annotation. The `ClassNames` property is required. If you specify only `ClassNames`, the object uses its value for both detection and annotation. You can specify `ClassDescriptions` in addition to `ClassNames` if you want to provide more details about the objects to detect.

You can specify these properties either when creating the object or as name-value arguments when calling the `detect` function.

- If you do not specify these properties at the time of object creation, you must specify at least the `ClassNames` name-value argument when you call the `detect` function.
- If you specify these properties at object creation and when calling the `detect` function, the values specified in the `detect` function override the values set in the detector object.

Input Arguments

`name` — Name of backbone network

`"swin-tiny"` (default) | `"swin-base"`

Name of the backbone network, specified as `"swin-tiny"` or `"swin-base"`.

- Use `"swin-tiny"` as the backbone for applications that require low latency and resource efficiency, such as in real-time or edge deployments. The network offers faster inference with lower memory usage than the `"swin-base"` network.
- Use `"swin-base"` for applications that require high accuracy in environments with sufficient computational resources.

The pretrained detector with Swin-Tiny as the backbone network is trained on the O365, GoldG, and Cap4M datasets. The pretrained detector with Swin-Base as the backbone network is trained on the COCO, O365, GoldG, Cap4M, OpenImage, ODinW-35, and RefCOCO datasets.

This argument sets the `ModelName` property as a character vector.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

`ClassNames` — Labels

string scalar | string array | cell array of character vectors | categorical vector

Labels, specified as a string scalar, string array, cell array of character vectors, or categorical vector.

- If you do not specify `ClassDescriptions`, the object uses the values in `ClassNames` as both output labels for annotation and natural language queries for detection. In this case, the total number of words across all elements, with each comma also counted as a word, must not exceed 255 words.
- If you specify `ClassDescriptions`, the object uses the values in `ClassNames` as output labels for annotation, and the word limit does not apply.

If you specify both `ClassNames` and `ClassDescriptions`, the number of elements in them must match.

This argument sets the `ClassNames` property.

Data Types: `char` | `string` | `cell` | `categorical`

`ClassDescriptions` — Natural language queries

string scalar | string array | cell array of character vectors | categorical vector

Natural language queries, specified as a string scalar, string array, cell array of character vectors, or categorical vector. The total number of words across all elements, with each comma also counted as a word, must not exceed 255 words. Each entry must correspond to an element in `ClassNames`. This argument sets the `ClassDescriptions` property.

In general, the `ClassDescriptions` are more detailed and descriptive than the `ClassNames` to guide the detection process. The queries provide additional context about an object, such as its appearance, color, size, or activity.

For an example, if you want to detect a brown-colored dog lying on grass, you can specify `ClassDescriptions` as `"brown dog lying on grass"` to query the Grounding DINO object detector, and specify the corresponding `ClassNames` as `"dog"` to label the detected object.

Data Types: `char` | `string` | `cell` | `categorical`

Properties

`ModelName` — Name of backbone network

`'swin-tiny'` (default) | `'swin-base'`

This property is read-only after object creation.

Name of the backbone network, represented as `'swin-tiny'` or `'swin-base'`.

Data Types: `char`

`ClassNames` — Labels

`[]` (default) | string scalar | string array | cell array of character vectors | categorical vector

This property is read-only after object creation.

Labels, specified as a string scalar, string array, cell array of character vectors, or categorical vector. The default value is `[]`. To set this property, specify the property value using the `ClassNames` name, value argument .

`ClassDescriptions` — Natural language queries

`[]` (default) | string scalar | string array | cell array of character vectors | categorical vector

This property is read-only after object creation.

Natural language queries, represented as a string scalar, string array, cell array of character vectors, or categorical vector. The default value is `[]`. To set this property, specify the property value using the `ClassDescriptions` name, value argument .

If you do not specify `ClassDescriptions` property, the object uses the values in `ClassNames` as both output labels for annotation and natural language queries for detection.

Object Functions

detect Detect objects using Grounding DINO object detector

Examples

Perform Zero-Shot Object Detection Using Grounding DINO

Read an input image into the workspace.

```
I = imread("visionteam.jpg");
```

Display the input image.

```
figure
imshow(I)
```



Create a Grounding DINO object detector using the Swin-Base network as the backbone network.

```
name = "swin-base";
detector = groundingDinoObjectDetector(name);
```

Specify the class names for the detector to use as output labels for the detection results.

```
labels = {'Holding paper','Holding jacket'};
```

Specify the class descriptions for the detector to use as text queries for performing object detection.

```
descriptions = {'Person holding paper','Person holding jacket'};
```

Detect objects in the image using the specified class names and descriptions.

```
[bboxes,scores,labels] = detect(detector,I,ClassNames=labels,ClassDescriptions=descriptions);
```

Format the detected labels and scores for image annotation.

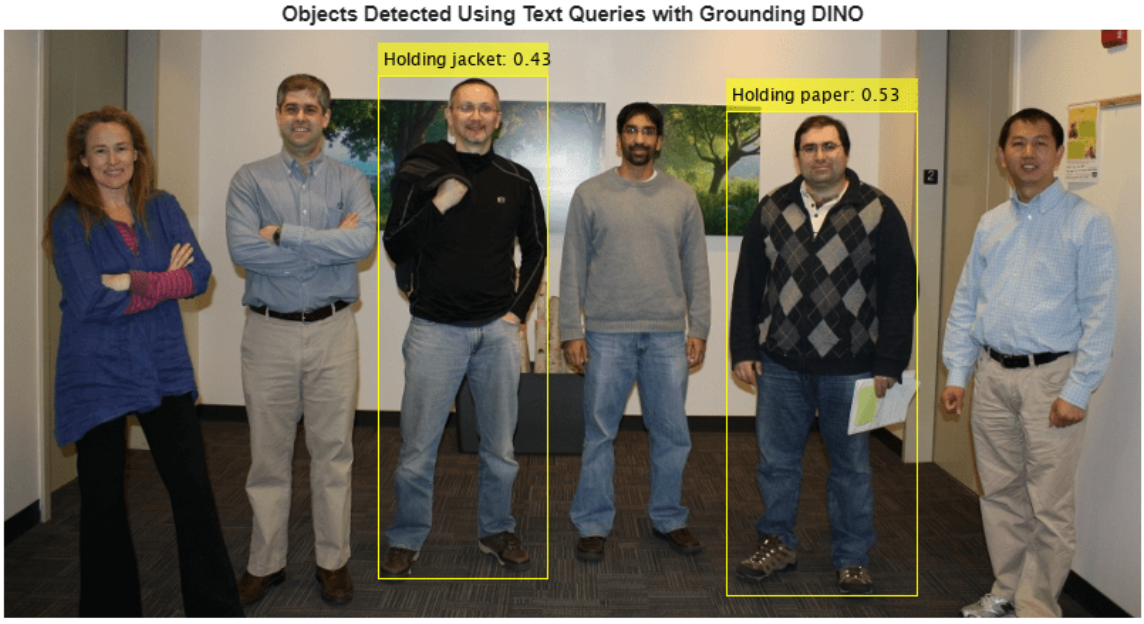
```
outputLabels = compose("%s: %.2f",string(labels),scores);
```

Annotate the detected objects in the image.

```
detections = insertObjectAnnotation(I,"rectangle",bboxes,outputLabels);
```

Display the image, annotated with the detection results.

```
imshow(detections)
title("Objects Detected Using Text Queries with Grounding DINO")
```



Detect License Plates Using Grounding DINO

This example uses a small vehicle dataset that contains 295 images. Many of these images come from the Caltech Cars 1999 and 2001 datasets, available at the Caltech Computational Vision website created by Pietro Perona and used with permission.

Unzip the vehicle images to the working folder.

```
fileNames = unzip("vehicleDatasetImages.zip");
```

Create an `imageDatastore` object to read the images for object detection.

```
imds = imageDatastore(fileNames);
```

Load a pretrained Grounding DINO object detector with a Swin-Base backbone network. Use the `classNames` name-value argument to specify text prompts for detecting the car and the license plate.

To improve detection accuracy and establish semantic context, specify both license plate and car as text prompts. This ensures that language-guided query selection correctly assigns the vehicle's large-scale features to car, preventing them from being falsely localized as the license plate.

```
detector = groundingDinoObjectDetector("swin-base",classNames=["License plate","car"]);
```

Read images from the image datastore using the `read` function. Detect cars and license plates in each image using the `detect` function of the `groundingDINOObjectDetector` object.

```
detectionResults = detect(detector,imds);
```

Visualize Detection Results

Extract the bounding boxes, detection scores, and labels from the results table. Iterate over the images in the datastore and filter detections to include only license plates. For each image, display the image and overlay bounding boxes with the corresponding attention scores when license plates are detected.

```
figure
allBoxes  = detectionResults.Boxes;
allScores = detectionResults.Scores;
allLabels = detectionResults.Labels;
for i = 1:length(imds.Files)
    img = readimage(imds,i);
    idx = (allLabels{i} == "License plate");
    plateBoxes = allBoxes{i}(idx,:);
    plateScores = cellstr(string(allScores{i}(idx)));

    imshow(img);

    if isempty(plateBoxes)
        title(sprintf("Image %d: No license plate detections",i));
    else
        annotatedImg = insertObjectAnnotation(img,'rectangle',plateBoxes,plateScores,...
            'LineWidth',3);
        imshow(annotatedImg);
        title(sprintf("Image %d: License plate(s) detected",i));
    end
    pause(0.1)
end
```



Version History

Introduced in R2026a

References

[1] Liu, Shilong, Zhaoyang Zeng, Tianhe Ren, et al. “Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection.” In *Computer Vision – ECCV 2024*, vol. 15105, edited by Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol. Springer Nature Switzerland, 2025. https://doi.org/10.1007/978-3-031-72970-6_3.

See Also

`rtmdetObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector`

Topics

“Get Started with Vision-Language Models”
“Automatically Label Ground Truth Using Vision-Language Model”

detect

Detect objects using Grounding DINO object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
detectionResults = detect(detector,ds)
[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects in an image or an array of images, `I`, using a Grounding DINO object detector, `detector`. The function returns the locations of objects detected in the input image as sets of bounding boxes.

`[bboxes,scores] = detect(detector,I)` also returns the class-specific attention scores for each bounding box.

`[bboxes,scores,labels] = detect(detector,I)` returns a categorical array of the labels assigned to the bounding boxes.

`detectionResults = detect(detector,ds)` detects objects in all the images stored in the datastore `ds`.

`[___] = detect(___,roi)` detects objects within the rectangular search region `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments.

Examples

Perform Zero-Shot Object Detection Using Grounding DINO

Read an input image into the workspace.

```
I = imread("visionteam.jpg");
```

Display the input image.

```
figure
imshow(I)
```



Create a Grounding DINO object detector using the Swin-Base network as the backbone network.

```
name = "swin-base";
detector = groundingDinoObjectDetector(name);
```

Specify the class names for the detector to use as output labels for the detection results.

```
labels = {'Holding paper','Holding jacket'};
```

Specify the class descriptions for the detector to use as text queries for performing object detection.

```
descriptions = {'Person holding paper','Person holding jacket'};
```

Detect objects in the image using the specified class names and descriptions.

```
[bboxes,scores,labels] = detect(detector,I,ClassNames=labels,ClassDescriptions=descriptions);
```

Format the detected labels and scores for image annotation.

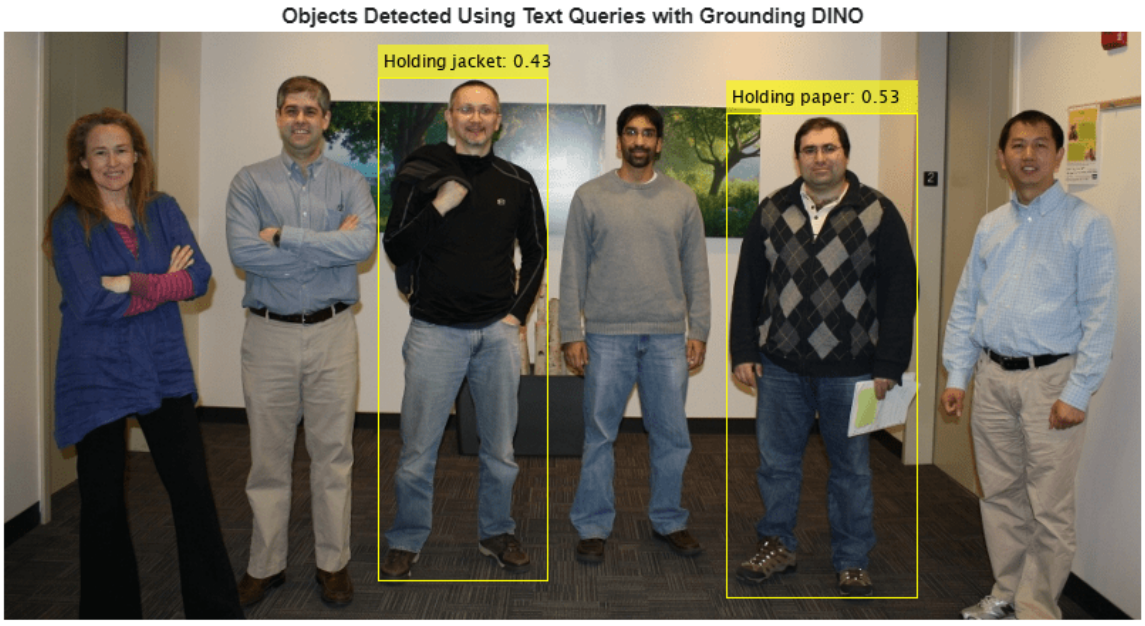
```
outputLabels = compose("%s: %.2f",string(labels),scores);
```

Annotate the detected objects in the image.

```
detections = insertObjectAnnotation(I,"rectangle",bboxes,outputLabels);
```

Display the image, annotated with the detection results.

```
imshow(detections)
title("Objects Detected Using Text Queries with Grounding DINO")
```



Detect License Plates Using Grounding DINO

This example uses a small vehicle dataset that contains 295 images. Many of these images come from the Caltech Cars 1999 and 2001 datasets, available at the Caltech Computational Vision website created by Pietro Perona and used with permission.

Unzip the vehicle images to the working folder.

```
fileNames = unzip("vehicleDatasetImages.zip");
```

Create an `imageDatastore` object to read the images for object detection.

```
imds = imageDatastore(fileNames);
```

Load a pretrained Grounding DINO object detector with a Swin-Base backbone network. Use the `classNames` name-value argument to specify text prompts for detecting the car and the license plate.

To improve detection accuracy and establish semantic context, specify both license plate and car as text prompts. This ensures that language-guided query selection correctly assigns the vehicle's large-scale features to car, preventing them from being falsely localized as the license plate.

```
detector = groundingDinoObjectDetector("swin-base",classNames=["License plate","car"]);
```

Read images from the image datastore using the `read` function. Detect cars and license plates in each image using the `detect` function of the `groundingDINOObjectDetector` object.

```
detectionResults = detect(detector,imds);
```

Visualize Detection Results

Extract the bounding boxes, detection scores, and labels from the results table. Iterate over the images in the datastore and filter detections to include only license plates. For each image, display the image and overlay bounding boxes with the corresponding attention scores when license plates are detected.

```
figure
allBoxes  = detectionResults.Boxes;
allScores = detectionResults.Scores;
allLabels = detectionResults.Labels;
for i = 1:length(imds.Files)
    img = readimage(imds,i);
    idx = (allLabels{i} == "License plate");
    plateBoxes = allBoxes{i}(idx,:);
    plateScores = cellstr(string(allScores{i}(idx)));

    imshow(img);

    if isempty(plateBoxes)
        title(sprintf("Image %d: No license plate detections",i));
    else
        annotatedImg = insertObjectAnnotation(img,'rectangle',plateBoxes,plateScores,...
            'LineWidth',3);
        imshow(annotatedImg);
        title(sprintf("Image %d: License plate(s) detected",i));
    end
    pause(0.1)
end
```



Input Arguments

`detector` — Grounding DINO object detector

`groundingDinoObjectDetector` object

Grounding DINO object detector, specified as a `groundingDinoObjectDetector` object.

`I` — Test image

matrix | 3-D array

Test image, specified as a matrix for a grayscale image or a 3-D array of size *height*-by-*width*-by-3 for an RGB image.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Test images

`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Test images, specified as an `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing the full paths of the test images. The images in the datastore must be grayscale or RGB images.

`roi` — Search region of interest

vector of form [*x y width height*]

Search region of interest, specified as a vector of form [*x y width height*]. The values of *x* and *y* specify the coordinates of the upper-left corner of a rectangular region, and *width* and *height* specify its size in pixels.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,ExecutionEnvironment="gpu")` to use the GPU to run the detector.

`ClassNames` — Labels

string scalar | string array | cell array of character vectors | categorical vector

Labels, specified as a string scalar, string array, cell array of character vectors, or categorical vector.

- If you do not specify `ClassDescriptions`, the object uses the values in `ClassNames` as both output labels for annotation and natural language queries for detection. In this case, the total number of words across all elements, with each comma also counted as a word, must not exceed 255 words.
- If you specify `ClassDescriptions`, the object uses the values in `ClassNames` as output labels for annotation, and the word limit does not apply.

If you specify both `ClassNames` and `ClassDescriptions`, the number of elements in them must match.

This argument sets the `ClassNames` property.

Data Types: `char` | `string` | `cell` | `categorical`

`ClassDescriptions` — Natural language queries

string scalar | string array | cell array of character vectors | categorical vector

Natural language queries, specified as a string scalar, string array, cell array of character vectors, or categorical vector. The total number of words across all elements, with each comma also counted as a word, must not exceed 255 words. Each entry must correspond to an element in `ClassNames`. This argument sets the `ClassDescriptions` property.

In general, the `ClassDescriptions` are more detailed and descriptive than the `ClassNames` to guide the detection process. The queries provide additional context about an object, such as its appearance, color, size, or activity.

For an example, if you want to detect a brown-colored dog lying on grass, you can specify `ClassDescriptions` as `"brown dog lying on grass"` to query the Grounding DINO object detector, and specify the corresponding `ClassNames` as `"dog"` to label the detected object.

Data Types: `char` | `string` | `cell` | `categorical`

`MinSize` — Minimum region size

`[1 1]` (default) | vector of form [*height width*]

Minimum region size, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest region that can contain the object.

`MaxSize` — Maximum region size

`size(I)` (default) | vector of the form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest region that contain the object.

By default, `MaxSize` is set to the height and width of the input image `I`. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test image.

`MiniBatchSize` — Minimum batch size
4 (default) | positive integer

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of image. Images are grouped into minibatches and processed as a batch, which can improve computational efficiency at the cost of increased memory demand.

`Threshold` — Detection threshold
`0.25` (default) | scalar in range [0, 1]

Detection threshold, specified as a scalar in the range [0, 1]. The function removes detections that have attention scores less than this threshold value. To reduce false positives, increase this value.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Returns only the strongest bounding box for each object. After object detection, the `detect` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their attention scores.

  By default, the `detect` function make this call to the `selectStrongestBboxMulticlass` function:

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.5);
  ```

- `false` — Returns all detected bounding boxes. You can then write your own custom function to eliminate overlapping bounding boxes.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have a Parallel Computing Toolbox™ license and a CUDA enabled NVIDIA® GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

Output Arguments

`bboxes` — Locations of detected objects
M-by-4 matrix

Locations of the detected objects within the input image, returned as an *M*-by-4 matrix. *M* is the number of bounding boxes detected in the image. Each row of the matrix is of the form [*x y width height*]. The *x* and *y* values specify the coordinates of the upper-left corner, and *width* and *height* specify the size, of the corresponding bounding box, in pixels.

`scores` — Attention scores
M-by-1 numeric vector

Attention scores for each bounding box, returned as an *M*-by-1 numeric vector. *M* is the number of bounding boxes detected in the image. Each element indicates the attention score for a bounding box in the corresponding image, and values are in the range [0, 1].

`labels` — Labels for bounding boxes
M-by-1 categorical vector

Labels for bounding boxes, returned as an *M*-by-1 categorical vector. *M* is the number of bounding boxes detected in the image.

`detectionResults` — Detection results
three-column table

Detection results when the input is a datastore, `ds`, returned as a table with these columns, in which each row corresponds to an image.

`bboxes`	`scores`	`labels`
Predicted bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y width height*], where: • *M* is the number of detections in the image. • *x* and `y` specify the coordinates of the upper-left corner of the rectangle, in pixels. • *width* specifies the width of the rectangle, which is its length along the *x*-axis, in pixels. • *height* specifies the height of the rectangle, which is its length along the *y*-axis, in pixels.	Attention scores for each bounding box, returned as an *M*-by-1 numeric vector with values in the range [0, 1].	Labels assigned to the bounding boxes, returned as an *M*-by-1 categorical vector.

Version History

Introduced in R2026a

References

[1] Liu, Shilong, Zhaoyang Zeng, Tianhe Ren, et al. “Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection.” In *Computer Vision – ECCV 2024*, vol. 15105, edited by Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol. Springer Nature Switzerland, 2025. https://doi.org/10.1007/978-3-031-72970-6_3.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`rtmdetObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector` | `groundingDinoObjectDetector`

Topics

“Get Started with Vision-Language Models”
“Automatically Label Ground Truth Using Vision-Language Model”

faceDetector

Detect faces using pretrained RetinaFace face detector

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for RetinaFace Face Detection add-on.

The `faceDetector` function object stores a pretrained RetinaFace face detector used to detect faces in images. The RetinaFace face detector is trained on the WIDER FACE data set. When creating the `faceDetector` object, you can load a RetinaFace face detector that uses either MobileNet-0.25 or ResNet-50 as the backbone network.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for RetinaFace Face Detection. You can install the Computer Vision Toolbox Model for RetinaFace Face Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = faceDetector
detector = faceDetector(name)
detector = faceDetector( ___ ,InputSize=inputSize)
```

Description

`detector = faceDetector` returns a RetinaFace face detector object that uses MobileNet-0.25 as the backbone network. The lightweight architecture of MobileNet-0.25 enables the face detector to maintain a good balance between detection accuracy and computational efficiency.

`detector = faceDetector(name)` returns a pretrained RetinaFace face detector object that uses MobileNet-0.25 or ResNet-50 as the backbone network. You can specify the name of the face detector by using the input argument `name`, which determines the choice of backbone network.

`detector = faceDetector( ___ ,InputSize=inputSize)` specifies the image size to use for inference in addition to any combination of input arguments from previous syntaxes.

Input Arguments

`name` — Name of RetinaFace face detector

`"small-network"` | `"large-network"`

Name of the RetinaFace face detector, specified as one of these values.

- `"small-network"` — Specify this value to use MobileNet-0.25 as the backbone network.

The small network, with its few layers, provides faster inference times due to its reduced parameters and computational complexity. Use the small network to detect faces in images with simple and uniform backgrounds, where facial features are distinguishable. Additionally, you can use the small network to detect faces in large batches of images, where detection speed and computational efficiency are crucial.

- `"large-network"` — Specify this value to use ResNet-50 as the backbone network.

  The large network, with its numerous layers, provides higher accuracy in detection results at the cost of reduced computational efficiency. Use the large network to achieve high accuracy in detecting faces within images that have complex or cluttered backgrounds, as well as in images containing numerous faces.

The `name` argument sets the `ModelName` property.

Data Types: `char` | `string`

`inputSize` — Inference image size
`[640 640]` (default) | two-element row vector of integers

Inference image size, specified as a two-element row vector of integers of the form `[H W]`. H and W correspond to the height and width of the input image, respectively.

If the resolution of the face class to be detected in the test image is too small, increase the `inputSize` value for better detection accuracy. However, increasing the `inputSize` value also increases the computational cost.

The `inputSize` argument sets the `InputSize` property.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Properties

`ModelName` — Name of RetinaFace face detector
`'small-network'` (default) | `'large-network'`

Name of the RetinaFace face detector, represented as `'small-network'` or `'large-network'`. To set this property, use the `name` argument, when calling the `faceDetector` function.

This property is read-only after object creation.

`ClassNames` — Name of object class
`'face'`

This property is read-only.

Name of the object class to detect, represented as `'face'`.

`InputSize` — Image size used for inference
`[640 640]` (default) | two-element row vector of integers

Image size used for inference, represented as a two-element row vector of integers of the form `[H W]`. H and W correspond to the height and width of the input image, respectively.

This property is read-only after object creation. By default, the `InputSize` property value is set to the network's input size `[640 640]`, which is the size of the training images. To set this property, use the `inputSize` argument when calling the `faceDetector` function.

The `detect` function resizes the test image to the dimensions specified by the `InputSize` property before performing the detections.

Object Functions

detect Detect faces in images using deep learning based face detector

Examples

Detect Faces in Image Using RetinaFace Detector

Read an input image into the MATLAB® workspace.

```
I = imread("visionteam1.jpg");
```

Create a face detector object using the `faceDetector` function. The default configuration of the object loads a small, pretrained RetinaFace deep learning detector for face detection. The small network uses MobileNet-0.25 as the backbone network.

```
detector = faceDetector
```

```
detector =
  faceDetector with properties:

     ModelName: "small-network"
    ClassNames: face
     InputSize: [640 640]
```

Detect faces in the image using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes, detection scores, and labels for the detected faces.

```
[bboxes,scores,labels] = detect(detector,I);
```

Overlay bounding boxes, labels, and scores on the image using the `insertObjectAnnotation` function.

```
detectedImg = insertObjectAnnotation(I,"rectangle",bboxes,scores);
```

Display the detection results.

```
table(bboxes,scores,labels)
```

```
ans=6×3 table
             bboxes              scores     labels
    ____________________________    _______    ______

    571     70     38     61    0.99237     face
    333     99     31     40    0.98568     face
    217    122     21     31    0.96641     face
    107    124     37     47    0.99897     face
    510    128     29     38    0.97089     face
```

```
    648    132     26     38    0.99424     face
```

```
figure
imshow(detectedImg)
```



Detect Faces in Video Using RetinaFace Detector

Create a face detector object using the `faceDetector` function. Specify the detector name as "`large-network`". This configuration loads a pretrained RetinaFace face detector with ResNet-50 as the backbone network for face detection.

```
detector = faceDetector("large-network");
```

Create a `VideoReader` object to read video data from a video file.

```
reader = VideoReader('handshake_right.avi');
```

Configure a `VideoPlayer` object to display the video frames and the face detection results.

```
videoPlayer = vision.VideoPlayer(Position=[0 0 400 400]);
```

Read and iterate over each frame in the video using a `while` loop. Perform these steps to detect faces and display the detection results.

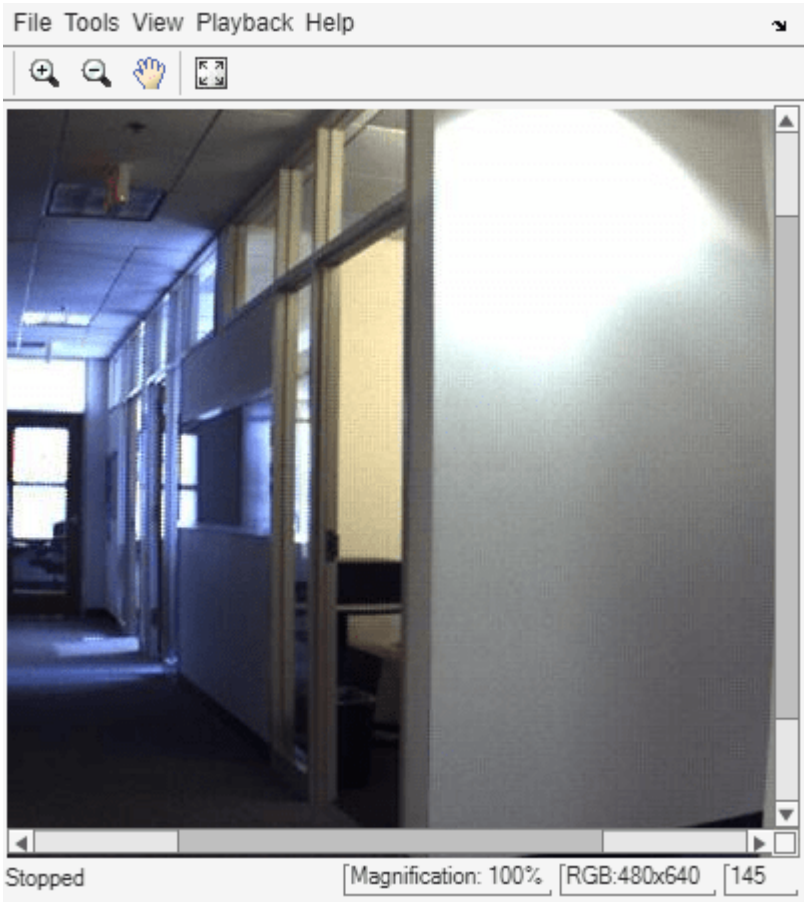
- Step 1: Read the current video frame with the `readFrame` function of the `VideoReader` object.
- Step 2: Detect faces in the video frame using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes and detection scores for the detected faces.
- Step 3: Overlay bounding boxes and scores on the video frame using the `insertObjectAnnotation` function.

- Step 4: Display the annotated frame using the `step` function of the `VideoPlayer` object.

```
while hasFrame(reader)
    % Step 1 %
    videoFrame = readFrame(reader);
    % Step 2 %
    [bboxes,scores] = detect(detector,videoFrame);
    % Step 3 %
    videoFrame= insertObjectAnnotation(videoFrame,"rectangle",bboxes,scores);
    % Step 4 %
    step(videoPlayer,videoFrame)
end
```

Call the `release` function to free up the resources allocated to the `VideoPlayer` object.

```
release(videoPlayer)
```



Detect and Blur Faces in Video

Create a face detector object using the `faceDetector` function. By default, the function uses the RetinaFace detector with a small backbone network for face detection.

```
detector = faceDetector;
```

Create a `VideoReader` object to read video data from a video file.

```
reader = VideoReader('tilted_face.avi');
```

Configure a `VideoPlayer` object to display the video frames and the face detection results.

```
videoPlayer = vision.VideoPlayer(Position=[0 0 600 600]);
```

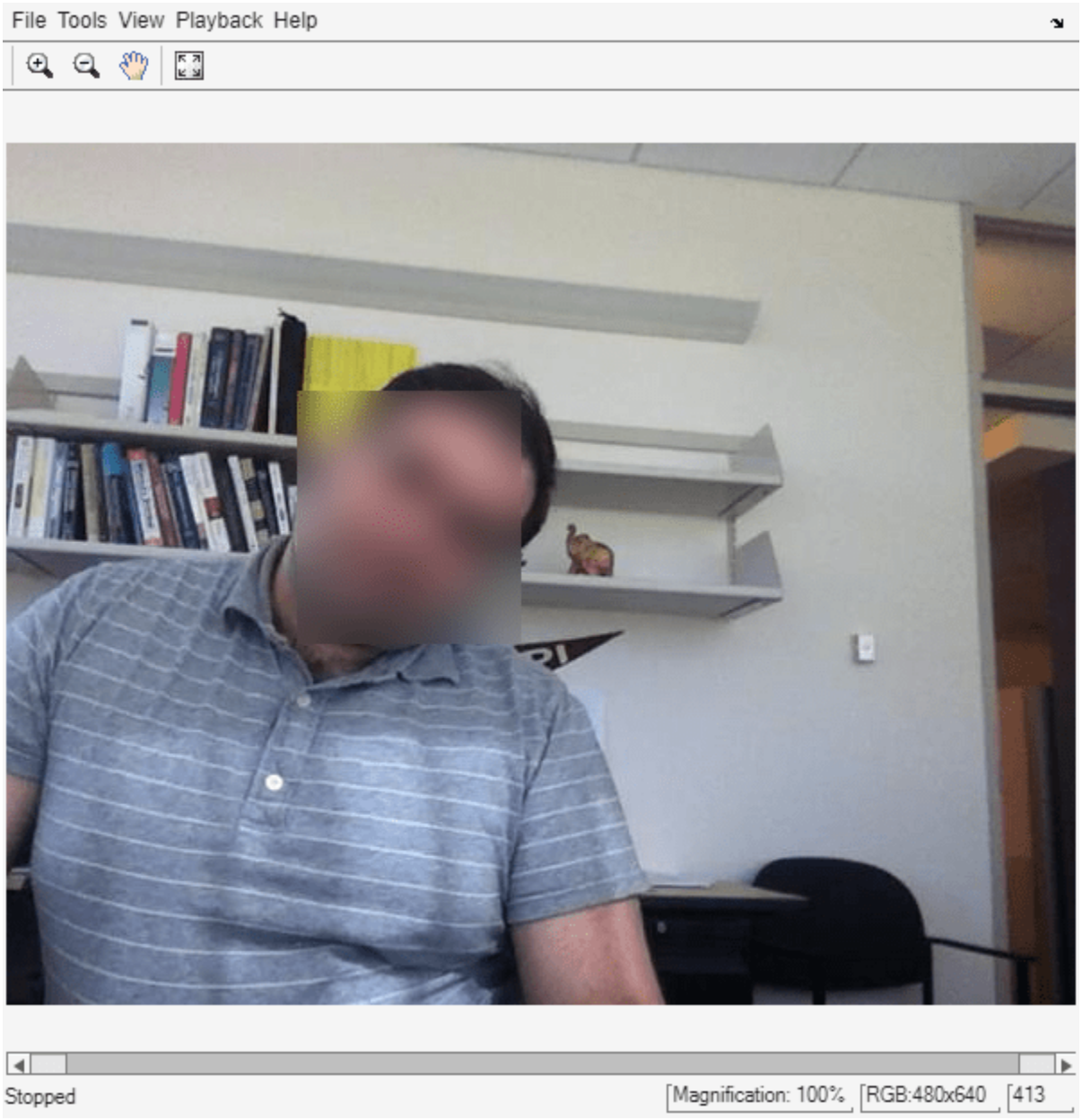
Read and iterate over each frame in the video using a `while` loop. Perform these steps to detect faces and display the detection results.

- Step 1: Read the current video frame with the `readFrame` function of the `VideoReader` object.
- Step 2: Detect faces in the video frame using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes for the detected faces.
- Step 3: If the current frame has detected faces, use the helper function `helperBlurFaces` to blur the faces in the frame. The helper function applies Gaussian filtering to blur the areas of the frame defined by the bounding boxes. This effectively obscures the detected faces.
- Step 4: Display the processed frame using the `step` function of the `VideoPlayer` object.

```
while hasFrame(reader)
    % Step 1 %
    videoFrame = readFrame(reader);
    % Step 2 %
    bboxes = detect(detector,videoFrame,Threshold=0.2);
    % Step 3 %
    if ~isempty(bboxes)
        videoFrame = helperBlurFaces(videoFrame,bboxes);
    end
    % Step 4 %
    step(videoPlayer,videoFrame)
end
```

Call the `release` function to free up the resources allocated to the `VideoPlayer` object.

```
release(videoPlayer)
```



The `helperBlurFaces` function applies Gaussian filtering to the regions defined by each bounding box, which correspond to detected faces.

```
function I = helperBlurFaces(I,bbox)
for j=1:size(bbox,1)
    xbox = round(bbox(j,:));
    subImage = imcrop(I,xbox);
    blurred = imgaussfilt(subImage,12);
    I(xbox(2):xbox(2)+xbox(4),xbox(1):xbox(1)+xbox(3),1:end) = blurred;
end
end
```

Version History

Introduced in R2025a

References

[1] Deng, Jiankang, Jia Guo, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. “RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild.” In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5202–11. Seattle, WA, USA: IEEE, 2020. https://doi.org/10.1109/CVPR42600.2020.00525.

[2] Yang, Shuo, Ping Luo, Chen Change Loy, and Xiaoou Tang. “WIDER FACE: A Face Detection Benchmark.” In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5525–33. Las Vegas, NV, USA: IEEE, 2016. https://doi.org/10.1109/CVPR.2016.596.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `faceDetector` object supports the generation of generic code only, which does not depend on any third-party libraries.
- The `roi` argument of the `detect` object function must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- The `AutoResize` name-value argument of the `detect` object function must be a code generation constant (`coder.const()`).
- The `detect` object function supports only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value arguments.
- The `detect` object function does not support code generation for the `ds` input argument, which specifies a datastore of test images.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`vision.CascadeObjectDetector` | `peopleDetector`

Topics

“Choose an Object Detector”
“Code Generation for People Detection Using Deep Learning”

detect

Detect faces in images using deep learning based face detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
detectionResults = detect(detector,ds)
[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for RetinaFace Face Detection add-on.

`bboxes = detect(detector,I)` detects faces within a single image or an array of images, `I`, using a pretrained RetinaFace face detector, `detector`. The `detect` function returns the locations of detected faces in the input image as a set of bounding boxes.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for RetinaFace Face Detection. You can install the Computer Vision Toolbox Model for RetinaFace Face Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[bboxes,scores] = detect(detector,I)` returns the confidence scores for the bounding boxes along with their locations.

`[bboxes,scores,labels] = detect(detector,I)` also returns a categorical array of the labels assigned to the bounding boxes.

`detectionResults = detect(detector,ds)` returns a table containing the predicted face bounding boxes, their associated confidence scores, and the corresponding labels for all the images in the input datastore `ds`.

`[___] = detect(___,roi)` detects faces within the rectangular region of interest `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments. For example, `Threshold=0.75` specifies a detection threshold of 0.75.

Examples

Detect Faces in Image Using RetinaFace Detector

Read an input image into the MATLAB® workspace.

```
I = imread("visionteam1.jpg");
```

Create a face detector object using the `faceDetector` function. The default configuration of the object loads a small, pretrained RetinaFace deep learning detector for face detection. The small network uses MobileNet-0.25 as the backbone network.

```
detector = faceDetector
```

```
detector =
  faceDetector with properties:

     ModelName: "small-network"
    ClassNames: face
     InputSize: [640 640]
```

Detect faces in the image using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes, detection scores, and labels for the detected faces.

```
[bboxes,scores,labels] = detect(detector,I);
```

Overlay bounding boxes, labels, and scores on the image using the `insertObjectAnnotation` function.

```
detectedImg = insertObjectAnnotation(I,"rectangle",bboxes,scores);
```

Display the detection results.

```
table(bboxes,scores,labels)
```

```
ans=6×3 table
             bboxes                scores     labels
    ___________________________    _______    ______

    571     70     38     61    0.99237     face
    333     99     31     40    0.98568     face
    217    122     21     31    0.96641     face
    107    124     37     47    0.99897     face
    510    128     29     38    0.97089     face
    648    132     26     38    0.99424     face
```

```
figure
imshow(detectedImg)
```



Detect Faces Within Region of Interest

Create a face detector object using the `faceDetector` function. Specify the detector name as "`large-network`". This configuration loads a pretrained RetinaFace face detector with ResNet-50 as the backbone network for face detection. The network has many layers and offers improved detection accuracy.

```
detector = faceDetector("large-network")
```

```
detector =
  faceDetector with properties:

     ModelName: "large-network"
    ClassNames: face
     InputSize: [640 640]
```

Read an input image into the MATLAB® workspace.

```
I = imread("boats.png");
```

Specify a region of interest (ROI) in the image to detect faces.

```
roi = [5 400 400 200];
```

Display the image and the ROI.

```
roiImg = insertObjectAnnotation(I,"rectangle",roi,"ROI");
figure
imshow(roiImg)
```



Detect faces in the specified ROI using the `detect` function of the `faceDetector` object.

```
[bboxes,scores,labels] = detect(detector,I,roi);
```

Display the computed bounding boxes, scores, and the corresponding labels as a table.

```
table(bboxes,scores,labels)
```

```
ans=2×3 table
            bboxes              scores     labels
    ____________________________    _______    ______

    253    551      7     10    0.94679     face
    222    557      7     10     0.9767     face
```

Overlay bounding boxes and scores on the image using the `insertObjectAnnotation` function.

```
detectedImg = insertObjectAnnotation(roiImg,"rectangle",bboxes,scores);
```

Display the detection results.

```
figure
imshow(detectedImg)
```



Detect Faces in Video Using RetinaFace Detector

Create a face detector object using the `faceDetector` function. Specify the detector name as "`large-network`". This configuration loads a pretrained RetinaFace face detector with ResNet-50 as the backbone network for face detection.

```
detector = faceDetector("large-network");
```

Create a `VideoReader` object to read video data from a video file.

```
reader = VideoReader('handshake_right.avi');
```

Configure a `VideoPlayer` object to display the video frames and the face detection results.

```
videoPlayer = vision.VideoPlayer(Position=[0 0 400 400]);
```

Read and iterate over each frame in the video using a `while` loop. Perform these steps to detect faces and display the detection results.

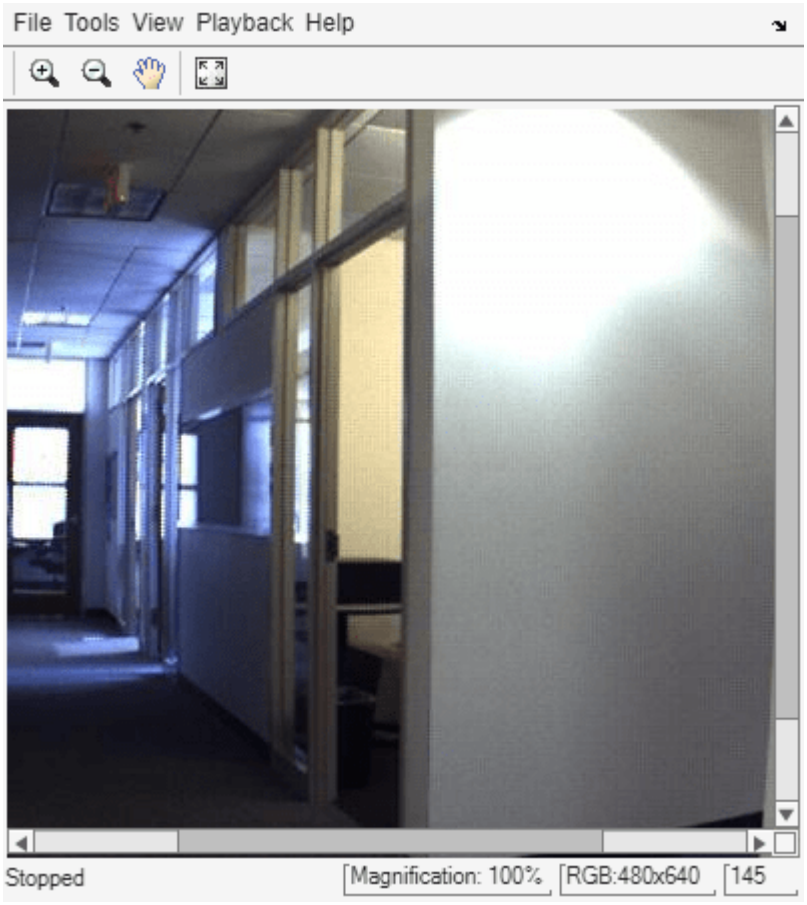
- Step 1: Read the current video frame with the `readFrame` function of the `VideoReader` object.
- Step 2: Detect faces in the video frame using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes and detection scores for the detected faces.
- Step 3: Overlay bounding boxes and scores on the video frame using the `insertObjectAnnotation` function.
- Step 4: Display the annotated frame using the `step` function of the `VideoPlayer` object.

```
while hasFrame(reader)
    % Step 1 %
    videoFrame = readFrame(reader);
```

```
    % Step 2 %
    [bboxes,scores] = detect(detector,videoFrame);
    % Step 3 %
    videoFrame= insertObjectAnnotation(videoFrame,"rectangle",bboxes,scores);
    % Step 4 %
    step(videoPlayer,videoFrame)
end
```

Call the `release` function to free up the resources allocated to the `VideoPlayer` object.

```
release(videoPlayer)
```



Detect and Blur Faces in Video

Create a face detector object using the `faceDetector` function. By default, the function uses the RetinaFace detector with a small backbone network for face detection.

```
detector = faceDetector;
```

Create a `VideoReader` object to read video data from a video file.

```
reader = VideoReader('tilted_face.avi');
```

Configure a `VideoPlayer` object to display the video frames and the face detection results.

```
videoPlayer = vision.VideoPlayer(Position=[0 0 600 600]);
```

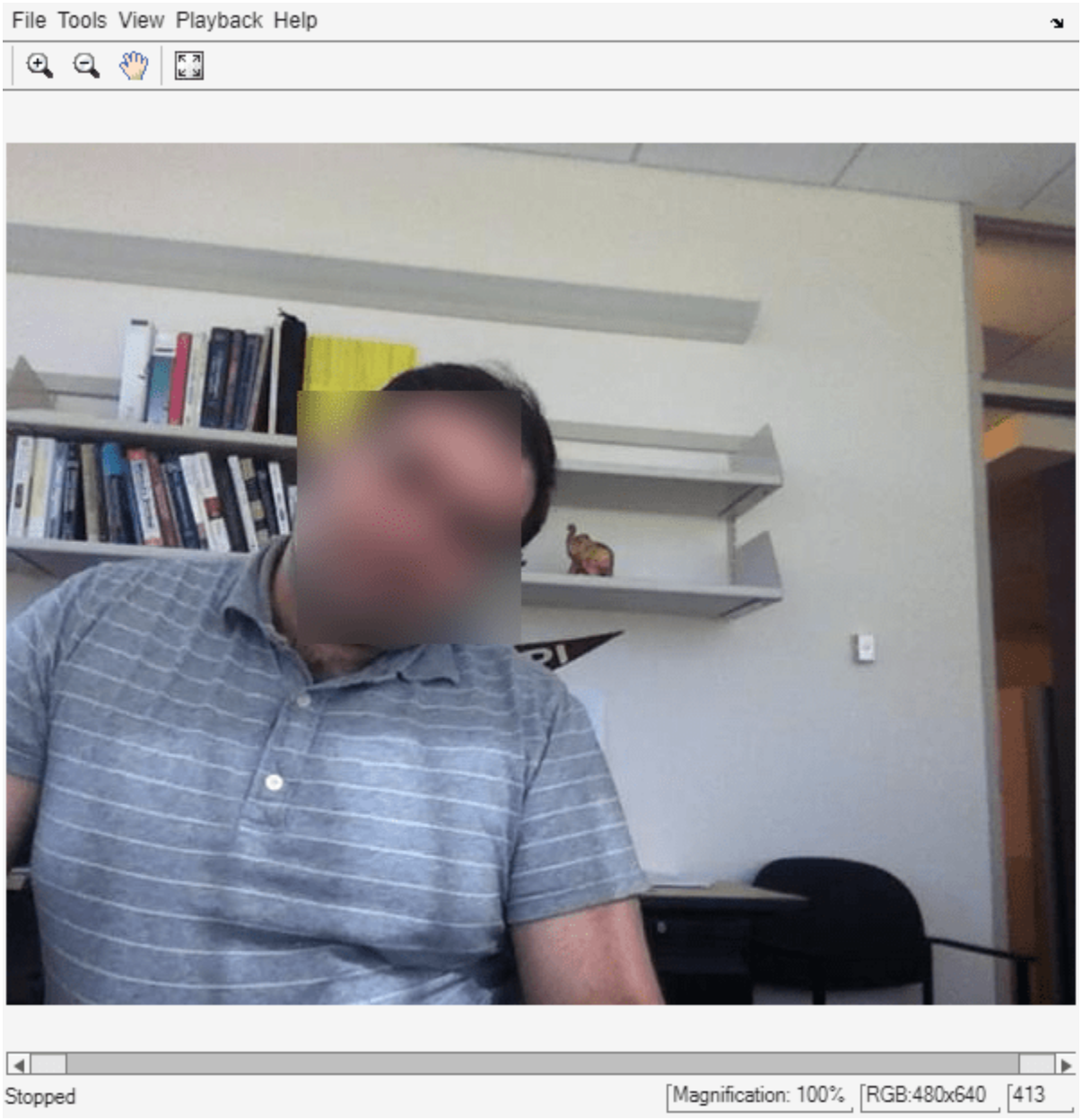
Read and iterate over each frame in the video using a `while` loop. Perform these steps to detect faces and display the detection results.

- Step 1: Read the current video frame with the `readFrame` function of the `VideoReader` object.
- Step 2: Detect faces in the video frame using the `detect` function of the `faceDetector` object. The `detect` function returns bounding boxes for the detected faces.
- Step 3: If the current frame has detected faces, use the helper function `helperBlurFaces` to blur the faces in the frame. The helper function applies Gaussian filtering to blur the areas of the frame defined by the bounding boxes. This effectively obscures the detected faces.
- Step 4: Display the processed frame using the `step` function of the `VideoPlayer` object.

```
while hasFrame(reader)
    % Step 1 %
    videoFrame = readFrame(reader);
    % Step 2 %
    bboxes = detect(detector,videoFrame,Threshold=0.2);
    % Step 3 %
    if ~isempty(bboxes)
        videoFrame = helperBlurFaces(videoFrame,bboxes);
    end
    % Step 4 %
    step(videoPlayer,videoFrame)
end
```

Call the `release` function to free up the resources allocated to the `VideoPlayer` object.

```
release(videoPlayer)
```



The `helperBlurFaces` function applies Gaussian filtering to the regions defined by each bounding box, which correspond to detected faces.

```
function I = helperBlurFaces(I,bbox)
for j=1:size(bbox,1)
    xbox = round(bbox(j,:));
    subImage = imcrop(I,xbox);
    blurred = imgaussfilt(subImage,12);
    I(xbox(2):xbox(2)+xbox(4),xbox(1):xbox(1)+xbox(3),1:end) = blurred;
end
end
```

Input Arguments

`detector` — Pretrained RetinaFace face detector

`faceDetector` object

Pretrained RetinaFace face detector, specified as a `faceDetector` object. The face detector has been trained on the WIDER FACE data set.

`I` — Test images

matrix | 3-D numeric array | 4-D numeric array

Test images, specified as one of these values:

- A matrix of form *H*-by-*W* for a grayscale image.
- A 3-D numeric array of form *H*-by-*W*-by-*3* for an RGB image.
- A 4-D numeric array of form *H*-by-*W*-by-*C*-by-*T* for a batch of test images.

H and *W* are the height and width of the images, respectively. *C* is the number of color channels. The value of *C* is 1 for grayscale images and 3 for RGB images. *T* is the number of images in the batch.

When the test image size does not match the network input size, the detector resizes the input image to the value of the `InputSize` property of `detector`, unless you specify `AutoResize` as `false`.

The detector is sensitive to the intensity range of the test images. It was trained on images with an intensity range of [0, 255]. For accurate results, ensure that the test images also have an intensity range of [0, 255].

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore of test images

`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Datastore of test images, specified as an `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing the full filenames of the test images. The images in the datastore must be grayscale or RGB images.

`roi` — Region of interest to search

vector of form [*x y width height*]

Region of interest (ROI) to search, specified as a vector of the form [*x y width height*]. The first two elements of the vector specify the coordinates of the upper-left corner of a region, and the third and fourth elements specify the size of that region, in pixels. If the input data is a datastore, the `detect` function applies the same ROI to every image in the datastore.

Note To specify the ROI to search, you must specify `AutoResize` value as `true`, enabling the function to automatically resize the input test images to the network input size.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.25)` specifies a detection threshold of 0.25.

`Threshold` — Detection threshold
`0.5` (default) | scalar in range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. The function removes detections that have scores less than this threshold value.

- To reduce false positives, at the possible expense of missing some detections, increase this value.
- To increase the sensitivity of the detector for detecting faces under challenging lighting conditions, pose variations, and occlusion, decrease the detection threshold. However, this might result in false positives.

`SelectStrongest` — Strongest bounding box selection
`true` or `1` (default) | `false` or `0`

Strongest bounding box selection, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Returns only the strongest bounding box for each detected face. The `detect` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `detect` function uses this call to the `selectStrongestBboxMulticlass` function.

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.45);
  ```

- `false` — Return all detected bounding boxes. You can write a custom function to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of form [*height width*]

Minimum region size containing a face, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest face in the test image. When you know the minimum size, you can reduce computation time by setting `MinSize` to that value.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest face in the test image.

By default, `MaxSize` is set to the height and width of the input image `I`. To reduce computation time, set this value to the known maximum region size in which to detect a face in the input test image.

`MiniBatchSize` — Minimum batch size
`16` (default) | positive integer

Minimum batch size, specified as a positive integer. Adjust the `MiniBatchSize` value to help process a large collection of images. The `detect` function groups images into minibatches of the specified size and processes them as a batch, which can improve computational efficiency at the cost of increased memory requirements. Decrease the minibatch size to use less memory.

`AutoResize` — Automatic resizing of input images
`true` or `1` (default) | `false` or `0`

Automatic resizing of the input images to preserve the aspect ratio, specified as a numeric or logical `1` (`true`) or `0` (`false`). When you specify `AutoResize` as `true`, the `detect` function resizes images to the nearest `InputSize` dimension, while preserving the aspect ratio. Specify `AutoResize` as `false` when performing image tiling-based inference, or inference at full test image size.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as one of these values:

- `"auto"` — Use a GPU if Parallel Computing Toolbox is installed and a supported GPU device is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply a number of compatible optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available only when using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the `detect` function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Do not use acceleration.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits on subsequent calls with compatible parameters, at the expense of an increased initial run time. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is available only for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is available only when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

bboxes — Locations of detected faces

M-by-4 matrix | *B*-by-1 cell array

Locations of the detected faces within the input image or images, returned as one of these options:

- *M*-by-4 matrix — Returned when the input is a single test image. *M* is the number of bounding boxes detected in an image. Each row of the matrix is of the form [*x y width height*]. The *x* and *y* values specify the coordinates of the upper-left corner, and *width* and *height* specify the size, of the corresponding bounding box, in pixels.
- *B*-by-1 cell array — Returned when the input is a batch of images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-4 matrix specifying the bounding boxes detected within the corresponding image.

scores — Detection confidence scores

M-by-1 numeric vector | *B*-by-1 cell array

Detection confidence scores for each bounding box, returned as one of these options:

- *M*-by-1 numeric vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — Returned when the input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

Each confidence score value is in the range `[0, 1]`.

labels — Labels for bounding boxes

M-by-1 categorical vector | *B*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — Returned when the input is a batch of test images. *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-1 categorical vector containing the class name.

By default, the output label value is `"face"`.

detectionResults — Detection results

three-column table

Detection results when the input is a datastore of test images, `ds`, returned as a table with these columns:

bboxes	scores	labels
Predicted bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y width height*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the coordinates of the upper-left corner of the rectangle, in pixels. • *width* specifies the width of the rectangle, which is its length along the *x*-axis, in pixels. • *height* specifies the height of the rectangle, which is its length along the *y*-axis, in pixels.	Confidence scores of the detected class for each bounding box, returned as an *M*-by-1 numeric vector with values in the range `[0, 1]`.	Labels assigned to the bounding boxes, returned as an *M*-by-1 categorical vector. By default, the value is `"face"`.

Version History

Introduced in R2025a

References

[1] Deng, Jiankang, Jia Guo, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. “RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild.” In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5202–11. Seattle, WA, USA: IEEE, 2020. https://doi.org/10.1109/CVPR42600.2020.00525.

[2] Yang, Shuo, Ping Luo, Chen Change Loy, and Xiaoou Tang. “WIDER FACE: A Face Detection Benchmark.” In *2016 IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 5525–33. Las Vegas, NV, USA: IEEE, 2016. https://doi.org/10.1109/CVPR.2016.596.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `roi` argument of the `detect` object function must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- The `AutoResize` name-value argument of the `detect` object function must be a code generation constant (`coder.const()`).
- The `detect` object function supports only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value arguments.

- The `detect` object function does not support code generation for the `ds` input argument, which specifies a datastore of test images.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`faceDetector` | `vision.CascadeObjectDetector`

Topics
“Code Generation for People Detection Using Deep Learning”

multiCameraParameters

Store multi-camera system parameters

Description

The `multiCameraParameters` object stores the intrinsic and extrinsic parameters of a multi-camera system estimated during multi-camera calibration.

Creation

Create a `multiCameraParameters` object using the `estimateMultiCameraParameters` function.

Properties

Extrinsic Parameters

`CameraPoses` — Pose of each camera relative to origin camera
numCameras-element vector of `rigidtform3d` objects

This property is read-only.

Pose of each camera relative to the origin camera `ReferenceCameraIndex`, specified as a *numCameras*-element vector of `rigidtform3d` objects. *numCameras* represents the number of cameras.

`ReferenceCameraIndex` — Index of origin camera
integer

This property is read-only.

Index of the origin camera, specified as an integer. This camera defines the reference coordinate system, and all other camera poses are defined relative to this reference camera. The pose of the reference camera is represented as `rigidtform3d(eye(4))`.

`NumCameras` — Number of cameras in system
integer

Number of cameras in the system, specified as an integer.

Intrinsic Parameters

`Intrinsics` — Intrinsic parameters for all cameras in multi-camera system
numCameras-element cell array

This property is read-only.

Intrinsic parameters for all cameras in the multi-camera system, specified as a *numCameras*-element cell array containing `cameraIntrinsics` objects, `fisheyeIntrinsics` objects, or a combination of both. *numCameras* is the number of cameras.

Accuracy of Estimated Parameters

`MeanReprojectionError` — Average Euclidean distance
scalar

This property is read-only.

Average Euclidean distance between reprojected points and detected points over all image pairs, specified as a scalar in pixels.

Data Types: `double`

`MeanReprojectionErrorPerCamera` — Average reprojection error for each camera
numCameras-element vector

This property is read-only.

Average reprojection error for each camera, specified a *numCameras*-element vector. *numCameras* is the number of cameras.

`MeanReprojectionErrorPerImage` — Average reprojection error of each image
numViews-by-*numCameras* vector

This property is read-only.

Average reprojection error of each image in pixels, specified as an *numViews*-by-*numCameras* vector.

`MeanReprojectionErrorPerView` — Average reprojection error of each view
numViews-by-1 vector

This property is read-only.

Average reprojection error of each view in pixels, specified as an *numViews*-by-1 vector

`ReprojectedPoints` — World points reprojected onto calibration images
numKeyPoints-by-2-by-*numViews*-by-*numCameras*-by-*patternCount* matrix

This property is read-only.

World points reprojected onto calibration images, specified as a *numKeyPoints*-by-2-by-*numViews*-by-*numCameras*-by-*patternCount* matrix.

Settings Used to Estimate Multi-Camera Parameters

`PatternCount` — Number of unique calibration patterns captured in calibration images
scalar

This property is read-only.

Number of unique calibration patterns captured in the calibration images, specified as a scalar.

`PatternPoses` — Poses of calibration patterns relative to reference pattern
P-element vector

This property is read-only.

Poses of calibration patterns relative to the reference pattern, specified as a *P*-element vector of `rigidtform3d` objects, where *P* represents the number of calibration patterns. The reference pattern is the first element of the `PatternPoses` vector, stored as `PatternPoses(1)`. Subsequent elements represent the poses of other patterns relative to this reference pattern. The reference pattern's pose is given by `rigidtform3d(eye(4))`.

`NumViews` — Number of pattern views to estimate camera poses
integer

This property is read-only.

Number of pattern views to estimate camera poses, specified as an integer.

`CovisibilityMatrix` — Covisibility of cameras
numCameras-by-*numCameras* logical matrix

This property is read-only.

Covisibility of cameras based on the calibration images, specified as a *numCameras*-by-*numCameras* logical matrix. *numCameras* is the number of cameras.

`WorldPoints` — World coordinates of pattern keypoints
numKeyPoints-by-2 matrix

This property is read-only.

World coordinates of pattern keypoints in the calibration pattern, specified as a *numKeyPoints*-by-2 matrix. *numKeyPoints* is the number of keypoints in the pattern.

`WorldUnits` — Units of world points
`"mm"` (default) | character vector | string scalar

This property is read-only.

Units of world points, specified as a character vector or string scalar. This value describes the unit of measure.

Examples

Calibrate a Multi-Camera System using a ChArUco board

Estimate the relative position and orientation of six cameras with overlapping fields of view by using calibration images that contain a single ChArUco board.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-charuco
calibImagesDir = fullfile(pwd,"overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (52 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (52 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
numKeyPoints = prod(patternDims - 1);
minMarkerId = 144;
```

Detect the key points of the ChArUco board in the calibration images.

```
imagePoints = detectPatternPoints(imageFiles,"charuco-board",patternDims, ...
    markerFamily,checkerSize,markerSize,MinMarkerID=minMarkerId);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world points for the pattern.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

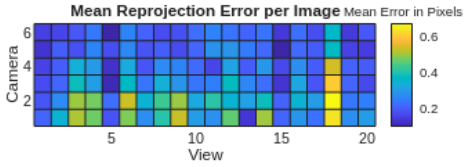
```
ld = load("sixCameraIntrinsics.mat");
```

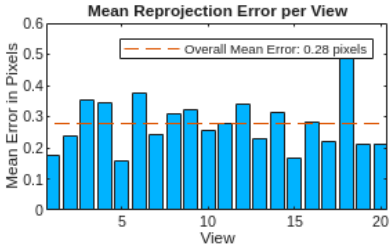
Perform multi-camera calibration.

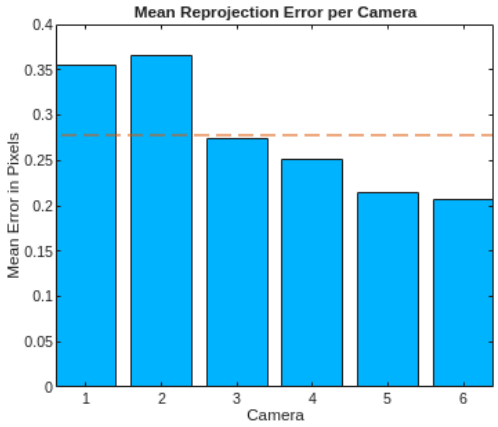
```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

Visualize the calibration accuracy.

```
figure(Position=[100,100,1000,400])
showReprojectionErrors(params)
```

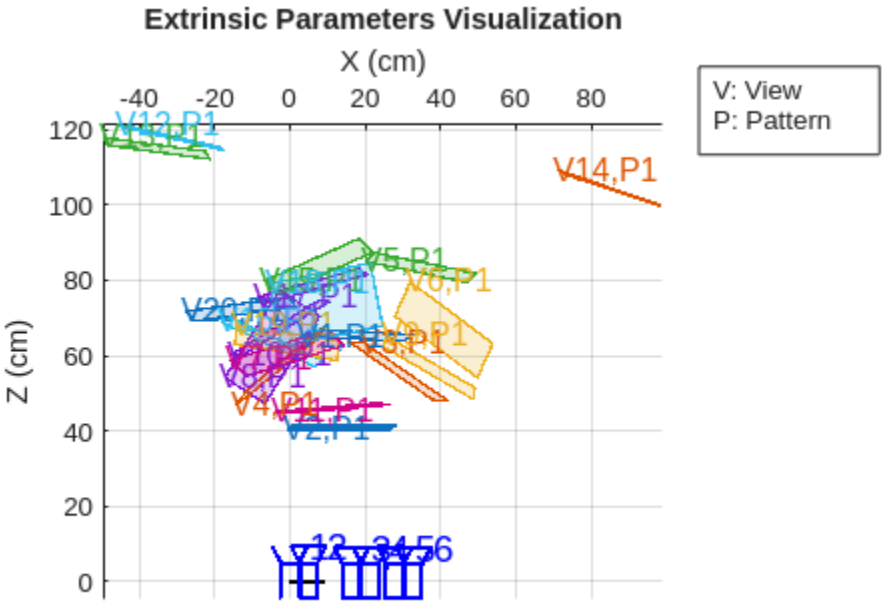






Visualize the camera extrinsic parameters.

```
figure
showExtrinsics(params)
view(2)
```



Version History

Introduced in R2025a

R2026a: New properties to support multiple calibration patternsPatternCount and PatternPoses properties

Added the `PatternCount`, `PatternPoses` `MeanReprojectionErrorPerView`, `MeanReprojectionErrorPerImage`, and `ReprojectedPoints` properties to support multiple calibration patterns.

R2026a: Computation of the `MeanReprojectionError` property has changed

The `MeanReprojectionError` property is computed as the mean of `MeanReprojectionErrorPerImage` over all images, omitting `NaN` values.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Objects
`cameraParameters` | `fisheyeParameters`

Functions
`patternWorldPoints` | `estimateMultiCameraParameters` | `detectPatternPoints` | `showReprojectionErrors` | `showExtrinsics`

Topics
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“3-D Motion Reconstruction Using Multiple Cameras”
“Import Stereo Camera Parameters from ROS”

peopleDetector

Detect people using pretrained deep learning object detector

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for RTMDet Object Detection add-on.

The `peopleDetector` object creates a deep learning object detector, trained on the COCO data set, to detect people in an image of arbitrary size. This feature also requires the Deep Learning Toolbox product.

Creation

Syntax

```
detector = peopleDetector
detector = peopleDetector(name)
detector = peopleDetector( ___ ,Name=Value)
```

Description

`detector = peopleDetector` returns a deep learning-based people detector object that is pretrained on the COCO data set for people detection.

`detector = peopleDetector(name)` returns a deep learning-based people detector object of the specified model `name` that is pretrained on the COCO data set.

`detector = peopleDetector( ___ ,Name=Value)` sets the `InputSize` property of the detector using name-value arguments in addition to any combination of arguments from previous syntaxes.

Input Arguments

`name` — Name of pretrained people detector model
`"small-network"` (default) | `"tiny-network"` | `"medium-network"` | `"large-network"`

Name of the pretrained people detector model trained on the COCO data set, specified as one of these values.

- `"tiny-network"` — This pretrained people detector deep learning model has the fewest filters and fewest convolutional layers. Use the `"tiny-network"` model when you have few available computational resources and require the fastest performance in terms of speed and real-time inference.
- `"small-network"` — Use `"small-network"` for a higher level of accuracy than `"tiny-network"` when moderate computational resources are available.
- `"medium-network"` — Use `"medium-network"` for a higher level of accuracy than `"small-network"` when more computational resources, such as GPU hardware, are available.

- `"large-network"` — This pretrained people detector deep learning model has the largest number of filters and convolutional layers. Use the `"large-network"` network for the highest level accuracy in people detection at the expense of computational cost and speed.

Data Types: `char` | `string`

Properties

`ModelName` — Name for people detector

`'small-network'` (default) | `'tiny-network'` | `'medium-network'` | `'large-network'`

This property is read-only.

Name for the people detector, stored as a character vector. By default, this property value is set to the value of the `name` input argument.

`ClassNames` — Name of object class

`{'person'}`

This property is read-only.

Name of the object class to be detected, stored as a cell array of character vector. By default, the value is `'person'`.

`InputSize` — Image size used for inference

`[640 640]` (default) | two-element row vector of integers

Image size to be used for inference, stored as a two-element row vector of integers of the form `[`*`H W`*`]`. *H* and *W* correspond to the height and width of the input image, respectively.

By default, the `InputSize` property value is set to the network's input size `[640 640]`, which is the size of the training images.

To set a custom value for this property, specify the property and its value as a name,value pair when creating the object. For example, `InputSize=[224 224]` sets the `InputSize` property to `[224 224]`.

The `detect` function resizes the test image to the dimensions specified by the `InputSize` property before performing the detections. If the resolution of the person class to be detected is too small in the test image, increase the `InputSize` property value for better detection accuracy. However, increasing the `InputSize` property value will also increase the computational cost.

Object Functions

detect Detect people using pretrained deep learning-based people detector

Examples

Detect People Using People Detector

Load the `"large-network"` pretrained deep learning-based people detector model.

```
detector = peopleDetector("large-network");
```

Read an image into the workspace.

```
I = imread("occlusion.png");
```

Detect all the people in the image regardless of their scale, orientation, or occlusion level.

```
[bboxes,scores] = detect(detector,I);
```

Annotate the detected people with bounding boxes, and display their detection scores.

```
detectedImg = insertObjectAnnotation(I,"rectangle",bboxes,scores);
figure
imshow(detectedImg)
```



Version History

Introduced in R2024b

R2025a: Generate C/C++ code using MATLAB Coder

The `peopleDetector` object now supports the generation of C/C++ code (requires MATLAB Coder).

R2025a: Generate CUDA code using GPU Coder

The `peopleDetector` object now supports the generation of CUDA code (requires GPU Coder™).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `peopleDetector` object supports the generation of generic code only, which does not depend on any third-party libraries.
- The `roi` argument of the `detect` object function must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- The `AutoResize` name-value argument of the `detect` object function must be a code generation constant (`coder.const()`).
- The `detect` object function supports only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value arguments.
- The `detect` object function does not support code generation for the `ds` input argument, which specifies a datastore of test images.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`detect` | `peopleDetectorACF` | `rtmdetObjectDetector` | `ssdObjectDetector` | `yolov2ObjectDetector` | `yolov3ObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector` | `faceDetector`

detect

Detect people using pretrained deep learning-based people detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
detectionResults = detect(detector,ds)
[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects people within a single image or a batch of images, `I`, using a pretrained deep learning-based people detector, `detector`. The `detect` function returns the locations of detected people in the input image as a set of bounding boxes.

Note This functionality requires Deep Learning Toolbox.

`[bboxes,scores] = detect(detector,I)` returns the confidence scores for the bounding boxes along with their locations.

`[bboxes,scores,labels] = detect(detector,I)` also returns a categorical array of person labels assigned to the bounding boxes.

`detectionResults = detect(detector,ds)` returns a table of predicted people for all the images in the input datastore `ds`.

`[___] = detect(___,roi)` detects people within the rectangular region of interest `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments. For example, `Threshold=0.75` specifies a detection threshold of 0.75.

Examples

Detect People Within ROI Using Pretrained People Detector

Load the default pretrained people detector model.

```
detector = peopleDetector;
```

Read a test image into the workspace.

```
I = imread("boats.png");
```

Specify a rectangular region of interest (ROI) that encloses the first boat within the test image.

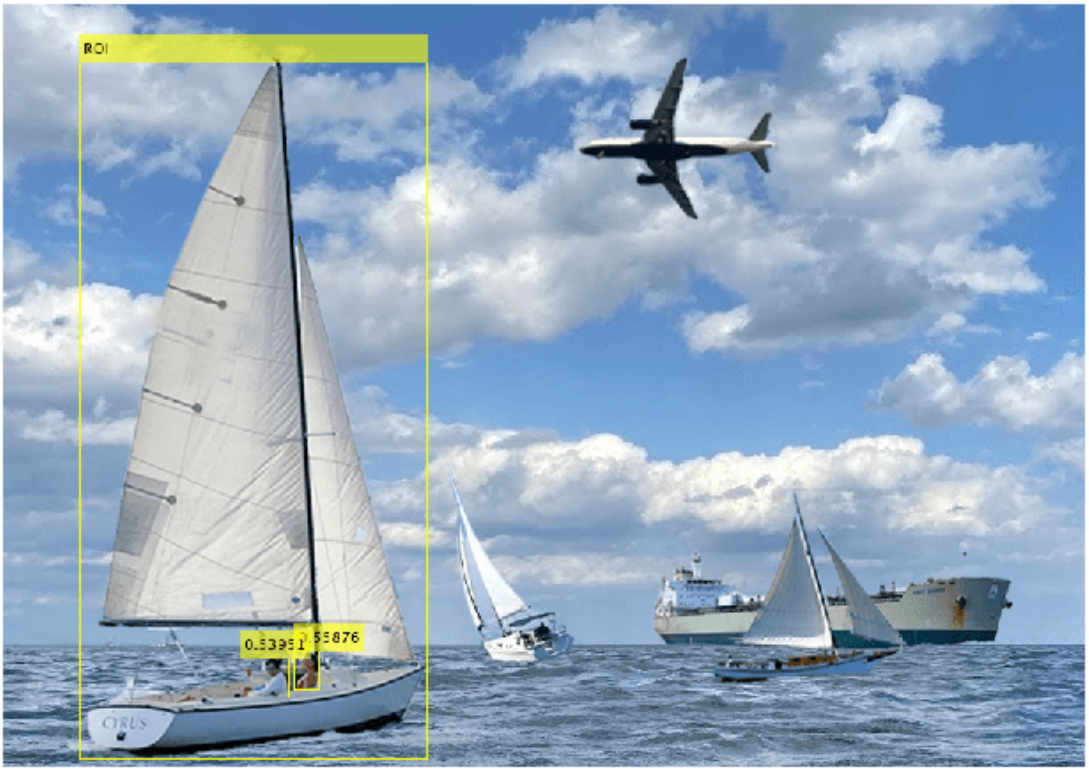
```
roiBox = [65 50 295 590];
```

Detect people in the first boat in the image using a `Threshold` value of 0.45.

```
[bboxes,scores] = detect(detector,I,roiBox,Threshold=0.45);
```

Annotate the detected people with bounding boxes and their detection scores.

```
img = insertObjectAnnotation(I,"rectangle",roiBox,"ROI");
detectedImg = insertObjectAnnotation(img,"rectangle",bboxes,scores);
figure
imshow(detectedImg)
```



Input Arguments

`detector` — people detector

`peopleDetector` object

People detector, specified as a `peopleDetector` object.

`I` — Test images

matrix | 3-D numeric array | 4-D numeric array

Test images, specified as one of these values:

- A matrix of form *H*-by-*W* for a grayscale image.
- A 3-D numeric array of form *H*-by-*W*-by-*3* for an RGB image.
- A 4-D numeric array of form *H*-by-*W*-by-*C*-by-*T* for a batch of test images.

H and *W* are the height and width of the images, respectively. *C* is the number of color channels. The value of *C* is:

- 1 for grayscale images.
- 3 for RGB images.

T is the number of images in the batch.

When the test image size does not match the network input size, the detector resizes the input image to the value of the `InputSize` property of `detector`, unless you specify `AutoResize` as `false`.

The detector is sensitive to the range of the test image. Therefore, ensure that the test image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore of test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Datastore of test images, specified as an `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing the full filenames of the test images. The images in the datastore must be grayscale or RGB images.

`roi` — Region of interest to search
vector of form [*x y width height*]

Region of interest (ROI) to search, specified as a vector of the form [*x y width height*]. The vector specifies the upper-left corner and size of a region, in pixels. If the input data is a datastore, the `detect` function applies the same ROI to every image in the datastore.

Note To specify the ROI to search, the `AutoResize` value must be `true`, enabling the function to automatically resize the input test images to the network input size.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.5)` specifies a detection threshold of 0.5.

`Threshold` — Detection threshold
`0.25` (default) | scalar in range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. The function removes detections that have scores less than this threshold value. To reduce false positives, at the possible expense of missing some detections, increase this value.

`SelectStrongest` — Strongest bounding box selection
`true` or `1` (default) | `false` or `0`

Strongest bounding box selection, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Returns only the strongest bounding box for each detected person. The `detect` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `detect` function uses this call to the `selectStrongestBboxMulticlass` function.

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.45);
  ```

- `false` — Return all detected bounding boxes. You can write a custom function to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of form [*height width*]

Minimum region size containing a person, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest person that the trained network can detect. When you know the minimum size, you can reduce computation time by setting `MinSize` to that value.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest person that the trained network can detect.

By default, `MaxSize` is set to the height and width of the input image `I`. To reduce computation time, set this value to the known maximum region size in which to detect people in the input test image.

`MiniBatchSize` — Minimum batch size
`16` (default) | positive integer

Minimum batch size, specified as a positive integer. Adjust the `MiniBatchSize` value to help process a large collection of images. The `detect` function groups images into minibatches of the specified size and processes them as a batch, which can improve computational efficiency at the cost of increased memory demand. Increase the minibatch size to decrease processing time. Decrease the minibatch size to use less memory.

`AutoResize` — Automatic resizing of input images
`true` or `1` (default) | `false` or `0`

Automatic resizing of input images to preserve the aspect ratio, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `AutoResize` is set to `true`, the `detect` function resizes images to the nearest `InputSize` dimension, while preserving the aspect ratio. Set `AutoResize` to logical `false` when performing image tiling-based inference, or inference at full test image size.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as one of these values:

- `"auto"` — Use a GPU if Parallel Computing Toolbox is installed and a supported GPU device is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply a number of compatible optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available only when using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the `detect` function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Do not use acceleration.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits on subsequent calls with compatible parameters, at the expense of an increased initial run time. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is available only for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is available only when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

`bboxes` — Locations of people detected
M-by-4 matrix | *B*-by-1 cell array

Locations of people detected within the input image or images, returned as one of these options:

- *M*-by-4 matrix — Returned when the input is a single test image. *M* is the number of bounding boxes detected in an image. Each row of the matrix is of the form [*x y width height*]. The *x* and *y* values specify the upper-left corner coordinates, and *width* and *height* specify the size, of the corresponding bounding box, in pixels.

- *B*-by-1 cell array — Returned when the input is a batch of images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-4 matrix specifying the bounding boxes detected within the corresponding image.

`scores` — Detection confidence scores

M-by-1 numeric vector | *B*-by-1 cell array

Detection confidence scores for each bounding box in the range `[0, 1]`, returned as one of these options:

- *M*-by-1 numeric vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — Returned when the input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

`labels` — Labels for bounding boxes

M-by-1 categorical vector | *B*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — Returned when the input is a batch of test images. *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-1 categorical vector containing the class name.

`detectionResults` — Detection results

three-column table

Detection results when the input is a datastore of test images, `ds`, returned as a table with these columns:

`bboxes`	`scores`	`labels`
Predicted bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner coordinates of the rectangle, in pixels. • *w* specifies the width of the rectangle, which is its length along the *x*-axis, in pixels. • *h* specifies the height of the rectangle, which is its length along the *y*-axis, in pixels.	Confidence scores of the person class for each bounding box, returned as an *M*-by-1 numeric vector with values in the range `[0, 1]`.	Predicted person class labels assigned to the bounding boxes, returned as an *M*-by-1 categorical vector.

Version History

Introduced in R2024b

R2025a: Generate C/C++ code using MATLAB Coder

The `detect` object function now supports the generation of C/C++ code (requires MATLAB Coder).

R2025a: Generate CUDA code using GPU Coder

The `detect` object function now supports the generation of CUDA code (requires GPU Coder).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `peopleDetector` object supports the generation of generic code only, which does not depend on any third-party libraries.
- The `roi` argument of the `detect` object function must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- The `AutoResize` name-value argument of the `detect` object function must be a code generation constant (`coder.const()`).
- The `detect` object function supports only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value arguments.
- The `detect` object function does not support code generation for the `ds` input argument, which specifies a datastore of test images.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`peopleDetector` | `peopleDetectorACF` | `rtmdetObjectDetector` | `ssdObjectDetector` | `yolov2ObjectDetector` | `yolov3ObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector`

rtmdetObjectDetector

Detect objects using RTMDet object detector

Description

The `rtmdetObjectDetector` object creates an Real-Time Model for object Detection (RTMDet), which is a one-stage, real-time, anchor-free object detector, for detecting objects in an image of arbitrary size. Using this object, you can create a pretrained RTMDet object detector by using RTMDet deep learning networks trained on the COCO data set.

Creation

Syntax

```
detector = rtmdetObjectDetector
detector = rtmdetObjectDetector(name)
detector = rtmdetObjectDetector( ___ ,Name=Value)
```

Description

`detector = rtmdetObjectDetector` creates a RTMDet object detector trained to detect 80 object classes from the COCO data set using a CSP-Darknet network.

`detector = rtmdetObjectDetector(name)` creates a pretrained RTMDet object detector by using the specified RTMDet deep learning network name trained on the COCO data set.

`detector = rtmdetObjectDetector( ___ ,Name=Value)` sets the `InputSize`, and `NormalizationStatistics` properties of the detector using one or more name-value arguments in addition to any combination of arguments from previous syntaxes.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for RTMDet Object Detection. You can install the Computer Vision Toolbox Model for RTMDet Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Input Arguments

`name` — Name of pretrained RTMDet deep learning network
`"small-network-coco"` (default) | `"tiny-network-coco"` | `"medium-network-coco"` | `"large-network-coco"`

Name of the pretrained RTMDet deep learning network, created using CSP-DarkNet as the base network and trained on the COCO data set, specified as one of these values.

- `"tiny-network-coco"` — Pretrained RTMDet-tiny deep learning network. This RTMDet network has the fewest filters and fewest convolutional layers. Use RTMDet-tiny when you have few

available computational resources, and require the fastest performance in terms of speed and real-time inference.

- `"small-network-coco"` — Pretrained RTMDet-small deep learning network. Use RTMDet-small for a higher level of accuracy than RTMDet-tiny when moderate computational resources are available.
- `"medium-network-coco"` — Pretrained RTMDet-medium deep learning network. Use RTMDet-medium for a higher level of accuracy than RTMDet-small when more computational resources, such as GPU hardware are available.
- `"large-network-coco"` — Pretrained RTMDet-large deep learning network. This RTMDet network has the largest number of filters and convolutional layers. Use the RTMDet-large network for the highest level accuracy in object detection at the expense of computational cost and speed.

Tip To improve accuracy and the performance of complex detection tasks, you can increase the pretrained model size at the expense of decreased inference speed, larger computation time, and greater memory requirements. Use RTMDet-tiny for real-time inference at the expense of detection accuracy. Use RTMDet-small for near-real-time applications due to fast inference. Use RTMDet-medium and RTMDet-large when sufficient computational resources are available for the largest accuracy, at the expense of lower computation speeds.

Data Types: `char` | `string`

Properties

`ModelName` — Name for object detector

character vector

This property is read-only.

Name for the object detector, stored as a character vector. By default, this property value is set to the value of the `name` input argument.

`ClassNames` — Names of object classes

cell array of character vectors

This property is read-only.

Names of 80 object classes in COCO data set stored as a cell array. The detector is pre-trained on the COCO data set.

`InputSize` — Image size to be used for inference

`[640 640 3]` (default) | three-element row vector of integers

This property is read-only.

Image size to be used for inference, stored as a three-element row vector of integers of the form `[`*`H W C`*`]`, where *H*, *W*, and *C* correspond to the height, width, and number of channels in the input image, respectively.

By default, the `InputSize` property value is set to the network's input size `[640 640 3]`, which is the size of the training images.

To set a custom value for this property, specify the property and its value as a name,value pair when creating the object. For example, `InputSize=[224 224 3]` sets the `InputSize` property to `[224 224 3]`.

The `detect` function resizes the test image to the dimensions specified by the `InputSize` property before performing the detections.

`NormalizationStatistics` — *Z*-score normalization values for the object detector
1-by-1 structure (default) | structure

Z-score normalization statistics, stored as a structure with the fields `Mean` and `StandardDeviation`, which contain the values from the COCO data set:

Field	Description	Default Value
`Mean`	1-by-*C* vector of means per channel.	`[123.6750 116.2800 103.5300]`
`StandardDeviation`	1-by-*C* vector of standard deviations per channel.	`[58.3950 57.1200 57.3750]`

The number of channels *C* must match the number of channels in the image dimensions specified by `InputSize`.

Object Functions

detect Detect objects using RTMDet object detector

Examples

Detect Objects Using Pretrained RTMDet Object Detector

Specify the name of a pretrained RTMDet deep learning network.

```
modelname = "medium-network-coco";
```

Create an RTMDet object detector by using the pretrained RTMDet network.

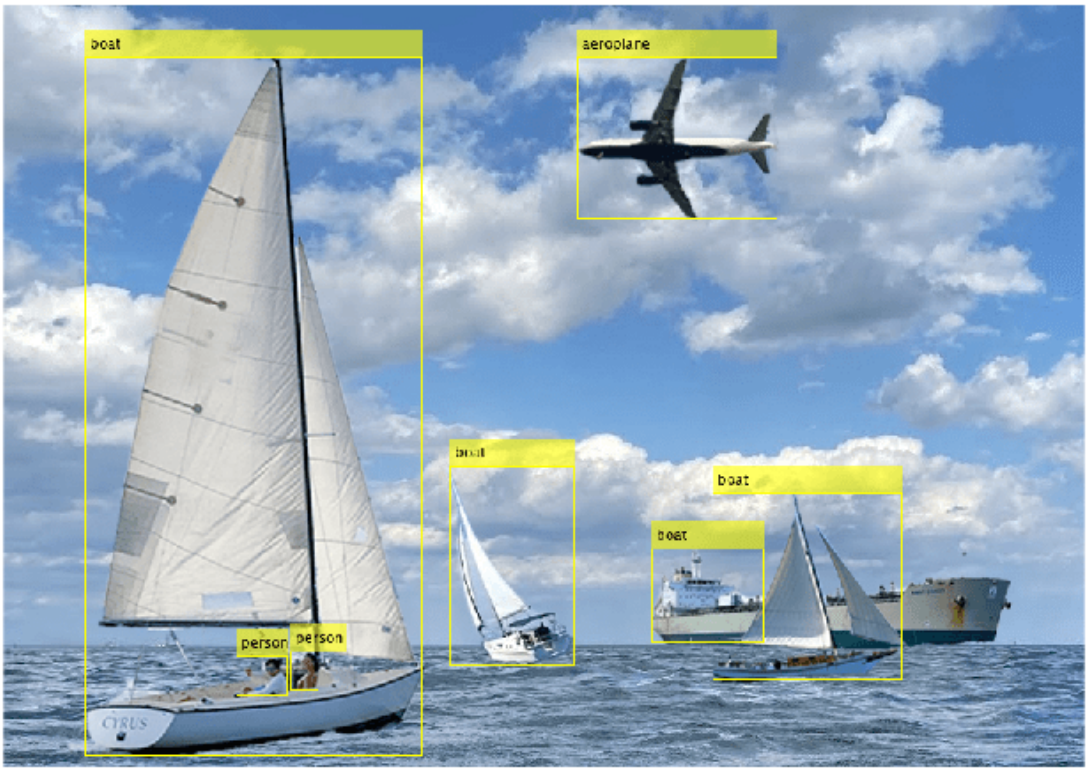
```
detector = rtmdetObjectDetector(modelname);
```

Read a test image into the workspace, and detect objects in it by using the pretrained RTMDet object detector with a `Threshold` value of 0.55.

```
img = imread("boats.png");
[bboxes,scores,labels] = detect(detector,img,Threshold=0.55);
```

Display the bounding boxes and predicted class labels of the detected objects.

```
detectedImg = insertObjectAnnotation(img,"rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Version History

Introduced in R2024b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`evaluateObjectDetection` | `ssdObjectDetector` | `yolov2ObjectDetector` | `yolov3ObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector`

Topics

“Get Started with Object Detection Using Deep Learning”
“Get Started with Image Preprocessing and Augmentation for Deep Learning”
“Choose an Object Detector”

detect

Detect objects using RTMDet object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
detectionResults = detect(detector,ds)
[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or a batch of images, `I`, using an RTMDet object detector, `detector`. The `detect` function returns the locations of detected objects in the input image as a set of bounding boxes.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for RTMDet Object Detection. You can install the Computer Vision Toolbox Model for RTMDet Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[bboxes,scores] = detect(detector,I)` returns the class-specific confidence scores for the bounding boxes along with their locations.

`[bboxes,scores,labels] = detect(detector,I)` also returns a categorical array of labels assigned to the bounding boxes.

`detectionResults = detect(detector,ds)` returns a table of predicted objects for all the images in the input datastore `ds`.

`[___] = detect(___,roi)` detects objects within the rectangular region of interest `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments. For example, `Threshold=0.5` specifies a detection threshold of 0.5.

Examples

Detect Objects Using Pretrained RTMDet Object Detector

Specify the name of a pretrained RTMDet deep learning network.

```
modelname = "medium-network-coco";
```

Create an RTMDet object detector by using the pretrained RTMDet network.

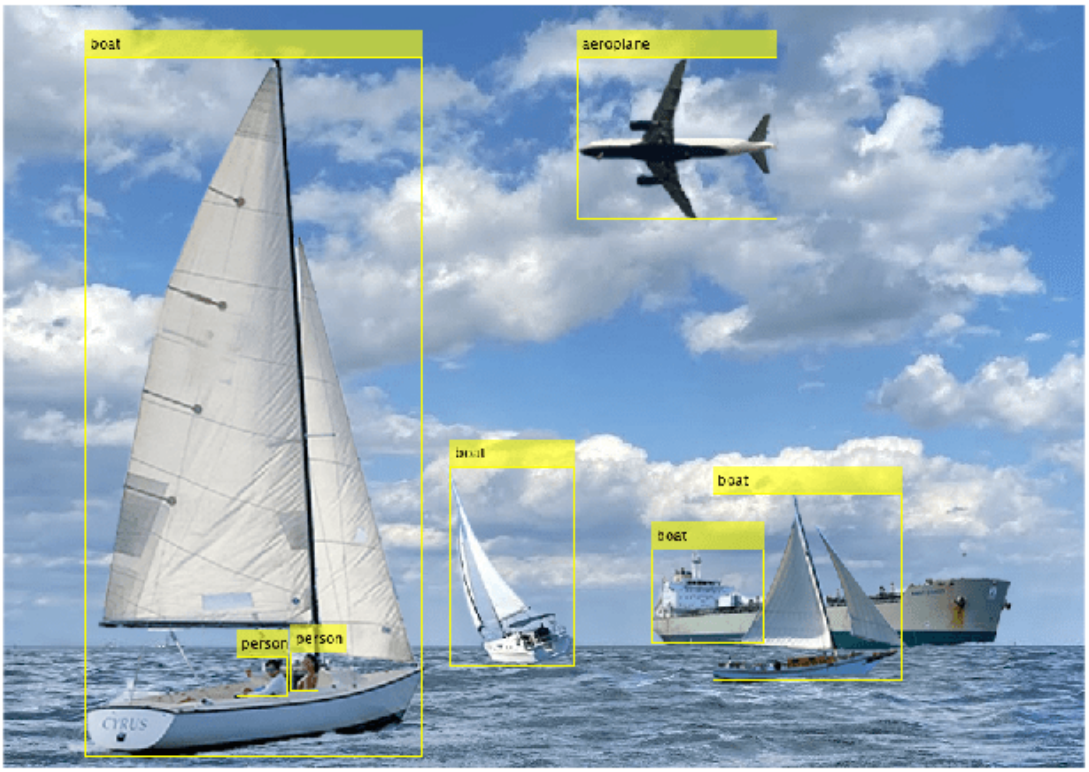
```
detector = rtmdetObjectDetector(modelname);
```

Read a test image into the workspace, and detect objects in it by using the pretrained RTMDet object detector with a `Threshold` value of 0.55.

```
img = imread("boats.png");
[bboxes,scores,labels] = detect(detector,img,Threshold=0.55);
```

Display the bounding boxes and predicted class labels of the detected objects.

```
detectedImg = insertObjectAnnotation(img,"rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects in Image Datastore Using RTMDet Object Detector

Load a pretrained RTMDet object detector.

```
detector = rtmdetObjectDetector("tiny-network-coco");
```

Read the test datastore, and store it as an image datastore object.

```
location = fullfile(matlabroot,"toolbox","vision","visiondata","stopSignImages");
imds = imageDatastore(location);
```

Detect objects in the test datastore. Set the `Threshold` value to 0.6 and `MiniBatchSize` value to 16.

```
detectionResults = detect(detector,imds,Threshold=0.6,MiniBatchSize=16);
```

Read an image from the test data set, and extract the corresponding detection results.

```
num = 4;
I = readimage(imds,num);
bboxes = detectionResults.Boxes{num};
labels = detectionResults.Labels{num};
scores = detectionResults.Scores{num};
```

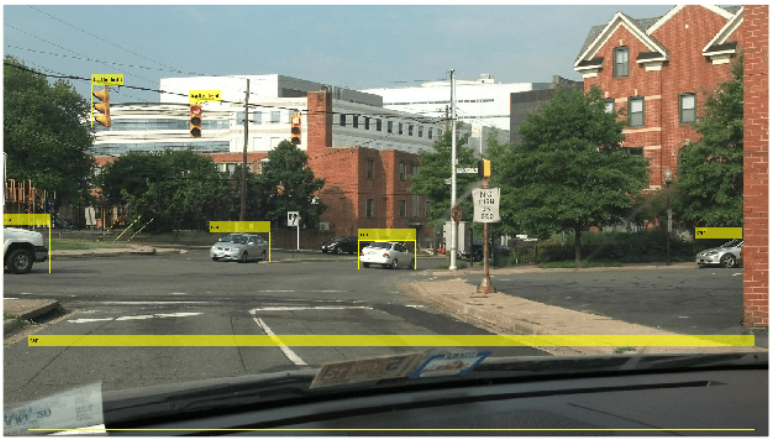
Display the table of detection results.

```
results = table(bboxes,labels,scores)
```

```
results=7×3 table
      189.3022    171.7018       40.2153     90.0829    traffic light    0.7034
             1    470.0854       99.5766    103.3941              car    0.6517
      439.7054    486.3765      128.5373     63.7707              car    0.7572
      758.2266    502.1126      122.3475     63.5290              car    0.7175
    1.4764e+03    498.0511      100.5210     63.9980              car    0.6535
      395.7627    207.2426       30.0126     76.8816    traffic light    0.6713
       53.3276    727.1694    1.5507e+03    180.0476              car    0.6140
```

Display the bounding boxes and predicted class labels of the detected objects in the image selected from the test datastore.

```
detectedImg = insertObjectAnnotation(I,"rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects Within ROI by Using RTMDet Object Detector

Load a pretrained RTMDet object detector.

```
detector = rtmdetObjectDetector("large-network-coco");
```

Read a test image.

```
img = imread("parkinglot_left.png");
```

Specify a rectangular region of interest (ROI) within the test image.

```
roiBox = [525 120 50 75];
```

Detect objects within the specified ROI.

```
[bboxes,scores,labels] = detect(detector,img,roiBox);
```

Display the bounding boxes and corresponding class labels of the detected objects within the ROI.

```
img = insertObjectAnnotation(img,"rectangle",roiBox,"ROI",AnnotationColor="yellow");
detectedImg = insertObjectAnnotation(img,"rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Input Arguments

`detector` — RTMDet object detector

`rtmdetObjectDetector` object

RTMDet object detector, specified as an `rtmdetObjectDetector` object.

`I` — Test images

H-by-*W*-by-*C* numeric array | *H*-by-*W*-by-*C*-by-*B* numeric array

Test images, specified as a numeric array of size *H*-by-*W*-by-*C* or *H*-by-*W*-by-*C*-by-*B*. You must specify real and nonsparse grayscale or RGB images.

- *H* — Height of the input images.
- *W* — Width of the input images.
- *C* — Number of channels. The channel size of each image must be equal to the input channel size of the network. For example, for grayscale images, *C* must be `1`. For RGB color images, it must be `3`.
- *B* — Number of test images in the batch. The `detect` function computes the object detection results for each test image in the batch.

When the test image size does not match the network input size, the detector resizes the input image to the value of the `InputSize` property of `detector`, unless you specify `AutoResize` as `false`.

The detector is sensitive to the range of the test image. Therefore, ensure that the test image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore of test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Datastore of test images, specified as an `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing the full filenames of the test images. The images in the datastore must be grayscale or RGB images.

`roi` — Region of interest to search
vector of the form [*x y width height*]

Region of interest (ROI) to search, specified as a vector of form [*x y width height*]. The vector specifies the upper-left corner and size of a region, in pixels. If the input data is a datastore, the `detect` function applies the same ROI to every image.

Note To specify the ROI to search, the `AutoResize` value must be `true`, enabling the function to automatically resize the input test images to the network input size.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.25)` specifies a detection threshold of 0.25.

`Threshold` — Detection threshold
`0.25` (default) | scalar in range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. The function removes detections that have scores less than this threshold value. To reduce false positives, at the possible expense of missing some detections, increase this value.

`SelectStrongest` — Strongest bounding box selection
`true` or `1` (default) | `false` or `0`

Strongest bounding box selection, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Returns only the strongest bounding box for each object. The `detect` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `detect` function uses this call to the `selectStrongestBboxMulticlass` function.

```
selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                              RatioType="Union", ...
                              OverlapThreshold=0.5);
```

- `false` — Return all detected bounding boxes. You can write a custom function to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of form [*height width*]

Minimum region size containing an object, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest object that the trained network can detect. When you know the minimum size, you can reduce computation time by setting `MinSize` to that value.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest object that the trained network can detect.

By default, `MaxSize` is set to the height and width of the input image `I`. To reduce computation time, set this value to the known maximum region size in which to detect objects in the input test image.

`MiniBatchSize` — Minimum batch size
`16` (default) | positive integer

Minimum batch size, specified as a positive integer. Adjust the `MiniBatchSize` value to help process a large collection of images. The `detect` function groups images into minibatches of the specified size and processes them as a batch, which can improve computational efficiency at the cost of increased memory demand. Decrease the minibatch size to use less memory.

`AutoResize` — Automatic resizing of input images
`true` or `1` (default) | `false` or `0`

Automatic resizing of input images to preserve the aspect ratio, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `AutoResize` is set to `true`, the `detect` function resizes images to the nearest `InputSize` dimension, while preserving the aspect ratio. Set `AutoResize` to logical `false` or `0` when performing image tiling-based inference.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as one of these values:

- `"auto"` — Use a GPU if Parallel Computing Toolbox is installed and a supported GPU device is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply a number of compatible optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available only when using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the `detect` function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Do not use acceleration.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits on subsequent calls with compatible parameters, at the expense of an increased initial run time. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is available only for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is available only when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

bboxes — Locations of objects detected

M-by-4 matrix | *B*-by-1 cell array

Locations of objects detected within the input image or images, returned as one of these options:

- *M*-by-4 matrix — Returned when the input is a single test image. *M* is the number of bounding boxes detected in an image. Each row of the matrix is of the form [*x y width height*]. The *x* and *y* values specify the upper-left corner coordinates, and *width* and *height* specify the size, of the corresponding bounding box, in pixels.
- *B*-by-1 cell array — Returned when the input is a batch of images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-4 matrix specifying the bounding boxes detected within the corresponding image.

scores — Detection confidence scores

M-by-1 numeric vector | *B*-by-1 cell array

Detection confidence scores for each bounding box in the range `[0, 1]`, returned as one of these options:

- *M*-by-1 numeric vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.

- *B*-by-1 cell array — Returned when the input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

A higher score indicates higher confidence in the detection. The confidence score for each detection is a product of the corresponding objectness score and maximum class probability. The objectness score is the probability that the object in the bounding box belongs to a class in the image. The maximum class probability is the largest probability that a detected object in the bounding box belongs to a particular class.

`labels` — Labels for bounding boxes

M-by-1 categorical vector | *B*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector — Returned when the input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — Returned when the input is an array of test images. *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-1 categorical vector containing the names of the object classes.

`detectionResults` — Detection results

three-column table

Detection results when the input is a datastore of test images, `ds`, returned as a table with these columns:

`bboxes`	`scores`	`labels`
Predicted bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner coordinates of the rectangle, in pixels. • *w* specifies the width of the rectangle, which is its length along the *x*-axis, in pixels. • *h* specifies the height of the rectangle, which is its length along the *y*-axis, in pixels.	Class-specific confidence scores for each bounding box, returned as an *M*-by-1 numeric vector with values in the range `[0, 1]`.	Predicted object labels assigned to bounding boxes, returned as an *M*-by-1 categorical vector. All categorical data returned by the datastore must contain the same categories.

Version History

Introduced in R2024b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `MiniBatchSize` name-value argument is not supported for code generation when the image input `I` is an *H*-by-*W*-by-*C*-by-*B* numeric array of images. All images will processed in one batch.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`rtmdetObjectDetector` | `imageDatastore` | `CombinedDatastore` | `TransformedDatastore` | `selectStrongestBboxMulticlass` | `evaluateObjectDetection`

Topics

“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Datastores for Deep Learning” (Deep Learning Toolbox)

opticalFlowRAFT

Estimate optical flow using RAFT deep learning algorithm

Description

Use the `opticalFlowRAFT` object to estimate the motion direction and velocity between previous and current video frames using the recurrent all-pairs field transforms (RAFT) algorithm. This algorithm uses a deep learning network trained on the Kubric [3 on page 2-90] dataset. The RAFT optical flow estimation algorithm outperforms approaches like Farneback by delivering greater accuracy, particularly in areas with minimal texture, motion blur, and under difficult camera movements. It provides dense (per-pixel) and highly accurate estimations, but requires more time and memory. For quicker but less precise dense optical flow estimation, opt for `opticalFlowFarneback`, a traditional vision algorithm that does not rely on deep learning.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for RAFT Optical Flow Estimation. You can install the Computer Vision Toolbox Model for RAFT Optical Flow Estimation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
flowModel = opticalFlowRAFT
```

Description

`flowModel = opticalFlowRAFT` returns an optical flow object that estimates the motion direction and velocity between the previous and current video frames.

Object Functions

estimateFlow	Estimate optical flow between two frames
reset	Reset the internal state of the optical flow estimation object

Examples

Estimate Optical Flow in Consecutive Image Frames

Create a RAFT optical flow object.

```
flowModel = opticalFlowRAFT;
```

Create an object to read the input video file.

```
vidReader = VideoReader("visiontraffic.avi",CurrentTime=11);
```

Create a custom figure window to visualize the optical flow vectors.

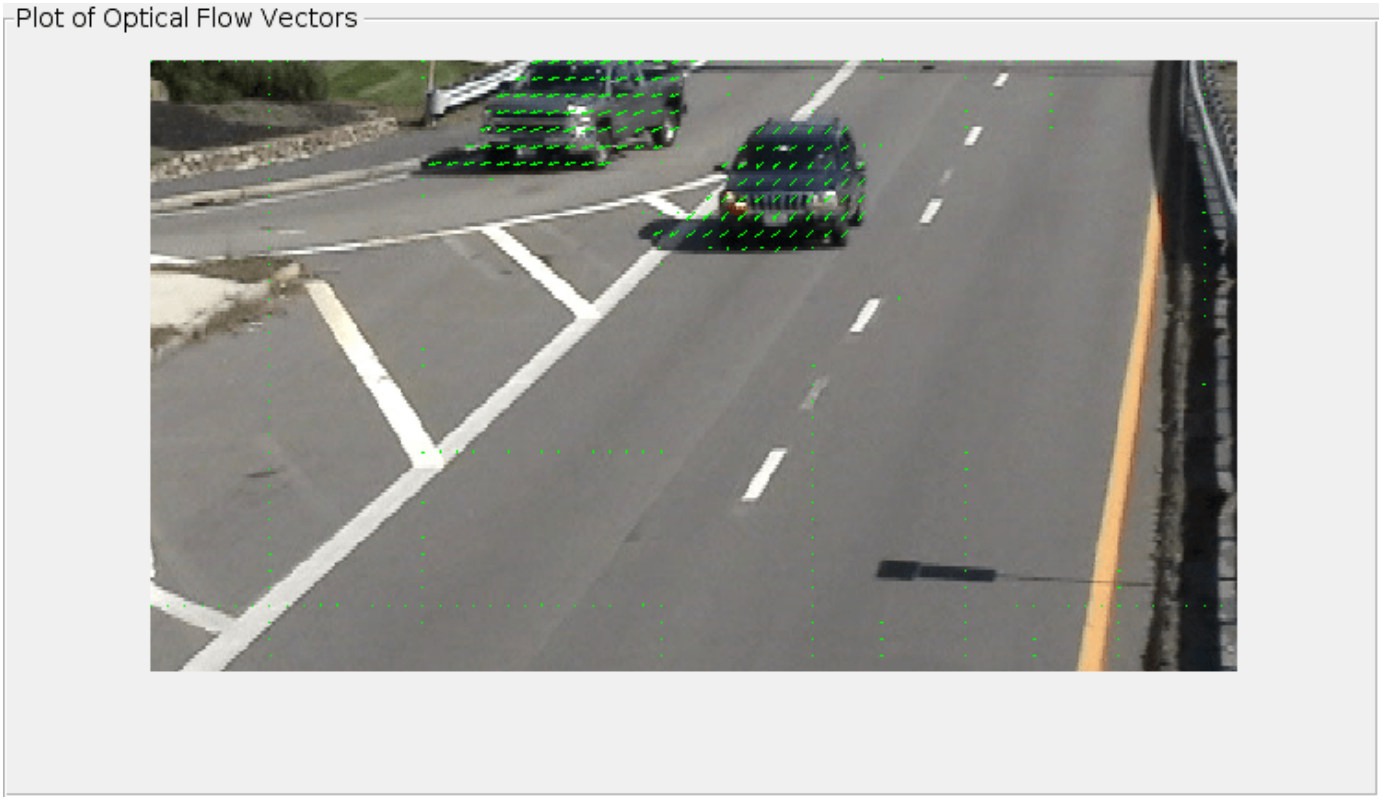
```
h = figure;
movegui(h);
hViewPanel = uipanel(h, Position=[0 0 1 1], Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

Read consecutive image frames to estimate optical flow. Display the current frame and overlay optical flow vectors using a quiver plot. The `estimateFlow` function calculates the optical flow between two consecutive frames.

Note that the function internally stores the previous frame and utilizes it implicitly for optical flow estimation. Consequently, when the function is called for the first time on a sequence of frames, it will return a zero flow. This is because, in the absence of a genuine previous frame, the initial frame is treated as both the current and previous frame, leading to no detectable motion between the two. This is consistent with the argument structure and behavior of established optical flow estimation methods, such as `opticalFlowFarneback`.

```
while hasFrame(vidReader)
    frame = readFrame(vidReader);
    flow = estimateFlow(flowModel,frame);

    imshow(frame)
    hold on
    plot(flow,DecimationFactor=[10 10],ScaleFactor=0.45,Parent=hPlot,color="g");
    hold off
    pause(10^-3)
end
```



Reset the `opticalFlowRAFT` object after the video processing has completed. This clears the internal state of the object, including the saved previous frame.

```
reset(flowModel);
```

Memory-Efficient RAFT Optical Flow for Large Images Using Rescaling

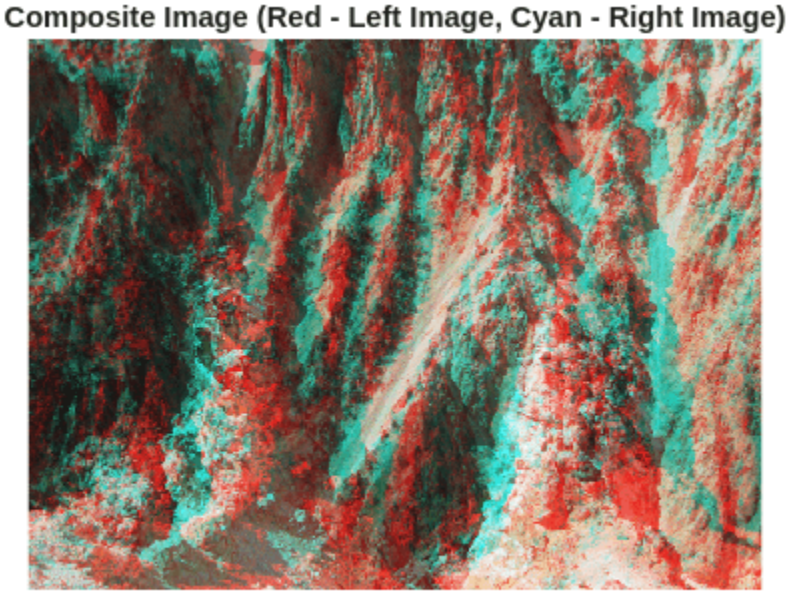
RAFT provides accurate optical flow estimation, but it can be memory-intensive when applied to large images or run on GPUs with limited memory. In such cases, processing high-resolution frames directly may lead to out-of-memory (OOM) errors. A practical solution is to downscale the input images before passing them to the RAFT model, compute optical flow at the reduced resolution, and then rescale the resulting flow field back to the original size of the imagery. This example demonstrates how to apply this memory-efficient strategy, allowing you to use RAFT on larger images or resource-constrained hardware while preserving meaningful flow estimates.

Load Image Pair

Load two images and visualize their pixel-wise differences due to camera motion using a red-cyan composite image.

```
I1 = imread("yellowstone_left.png");
I2 = imread("yellowstone_right.png");

figure
imshow(stereoAnaglyph(I1, I2))
title("Composite Image (Red - Left Image, Cyan - Right Image)")
```



Display the spatial dimensions of the images.

```
disp(size(I1, [1 2]))
```

```
   480   640
```

Create `opticalFlowRAFT` Object

Create an `opticalFlowRAFT` object for computing optical flow between two images.

```
raft = opticalFlowRAFT;
```

Estimate Optical Flow on Full Resolution Images

Estimate optical flow on the images at original resolution. This gives the most accurate result but may cause into slow runtime and high memory consumption.

```
estimateFlow(raft, I1);
flowFull = estimateFlow(raft, I2);
```

Estimate Optical Flow on Downsampled Images

Create Downscaled Images

Scale the image dimensions by a factor of 0.5 for illustrative purpose. A small resolution image can result in faster optical flow computation and less memory consumption.

```
I1Small = imresize(I1, 0.5);
I2Small = imresize(I2, 0.5);
```

Display the resized image dimensions.

```
disp(size(I1Small, [1 2]))
```

```
   240   320
```

Estimate Flow on Downscaled Images

Compute optical flow on downscaled images. Use the same opticalFlowRAFT model instantiated earlier after resetting it.

```
reset(raft);
estimateFlow(raft, I1Small);
flowLowRes = estimateFlow(raft, I2Small);
```

Scale Estimated Flow to Original Resolution

Upsample and scale the lower resolution optical flow back to the original image resolution.

```
origDims = size(I1, [1 2]);
scaleX = origDims(2) / size(flowLowRes.Magnitude, 2);
scaleY = origDims(1) / size(flowLowRes.Magnitude, 1);

flowX = imresize(flowLowRes.Vx, origDims, "bilinear") * scaleX;
flowY = imresize(flowLowRes.Vy, origDims, "bilinear") * scaleY;

flowUpscaled = opticalFlow(flowX, flowY);
```

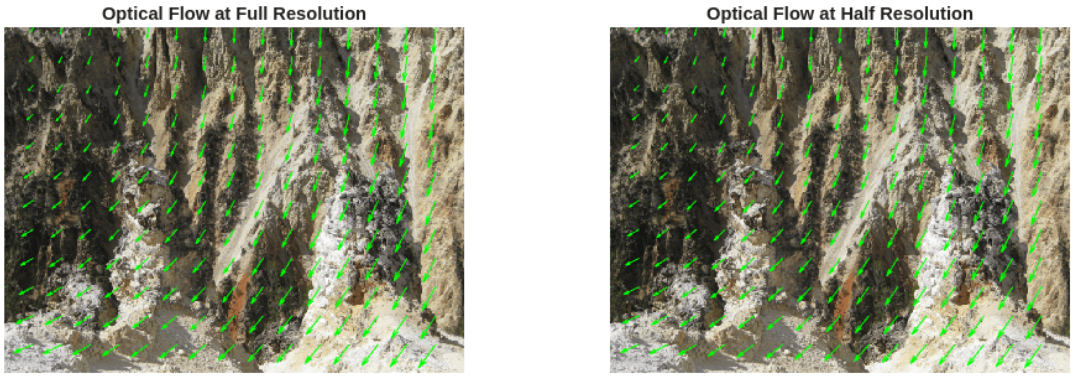
Compare Optical Flow at Original and Downscaled Resolutions

Visually compare optical flow computed directly on the original high-resolution images with optical flow computed on downscaled images and then rescaled back to the original size. This comparison

illustrates how the downscale-upscale workflow approximates the original flow while reducing memory usage.

```
figure
subplot(1,2,1)
imshow(I1)
hold on
plot(flowFull, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Full Resolution")

subplot(1,2,2)
imshow(I1)
hold on
plot(flowUpscaled, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Half Resolution")
```



The bilinear interpolation in the image resizing operations can introduce numerical inaccuracies and overly-smoothed results in the upscaled optical flow. This reduced accuracy is a trade-off against the lower memory consumption when optical flow computation is performed on lower resolution imagery.

Version History

Introduced in R2024b

References

[1] Teed, Zachary, and Jia Deng. *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow*. Proceedings of the 16th European Conference on Computer Vision. 2020.

[2] Shah, Neelay, Prajnan Goswami, and Huaizu Jiang. *EzFlow: A modular PyTorch library for optical flow estimation using neural networks*. 2021. Web. https://github.com/neu-vi/ezflow.

[3] Greff, Klaus, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J. Fleet et al. *Kubric: A scalable dataset generator*. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 3749-3761. 2022.

See Also

Objects
`opticalFlow` | `opticalFlowFarneback` | `opticalFlowLKDoG` | `opticalFlowHS` | `opticalFlowLK`

Functions
`quiver`

Topics
“Dense 3-D Reconstruction from Two Views Using RAFT Optical Flow”
“Dense 3-D Reconstruction from Multiple Views Using RAFT Optical Flow”
“Compare RAFT Optical Flow and Semi-Global Matching for Stereo Reconstruction”
“Automate Labeling of Objects in Video Using RAFT Optical Flow”
“Stabilize Video Using Optical Flow”
“What Is Optical Flow?”

estimateFlow

Estimate optical flow between two frames

Syntax

```
flow = estimateFlow(flowModel,I)
flow = estimateFlow(flowModel,I,Name=Value)
```

Description

`flow = estimateFlow(flowModel,I)` estimates the optical flow between the current frame `I` and the previous frame using the recurrent all-pairs field transforms (RAFT) deep learning algorithm.

RAFT optical flow estimation outperforms methods like Farneback by delivering higher accuracy, particularly in areas with minimal texture and under difficult camera movements.

`flow = estimateFlow(flowModel,I,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `MaxIterations=10` sets the number of refinement iterations to `10`.

Examples

Estimate Optical Flow in Consecutive Image Frames

Create a RAFT optical flow object.

```
flowModel = opticalFlowRAFT;
```

Create an object to read the input video file.

```
vidReader = VideoReader("visiontraffic.avi",CurrentTime=11);
```

Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h, Position=[0 0 1 1], Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

Read consecutive image frames to estimate optical flow. Display the current frame and overlay optical flow vectors using a quiver plot. The `estimateFlow` function calculates the optical flow between two consecutive frames.

Note that the function internally stores the previous frame and utilizes it implicitly for optical flow estimation. Consequently, when the function is called for the first time on a sequence of frames, it will return a zero flow. This is because, in the absence of a genuine previous frame, the initial frame is treated as both the current and previous frame, leading to no detectable motion between the two. This is consistent with the argument structure and behavior of established optical flow estimation methods, such as `opticalFlowFarneback`.

```
while hasFrame(vidReader)
    frame = readFrame(vidReader);
    flow = estimateFlow(flowModel,frame);

    imshow(frame)
    hold on
    plot(flow,DecimationFactor=[10 10],ScaleFactor=0.45,Parent=hPlot,color="g");
    hold off
    pause(10^-3)
end
```



Reset the `opticalFlowRAFT` object after the video processing has completed. This clears the internal state of the object, including the saved previous frame.

```
reset(flowModel);
```

Memory-Efficient RAFT Optical Flow for Large Images Using Rescaling

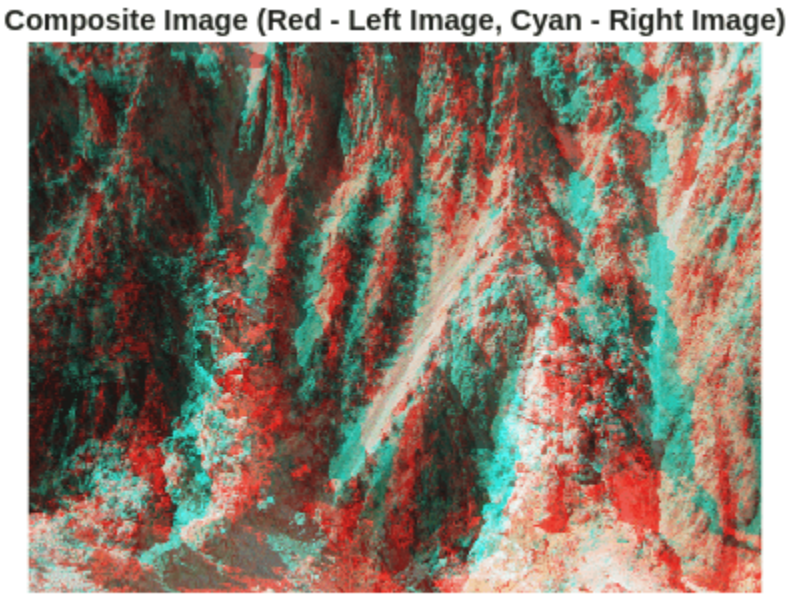
RAFT provides accurate optical flow estimation, but it can be memory-intensive when applied to large images or run on GPUs with limited memory. In such cases, processing high-resolution frames directly may lead to out-of-memory (OOM) errors. A practical solution is to downscale the input images before passing them to the RAFT model, compute optical flow at the reduced resolution, and then rescale the resulting flow field back to the original size of the imagery. This example demonstrates how to apply this memory-efficient strategy, allowing you to use RAFT on larger images or resource-constrained hardware while preserving meaningful flow estimates.

Load Image Pair

Load two images and visualize their pixel-wise differences due to camera motion using a red-cyan composite image.

```
I1 = imread("yellowstone_left.png");
I2 = imread("yellowstone_right.png");

figure
imshow(stereoAnaglyph(I1, I2))
title("Composite Image (Red - Left Image, Cyan - Right Image)")
```



Display the spatial dimensions of the images.

```
disp(size(I1, [1 2]))
```

```
   480   640
```

Create `opticalFlowRAFT` Object

Create an `opticalFlowRAFT` object for computing optical flow between two images.

```
raft = opticalFlowRAFT;
```

Estimate Optical Flow on Full Resolution Images

Estimate optical flow on the images at original resolution. This gives the most accurate result but may cause into slow runtime and high memory consumption.

```
estimateFlow(raft, I1);
flowFull = estimateFlow(raft, I2);
```

Estimate Optical Flow on Downsampled Images

Create Downscaled Images

Scale the image dimensions by a factor of 0.5 for illustrative purpose. A small resolution image can result in faster optical flow computation and less memory consumption.

```
I1Small = imresize(I1, 0.5);
I2Small = imresize(I2, 0.5);
```

Display the resized image dimensions.

```
disp(size(I1Small, [1 2]))
```

```
   240   320
```

Estimate Flow on Downscaled Images

Compute optical flow on downscaled images. Use the same opticalFlowRAFT model instantiated earlier after resetting it.

```
reset(raft);
estimateFlow(raft, I1Small);
flowLowRes = estimateFlow(raft, I2Small);
```

Scale Estimated Flow to Original Resolution

Upsample and scale the lower resolution optical flow back to the original image resolution.

```
origDims = size(I1, [1 2]);
scaleX = origDims(2) / size(flowLowRes.Magnitude, 2);
scaleY = origDims(1) / size(flowLowRes.Magnitude, 1);

flowX = imresize(flowLowRes.Vx, origDims, "bilinear") * scaleX;
flowY = imresize(flowLowRes.Vy, origDims, "bilinear") * scaleY;

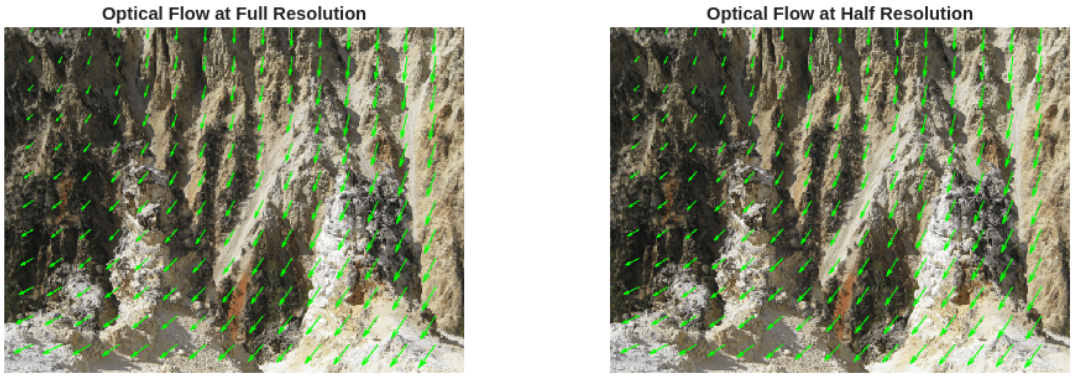
flowUpscaled = opticalFlow(flowX, flowY);
```

Compare Optical Flow at Original and Downscaled Resolutions

Visually compare optical flow computed directly on the original high-resolution images with optical flow computed on downscaled images and then rescaled back to the original size. This comparison illustrates how the downscale-upscale workflow approximates the original flow while reducing memory usage.

```
figure
subplot(1,2,1)
imshow(I1)
hold on
plot(flowFull, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Full Resolution")

subplot(1,2,2)
imshow(I1)
hold on
plot(flowUpscaled, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Half Resolution")
```



The bilinear interpolation in the image resizing operations can introduce numerical inaccuracies and overly-smoothed results in the upscaled optical flow. This reduced accuracy is a trade-off against the lower memory consumption when optical flow computation is performed on lower resolution imagery.

Input Arguments

`flowModel` — Optical flow object

`opticalFlowRAFT` object

Optical flow object, specified as an `opticalFlowRAFT` object.

`I` — Current video frame

2-D grayscale image | RGB image

Current video frame, specified as a 2-D grayscale or RGB image. When the input image is of type `uint8` or `int16`, the pixel values must be in the range `[0,255]`. When the input image is of type `single` or `double`, the pixels values must be in the range `[0,1]`. The RAFT model requires input images to have a minimum dimension of more than 57 pixels on the shortest side.

When initializing motion estimation in a video, the first frame is duplicated as its own "previous frame" to ensure the calculated motion (optical flow) starts from zero, as there is no actual prior frame to compare with.

Data Types: `single` | `double` | `int16` | `uint8`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `estimateFlow(flowModel,I,MaxIterations=10)` sets the number of refinement iterations to `10`.

`MaxIterations` — Number of refinement iterations

`12` (default) | integer

Number of refinement iterations, specified as an integer. Increasing the number of refinement iterations enhances the precision of the optical flow estimation from its initial prediction by iteratively

refining the calculation. Decreasing this value accelerates the algorithm's execution but compromises on accuracy. A typical value is 12.

`Tolerance` — Acceptable difference between consecutive refinement iterations
`1e-5` (default) | scalar

Acceptable difference between consecutive refinement iterations, specified as a scalar. The tolerance level set for iterations in estimating optical flow, beginning with an initial prediction, determines the acceptable deviation between each successive iteration. If the variation in optical flow between two successive updates falls below this tolerance threshold, the iteration process is halted. Opting for larger tolerance values can speed up the process but at the expense of reduced accuracy.

`ExecutionEnvironment` — Processing execution environment
`"auto"` (default) | `"CPU"` | `"GPU"`

Processing execution environment to perform the model's inference, specified as `"cpu"`, `"gpu"`, or `"auto"`.

- `"auto"` — Use a local GPU if one is available. Otherwise, use the local CPU.
- `"cpu"` — Use the local CPU.
- `"gpu"` — Use the local GPU.

`Acceleration` — Acceleration
`"auto"` (default) | `"none"`

Acceleration, specified as `"auto"` or `"none"`. The `"auto"` acceleration setting increases memory usage but also speeds up execution.

Output Arguments

`flow` — Optical flow velocity matrices
`opticalFlow` object

Optical flow velocity matrices, returned as an `opticalFlow` object.

Tips

- Using RAFT for optical flow estimation on a GPU requires a minimum of 12 GB of memory.
- The RAFT model, being fully convolutional, can process images of any size in theory, with the only limitation being the available GPU memory.

Version History

Introduced in R2024b

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `estimateFlow` function uses a GPU if one is available. You can specify the hardware that the `estimateFlow` function uses by setting the `ExecutionEnvironment` name-value argument.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Objects
`opticalFlowRAFT`

Functions
`reset` | `imwarp`

Topics
“Automate Labeling of Objects in Video Using RAFT Optical Flow”

reset

Reset the internal state of the optical flow estimation object

Syntax

```
reset(flowModel)
```

Description

`reset(flowModel)` resets the internal state of the optical flow estimation object, `flowModel`. The previous frame is reset to its initial value which is a uniform image of intensity value `0`. To reuse an `opticalFlowRAFT` object for a new video, call `reset` to clear cached frame information from a previously processed video.

Examples

Estimate Optical Flow in Consecutive Image Frames

Create a RAFT optical flow object.

```
flowModel = opticalFlowRAFT;
```

Create an object to read the input video file.

```
vidReader = VideoReader("visiontraffic.avi",CurrentTime=11);
```

Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h, Position=[0 0 1 1], Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

Read consecutive image frames to estimate optical flow. Display the current frame and overlay optical flow vectors using a quiver plot. The `estimateFlow` function calculates the optical flow between two consecutive frames.

Note that the function internally stores the previous frame and utilizes it implicitly for optical flow estimation. Consequently, when the function is called for the first time on a sequence of frames, it will return a zero flow. This is because, in the absence of a genuine previous frame, the initial frame is treated as both the current and previous frame, leading to no detectable motion between the two. This is consistent with the argument structure and behavior of established optical flow estimation methods, such as `opticalFlowFarneback`.

```
while hasFrame(vidReader)
    frame = readFrame(vidReader);
    flow = estimateFlow(flowModel,frame);

    imshow(frame)
    hold on
```

```
    plot(flow,DecimationFactor=[10 10],ScaleFactor=0.45,Parent=hPlot,color="g");
    hold off
    pause(10^-3)
end
```



Reset the `opticalFlowRAFT` object after the video processing has completed. This clears the internal state of the object, including the saved previous frame.

```
reset(flowModel);
```

Memory-Efficient RAFT Optical Flow for Large Images Using Rescaling

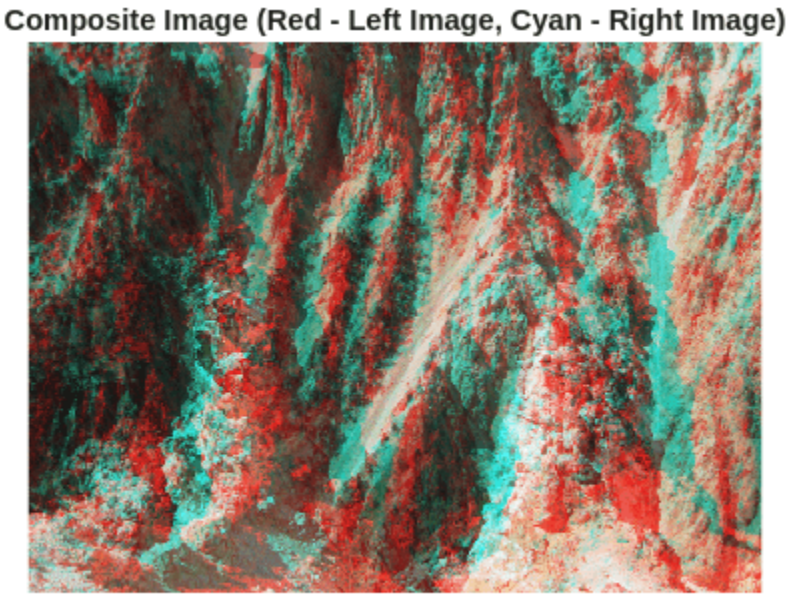
RAFT provides accurate optical flow estimation, but it can be memory-intensive when applied to large images or run on GPUs with limited memory. In such cases, processing high-resolution frames directly may lead to out-of-memory (OOM) errors. A practical solution is to downscale the input images before passing them to the RAFT model, compute optical flow at the reduced resolution, and then rescale the resulting flow field back to the original size of the imagery. This example demonstrates how to apply this memory-efficient strategy, allowing you to use RAFT on larger images or resource-constrained hardware while preserving meaningful flow estimates.

Load Image Pair

Load two images and visualize their pixel-wise differences due to camera motion using a red-cyan composite image.

```
I1 = imread("yellowstone_left.png");
I2 = imread("yellowstone_right.png");
```

```
figure
imshow(stereoAnaglyph(I1, I2))
title("Composite Image (Red - Left Image, Cyan - Right Image)")
```



Display the spatial dimensions of the images.

```
disp(size(I1, [1 2]))
```

```
   480   640
```

Create `opticalFlowRAFT` Object

Create an `opticalFlowRAFT` object for computing optical flow between two images.

```
raft = opticalFlowRAFT;
```

Estimate Optical Flow on Full Resolution Images

Estimate optical flow on the images at original resolution. This gives the most accurate result but may cause into slow runtime and high memory consumption.

```
estimateFlow(raft, I1);
flowFull = estimateFlow(raft, I2);
```

Estimate Optical Flow on Downsampled Images

Create Downscaled Images

Scale the image dimensions by a factor of 0.5 for illustrative purpose. A small resolution image can result in faster optical flow computation and less memory consumption.

```
I1Small = imresize(I1, 0.5);
I2Small = imresize(I2, 0.5);
```

Display the resized image dimensions.

```
disp(size(I1Small, [1 2]))
```

```
   240   320
```

Estimate Flow on Downscaled Images

Compute optical flow on downscaled images. Use the same opticalFlowRAFT model instantiated earlier after resetting it.

```
reset(raft);
estimateFlow(raft, I1Small);
flowLowRes = estimateFlow(raft, I2Small);
```

Scale Estimated Flow to Original Resolution

Upsample and scale the lower resolution optical flow back to the original image resolution.

```
origDims = size(I1, [1 2]);
scaleX = origDims(2) / size(flowLowRes.Magnitude, 2);
scaleY = origDims(1) / size(flowLowRes.Magnitude, 1);

flowX = imresize(flowLowRes.Vx, origDims, "bilinear") * scaleX;
flowY = imresize(flowLowRes.Vy, origDims, "bilinear") * scaleY;

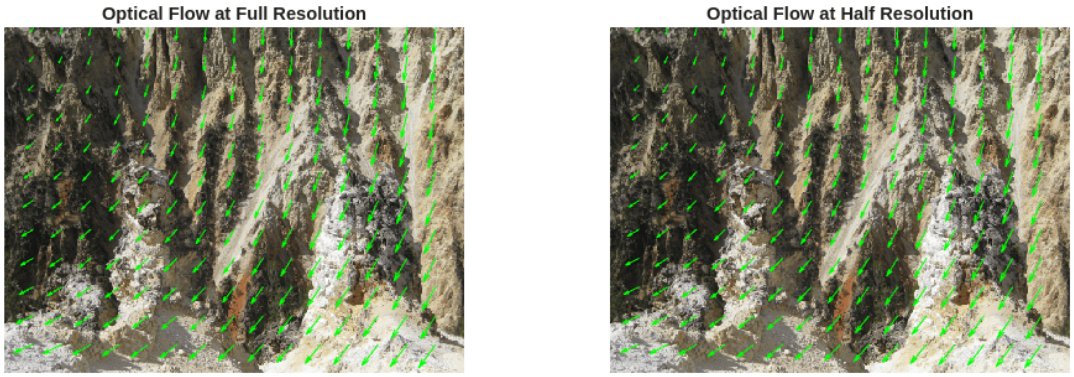
flowUpscaled = opticalFlow(flowX, flowY);
```

Compare Optical Flow at Original and Downscaled Resolutions

Visually compare optical flow computed directly on the original high-resolution images with optical flow computed on downscaled images and then rescaled back to the original size. This comparison illustrates how the downscale-upscale workflow approximates the original flow while reducing memory usage.

```
figure
subplot(1,2,1)
imshow(I1)
hold on
plot(flowFull, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Full Resolution")

subplot(1,2,2)
imshow(I1)
hold on
plot(flowUpscaled, DecimationFactor=[40 40], ScaleFactor=0.75, Color="g");
hold off
title("Optical Flow at Half Resolution")
```



The bilinear interpolation in the image resizing operations can introduce numerical inaccuracies and overly-smoothed results in the upscaled optical flow. This reduced accuracy is a trade-off against the lower memory consumption when optical flow computation is performed on lower resolution imagery.

Input Arguments

`flowModel` — Object for optical flow estimation

`opticalFlowRAFT` object

Object for optical flow estimation, specified as an `opticalFlowRAFT` object.

Version History

Introduced in R2024b

See Also

Objects

`opticalFlowRAFT`

Functions

`estimateFlow` | `imwarp`

Topics

“Automate Labeling of Objects in Video Using RAFT Optical Flow”

dbowLoopDetector

Detect loop closure using visual features

Description

Use `dbowLoopDetector` object to create loop detector database.

Creation

Syntax

```
loopDetector = dbowLoopDetector()
loopDetector = dbowLoopDetector(bag)
```

Description

`loopDetector = dbowLoopDetector()` creates a loop detector database using a default internal vocabulary. It detects loop closures in visual simultaneous localization and mapping (vSLAM) using visual features.

`loopDetector = dbowLoopDetector(bag)` creates a loop detector database specified by feature descriptors in `bag`.

Object Functions

addVisualFeatures	Add image features to database
detectLoop	Detect loop closure using visual features

Examples

Implement Loop Closure Detection and Add New Image

Load an existing binary visual vocabulary for loop detection.

```
bag = bagOfFeaturesDBoW("bagOfFeatures.bin.gz");
```

Initialize the loop detector using the loaded vocabulary.

```
loopDetector = dbowLoopDetector(bag);
```

Read a new image and detect ORB features in the image.

```
I = imread("cameraman.tif");
points = detectORBFeatures(I);
imshow(I)
hold on
plot(points,Showscale=false)
```



Extract ORB features from the detected points in the image. The `extractFeatures` function returns features and their corresponding locations. This code focuses on only the features for loop closure detection.

```
features = extractFeatures(I,points);
```

Perform loop closure detection with the extracted features.

```
loopViewIds = detectLoop(loopDetector,features);
```

Update the loop detector's database with features from the new image by specifying the new image as the 100th view in the sequence. This supports loop detection in future images similar to the current one.

```
viewId = 100;
addVisualFeatures(loopDetector,viewId,features);
```

Loop Closure Detection in a Sequence of Images

Load a pre-existing binary vocabulary for feature description.

```
bag = bagOfFeaturesDBoW("bagOfFeatures.bin.gz");
```

Initialize the loop detector with the loaded vocabulary.

```
loopDetector = dbowLoopDetector(bag);
```

Use a single image to simulate adding different views.

```
I = im2gray(imread("cameraman.tif"));
points = detectORBFeatures(I);
[features,points] = extractFeatures(I,points);
```

Initialize an image view set to manage and store views.

```
vSet = imageviewset;
```

Add the first view with random features for initialization.

```
zeroFeatures = binaryFeatures(zeros(size(points,1),32,"like",uint8(0)));
vSet = addView(vSet,1,"Features",zeroFeatures,"Points",points);
addVisualFeatures(loopDetector,1,zeroFeatures);
fig = figure;
tlo = tiledlayout(fig,1,3,TileSpacing='None');
for i = 1:3
    ax = nexttile(tlo);
    imshow(I,Parent=ax)
    title(["Camera View ", num2str(i)])
end
```



Sequentially add three views with actual features to simulate potential loop candidates.

```
for viewId = 2:4
    vSet = addView(vSet,viewId,"Features",features,"Points",points);
    addVisualFeatures(loopDetector,viewId,features);
end
```

Add two new connected views to the image sequence. First, add a previous view with a cropped section of the original image to represent the view from a camera. The resulting cropped image represents the previous frame in the sequence.

```
prevViewId = 5;
prevView = I(100:200,100:200);
```

```
figure
imshow(prevView)
title("Camera View 5")
```



Detect ORB features in the cropped frame with the pyramid level set to 3. Add the features to the image viewset as a new view.

```
prevPoints = detectORBFeatures(prevView,NumLevels=3);
[prevFeatures,prevPoints] = extractFeatures(prevView,prevPoints);
vSet = addView(vSet,prevViewId,"Features",prevFeatures,"Points",prevPoints);
addVisualFeatures(loopDetector,prevViewId,prevFeatures);
```

Add a current view, connected to the previous one with another cropped section.

```
currViewId = 6;
currView = I(50:200, 50:200);
currPoints = detectORBFeatures(currView,NumLevels=3);
[currFeatures, currPoints] = extractFeatures(currView,currPoints);
vSet = addView(vSet,currViewId,"Features",currFeatures,"Points",currPoints);
vSet = addConnection(vSet,prevViewId,currViewId,"Matches",[1:10; 1:10]');
imshow(currView)
title("Current Camera View 6")
```



Identify views connected to the current key frame.

```
covisViews = connectedViews(vSet, currViewId);
covisViewsIds = covisViews.ViewId;
```

Perform loop closure detection by comparing current features against those from connected views. Use 75% of the maximum score among connected views for the threshold.

```
relativeThreshold = 0.75;
loopViewIds = detectLoop(loopDetector,currFeatures,covisViewsIds,relativeThreshold)
```

```
loopViewIds = 1×3 int32 row vector

   2   3   4
```

The detected loop candidates show that the current camera view 6 is visually most similar to views 2,3, and 4.This suggests that these earlier views share significant visual features with the current view.

```
fig= figure;
tlo = tiledlayout(fig,1,4,TileSpacing='None');
for i = 1:4
    ax = nexttile(tlo);
    if i == 1
        imshow(currView,Parent=ax)
        title("Current Camera View 6")
    else
        imshow(I,Parent=ax)
        title(["Camera View ", num2str(i)])
    end

end
```



Version History

Introduced in R2024b

References

[1] Galvez-López, D., and J. D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences.” *IEEE Transactions on Robotics*, vol. 28, no. 5, Oct. 2012, pp. 1188–97. *DOI.org (Crossref)*, https://doi.org/10.1109/TRO.2012.2197158.

See Also

Objects

`bagOfFeaturesDBoW`

Topics

“Monocular Visual Simultaneous Localization and Mapping”

addVisualFeatures

Add image features to database

Syntax

```
addVisualFeatures(loopDetector,viewId,features)
```

Description

`addVisualFeatures(loopDetector,viewId,features)` adds ORB feature descriptors into the bag of words (BoW) loop detector database, `loopDetector`. The function adds `features` extracted from images uniquely identified by its view identifier `viewId`.

Examples

Add Image Features to Loop Detector Database

Create a loop detector.

```
loopDetector = dbowLoopDetector;
```

Get query image features and assign it as the 100th view.

```
viewId = 100;
I = imread("cameraman.tif");
points = detectORBFeatures(I);
[features,~] = extractFeatures(I,points);
```

Add image features for the 100th view ID to the loop detector database.

```
addVisualFeatures(loopDetector,viewId,features);
```

Input Arguments

`loopDetector` — BoW loop detector database
`dbowLoopDetector` object

BoW loop detector database, specified as a `dbowLoopDetector` object.

`viewId` — Unique view identifier
integer

Unique view identifier, specified as an integer.

`features` — ORB feature descriptors
`binaryFeatures` object

ORB feature descriptors, specified as a `binaryFeatures` object.

Version History

Introduced in R2024b

See Also

Objects

`dbowLoopDetector` | `bagOfFeaturesDBoW` | `binaryFeatures`

detectLoop

Detect loop closure using visual features

Syntax

```
[loopViewIDs,similarityScores] = detectLoop(loopDetector,features)
[loopViewIDs,similarityScores] = detectLoop( ___ ,connectedViewIds,
relativeThreshold)
[ ___ ] = detectLoop( ___ ,NumResults=Value)
```

Description

`[loopViewIDs,similarityScores] = detectLoop(loopDetector,features)` compares oriented FAST and rotated BRIEF (ORB) `features`, against features in a prepopulated `loopDetector` BoW database. It returns the indices of images, `loopViewIDs`, in the database that resemble the feature descriptors in `features` and it returns a quantitative measure of how similar each detected pair of images is in `similarityScores`.

By using the `detectLoop` function with the specified inputs, you can effectively search a database for images that are similar to a given set of query images, while ignoring those that are closely connected.

`[loopViewIDs,similarityScores] = detectLoop( ___ ,connectedViewIds, relativeThreshold)` finds images within a BoW database that are similar to an image represented by ORB feature descriptors, `features`, while ignoring those that are closely connected, specified by `connectedViewIds`. The `relativeThreshold` allows you to fine-tune the sensitivity of the search to ensure that only the most relevant, non-connected similar images are identified and returned.

`[ ___ ] = detectLoop( ___ ,NumResults=Value)` sets the maximum number of results to return in addition to all input arguments from the previous syntax. Specify `Value` as a positive integer. The default value is set to `20`.

Examples

Implement Loop Closure Detection and Add New Image

Load an existing binary visual vocabulary for loop detection.

```
bag = bagOfFeaturesDBoW("bagOfFeatures.bin.gz");
```

Initialize the loop detector using the loaded vocabulary.

```
loopDetector = dbowLoopDetector(bag);
```

Read a new image and detect ORB features in the image.

```
I = imread("cameraman.tif");
points = detectORBFeatures(I);
imshow(I)
```

```
hold on
plot(points,Showscale=false)
```



Extract ORB features from the detected points in the image. The `extractFeatures` function returns features and their corresponding locations. This code focuses on only the features for loop closure detection.

```
features = extractFeatures(I,points);
```

Perform loop closure detection with the extracted features.

```
loopViewIds = detectLoop(loopDetector,features);
```

Update the loop detector's database with features from the new image by specifying the new image as the 100th view in the sequence. This supports loop detection in future images similar to the current one.

```
viewId = 100;
addVisualFeatures(loopDetector,viewId,features);
```

Loop Closure Detection in a Sequence of Images

Load a pre-existing binary vocabulary for feature description.

```
bag = bagOfFeaturesDBoW("bagOfFeatures.bin.gz");
```

Initialize the loop detector with the loaded vocabulary.

```
loopDetector = dbowLoopDetector(bag);
```

Use a single image to simulate adding different views.

```
I = im2gray(imread("cameraman.tif"));
points = detectORBFeatures(I);
[features,points] = extractFeatures(I,points);
```

Initialize an image view set to manage and store views.

```
vSet = imageviewset;
```

Add the first view with random features for initialization.

```
zeroFeatures = binaryFeatures(zeros(size(points,1),32,"like",uint8(0)));
vSet = addView(vSet,1,"Features",zeroFeatures,"Points",points);
addVisualFeatures(loopDetector,1,zeroFeatures);
fig = figure;
tlo = tiledlayout(fig,1,3,TileSpacing='None');
for i = 1:3
    ax = nexttile(tlo);
    imshow(I,Parent=ax)
    title(["Camera View ", num2str(i)])
end
```



Sequentially add three views with actual features to simulate potential loop candidates.

```
for viewId = 2:4
    vSet = addView(vSet,viewId,"Features",features,"Points",points);
    addVisualFeatures(loopDetector,viewId,features);
end
```

Add two new connected views to the image sequence. First, add a previous view with a cropped section of the original image to represent the view from a camera. The resulting cropped image represents the previous frame in the sequence.

```
prevViewId = 5;
prevView = I(100:200,100:200);
```

```
figure
imshow(prevView)
title("Camera View 5")
```



Detect ORB features in the cropped frame with the pyramid level set to 3. Add the features to the image viewset as a new view.

```
prevPoints = detectORBFeatures(prevView,NumLevels=3);
[prevFeatures,prevPoints] = extractFeatures(prevView,prevPoints);
vSet = addView(vSet,prevViewId,"Features",prevFeatures,"Points",prevPoints);
addVisualFeatures(loopDetector,prevViewId,prevFeatures);
```

Add a current view, connected to the previous one with another cropped section.

```
currViewId = 6;
currView = I(50:200, 50:200);
currPoints = detectORBFeatures(currView,NumLevels=3);
[currFeatures, currPoints] = extractFeatures(currView,currPoints);
vSet = addView(vSet,currViewId,"Features",currFeatures,"Points",currPoints);
vSet = addConnection(vSet,prevViewId,currViewId,"Matches",[1:10; 1:10]');
imshow(currView)
title("Current Camera View 6")
```



Identify views connected to the current key frame.

```
covisViews = connectedViews(vSet, currViewId);
covisViewsIds = covisViews.ViewId;
```

Perform loop closure detection by comparing current features against those from connected views. Use 75% of the maximum score among connected views for the threshold.

```
relativeThreshold = 0.75;
loopViewIds = detectLoop(loopDetector,currFeatures,covisViewsIds,relativeThreshold)
```

```
loopViewIds = 1×3 int32 row vector

   2   3   4
```

The detected loop candidates show that the current camera view 6 is visually most similar to views 2,3, and 4.This suggests that these earlier views share significant visual features with the current view.

```
fig= figure;
tlo = tiledlayout(fig,1,4,TileSpacing='None');
for i = 1:4
    ax = nexttile(tlo);
    if i == 1
        imshow(currView,Parent=ax)
        title("Current Camera View 6")
    else
        imshow(I,Parent=ax)
        title(["Camera View ", num2str(i)])
    end

end
```



Input Arguments

`loopDetector` — BoW loop detector database

`dbowLoopDetector` object

BoW loop detector database, specified as `dbowLoopDetector` object. This prepopulated database of images is search to find similar images.

`features` — ORB feature descriptors

`binaryFeatures` object

ORB feature descriptors, specified as a `binaryFeatures` object. These features are used to search the database for similar images.

`connectedViewIds` — Connected view identifiers

M-element vector

Connected view identifiers, specified as an *M*-element vector. The length *M*, is the number of images considered connected at the time of the query. For example, if the value of `connectedViewIds = [2 5 7]`, this means that the images or views with indices 2,5, and 7 in the database are considered connected to the current query images. The function will ignore these indices during its search for similar images, focusing instead on finding matches among the non-connected images.

`relativeThreshold` — Relative threshold

scalar value

Relative threshold for filtering search results, specified as a scalar value between 0 and 1. Use this threshold to help exclude less similar images by setting a minimum similarity score based on the connected images scores.

A value of 0.75 (or 75%) is recommended for the `relativeThreshold`. This setting means that only images with similarity scores greater than 75% of the minimum score among the connected views are returned, ensuring a high relevance of the detected loops

Output Arguments

`loopViewIDs` — Image indices in loop detector database

M-element vector

Image indices in loop detector database, returned as an *M*-element vector. *M* is the number of queries. The indices points to an image that is similar to the query image, excluding those marked as connected. These indices allow you to locate the matching images within the database.

`similarityScores` — Similarity score between each matched image and query image

M-element vector

Similarity score between each matched image and query image, returned as an *M*-element vector. *M* is the number of queries. Higher scores indicate closer matches.

The function calculates the minimum similarity score among the connected views. It then multiplies this minimum score by the `relativeThreshold` to establish a cutoff score. Only non-connected images with similarity scores above this cut-off are considered significant and included in the results.

Version History

Introduced in R2024b

References

[1] Galvez-López, D., and J. D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences.” *IEEE Transactions on Robotics*, vol. 28, no. 5, Oct. 2012, pp. 1188–97. *DOI.org (Crossref)*, https://doi.org/10.1109/TRO.2012.2197158.

See Also

Objects

`dbowLoopDetector`

Topics

“Monocular Visual Simultaneous Localization and Mapping”

bagOfFeaturesDBoW

Bag of visual words using DBoW2 library

Description

Use the `bagOfFeaturesDBoW` object to create a bag of words (BoW) vocabulary to use for loop closure detection with vSLAM algorithms. This object uses the distributed bag of words DBoW2 library. This object supports only oriented FAST and rotated BRIEF (ORB) features.

Creation

Syntax

```
bag = bagOfFeaturesDBoW(imds)
bag = bagOfFeaturesDBoW(features)
bag = bagOfFeaturesDBoW( ___ ,Name=Value)
bag = bagOfFeaturesDBoW(vocabularyFileName)
```

Description

`bag = bagOfFeaturesDBoW(imds)` creates a bag of features from images specified by the datastore `imds`. The algorithm uses oriented fast and rotated brief (ORB) features to create the visual vocabulary.

`bag = bagOfFeaturesDBoW(features)` creates a bag of features from ORB feature descriptors specified by `features`.

`bag = bagOfFeaturesDBoW( ___ ,Name=Value)` sets properties by using one or more name-value arguments in addition to the previous syntax. For example, `Normalization="L2"` sets the normalization to `L2`.

`bag = bagOfFeaturesDBoW(vocabularyFileName)` loads an existing DBoW vocabulary file.

Input Arguments

`imds` — Images
`ImageDatastore` object

Images, specified as an `ImageDatastore` object.

`features` — ORB feature descriptors
cell array of `binaryFeatures`

ORB feature descriptors, specified as a cell array of `binaryFeatures` objects.

`vocabularyFileName` — DBoW vocabulary file
character string

DBoW vocabulary file, specified as a character string.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

`TreeProperties` — Vocabulary tree properties

`[5 10]` (default) | 2-element vector

Vocabulary tree properties, specified as a 2-element vector of the form [*depthLevel*,*branchingFactor*]. *depthLevel* is the number of levels in the vocabulary tree, specified as an integer. *branchingFactor* is the factor to control the amount that the vocabulary can grow at successive levels in the tree, specified as an integer.

The capacity of a vocabulary tree to represent visual words is determined by the formula: `branchingFactor` x `depthLevel`. Commonly, the *depthLevel* ranges from 1 to 6, while the `branchingFactor` varies between 10 and 500. To identify the optimal values for these parameters, conducting empirical analysis is recommended.

Increasing the branching factor enlarges the vocabulary, potentially improving the accuracy of image similarity assessments. However, this adjustment also leads to an increase in the time required to encode images. Implementing a vocabulary tree with multiple levels can facilitate the creation of vocabularies comprising over 10,000 visual words. Although this multi-level approach streamlines the encoding process for images associated with large vocabularies, it necessitates a more extended setup phase.

Alternatively, for vocabularies containing only 100 to 1,000 visual words, employing a tree with a single level is advisable. This configuration simplifies the structure and accelerates the creation process, albeit for smaller vocabularies.

`Normalization` — Type of normalization applied to features

`L1` (default) | `L2`

Type of normalization applied to the features, specified as `L1` or `L2`. Normalization is the method by which to evaluate the similarity or dissimilarity between the feature descriptors for the bag of features.

- `L1` — Calculates the sum of the absolute values of the vector elements.
- `L2` — Calculates the square root of the sum of the squared values of the vector elements.

Properties

`DepthLevel` — Number of levels in the vocabulary tree

integer

This property is read-only.

Number of levels in the vocabulary tree, returned as an integer. The `Normalization` property sets this value.

The `TreeProperties` name-value argument sets this property.

`BranchingFactor` — Number of branches of every node

integer

This property is read-only.

Number of branches of every node in the vocabulary tree, returned as an integer. The `Normalization` property sets this value.

The `TreeProperties` name-value argument sets this property.

`Normalization` — Type of normalization applied to features
`L1` (default) | `L2`

Type of normalization applied to the features, returned as an `L1` or `L2`.

The `Normalization` name-value argument sets this property.

Examples

Create Bag of Features Using an Existing Vocabulary File

Create a bag of features object using an existing distributed bag of words (DBoW) vocabulary file.

```
bag = bagOfFeaturesDBoW("bagOfFeatures.bin.gz")
```

```
bag =
  bagOfFeaturesDBoW with properties:

         DepthLevel: 5
    BranchingFactor: 10
      Normalization: 'L1'
```

Create Bag of Features Using Image Datastore

Create an image datastore for a set of images with stop signs.

```
folder=fullfile(toolboxdir("vision"),"visiondata","stopSignImages");
imds=imageDatastore(folder);
```

Visualize an image from the datastore.

```
imshow(preview(imds))
```



Create a bag of features vocabulary representation from the images in the datastore.

```
bag = bagOfFeaturesDBoW(imds)
```

```
bag =
  bagOfFeaturesDBoW with properties:

         DepthLevel: 5
    BranchingFactor: 10
      Normalization: 'L1'
```

Create Bag of Features Using Image Features

Read an image.

```
I = imread("cameraman.tif");
imshow(I)
```



Detect ORB features in the image.

```
points = detectORBFeatures(I);
imshow(I)
hold on
plot(points,ShowScale=false)
```



Extract ORB features from the detected points in the image. The `extractFeatures` function returns features and their corresponding locations. This code focuses on only the features for loop closure detection.

```
features = extractFeatures(I,points);
```

Create a bag of features using the extracted ORB features. Specify L2 normalization method to normalize image encodings in the bag.

```
bag = bagOfFeaturesDBoW({features},Normalization="L2")
```

```
bag =
  bagOfFeaturesDBoW with properties:

         DepthLevel: 5
    BranchingFactor: 10
      Normalization: 'L2'
```

You can now apply this bag to tasks such as loop closure detection in visual SLAM workflows using the `dbowLoopDetector` object.

Version History

Introduced in R2024b

References

[1] Galvez-López, D., and J. D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences.” *IEEE Transactions on Robotics*, vol. 28, no. 5, Oct. 2012, pp. 1188–97. *DOI.org (Crossref)*, https://doi.org/10.1109/TRO.2012.2197158.

See Also

Objects
`dbowLoopDetector`

Topics
“Monocular Visual Simultaneous Localization and Mapping”

External Websites
DBoW2

trajectoryErrorMetrics

Store accuracy metrics for trajectories

Description

The `trajectoryErrorMetrics` object stores accuracy metrics for trajectories, including both absolute trajectory error and relative trajectory error, across a sequence of poses.

Creation

Create a `trajectoryErrorMetrics` object by using the `compareTrajectories` function.

Properties

`AbsoluteError` — Absolute trajectory error
M-by-2 matrix (default)

This property is read-only.

Absolute trajectory error, specified as an *M*-by-2 matrix of the form [*absRotationError*,*absTranslationError*]. *M* specifies the number of poses. *absRotationError* specifies the absolute rotation error in degrees and *absTranslationError* specifies the absolute translation error. The *absRotationError* is the magnitude of the rotation angle in degrees, obtained by converting the rotation matrix in *AbsoluteError_i* to the angle-axis representation. The *absTranslationError* is the Euclidean norm of the translation vector derived from *AbsoluteError_i*.

The absolute trajectory pose error calculates the exact differences between where a robot thinks it is (estimated pose), and its actual location (ground truth), at specific times. It quantifies how accurately the estimated path aligns with the actual (ground truth) path. Mathematically, if P_i^{est} is the i-th estimated pose and P_i^{gt} is the corresponding ground truth pose, then the absolute pose error of the i-th estimated pose can be defined as the:

$$AbsoluteError_i = \left\| P_i^{est} - P_i^{gt} \right\|$$

or as,

$$AbsoluteError_i = P_i^{gt} \cdot (P_i^{est})^{-1}$$

`RelativeError` — Relative trajectory pose error
M-by-2 matrix (default)

This property is read-only.

Relative trajectory pose error, specified as an *M*-by-2 matrix of the form [*relRotationError*,*relTranslationError*]. *M* specifies the number of poses. *relRotationError* specifies the relative rotation error in degrees and *relTranslationError* specifies the relative translation error.

The *relRotationError* represents the magnitude of the rotation angle, measured in degrees. This value is obtained after converting the rotation matrix in $RelativeError_{i+\Delta}$ to the angle-axis representation.

The *relTranslationError* is the Euclidean norm of the translation component extracted from $RelativeError_{i+\Delta}$.

The relative pose error calculates the differences in the robot's movement over a set distance or interval, rather than its exact position at a specific point in time. It measures how much the robot's estimated change in position (estimated relative pose) deviates from the actual change in position (ground truth relative pose) over this interval. This helps in assessing the accuracy of the robot's perception of its own movement over time.

To compute relative pose error, compare pairs of poses $(P_i^{est}, P_{i+\Delta}^{est})$ from the estimated trajectory $(P_i^{gt}, P_{i+\Delta}^{gt})$ from the actual (ground truth) trajectory, spaced by a set distance.

The error is the difference in movement between these pairs.

$$RelativeError_{i+\Delta} = \left\| \left(P_i^{est}\right)^{-1} \bullet P_{i+\Delta}^{est} - \left(P_i^{gt}\right)^{-1} \bullet P_{i+\Delta}^{gt} \right\|$$

or,

$$RelativeError_{i+\Delta} = \left(P_i^{gt}\right)^{-1} \cdot P_{i+\Delta}^{gt} \cdot \left(\left(P_i^{est}\right)^{-1} \cdot P_{i+\Delta}^{est}\right)^{-1}$$

`AbsoluteRMSE` — Root mean square of absolute trajectory error
2-element vector (default)

This property is read-only.

Root mean square (RMS) of absolute trajectory error, specified as a 2-element vector of the form [*absRotationRMS*,*absTranslationRMS*]. The RMS is calculated as:

- *absRotationRMS* = `rms`(*absRotationRMS*)
- *absTranslationRMS* = `rms`(*absTranslationRMS*)

The absolute pose error can be summarized over the entire trajectory by taking the root mean square of the individual errors.

$$AbsoluteRMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} AbsoluteError_i^2}$$

`RelativeRMSE` — Root mean square of relative trajectory error
2-element vector (default)

This property is read-only.

Root mean square of relative (RMS) trajectory error, specified as a 2-element vector of the form [*relRotationRMS*,*relTranslationRMS*]. The RMS is calculated as:

- *relRotationRMS* = `rms`(*relRotationRMS)*
- *relTranslationRMS* = `rms`(*relTranslationRMS*)

The relative pose error can be summarized over the entire trajectory by taking the root mean square of the individual errors.

$$RelativeRMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} RelativeError_i^2}$$

Object Functions

plot Plot aligned estimated trajectory and error metrics

Version History

Introduced in R2024b

References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. 2012. "A Benchmark for the Evaluation of RGB-D SLAM Systems." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 573-580.

[2] Zhang, Zichao, and Davide Scaramuzza. 2018. "A Tutorial on Quantitative Trajectory Evaluation for Visual (-Inertial) Odometry." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 7244-7251.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`compareTrajectories`

Objects

`rgbdvslam` | `monovslam` | `stereovslam`

plot

Plot aligned estimated trajectory and error metrics

Syntax

```
plot(metrics,metricType)
ax = plot( ___ )
plot( ___ ,Name=Value)
```

Description

`plot(metrics,metricType)` plots the aligned estimated trajectory and error values, color-coding the trajectory based on the specified metric type `metricType`, and the error values, linearly mapped to colors in the current colormap.

`ax = plot( ___ )` additionally returns the axes handle.

`plot( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `ShowGroundTruth=false`, suppresses the display of the ground truth trajectory.

Input Arguments

`metrics` — Trajectory Error metrics

`trajectoryErrorMetrics`object

Trajectory Error metrics, specified as a `trajectoryErrorMetrics` object.

`metricType` — Metric type

`"absolute-rotation"` | `"absolute-translation` | `"relative-rotation"` | `"relative-translation"`

Metric type, specified as `"absolute-rotation"`, `"absolute-translation`, `"relative-rotation"`, or `"relative-translation"`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(metric,metricType,ShowGroundTruth=false)`, suppresses the display of the ground truth trajectory.

`VerticalAxis` — Point cloud vertical axis and direction

`"YDown"` (default) | `"ZUp"` | `"ZDown"` | `"XUp"` | `"XDown"` | `"YUp"`

Point cloud vertical axis and direction, specified as `"ZUp"`, `"ZDown"`, `"XUp"`, `"XDown"`, `"YUp"`, or `"YDown"`. This argument sets the axis of the point cloud as the vertical axis, with the positive direction based on the value specified.

`ShowGroundTruth` — Display ground truth trajectory
`false` (default) | `true`

Display ground truth trajectory, specified as `true` or `false`.

`Parent` — Axes on which to display visualization
`Axes` graphics object

Axes on which to display the visualization, specified as an `Axes` graphics object. To create an `Axes` object, use the `axes` function. To display the visualization in a new figure, leave `Parent` unspecified.

Output Arguments

`ax` — Axes handle
`Axes` graphics object (default)

Axes handle, returned as an `axes` graphics object.

Version History

Introduced in R2024b

See Also

Objects
`trajectoryErrorMetrics`

stereovslam

Feature-based visual simultaneous localization and mapping (vSLAM) and visual-inertial sensor fusion with stereo camera

Description

Use the `stereovslam` object to perform visual simultaneous localization and mapping (vSLAM) with stereo camera data. To learn more about visual SLAM, see “Implement Visual SLAM in MATLAB”.

The `stereovslam` object extracts Oriented FAST and Rotated BRIEF (ORB) features from incrementally read images, and then tracks those features to estimate camera poses, identify key frames, and reconstruct a 3-D environment. The vSLAM algorithm also searches for loop closures using the bag-of-features algorithm, and then optimizes the camera poses using pose graph optimization. You can enhance the accuracy and robustness of the SLAM by integrating this object with IMU data to perform visual-inertial sensor fusion.

Note The `stereovslam` object requires Navigation Toolbox™.

Creation

Syntax

```
vslam = stereovslam(intrinsics,baseline)
vslam = stereovslam(reprojectionMatrix,imageSize)
vslam = stereovslam( ___ ,imuParameters)
vslam = stereovslam( ___ ,PropertyName=Value)
```

Description

`vslam = stereovslam(intrinsics,baseline)` creates a stereo visual SLAM object, `vslam`, using the rectified stereo camera intrinsic parameters `intrinsics`, and the `baseline` distance between the rectified left and right cameras.

The object represents 3-D map points and camera poses in world coordinates, and assumes the camera pose of the first key frame is an identity `rigidtform3d` transform.

Note The `stereovslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`vslam = stereovslam(reprojectionMatrix,imageSize)` creates a stereo visual SLAM object `vslam` using the stereo camera reprojection matrix, `reprojectionMatrix`, and the image size, `imageSize`.

`vslam = stereovslam( ___ ,imuParameters)` performs stereo visual-inertial SLAM based on the specified `imuParameters`.

`vslam = stereovslam( ___ ,PropertyName=Value)` sets properties using one or more name-value arguments in addition to any combination of input arguments from previous syntaxes. For example, `MaxNumPoints=850` sets the maximum number of ORB feature points to extract from each image to `850`.

Input Arguments

`intrinsics` — Rectified stereo camera intrinsic parameters

`cameraIntrinsics` object

Rectified stereo camera intrinsic parameters, specified as a `cameraIntrinsics` object.

This argument sets the `Intrinsics` property.

`baseline` — Distance between rectified left and right cameras

nonnegative scalar

Distance between the rectified left and right cameras, specified as a nonnegative scalar. Stereo vSLAM algorithms typically track the primary (or left) camera, in which case the baseline is greater than zero. A negative baseline value indicates a negative disparity range, and the vSLAM algorithm tracks the secondary (or right) camera instead.

This argument sets the `Baseline` property.

`reprojectionMatrix` — Reprojection matrix

4-by-4 matrix

Reprojection matrix, specified as a 4-by-4 matrix of the form:

$$\begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/b & 0 \end{bmatrix},$$

where f and (cx,cy) are the focal length and principal point of rectified primary camera, respectively. b is the baseline of the virtual rectified stereo camera.

You can obtain the reprojection matrix by using the `rectifyStereoImages` function.

`imageSize` — Image size produced by camera

two-element vector

Image size produced by the camera, in pixels, specified as a two-element vector of the form [*nrows ncols*]. The elements *nrows* and *ncols* represent represents the number of rows and columns, respectively.

`imuParameters` — IMU parameters

`factorIMUParameters` object

IMU parameters, specified as a `factorIMUParameters` object. The object contains the noise, bias, and sample rate information about the inertial measurement unit (IMU).

Properties

Camera Parameters

`Intrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object

This property is read-only.

Camera intrinsic parameters, stored as a `cameraIntrinsics` object.

Use the `intrinsics` argument to set this property.

`Baseline` — Distance between rectified left and right cameras

nonnegative scalar

Distance between the rectified left and right cameras, stored as a nonnegative scalar. Stereo vSLAM algorithms typically track the primary (or left) camera, in which case the baseline is greater than zero. A negative baseline value indicates a negative disparity range, and the vSLAM algorithm tracks the secondary (or right) camera instead.

Use the `baseline` argument to set this property.

`DisparityRange` — Disparity range

`[0 48]` (default) | two-element vector of integers

This property is read-only.

Disparity range, specified as a two-element vector of integers of the form [*min max*]. The elements specify the minimum and maximum disparity, respectively. The range must be within the width of the image, and the difference between the minimum and the maximum must be divisible by 16.

`UniquenessThreshold` — Minimum value of uniqueness

`15` (default) | nonnegative integer

Minimum value of uniqueness, specified as a nonnegative integer.

The function marks the estimated disparity value K for a pixel as unreliable if:

$$v < V \times (1 + 0.01 \times \texttt{UniquenessThreshold}),$$

where V is the sum of the absolute difference (SAD) corresponding to the disparity value K, and v is the smallest SAD value over the whole disparity range, excluding K, K–1, and K+1.

Increasing the value of `UniquenessThreshold` results in the function marking disparity values for more pixels as unreliable. To turn off the uniqueness threshold, set this value to `0`.

Feature Extraction

`ScaleFactor` — Scale factor for image decomposition

`1.2` (default) | scalar greater than 1

This property is read-only after object creation.

Scale factor for image decomposition, stored as a scalar greater than 1. The scale factor is also referred to as the pyramid decimation ratio. Increasing the value of `ScaleFactor` reduces the

number of pyramid levels, but reduce computation time. Decreasing this value (down to just over 1), increases the number of pyramid levels, which can improve tracking performance, at the cost of computation speed. The scale value at each level of decomposition is `ScaleFactor`$^{(level\text{-}1)}$, where *level* is any value in the range [0, `Numlevels`-1]. Given the input image of size *M*-by-*N*, the image size at each level of composition is M_{level}-by-N_{level}, where:

$$M_{level} = \frac{M}{ScaleFactor^{(level\ -\ 1)}}$$

$$N_{level} = \frac{N}{ScaleFactor^{(level\ -\ 1)}}$$

`NumLevels` — Number of decomposition levels
`8` (default) | integer greater than or equal to `1`

This property is read-only after object creation.

Number of decomposition levels, specified as an integer greater than or equal to `1`. Increase this value to extract keypoints from the image at more levels of decomposition. Along with the `ScaleFactor` value, `NumLevels` controls the number of pyramid levels on which the object evaluates feature points.

The image size at each decomposition level limits the number of levels at which you can extract keypoints. The image size at a level of decomposition must be at least 63-by-63 for keypoint detection. The maximum level of decomposition is calculated as

$$level_{\max} = \text{floor}\left(\frac{\log(\min(M,N))\ -\ \log(63)}{\log(\text{ScaleFactor})}\right) + 1$$

If either the default value or the specified value of `NumLevels` is greater than $level_{max}$, the object modifies `NumLevels` to $level_{max}$ and returns a warning.

`MaxNumPoints` — Maximum number of ORB feature points uniformly extracted from each image
`1000` | positive integer

This property is read-only after object creation.

Maximum number of ORB feature points uniformly extracted from each image, specified as a positive integer. Values are typically in the range of [800, 2000], depending on the resolution of the image. When the number of extracted features is less than the value of `MaxNumPoints`, then the object uses all feature points.

Tracking

`TrackFeatureRange` — Range for the number of feature points to identify key frame
`[30 100]` | two-element vector of positive integers

This property is read-only after object creation.

Key frame feature point range, stored as a two-element vector of positive integers in the form [*lowerLimit upperLimit*]. This property specifies the minimum and maximum numbers of tracked features points a frame must contain for the object to identify it as a key frame. The `TrackFeatureRange` and the `SkipMaxFrames` properties enable you to control the frequency at which frames in the tracking process become key frames.

The success of tracking depends on the number of tracked points in the current frame, with one of these results:

- Tracking is lost — The number of tracked feature points in the current frame is less than the *lowerLimit* set by the `TrackFeatureRange` argument. This indicates that the image does not contain enough features, or that the camera is moving too fast. To improve the tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` property and decrease the `SkipMaxFrames` property to add key frames more frequently.
- Tracking is successful — The object identifies the current frame as a key frame. The number of tracked feature points in the current frame is in the range set by `TrackFeatureRange`.
- Tracking adds key frames too frequently — The number of tracked feature points in the current frame is greater than the *upperLimit* set by the `TrackFeatureRange` property. This indicates that the camera is moving very slowly, which produces an unnecessary number of key frames. To improve the tracking, you can reduce the frequency of adding key frames by increasing the value of the `SkipMaxFrames` property.

For more details, see the `addFrame` object function and “How to Improve Accuracy in Visual SLAM”.

`SkipMaxFrames` — Maximum number of skipped frames
`20` | positive integer

This property is read-only after object creation.

Maximum number of skipped frames, stored as a positive integer. When the number of tracked features is consistently greater than the *upperLimit* set by the `TrackFeatureRange` property, use the `SkipMaxFrames` property to control the frequency at which the object adds new key frames. The object identifies the current frame as a key frame when the number of skipped frames since the most recently added key frame equals the value of `SkipMaxFrames`. For more details, see “How to Improve Accuracy in Visual SLAM”.

`LoopClosureThreshold` — Minimum number of matched feature points between loop closure key frames
`60` | positive integer

This property is read-only after object creation.

Minimum number of matched feature points between loop closure key frames, stored as a positive integer.

`CustomBagOfFeatures` — Custom bag of features for loop detection
`[]` (default) | `bagOfFeaturesDBoW` object

This property is read-only after object creation.

Custom bag of features for loop detection, specified as a `bagOfFeaturesDBoW` object. The `bagOfFeaturesDBoW` enables you to create a custom bag of words (BoW) from feature descriptors, alongside options to utilize a built-in vocabulary or load a custom one from a specified file.

`Verbose` — Progress information display
`[]` (default) | `1` | `2` | `3`

This property is read-only after object creation.

Progress information display, specified as `[]`, `1`, `2`, or `3`. Paths to the location of log files, when they are created, is displayed on the command line. For more details, see “How to Improve Accuracy in Visual SLAM”.

Verbose value	Display description	Display location
`[]` or `false`	Display is turned off	—
`1` or `true`	Stages of vSLAM execution	Command window
`2`	Stages of vSLAM execution, with details on how the frame is processed, such as the artifacts used to initialize the map.	Log file in a temporary folder
`3`	Stages of vSLAM, artifacts used to initialize the map, poses and map points before and after bundle adjustment, and loop closure optimization data.	Log file in a temporary folder

IMU Fusion

`CameraToIMUTransform` — IMU sensor transformation
`rigidtform3d`

This property is read-only after object creation.

IMU sensor transformation, specified as a `rigidtform3d` object. The transformation describes the rotation and translation of the camera in the coordinate system of the IMU sensor.

`NumPosesThreshold` — Number of estimated camera poses to trigger camera-IMU alignment
`30` (default) | integer equal to or greater than 2

This property is read-only after object creation.

Number of estimated camera poses to trigger camera-IMU alignment, specified as an integer equal to or greater than 2. The process of aligning camera and IMU data is initiated after a specific number of camera poses have been estimated. This alignment serves two primary purposes: first, to estimate a scale factor that translates the up-to-scale results from a monocular camera into actual world units (meters), and second, to synchronize the IMU and camera frames, effectively eliminating the influence of gravity on accelerometer data. The timing for this alignment, determined by a threshold for the number of camera poses, is key to its success. A threshold set too low may not provide enough data for accurate calibration, while one set too high risks incorporating noise from measurement drift into the calibration. For a more in-depth understanding of this calibration technique, see the `estimateGravityRotationAndPoseScale` function.

`AlignmentFraction` — Subset of pose estimates
`0.8` (default) | scalar in the range (0,1]

This property is read-only after object creation.

Subset of pose estimates, specified as a scalar in the range of (0,1]. This value specifies a fraction of the number of recent pose estimates, calculated as `round(NumPosesThreshold*AlignmentFraction)`, for use in the camera-IMU alignment

process. It effectively filters out initial, potentially noisy pose estimates, ensuring only the most relevant data contributes to the alignment for improved accuracy.

Object Functions

addFrame	Add pair of color and depth images to stereo visual SLAM object
checkStatus	Check status of stereo visual SLAM object
hasNewKeyFrame	Check if new key frame added in stereo visual SLAM object
isDone	End-of-processing status for stereo visual SLAM object
mapPoints	Build 3-D map of world points from stereo vSLAM object
plot	Plot 3-D map points and estimated camera trajectory in stereo visual SLAM
poses	Absolute camera poses of stereo key frames
reset	Reset stereo visual SLAM object

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
```

```
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

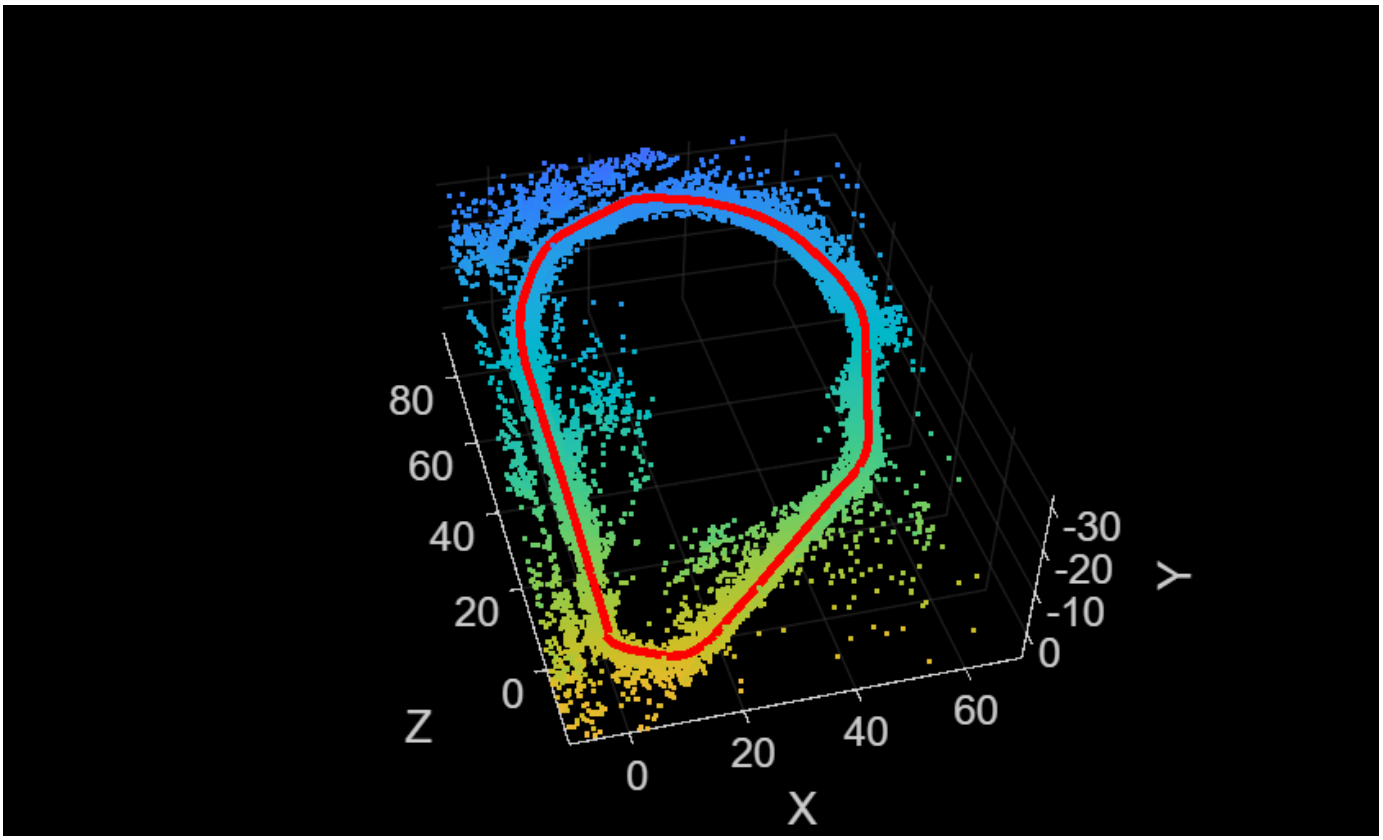
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Tips

- The `stereovslam` object:
  - Does not account for lens distortion. You can undistort the images by using the `undistortImage` function before adding images.
  - Assumes images have been rectified. You can rectify undistorted stereo images by using the `rectifyStereoImages` function before adding images.
  - Runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the most recently added frame.
- The camera poses are the poses of the primary camera, which corresponds to the input image `I1` added by the `addFrame` object function.
- The object represents 3-D map points and camera poses in world coordinates. The object assumes the camera pose of the first key frame is an identity `rigidtform3d` transform.

Version History

Introduced in R2024a

R2025a: `stereovslam` available in Navigation Toolbox with Computer Vision Toolbox license

You can use the `stereovslam` object in Navigation Toolbox if you have a Computer Vision Toolbox license.

R2024b: Set disparity map for stereo vSLAM images

Use the `DisparityMap` name-value argument of the `addFrame` function.

R2024b: Add ability to specify custom bag of words

Use the `CustomBagOfFeatures` name-value argument to specify a custom bag of words vocabulary as a `bagOfFeaturesDBoW` object.

R2024b: `Verbose` property enhanced to display progress information
Behavior changed in R2024b

The `Verbose` property added values of `1`, `2`, and `3` to display levels of progress information.

References

[1] Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

To compile the generated code for non-host target, these dependencies must be installed:

- OpenCV 4.7.0
- Ceres-solver 2.2.0 library — Download the source and build it as a shared library, on your remote device.
- eigen3 Library

When generating portable C code with a C++ compiler, you must specify hierarchical packing with non-minimal headers. For more information on packaging options, see the `packNGo` function.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`monovslam` | `rgbdvslam` | `factorIMUParameters` | `cameraIntrinsics` | `imageDatastore` | `ORBPoints`

Functions
`detectORBFeatures`

Topics
“Implement Visual SLAM in MATLAB”
“Performant and Deployable Stereo Visual SLAM with Fisheye Images”
“Monocular Visual Simultaneous Localization and Mapping”
“Monocular Visual-Inertial SLAM”

External Websites
What is SLAM?

addFrame

Add pair of color and depth images to stereo visual SLAM object

Syntax

```
addFrame(vslam,I1,I2)
addFrame( ___ ,gyro,accel)
addFrame( ___ ,DisparityMap=Value)
```

Description

`addFrame(vslam,I1,I2)` adds the pair of RGB or grayscale stereo images, `I1` and `I2`, to the stereo visual SLAM object `vslam`.

Note The `stereovslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`addFrame( ___ ,gyro,accel)` adds IMU gyroscope and acceleration measurements for visual-inertial fusion.

`addFrame( ___ ,DisparityMap=Value)` sets the disparity map for the stereo input images `I1` and `I2`. The disparity map must be a single or double matrix and the same width and height as the input images. If unspecified, the function defaults to calculating frame disparity using semi-global block matching.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);
```

```
    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

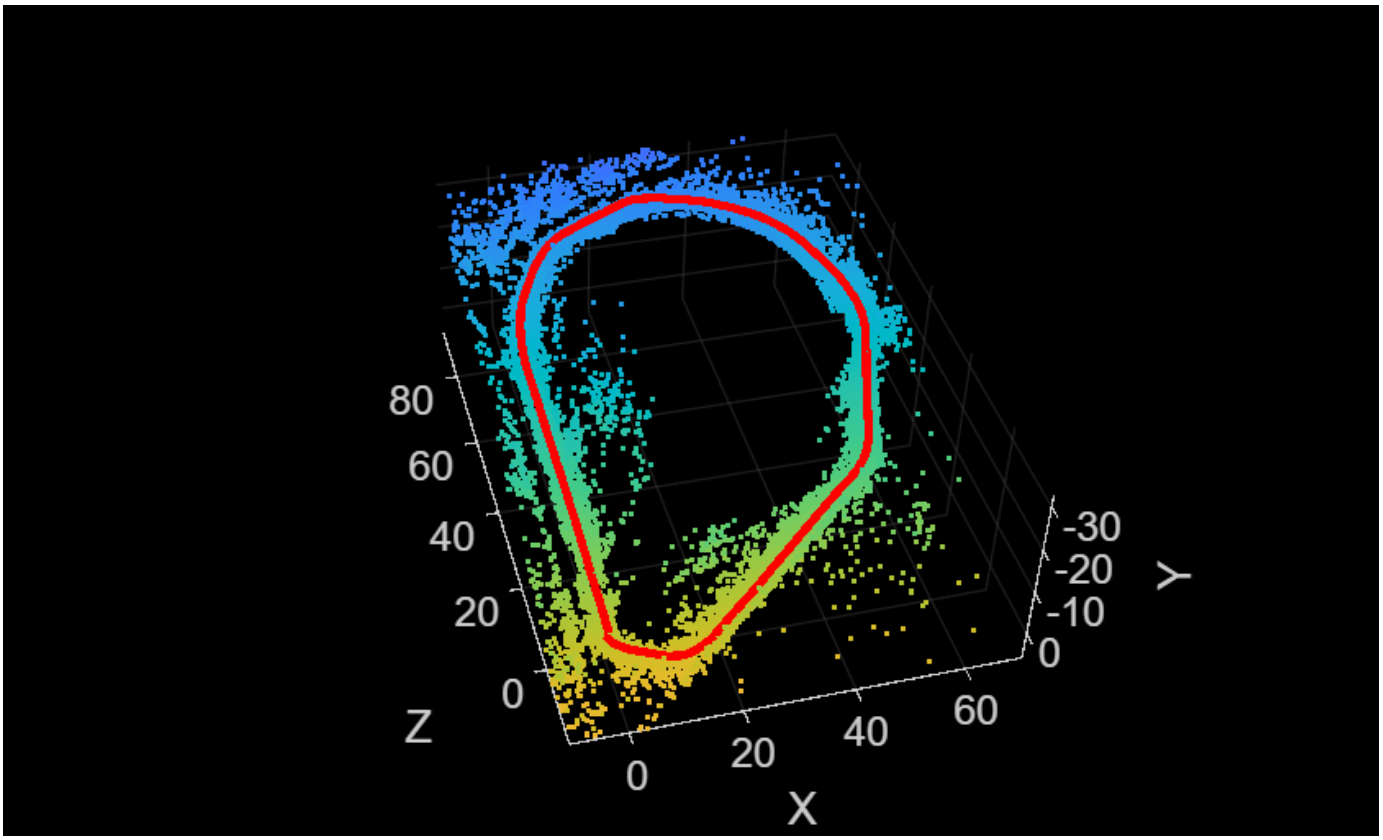
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

`I1` — Image 1

RGB image | grayscale image

Image 1, specified as a nonsparse RGB or grayscale image. The stereo visual slam object tracks the pose of camera 1, the left camera, which corresponds to image `I1` in the stereo camera system. The input images must be undistorted and rectified. You can undistort and then rectify the images by using the `undistortImage` and `rectifyStereoImages` functions, respectively, before adding the images.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`I2` — Image 2

RGB image | grayscale image

Image 2, specified as a nonsparse RGB or grayscale image. Image 2 corresponds to the right camera in the stereo camera system. The input images must be undistorted and rectified. You can undistort

and then rectify the images by using the `undistortImage` and `rectifyStereoImages` functions, respectively, before adding the images.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`gyro` — Gyroscope measurement
N-by-3 matrix

Gyroscope measurement, specified as an *N*-by-3 matrix. Each row of the matrix represents the *x*, *y*, and *z* components of the gyroscope measurement for a single frame, of the form [*gx*,*gy*,*gz*]. *N* represents the total number of frames. *N* represents the number of IMU measurements between the previous and current camera frames. The number of gyroscope and accelerometer measurements must be equal.

`accel` — Acceleration measurement
N-by-3 matrix

Acceleration measurement, specified as an *N*-by-3 matrix. Each row of the matrix represents the *x*, *y*, and *z* components of the acceleration measurement for a single frame, of the form [*ax*,*ay*,*az*]. *N* represents the number of IMU measurements between the previous and current camera frames. The number of gyroscope and accelerometer measurements must be equal.

Version History

Introduced in R2024a

R2024b: Set disparity map for stereo vSLAM images

Use the frame disparity by using the `DisparityMap` name-value argument.

See Also

Objects
`stereovslam`

Functions
`checkStatus` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `plot` | `poses` | `reset`

hasNewKeyFrame

Check if new key frame added in stereo visual SLAM object

Syntax

```
hasAdded = hasNewKeyFrame(vslam)
```

Description

`hasAdded = hasNewKeyFrame(vslam)` returns a logical value that indicates if a new key frame has been added within the stereo visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
```

```
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

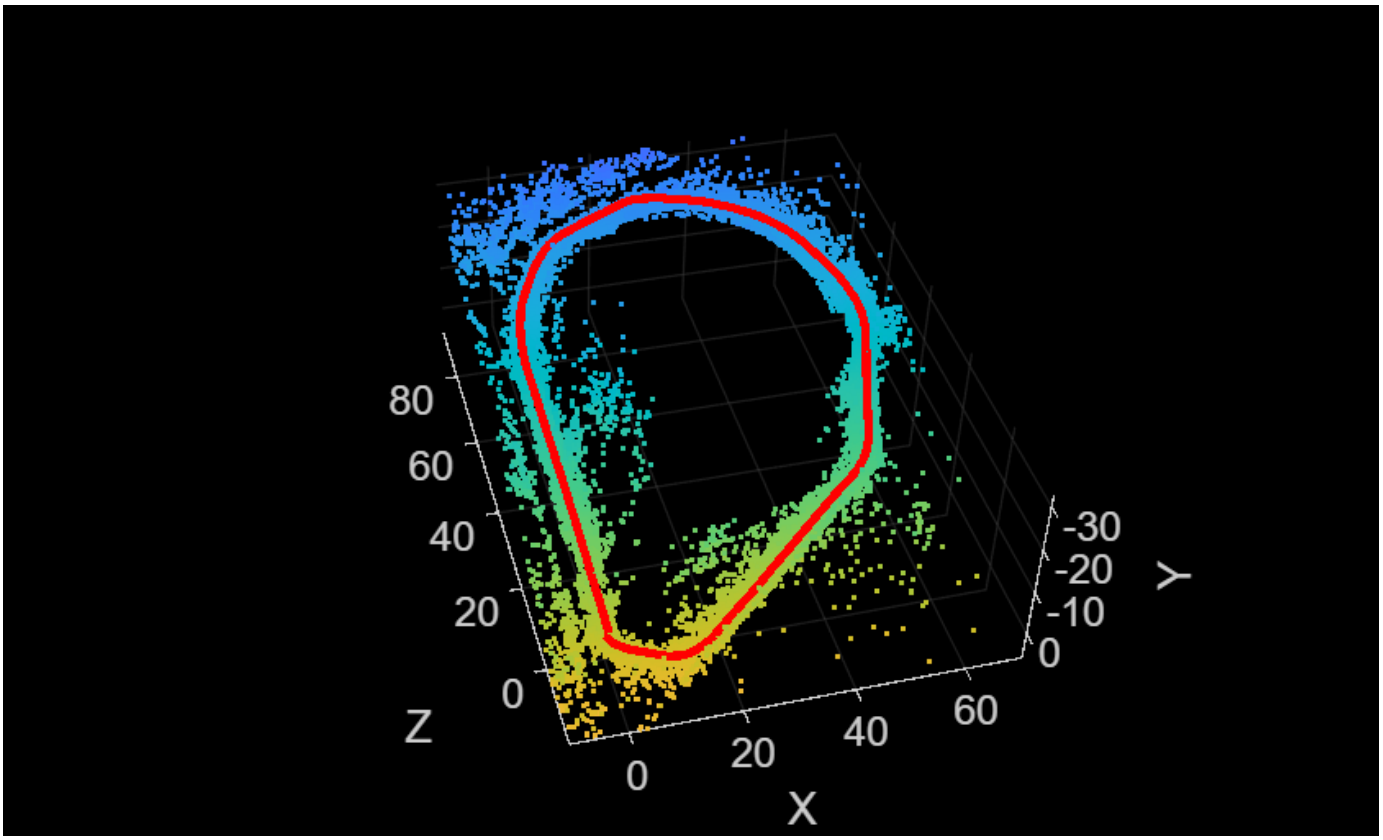
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object
`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Output Arguments

`hasAdded` — New key frame status
`false` or `0` | `true` or `1`

New key frame status, returned as a logical `0` (`false`) or `1` (`true`). A value of `true` indicates that a new key frame has been added within the stereo visual SLAM object `vslam`. A value of `false` indicates that no new key frame has been added. The object considers an image frame to be a key frame if the number of tracked feature points in the current frame satisfies one these conditions:

- The number of tracked feature points is within the limits specified by the `TrackFeatureRange` property of the `stereovslam` object `vslam`.
- The number of tracked feature points is greater than the upper limit specified by the `TrackFeatureRange` property of the `stereovslam` object `vslam`, and the number of skipped

frames since adding the last key frame equals the value of the `SkipMaxFrames` property of `vslam`.

The `hasNewKeyFrame` object function returns `true` only when the `stereovslam` object has added one or more new key frames since the last `hasNewKeyFrame` call. For example, if the `stereovslam` object adds only one key frame, then two consecutive calls to `hasNewKeyFrame` return `true` and `false`, respectively.

Version History

Introduced in R2024a

See Also

Objects
`stereovslam`

Functions
`addFrame` | `checkStatus` | `isDone` | `mapPoints` | `plot` | `poses` | `reset`

checkStatus

Check status of stereo visual SLAM object

Syntax

```
status = checkStatus(vslam)
```

Description

`status = checkStatus(vslam)` returns the current status of the stereo visual SLAM object. The frame the object is currently processing might be different than the most recently added frame.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
```

```
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

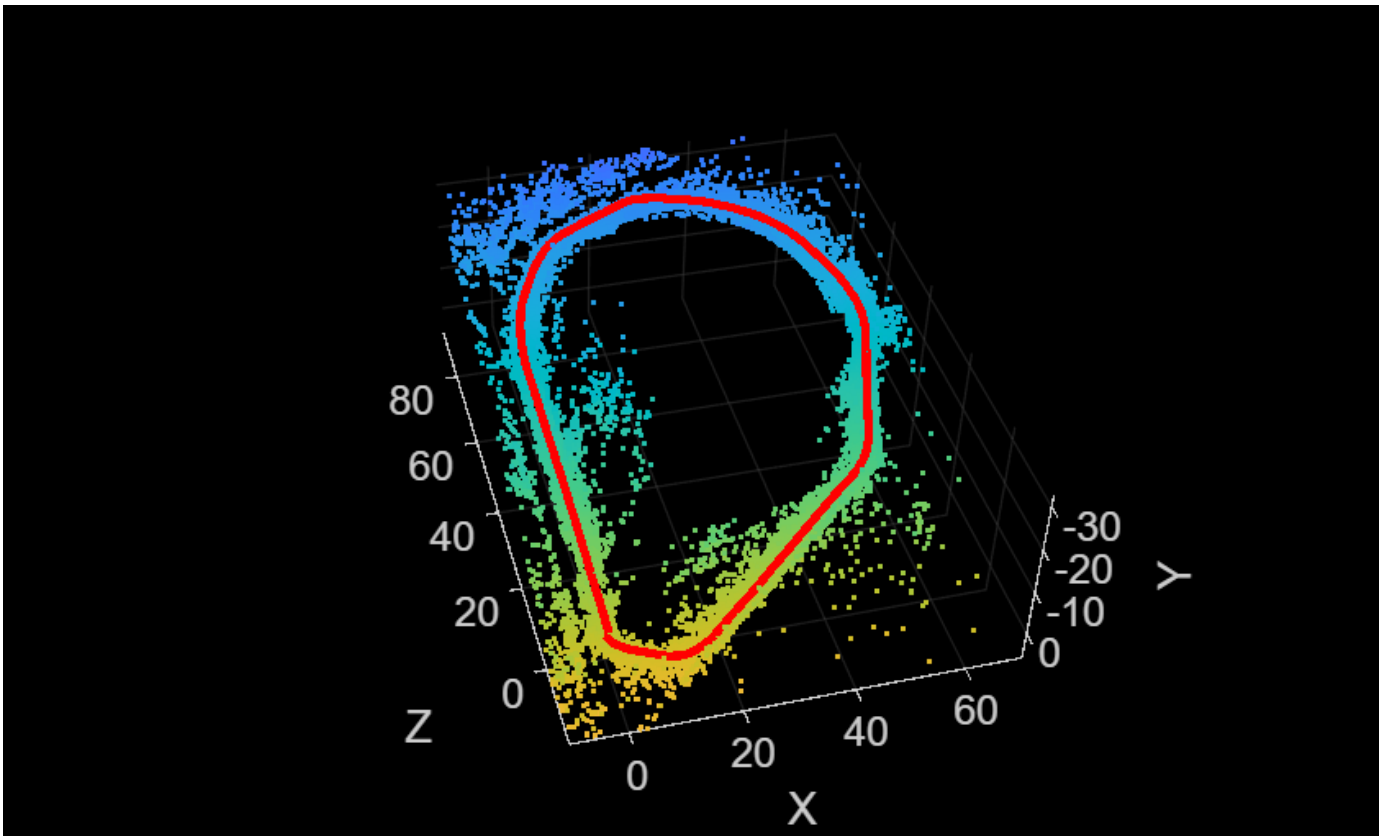
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Output Arguments

`status` — Current status of stereo visual SLAM object

`TrackingLost` enumeration | `TrackingSuccessful` enumeration | `FrequentKeyFrames` enumeration

Current status of the stereo visual SLAM object, returned as a `TrackingLost`, `TrackingSuccessful`, or `FrequentKeyFrames` enumeration. This table describes these enumerations.

Enumeration Value	Numeric Value	Description
`TrackingLost`	`uint8(0)`	Tracking is lost. The number of tracked feature points in the frame currently being processed is less than the lower limit of the `TrackFeatureRange` property of `vslam`. This indicates the image does not contain enough features, or that the camera is moving too fast. If the object does not accept enough frames as key frames, to improve tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` property and decrease the `SkipMaxFrames` property to add key frames more frequently.
`TrackingSuccessful`	`uint8(1)`	Tracking is successful. The number of tracked feature points in the frame currently being processed is between the lower limit and upper limit values of the `TrackFeatureRange` property of `vslam`.
`FrequentKeyFrames`	`uint8(2)`	Tracking adds key frames too frequently. The number of tracked feature points in the frame currently being processed is greater than the upper limit of the `TrackFeatureRange` property of `vslam`.

Version History

Introduced in R2024a

See Also

Objects
`stereovslam`

Functions
`addFrame` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `plot` | `poses` | `reset`

isDone

End-of-processing status for stereo visual SLAM object

Syntax

```
done = isDone(vslam)
```

Description

`done = isDone(vslam)` returns a logical value indicating whether the stereo visual SLAM object has finished processing its frames. A value of `true` indicates the object is in an idle state. A value of `false` indicates the object is processing one or more frames.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
```

```
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

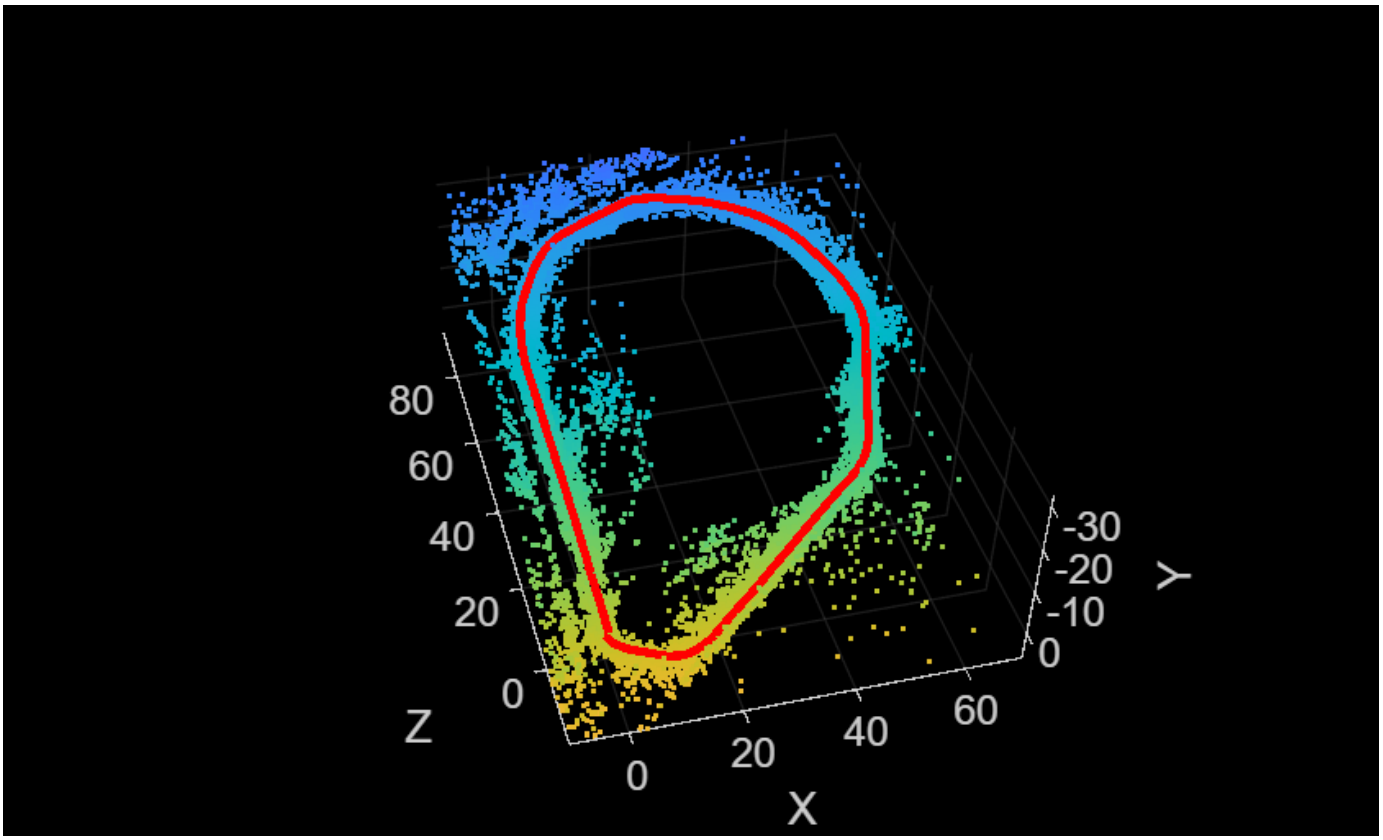
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Version History

Introduced in R2024a

See Also

Objects

`stereovslam`

Functions

`addFrame` | `checkStatus` | `hasNewKeyFrame` | `mapPoints` | `plot` | `poses` | `reset`

mapPoints

Build 3-D map of world points from stereo vSLAM object

Syntax

```
xyzPoints = mapPoints(vslam)
```

Description

`xyzPoints = mapPoints(vslam)` builds a sparse 3-D map of world points, `xyzPoints`, from the input images in the stereo visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
```

```
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

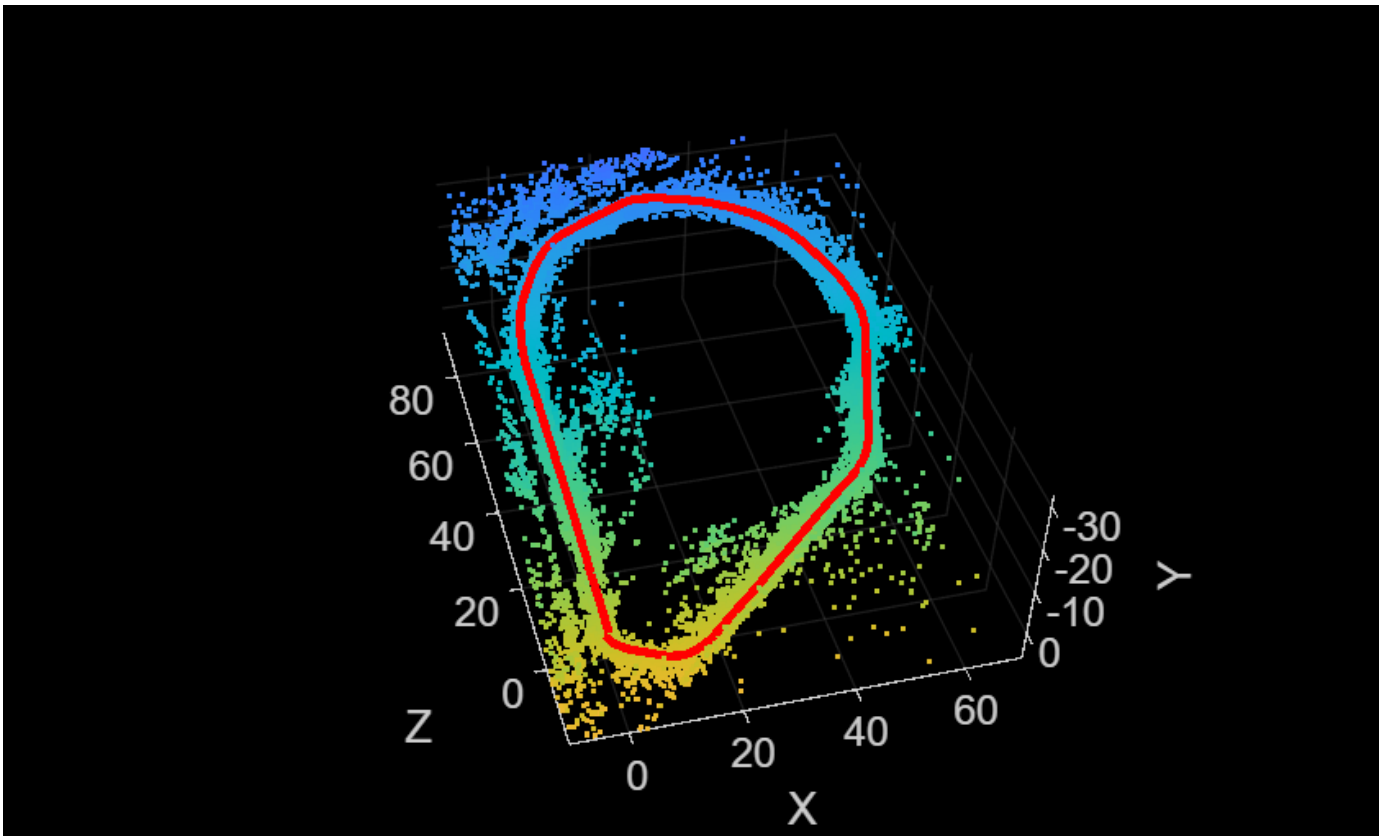
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Output Arguments

`xyzPoints` — World point coordinates of 3-D map

M-by-3 matrix

World point coordinates of 3-D map, returned as an *M*-by-3 matrix. *M* is the number of points, and each point is of the form [*X Y Z*].

Version History

Introduced in R2024a

See Also

Objects
`stereovslam`

Functions
`addFrame` | `checkStatus` | `hasNewKeyFrame` | `isDone` | `plot` | `poses` | `reset`

poses

Absolute camera poses of stereo key frames

Syntax

```
camPoses = poses(vslam)
[camPoses,keyFrameIDs] = poses(vslam)
```

Description

`camPoses = poses(vslam)` returns the absolute primary camera poses `camPoses` of the key frames from the visual simultaneous localization and mapping (vSLAM) object `vslam`.

`[camPoses,keyFrameIDs] = poses(vslam)` returns the IDs `keyFrameIDs` of the key frames corresponding to the primary camera poses. The IDs represent the order in which the image frames have been added to the `vslam` object by the `addFrame` object function.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

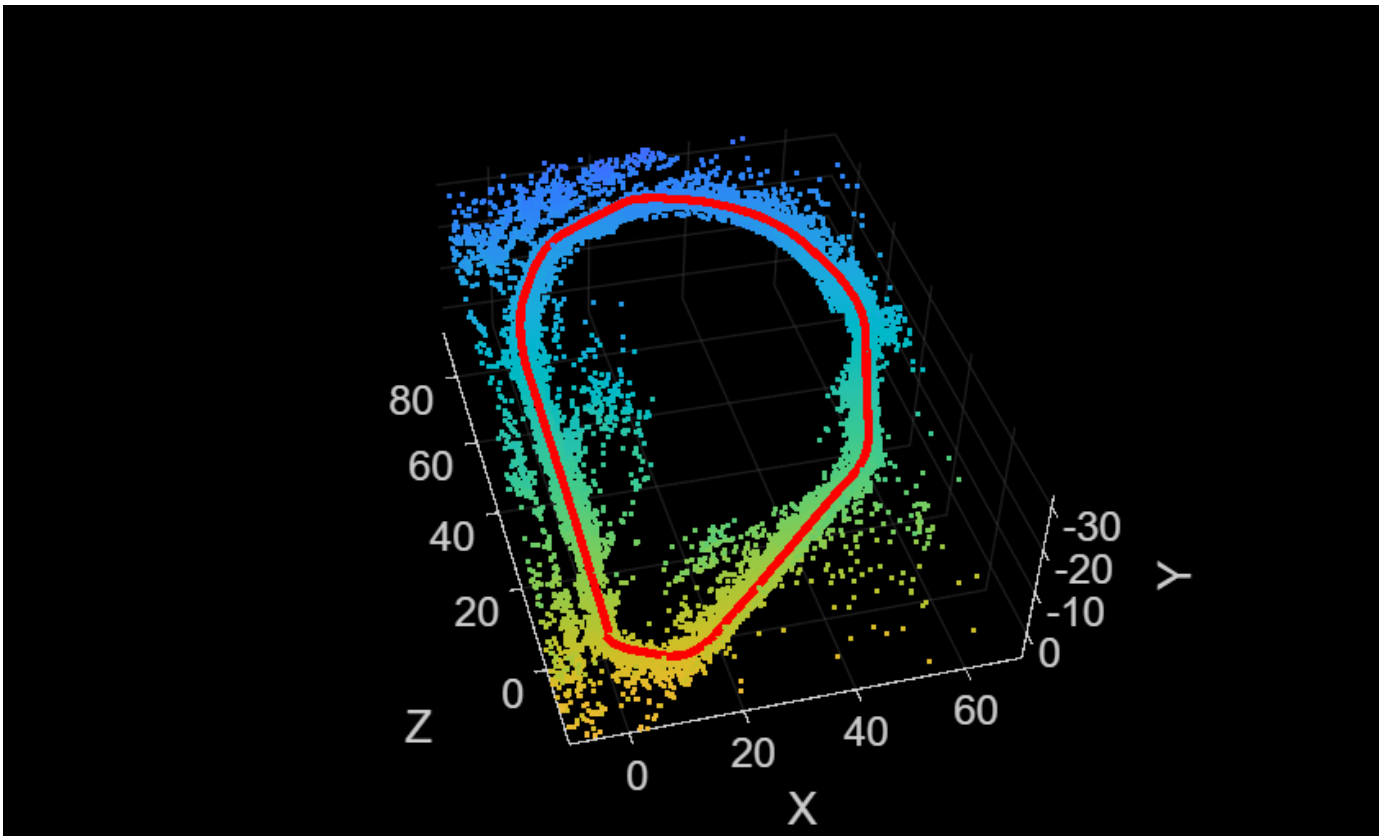
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object
`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Output Arguments

`camPoses` — Absolute primary camera poses of key frames
M-element array of `rigidtform3d` objects

Absolute primary camera poses of the key frames, returned as an *M*-element array of `rigidtform3d` objects.

`keyFrameIDs` — Key frame IDs for absolute primary camera poses
M-element array of integers

Key frame IDs for the absolute primary camera poses, returned as an *M*-element array of integers. Each element specifies the key frame ID for the camera pose in the corresponding element of `camPoses`. The IDs represent the order in which the image frames have been added to `vslam` by the `addFrame` object function.

Version History

Introduced in R2024a

See Also

Objects

`stereovslam`

Functions

`addFrame` | `checkStatus` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `plot` | `reset`

plot

Plot 3-D map points and estimated camera trajectory in stereo visual SLAM

Syntax

```
plot(vslam)
plot(vslam,Name=Value)
ax = plot( ___ )
```

Description

`plot(vslam)` plots the 3-D map points and estimated camera trajectory from the visual SLAM object `vslam`.

`plot(vslam,Name=Value)` specifies options using one or more name-value arguments. For example, `MarkerSize=10` sets the diameter of the marker size to 10 points.

`ax = plot( ___ )` returns the axes handle for the 3-D mapped points of the plot, using any combination of input arguments from previous syntaxes.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
```

```
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
imageSize = [384 512];
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

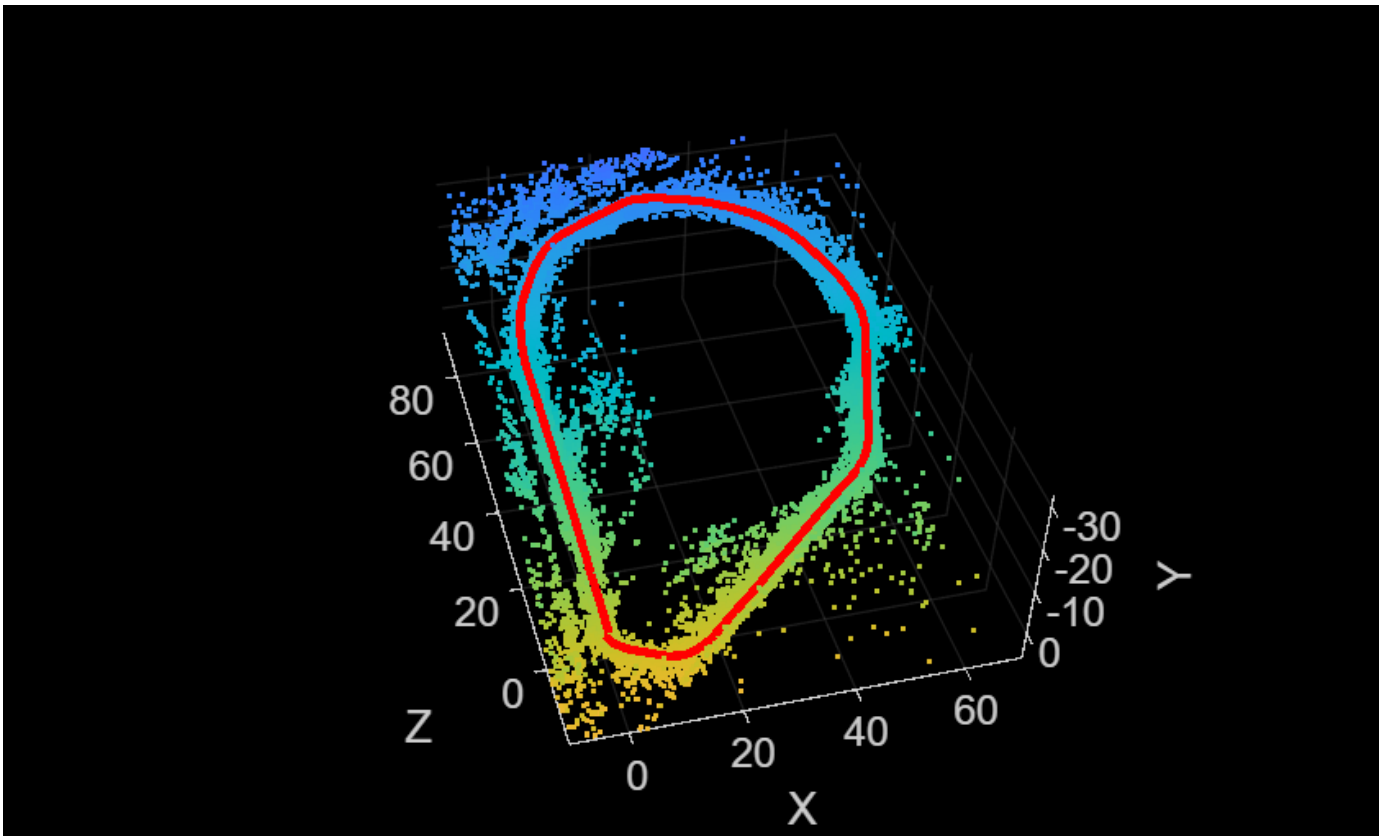
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(vslam,MarkerSize=10)`, sets the diameter of the marker to 10 points.

`MarkerSize` — Diameter of markers for 3-D map points

`6` (default) | positive integer

Diameter of markers for the 3-D map points, specified as a positive integer in points.

`MarkerColor` — Color of markers for 3-D map points

`[0 1 0]` (default) | RGB triplet | short or long color name

Color of the markers for the 3-D map points, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range [`0 1`].

`ColorSource` — Colormap source for the markers
`"Y"` (default) | `"X"` | `"Z"` | `"MarkerColor"`

Colormap source for markers, specified as `"X"`, `"Y"`, `"Z"`, or `"MarkerColor"`.

`CameraSize` — Width of camera base
`0.1` (default) | positive scalar

Width of the camera base, specified as a positive scalar in the data units of the axes.

`CameraColor` — Color of camera
`[1 0 0]` (default) | RGB triplet | short or long color name

Color of the camera, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range [`0 1`].

`Parent` — Axes for visualization
`Axes` object | `UIAxes` object

Axes for visualization, specified as an `Axes` graphics object or a `UIAxes` object. Use this name-value argument to visualize the plot in a UI for which you have specified `Figure` and `Axes` properties.

Output Arguments

`ax` — Axes handle
`Axes` graphics object

Axes handle, returned as an `axes` graphics object.

Version History

Introduced in R2024a

See Also

Objects
`stereovslam`

Functions
`addFrame` | `checkStatus` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `poses` | `reset`

reset

Reset stereo visual SLAM object

Syntax

`reset(vslam)`

Description

`reset(vslam)` resets all internal processes and clears all data in the `stereovslam` object.

Examples

Stereo Visual SLAM Using UTIAS Data Set

Perform stereo visual simultaneous localization and mapping (vSLAM) using the data from the UTIAS Long-Term Localization and Mapping Dataset provided by University of Toronto Institute for Aerospace Studies. You can download the data to a directory using a web browser, or by running this code:

```
ftpObj = ftp("asrl3.utias.utoronto.ca");
tempFolder = fullfile(tempdir);
dataFolder = [tempFolder,'2020-vtr-dataset\UTIAS-In-The-Dark\'];
zipFileName = [dataFolder,'run_000005.zip'];
folderExists = exist(dataFolder,"dir");
```

Create a folder in a temporary directory to save the downloaded file and extract its contents.

```
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading run_000005.zip (818 MB). This download can take a few minutes.")
    mget(ftpObj,"/2020-vtr-dataset/UTIAS-In-The-Dark/run_000005.zip",tempFolder);

    disp("Extracting run_000005.zip (818 MB) ...")
    unzip(zipFileName,dataFolder);
end
```

Create two `imageDatastore` objects to store the stereo images.

```
imgFolderLeft = [dataFolder,'\images\left\'];
imgFolderRight = [dataFolder,'\images\right\'];
imdsLeft = imageDatastore(imgFolderLeft);
imdsRight = imageDatastore(imgFolderRight);
```

Specify the intrinsic parameters and the baseline of the stereo camera, and use them to create a stereo visual SLAM object. The focal length, principal point, and image size is in pixels, and the baseline is in meters.

```
focalLength = [387.777 387.777];
principalPoint = [257.446 197.718];
imageSize = [384 512];
```

```
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
baseline = 0.239965;

vslam = stereovslam(intrinsics,baseline,MaxNumPoints=600, ...
    TrackFeatureRange=[30 120],SkipMaxFrames=5);
```

Process each pair of stereo images and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsLeft.Files)
    leftImage = readimage(imdsLeft,i);
    rightImage = readimage(imdsRight,i);
    addFrame(vslam,leftImage,rightImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

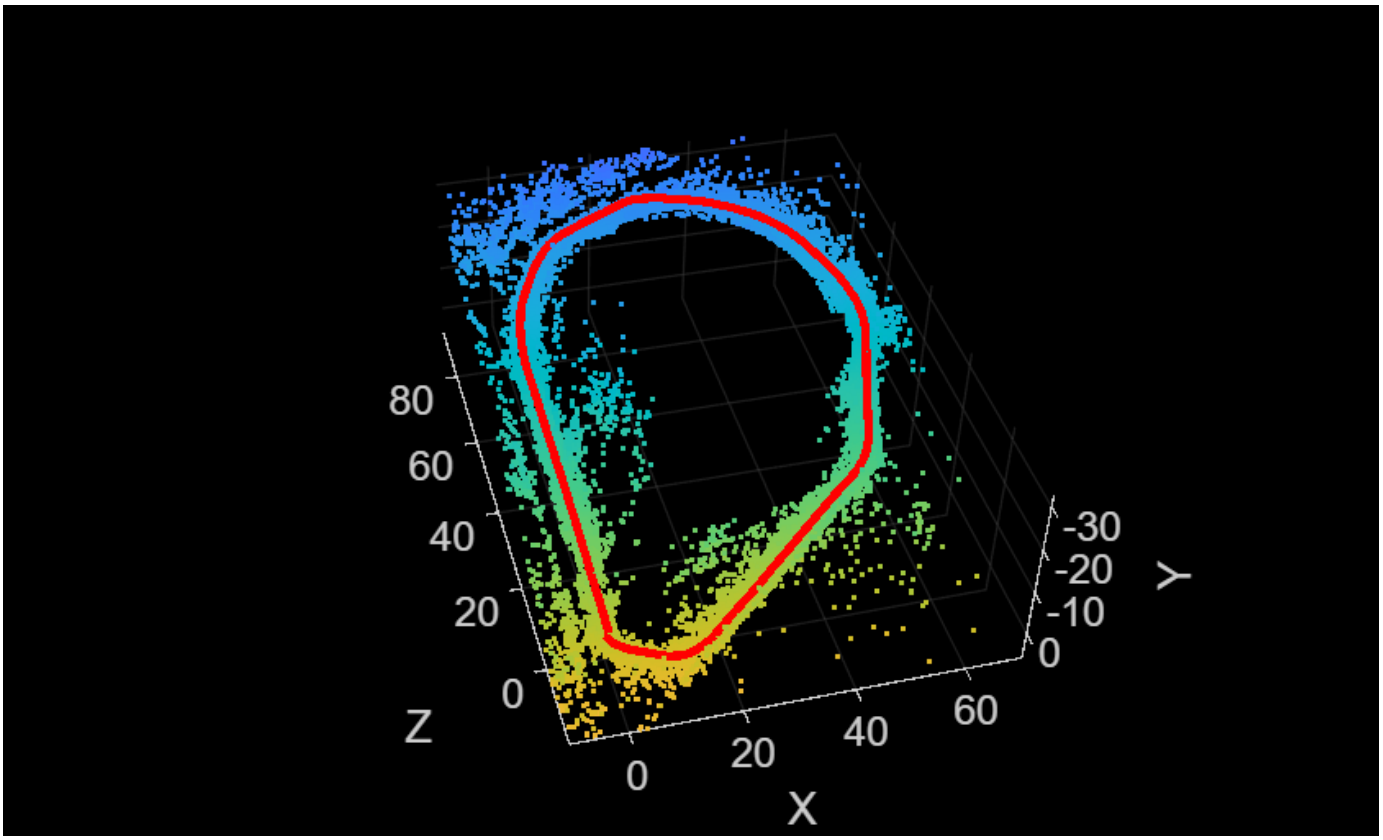
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam)
end
```




```
reset(vslam)
```

Input Arguments

`vslam` — Stereo visual SLAM object

`stereovslam` object

Stereo visual SLAM object, specified as a `stereovslam` object.

Version History

Introduced in R2024a

See Also

Objects

`stereovslam`

Functions

`addFrame` | `checkStatus` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `plot` | `poses`

rgbdvslam

Feature-based visual simultaneous localization and mapping (vSLAM) and visual-inertial sensor fusion with RGB-D camera

Description

Use the `rgbdvslam` object to perform visual simultaneous localization and mapping (vSLAM) with RGB-D camera data. RGB-D vSLAM combines depth information from sensors, such as RGB-D cameras or depth sensors, with RGB images to simultaneously estimate the camera pose and create a map of the environment. To learn more about visual SLAM, see “Implement Visual SLAM in MATLAB”.

The `rgbdvslam` object extracts Oriented FAST and Rotated BRIEF (ORB) features from incrementally read images, and then tracks those features to estimate camera poses, identify key frames, and reconstruct a 3-D environment. The vSLAM algorithm also searches for loop closures using the bag-of-features algorithm, and then optimizes the camera poses using pose graph optimization. You can enhance the accuracy and robustness of the SLAM by integrating this object with IMU data to perform visual-inertial sensor fusion.

Note The `rgbdvslam` object requires Navigation Toolbox.

Creation

Syntax

```
vslam = rgbdvslam(intrinsics)
vslam = rgbdvslam(intrinsics,depthScaleFactor)
vslam = rgbdvslam( ___ ,imuParameters)
vslam = rgbdvslam(intrinsics,PropertyName=Value)
```

Description

`vslam = rgbdvslam(intrinsics)` creates an RGB-D visual SLAM object, `vslam`, by using the specified camera intrinsic parameters.

The `rgbdvslam` object assumes the color and the depth images have been preregistered with one-to-one correspondence.

The object represents 3-D map points and camera poses in world coordinates, and assumes the camera pose of the first key frame is an identity `rigidtform3d` transform.

Note The `rgbdvslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`vslam = rgbdvslam(intrinsics,depthScaleFactor)` specifies the depth correction factor of the RGB-D camera, which the camera manufacturer usually provides. Use this syntax when the depth scale factor for the sensor is not equal to `1`.

`vslam = rgbdvslam( ___ ,imuParameters)` performs RGB-D visual-inertial SLAM based on the specified `imuParameters`.

`vslam = rgbdvslam(intrinsics,PropertyName=Value)` sets properties using one or more name-value arguments. For example, `MaxNumPoints=850` sets the maximum number of ORB feature points to extract from each image to `850`.

Input Arguments

`intrinsics` — Camera intrinsic parameters
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

This argument sets the `Intrinsics` property.

`depthScaleFactor` — Depth scale factor
scalar

Depth scale factor, specified as a scalar in real-world units, such as meters. The depth scale factor is the conversion factor that relates the depth values of the depth sensor to real-world distances, and is typically expressed in the same units as the depth measurements provided by the sensor, such as millimeters, centimeters, or meters. This value provides the necessary information to transform the depth measurements into the metric scale. Use the `depthScaleFactor` argument when the value for the sensor you are using is not equal to `1`.

For the world 3-D coordinates (X, Y, Z), where Z is the depth at any pixel coordinate (u, v), $Z = P/$ `depthScaleFactor`, where P represents the intensity value of the depth image at pixel (u, v).

This argument sets the `DepthScaleFactor` property.

`imuParameters` — IMU parameters
`factorIMUParameters` object

IMU parameters, specified as a `factorIMUParameters` object. The object contains the noise, bias, and sample rate information about the inertial measurement unit (IMU).

Properties

Camera Parameters

`Intrinsics` — Camera intrinsic parameters
`cameraIntrinsics` object

This property is read-only.

Camera intrinsic parameters, stored as a `cameraIntrinsics` object.

Use the `intrinsics` argument to set this property.

`DepthScaleFactor` — Depth scale factor

scalar

This property is read-only after object creation.

Depth scale factor, specified as a scalar in real-world units (e.g., meters). The depth scale factor depends on the depth sensor, and it is typically expressed in the same unit as the depth measurements provided by the sensor (e.g., millimeters, centimeters, or meters). It is the conversion factor that relates the depth values to real-world distances. In other words, it provides the necessary information to transform the depth measurements into metric scale.

For the world 3-D coordinates (X,Y,Z), where Z represents depth at any pixel coordinate (u,v), Z would be computed as $Z = P/$`depthScaleFactor`, where P represents the intensity value of the depth image at pixel (u,v).

Use the `depthScaleFactor` argument to set this property.

`DepthRange` — Depth range of RGB-D camera

`[0.5 5]` (default) | two-element vector

This property is read-only after object creation.

Depth range of the RGB-D camera, specified as a two-element vector, in world units. The range specifies the minimum and maximum depth values of the camera, which you can use to filter out invalid depth values in the depth images.

Feature Extraction

`ScaleFactor` — Scale factor for image decomposition

`1.2` (default) | scalar greater than 1

This property is read-only after object creation.

Scale factor for image decomposition, stored as a scalar greater than 1. The scale factor is also referred to as the pyramid decimation ratio. Increasing the value of `ScaleFactor` reduces the number of pyramid levels, but reduce computation time. Decreasing this value (down to just over 1), increases the number of pyramid levels, which can improve tracking performance, at the cost of computation speed. The scale value at each level of decomposition is $\texttt{ScaleFactor}^{(level\text{-}1)}$, where *level* is any value in the range [0, `Numlevels`-1]. Given the input image of size M-by-N, the image size at each level of composition is M_{level}-by-N_{level}, where:

$$M_{level} = \frac{M}{ScaleFactor^{(level\ -\ 1)}}$$

$$N_{level} = \frac{N}{ScaleFactor^{(level\ -\ 1)}}$$

`NumLevels` — Number of decomposition levels

`8` (default) | integer greater than or equal to `1`

This property is read-only after object creation.

Number of decomposition levels, specified as an integer greater than or equal to `1`. Increase this value to extract keypoints from the image at more levels of decomposition. Along with the `ScaleFactor` value, `NumLevels` controls the number of pyramid levels on which the object evaluates feature points.

The image size at each decomposition level limits the number of levels at which you can extract keypoints. The image size at a level of decomposition must be at least 63-by-63 for keypoint detection. The maximum level of decomposition is calculated as

$$level_{\max} = \text{floor}\left(\frac{\log(\min(M,N)) - \log(63)}{\log(\text{ScaleFactor})}\right) + 1$$

If either the default value or the specified value of `NumLevels` is greater than $level_{max}$, the object modifies `NumLevels` to $level_{max}$ and returns a warning.

`MaxNumPoints` — Maximum number of ORB feature points uniformly extracted from each image

`1000` | positive integer

This property is read-only after object creation.

Maximum number of ORB feature points uniformly extracted from each image, specified as a positive integer. Values are typically in the range of [800, 2000], depending on the resolution of the image. When the number of extracted features is less than the value of `MaxNumPoints`, then the object uses all feature points.

Tracking

`TrackFeatureRange` — Range for the number of feature points to identify key frame

`[30 100]` | two-element vector of positive integers

This property is read-only after object creation.

Key frame feature point range, stored as a two-element vector of positive integers in the form [*lowerLimit upperLimit*]. This property specifies the minimum and maximum numbers of tracked features points a frame must contain for the object to identify it as a key frame. The `TrackFeatureRange` and the `SkipMaxFrames` properties enable you to control the frequency at which frames in the tracking process become key frames.

The success of tracking depends on the number of tracked points in the current frame, with one of these results:

- Tracking is lost — The number of tracked feature points in the current frame is less than the *lowerLimit* set by the `TrackFeatureRange` argument. This indicates that the image does not contain enough features, or that the camera is moving too fast. To improve the tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` property and decrease the `SkipMaxFrames` property to add key frames more frequently.
- Tracking is successful — The object identifies the current frame as a key frame. The number of tracked feature points in the current frame is in the range set by `TrackFeatureRange`.
- Tracking adds key frames too frequently — The number of tracked feature points in the current frame is greater than the *upperLimit* set by the `TrackFeatureRange` property. This indicates that the camera is moving very slowly, which produces an unnecessary number of key frames. To improve the tracking, you can reduce the frequency of adding key frames by increasing the value of the `SkipMaxFrames` property.

For more details, see the `addFrame` object function and “How to Improve Accuracy in Visual SLAM”.

`SkipMaxFrames` — Maximum number of skipped frames

`20` | positive integer

This property is read-only after object creation.

Maximum number of skipped frames, stored as a positive integer. When the number of tracked features is consistently greater than the *upperLimit* set by the `TrackFeatureRange` property, use the `SkipMaxFrames` property to control the frequency at which the object adds new key frames. The object identifies the current frame as a key frame when the number of skipped frames since the most recently added key frame equals the value of `SkipMaxFrames`. For more details, see “How to Improve Accuracy in Visual SLAM”.

`LoopClosureThreshold` — Minimum number of matched feature points between loop closure key frames
`60` | positive integer

This property is read-only after object creation.

Minimum number of matched feature points between loop closure key frames, stored as a positive integer.

`CustomBagOfFeatures` — Custom bag of features for loop detection
`[]` (default) | `bagOfFeaturesDBoW` object

This property is read-only after object creation.

Custom bag of features for loop detection, specified as a `bagOfFeaturesDBoW` object. The `bagOfFeaturesDBoW` enables you to create a custom bag of words (BoW) from feature descriptors, alongside options to utilize a built-in vocabulary or load a custom one from a specified file.

`Verbose` — Progress information display
`[]` (default) | `1` | `2` | `3`

This property is read-only after object creation.

Progress information display, specified as `[]`, `1`, `2`, or `3`. Paths to the location of log files, when they are created, is displayed on the command line. For more details, see “How to Improve Accuracy in Visual SLAM”.

Verbose value	Display description	Display location
`[]` or `false`	Display is turned off	—
`1` or `true`	Stages of vSLAM execution	Command window
`2`	Stages of vSLAM execution, with details on how the frame is processed, such as the artifacts used to initialize the map.	Log file in a temporary folder
`3`	Stages of vSLAM, artifacts used to initialize the map, poses and map points before and after bundle adjustment, and loop closure optimization data.	Log file in a temporary folder

IMU Fusion

`CameraToIMUTransform` — IMU sensor transformation
`rigidtform3d`

This property is read-only after object creation.

IMU sensor transformation, specified as a `rigidtform3d` object. The transformation describes the rotation and translation of the camera in the coordinate system of the IMU sensor.

`NumPosesThreshold` — Number of estimated camera poses to trigger camera-IMU alignment
`30` (default) | integer equal to or greater than 2

This property is read-only after object creation.

Number of estimated camera poses to trigger camera-IMU alignment, specified as an integer equal to or greater than 2. The process of aligning camera and IMU data is initiated after a specific number of camera poses have been estimated. This alignment serves two primary purposes: first, to estimate a scale factor that translates the up-to-scale results from a monocular camera into actual world units (meters), and second, to synchronize the IMU and camera frames, effectively eliminating the influence of gravity on accelerometer data. The timing for this alignment, determined by a threshold for the number of camera poses, is key to its success. A threshold set too low may not provide enough data for accurate calibration, while one set too high risks incorporating noise from measurement drift into the calibration. For a more in-depth understanding of this calibration technique, see the `estimateGravityRotationAndPoseScale` function.

`AlignmentFraction` — Subset of pose estimates
`0.8` (default) | scalar in the range (0,1]

This property is read-only after object creation.

Subset of pose estimates, specified as a scalar in the range of (0,1]. This value specifies a fraction of the number of recent pose estimates, calculated as `round(NumPosesThreshold*AlignmentFraction)`, for use in the camera-IMU alignment process. It effectively filters out initial, potentially noisy pose estimates, ensuring only the most relevant data contributes to the alignment for improved accuracy.

Object Functions

addFrame	Add pair of color and depth images to RGB-D visual SLAM object
hasNewKeyFrame	Check if new key frame added in RGB-D visual SLAM object
checkStatus	Check status of visual RGB-D SLAM object
isDone	End-of-processing status for RGB-D visual SLAM object
mapPoints	Build 3-D map of world points from RGB-D vSLAM object
poses	Absolute camera poses of RGB-D vSLAM key frames
plot	Plot 3-D map points and estimated camera trajectory in RGB-D visual SLAM
reset	Reset RGB-D visual SLAM object

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
```

```
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

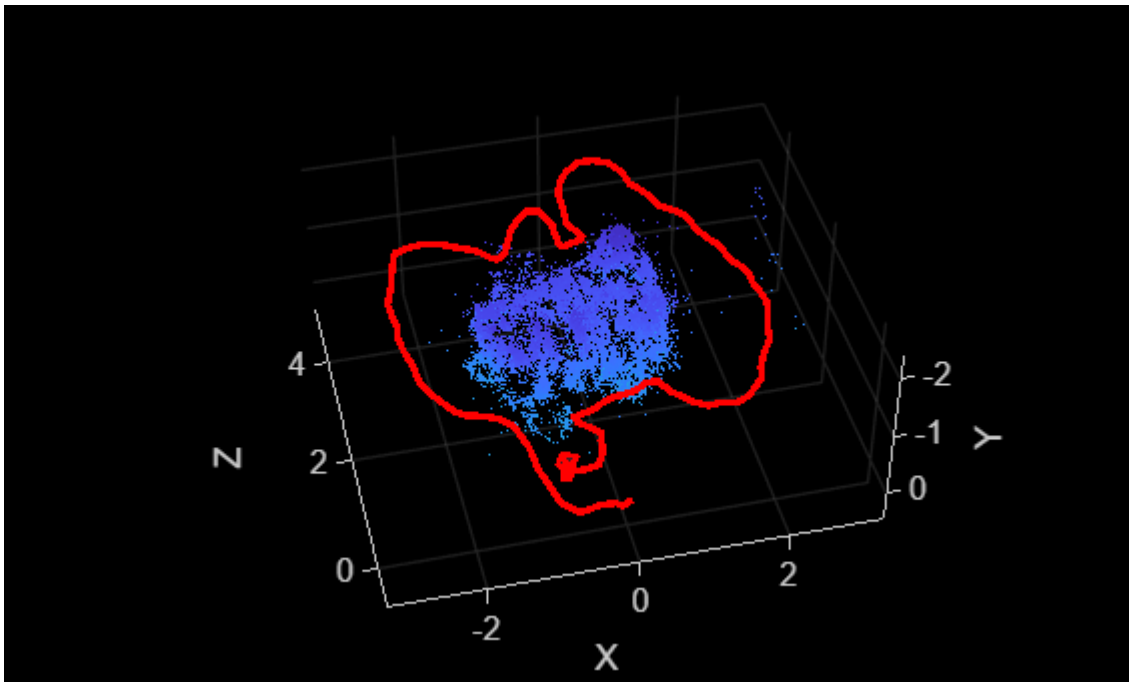
    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);

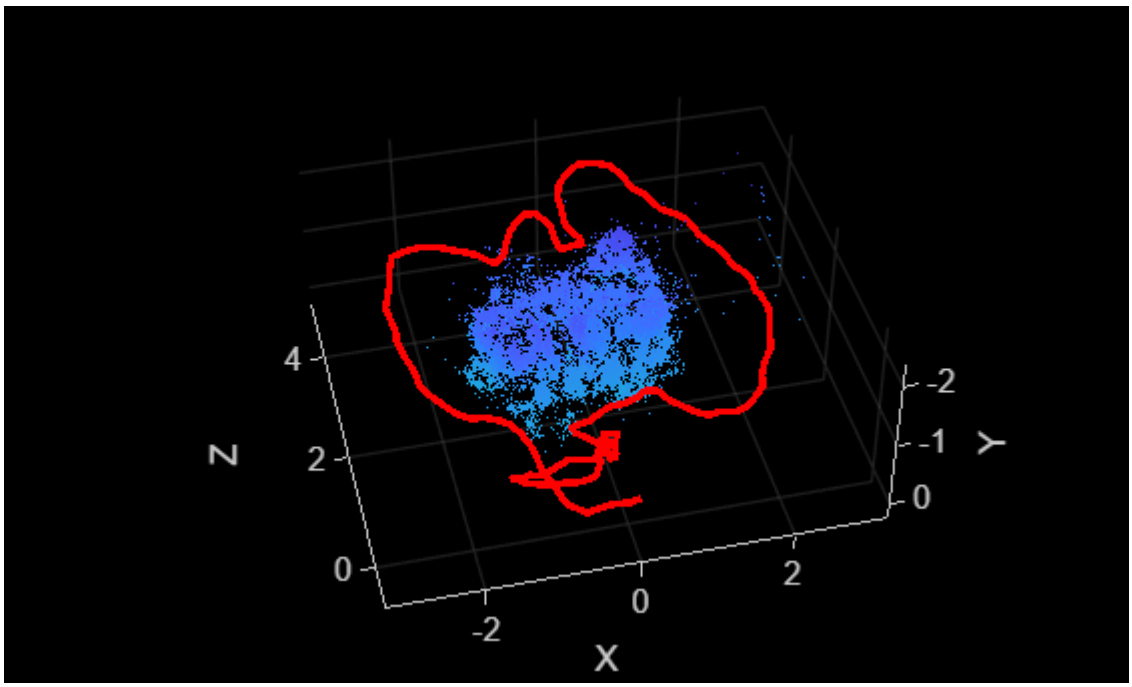
    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
```

```
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```



```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

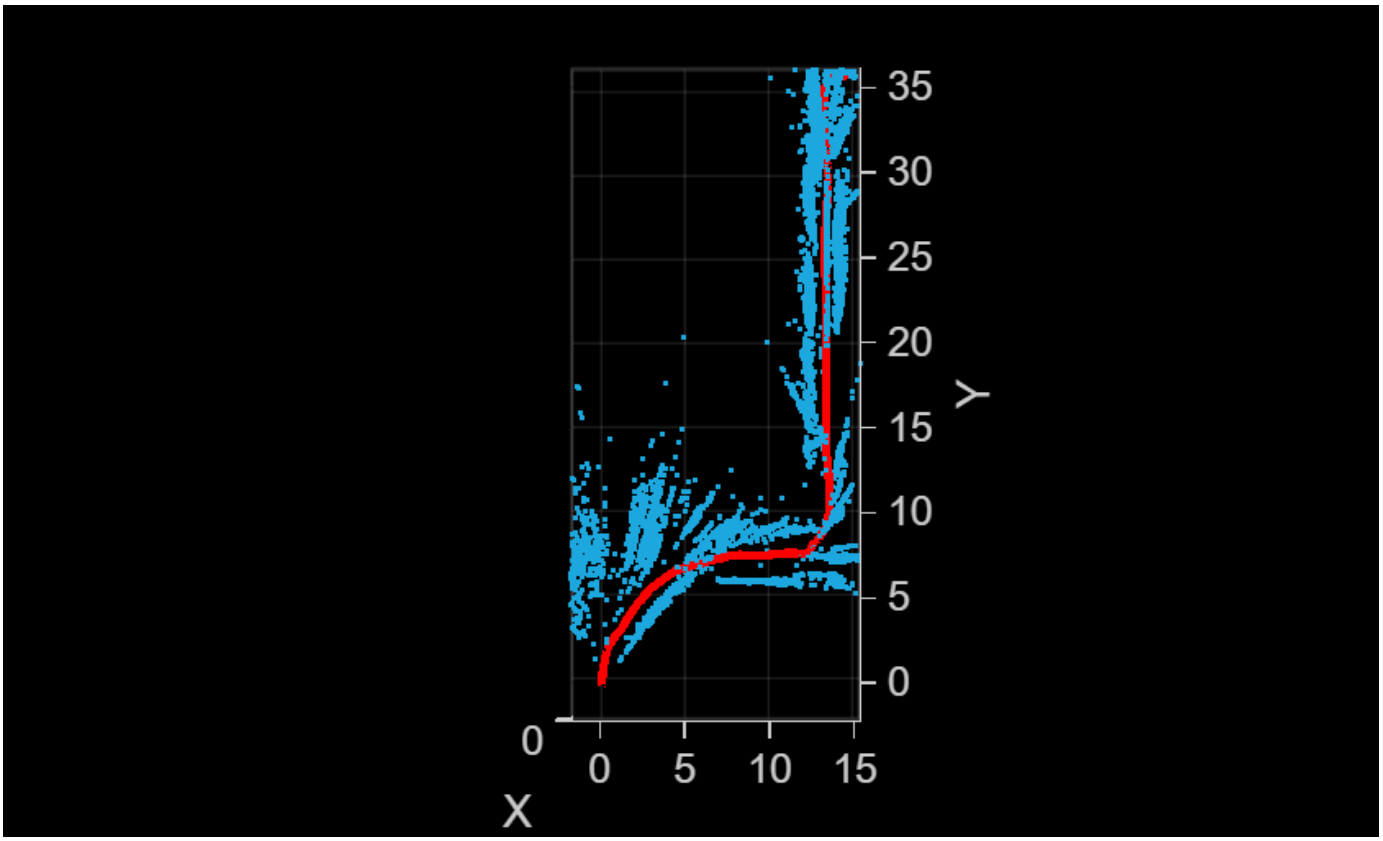
Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Version History

Introduced in R2024a

R2025a: `rgbdvslam` available in Navigation Toolbox with Computer Vision Toolbox license

You can use the `rgbdvslam` object in Navigation Toolbox if you have a Computer Vision Toolbox license.

R2024b: Add ability to specify custom bag of words

Use the `CustomBagOfFeatures` name-value argument to specify a custom bag of words vocabulary as a `bagOfFeaturesDBoW` object.

R2024b: `Verbose` property enhanced to display progress information
Behavior changed in R2024b

The `Verbose` property added values of `1`, `2`, and `3` to display levels of progress information.

References

[1] Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System." *IEEE Transactions on Robotics* 31, no. 5 (October 2015): 1147–63. https://doi.org/10.1109/TRO.2015.2463671.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

To compile the generated code for non-host target, these dependencies must be installed:

- OpenCV 4.7.0
- Ceres-solver 2.2.0 library — Download the source and build it as a shared library, on your remote device.
- eigen3 Library

When generating portable C code with a C++ compiler, you must specify hierarchical packing with non-minimal headers. For more information on packaging options, see the `packNGo` function.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`monovslam` | `factorIMUParameters` | `cameraIntrinsics` | `imageDatastore` | `ORBPoints`

Functions
`detectORBFeatures`

Topics
"Implement Visual SLAM in MATLAB"
"Visual SLAM with RGB-D Camera"
"Simulate RGB-D Visual SLAM System with Cosimulation in Gazebo and Simulink" (ROS Toolbox)
"Monocular Visual Simultaneous Localization and Mapping"
"Monocular Visual-Inertial SLAM"

External Websites
What is SLAM?

addFrame

Add pair of color and depth images to RGB-D visual SLAM object

Syntax

```
addFrame(vslam,colorImage,depthImage)
addFrame( ___ ,gyro,accel)
```

Description

`addFrame(vslam,colorImage,depthImage)` adds the pair of color and depth images, `colorImage` and `depthImage` respectively, to the RGB-D visual SLAM object `vslam`. The color and depth images must be preregistered with a one-to-one correspondence.

Note The `rgbdvslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`addFrame( ___ ,gyro,accel)` adds IMU gyroscope and acceleration measurements for visual-inertial fusion.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

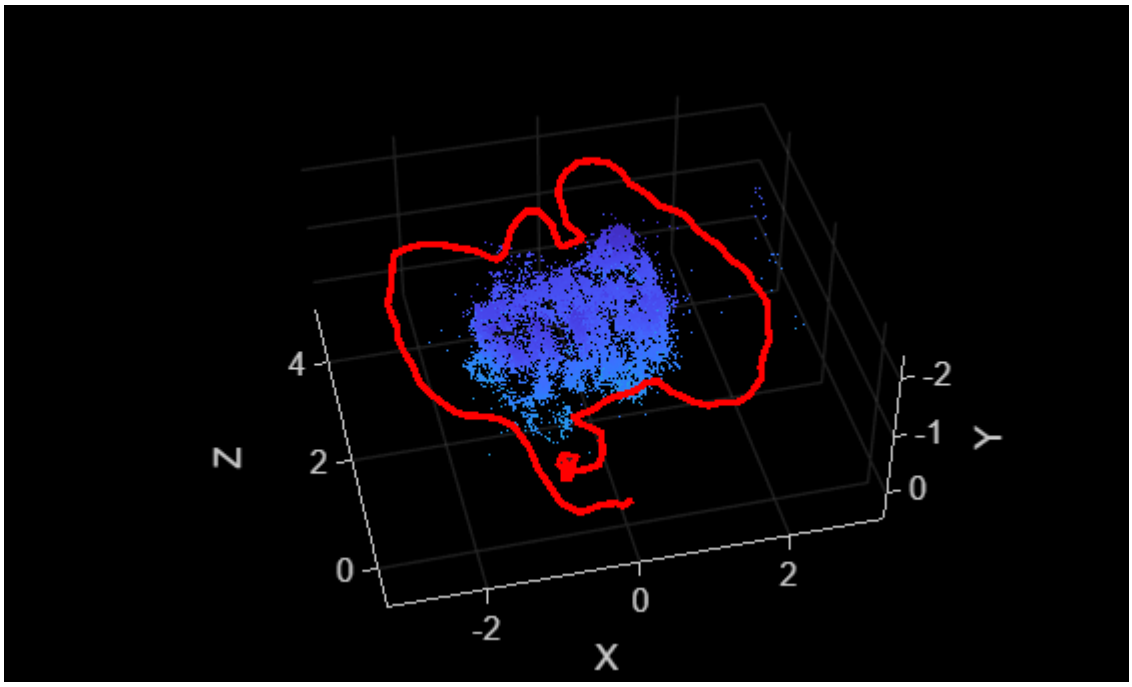
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

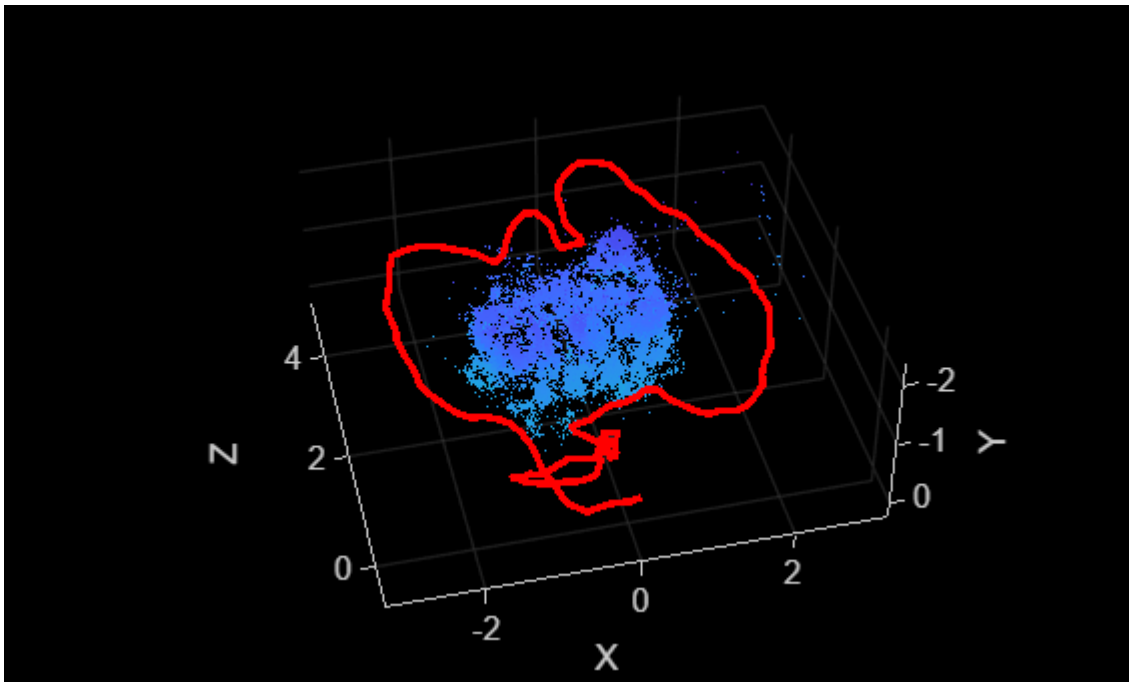
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

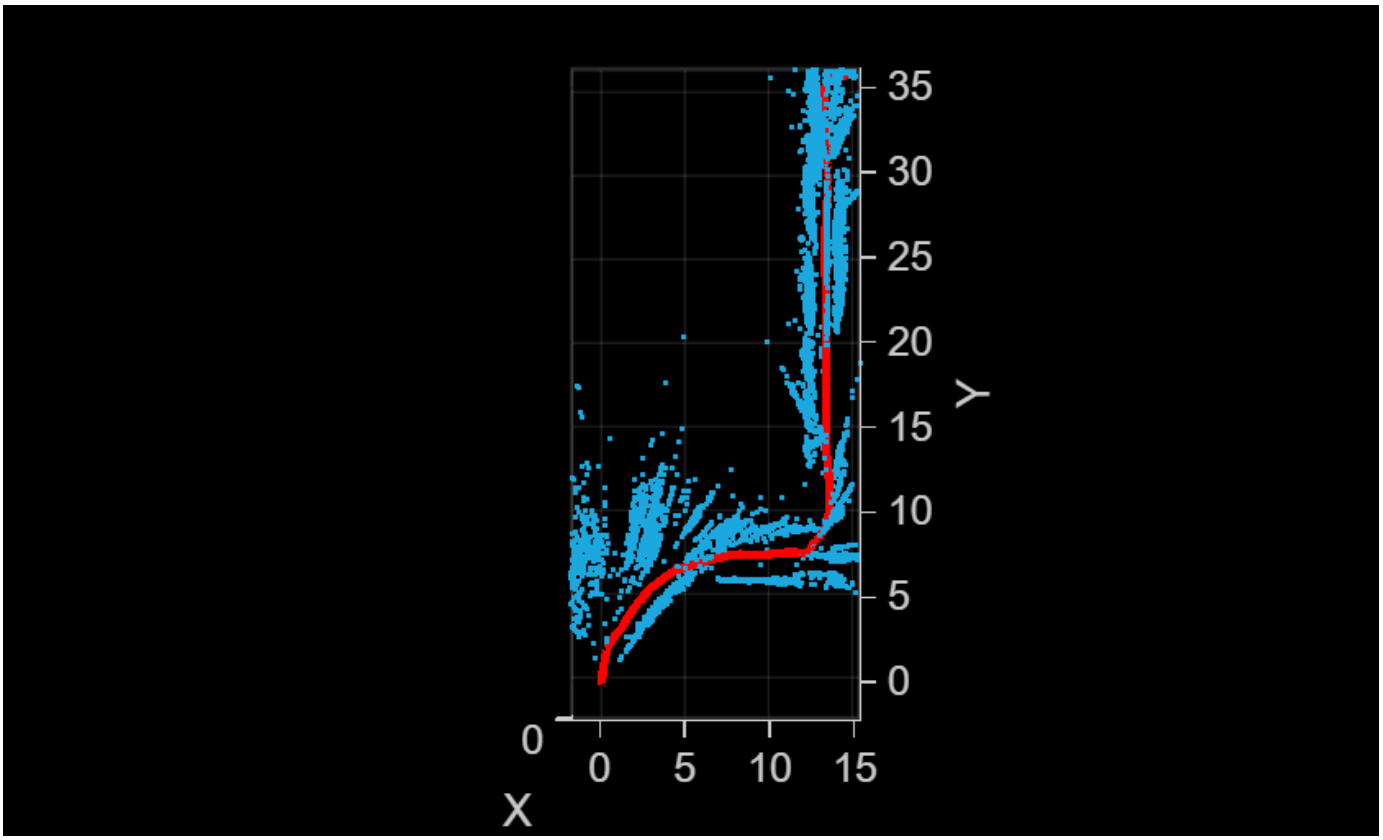
```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
```

```
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

`colorImage` — Color image
RGB image | grayscale image

Color image, specified as a nonsparse RGB or grayscale image. The color and depth images must be preregistered with a one-to-one correspondence.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`depthImage` — Depth image
grayscale image

Color image, specified as a nonsparse grayscale image. The color and depth images must be preregistered with a one-to-one correspondence.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`gyro` — Gyroscope measurement
N-by-3 matrix

Gyroscope measurement, specified as an *N*-by-3 matrix. Each row of the matrix represents the *x*, *y*, and *z* components of the gyroscope measurement for a single frame, of the form [*gx*,*gy*,*gz*]. *N* represents the total number of frames. *N* represents the number of IMU measurements between the previous and current camera frames. The number of gyroscope and accelerometer measurements must be equal.

`accel` — Acceleration measurement
N-by-3 matrix

Acceleration measurement, specified as an *N*-by-3 matrix. Each row of the matrix represents the *x*, *y*, and *z* components of the acceleration measurement for a single frame, of the form [*ax*,*ay*,*az*]. *N* represents the number of IMU measurements between the previous and current camera frames. The number of gyroscope and accelerometer measurements must be equal.

Version History
Introduced in R2024a

See Also

Objects
`rgbdvslam`

Functions
`hasNewKeyFrame` | `checkStatus` | `isDone` | `mapPoints` | `poses` | `plot` | `reset`

Topics
“Simulate RGB-D Visual SLAM System with Cosimulation in Gazebo and Simulink” (ROS Toolbox)

hasNewKeyFrame

Check if new key frame added in RGB-D visual SLAM object

Syntax

```
hasAdded = hasNewKeyFrame(vslam)
```

Description

`hasAdded = hasNewKeyFrame(vslam)` returns a logical value that indicates if a new key frame has been added within the RGB-D visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

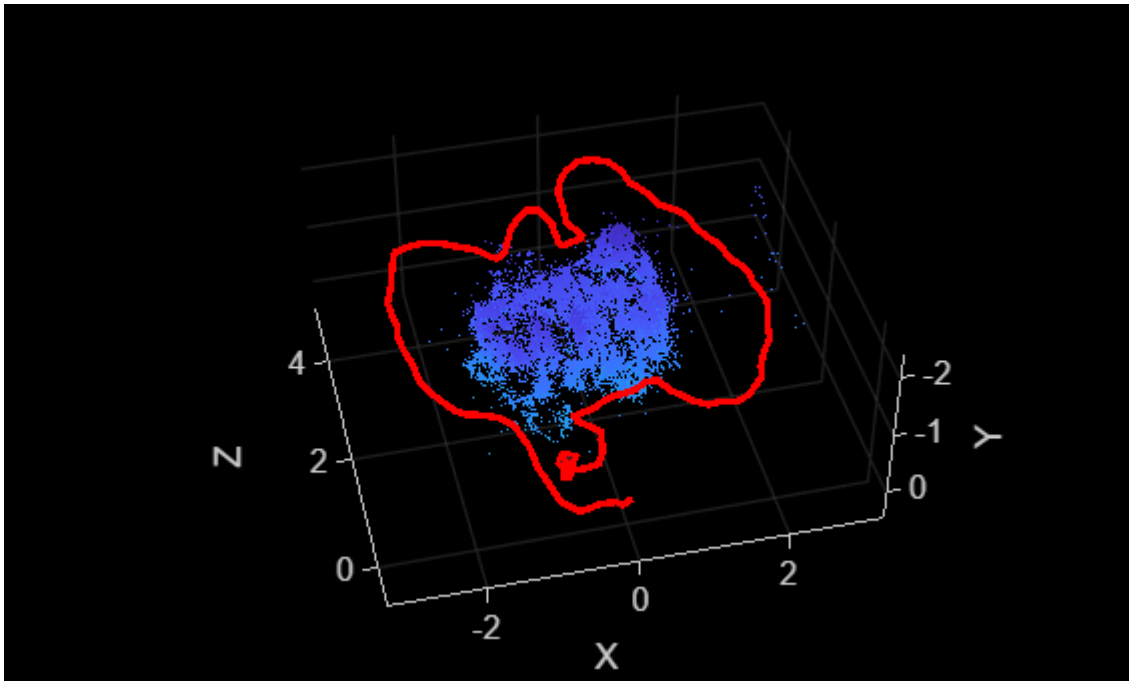
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

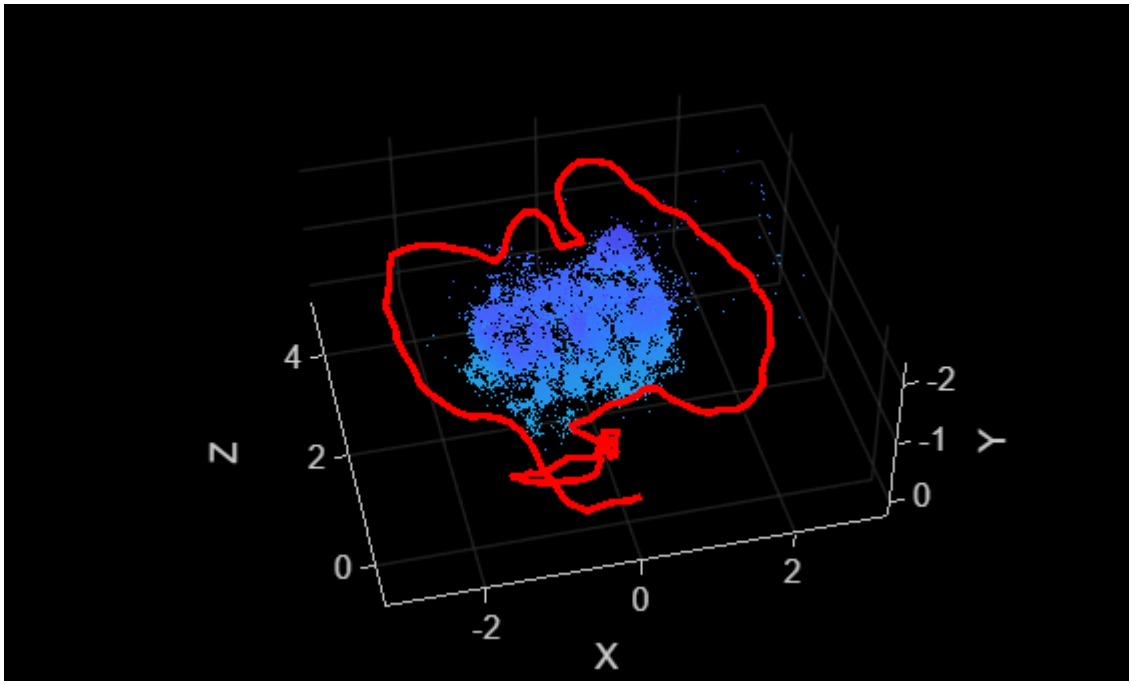
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

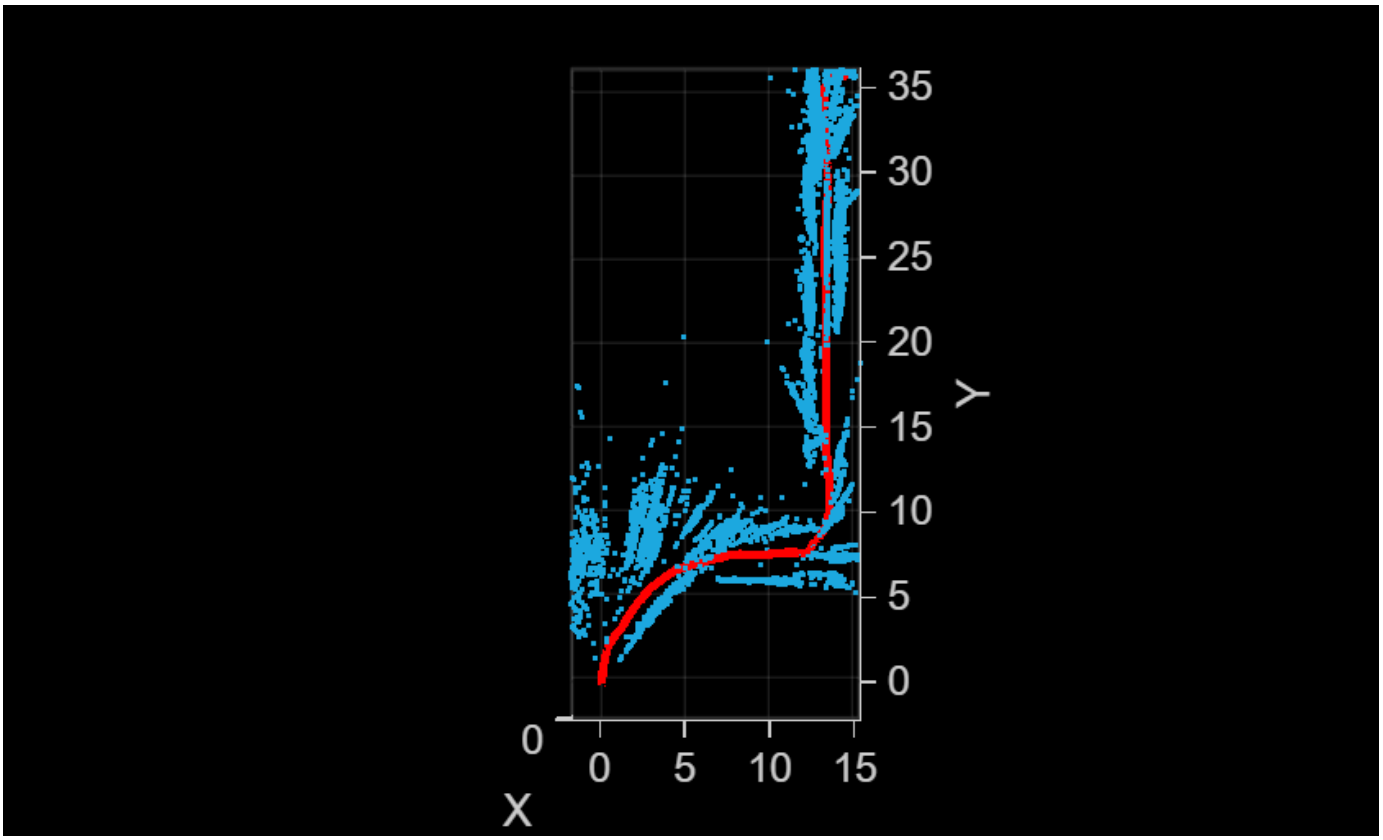
Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Output Arguments

`hasAdded` — New key frame status

`false` or `0` | `true` or `1`

New key frame status, returned as a logical `0` (`false`) or `1` (`true`). A value of `true` indicates that a new key frame has been added within the RGB-D visual SLAM object `vslam`. A value of `false` indicates that no new key frame has been added. The object considers an image frame to be a key frame if the number of tracked feature points in the current frame satisfies one these conditions:

- The number of tracked feature points is within the limits specified by the `TrackFeatureRange` property of the `rgbdvslam` object `vslam`.
- The number of tracked feature points is greater than the upper limit specified by the `TrackFeatureRange` property of the `rgbdvslam` object `vslam`, and the number of skipped

frames since adding the last key frame equals the value of the `SkipMaxFrames` property of `vslam`.

The `hasNewKeyFrame` object function returns `true` only when the `rgbdvslam` object has added one or more new key frames since the last `hasNewKeyFrame` call. For example, if the `rgbdvslam` object adds only one key frame, then two consecutive calls to `hasNewKeyFrame` return `true` and `false`, respectively.

Version History

Introduced in R2024a

See Also

Objects
`rgbdvslam`

Functions
`addFrame` | `checkStatus` | `isDone` | `mapPoints` | `poses` | `plot` | `reset`

checkStatus

Check status of visual RGB-D SLAM object

Syntax

```
status = checkStatus(vslam)
```

Description

`status = checkStatus(vslam)` returns the current status of the RGB-D visual SLAM object. The frame the object is currently processing might be different than the most recently added frame.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

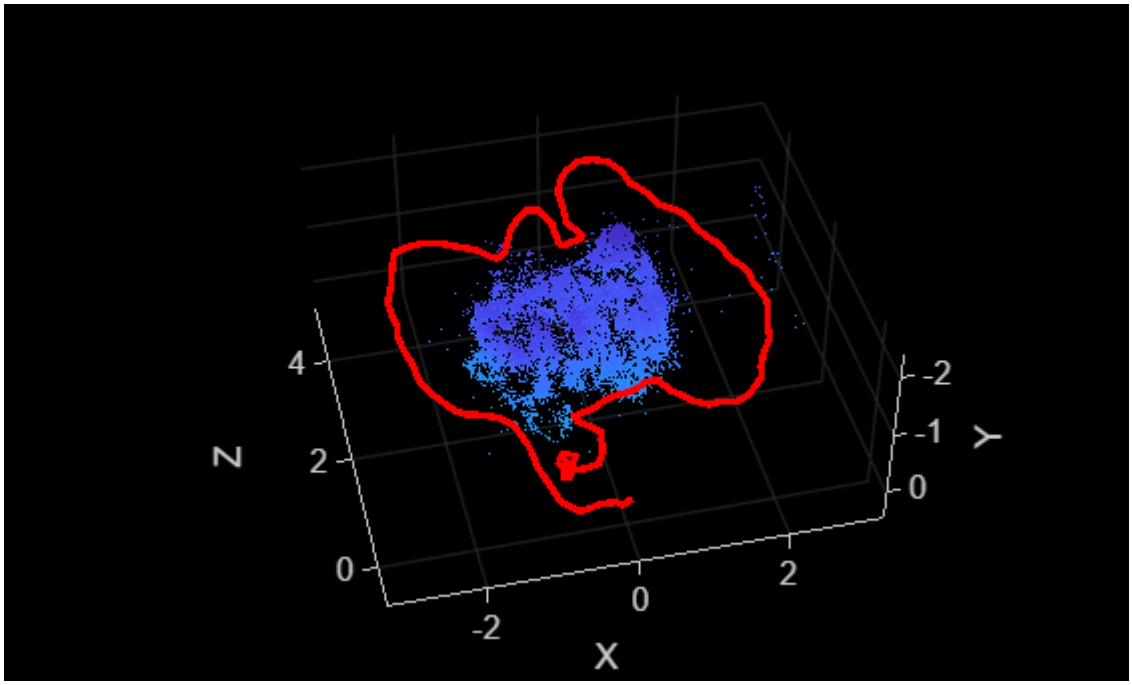
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

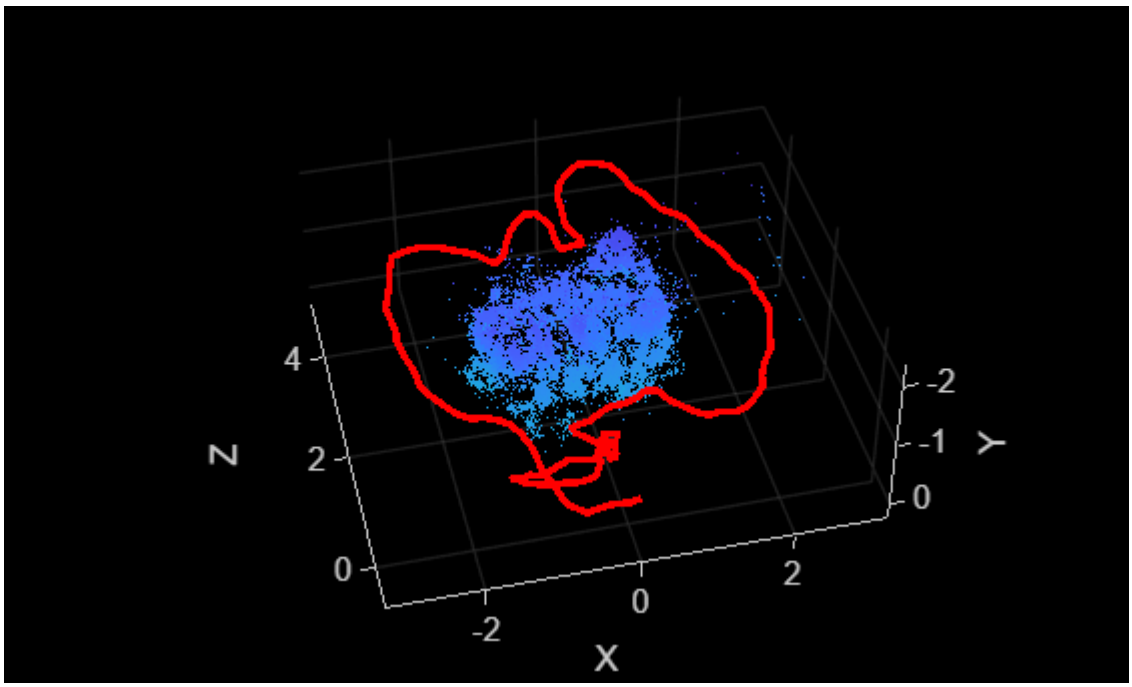
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Output Arguments

`status` — Current status of RGB-D visual SLAM object

`TrackingLost` enumeration | `TrackingSuccessful` enumeration | `FrequentKeyFrames` enumeration

Current status of the RGB-D visual SLAM object, returned as a `TrackingLost`, `TrackingSuccessful`, or `FrequentKeyFrames` enumeration. This table describes these enumerations.

Enumeration Value	Numeric Value	Description
`TrackingLost`	`uint8(0)`	Tracking is lost. The number of tracked feature points in the frame currently being processed is less than the lower limit of the `TrackFeatureRange` property of `vslam`. This indicates the image does not contain enough features, or that the camera is moving too fast. If the object does not accept enough frames as key frames, to improve tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` property and decrease the `SkipMaxFrames` property to add key frames more frequently.
`TrackingSuccessful`	`uint8(1)`	Tracking is successful. The number of tracked feature points in the frame currently being processed is between the lower limit and upper limit values of the `TrackFeatureRange` property of `vslam`.
`FrequentKeyFrames`	`uint8(2)`	Tracking adds key frames too frequently. The number of tracked feature points in the frame currently being processed is greater than the upper limit of the `TrackFeatureRange` property of `vslam`.

Version History

Introduced in R2024a

See Also

Objects
`rgbdvslam`

Functions
`addFrame` | `hasNewKeyFrame` | `isDone` | `mapPoints` | `poses` | `plot` | `reset`

isDone

End-of-processing status for RGB-D visual SLAM object

Syntax

```
done = isDone(vslam)
```

Description

`done = isDone(vslam)` returns a logical value indicating whether the RGB-D visual SLAM object has finished processing its frames. A value of `true` indicates the object is in an idle state. A value of `false` indicates the object is processing one or more frames.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

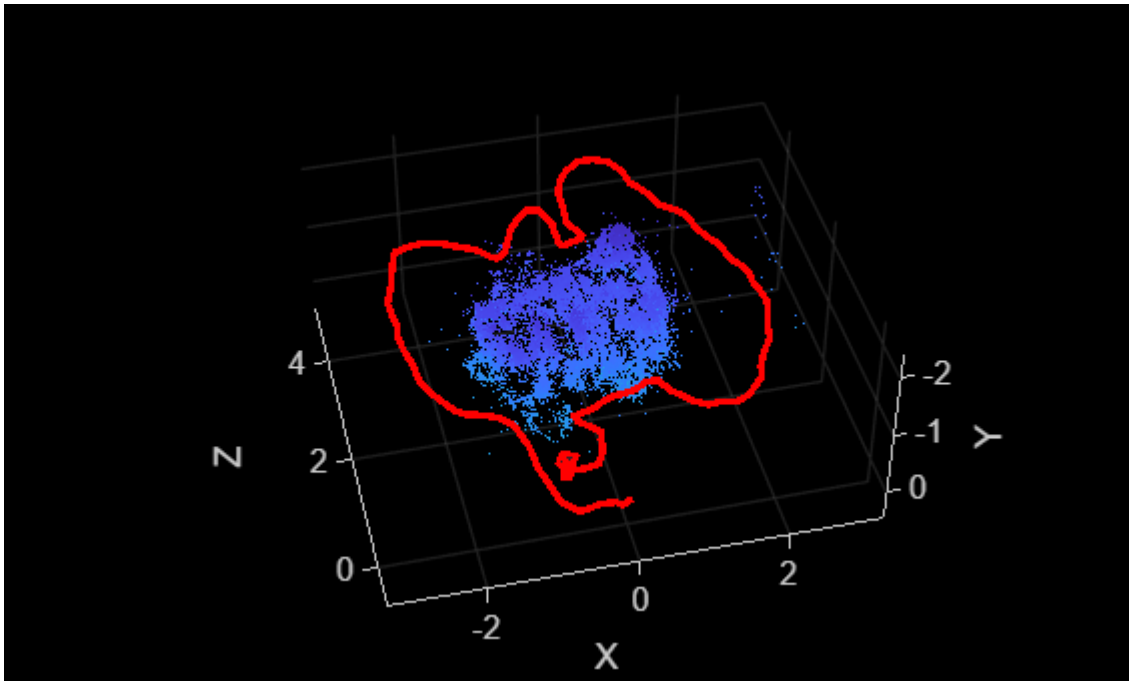
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

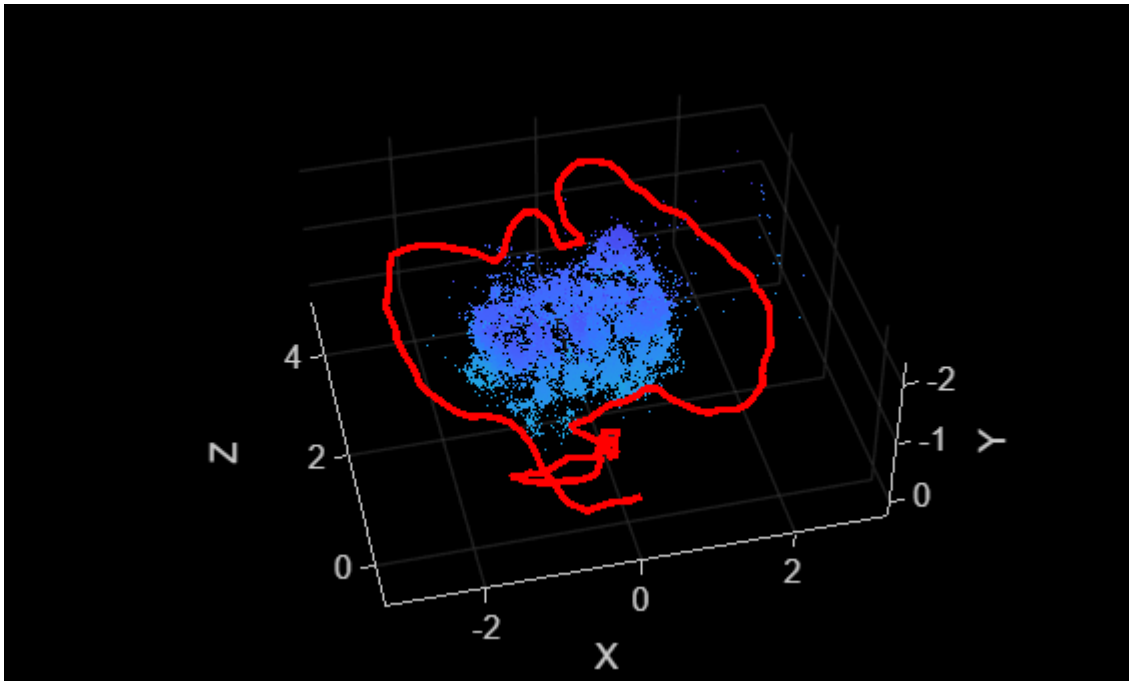
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

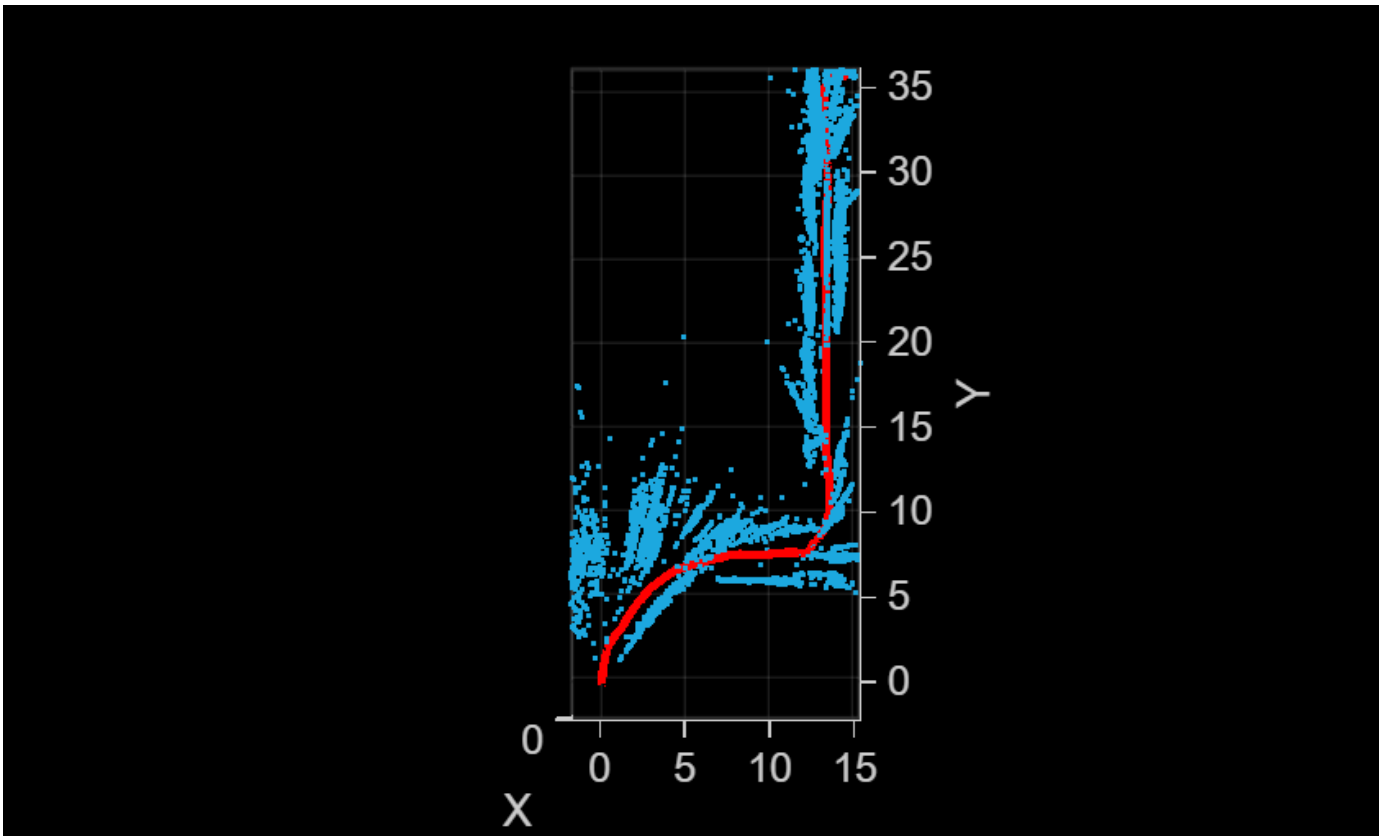
Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Version History

Introduced in R2024a

See Also

Objects

`rgbdvslam`

Functions

`addFrame` | `hasNewKeyFrame` | `checkStatus` | `mapPoints` | `poses` | `plot` | `reset`

mapPoints

Build 3-D map of world points from RGB-D vSLAM object

Syntax

```
xyzPoints = mapPoints(vslam)
```

Description

`xyzPoints = mapPoints(vslam)` builds a sparse 3-D map of world points, `xyzPoints`, from the input images in the RGB-D visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

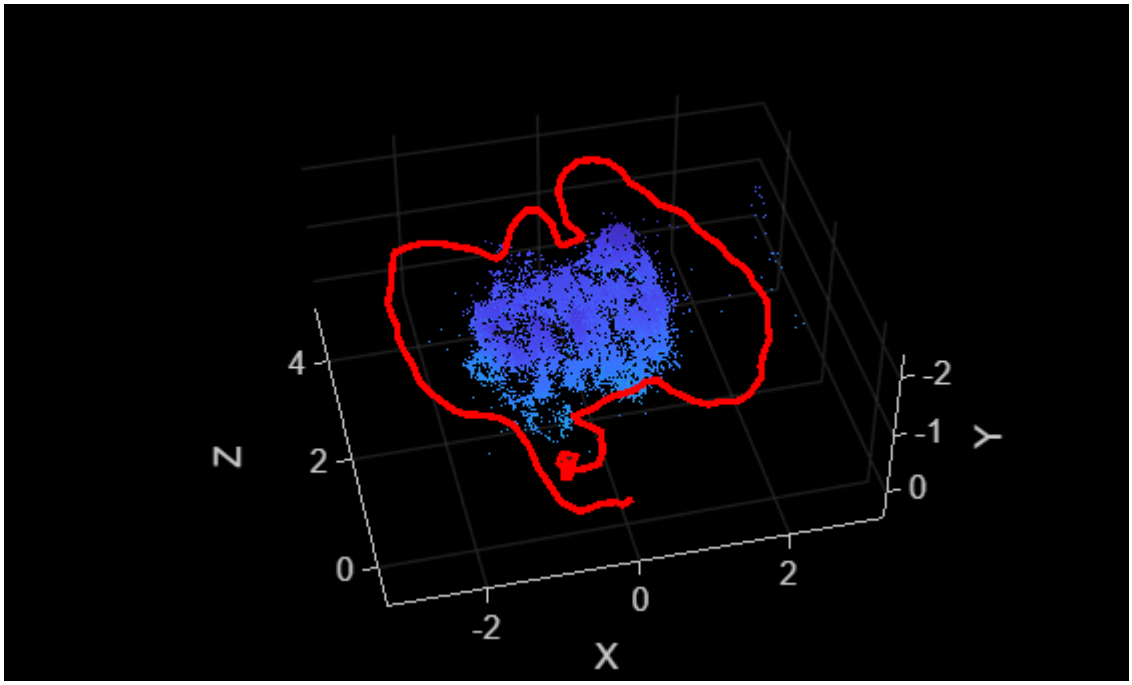
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

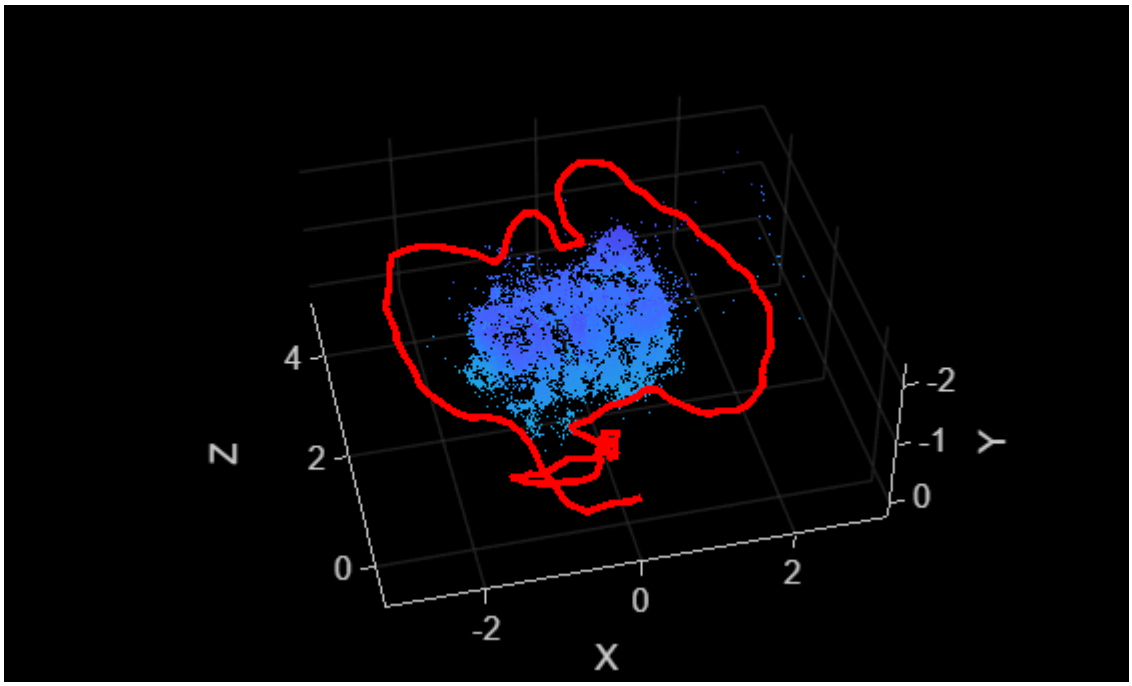
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Output Arguments

`xyzPoints` — World point coordinates of 3-D map

M-by-3 matrix

World point coordinates of 3-D map, returned as an *M*-by-3 matrix. *M* is the number of points, and each point is of the form [*X Y Z*].

Version History

Introduced in R2024a

See Also

Objects

`rgbdvslam`

Functions
`addFrame` | `hasNewKeyFrame` | `checkStatus` | `isDone` | `poses` | `plot` | `reset`

Topics
“Simulate RGB-D Visual SLAM System with Cosimulation in Gazebo and Simulink” (ROS Toolbox)

poses

Absolute camera poses of RGB-D vSLAM key frames

Syntax

```
camPoses = poses(vslam)
[camPoses,keyFrameIDs] = poses(vslam)
```

Description

`camPoses = poses(vslam)` returns the absolute camera poses `camPoses` of the key frames from the RGB-D visual simultaneous localization and mapping (vSLAM) object `vslam`.

`[camPoses,keyFrameIDs] = poses(vslam)` returns the IDs `keyFrameIDs` of the key frames corresponding to the camera poses. The IDs represent the order in which the image frames have been added to the `vslam` object by the `addFrame` object function.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

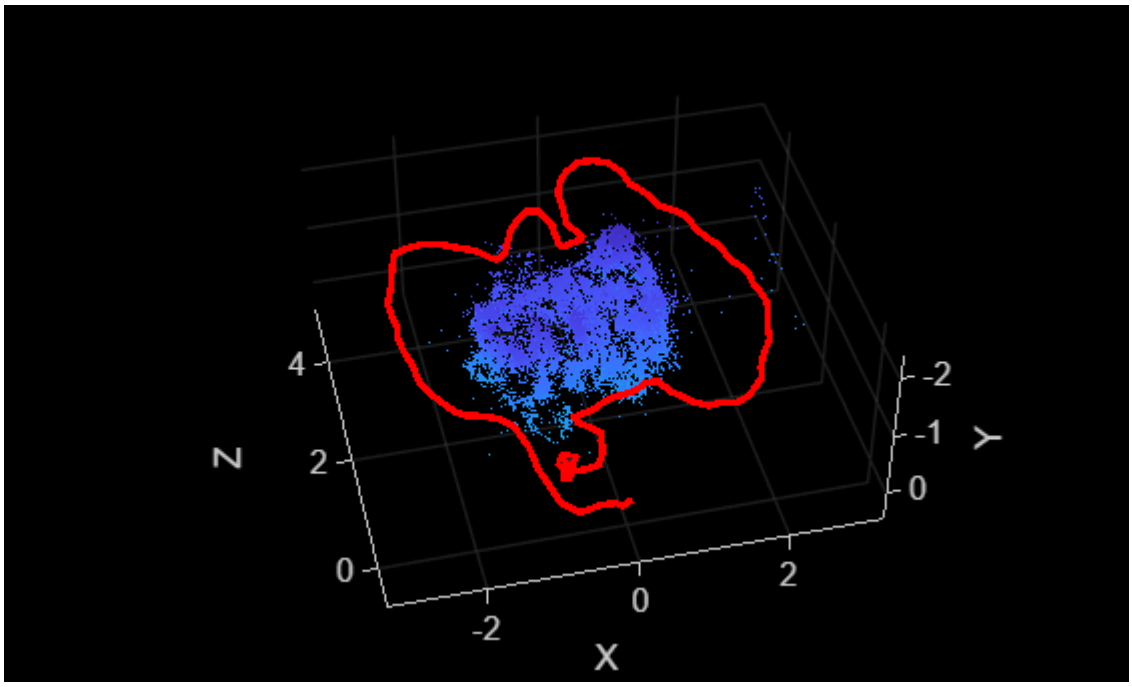
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

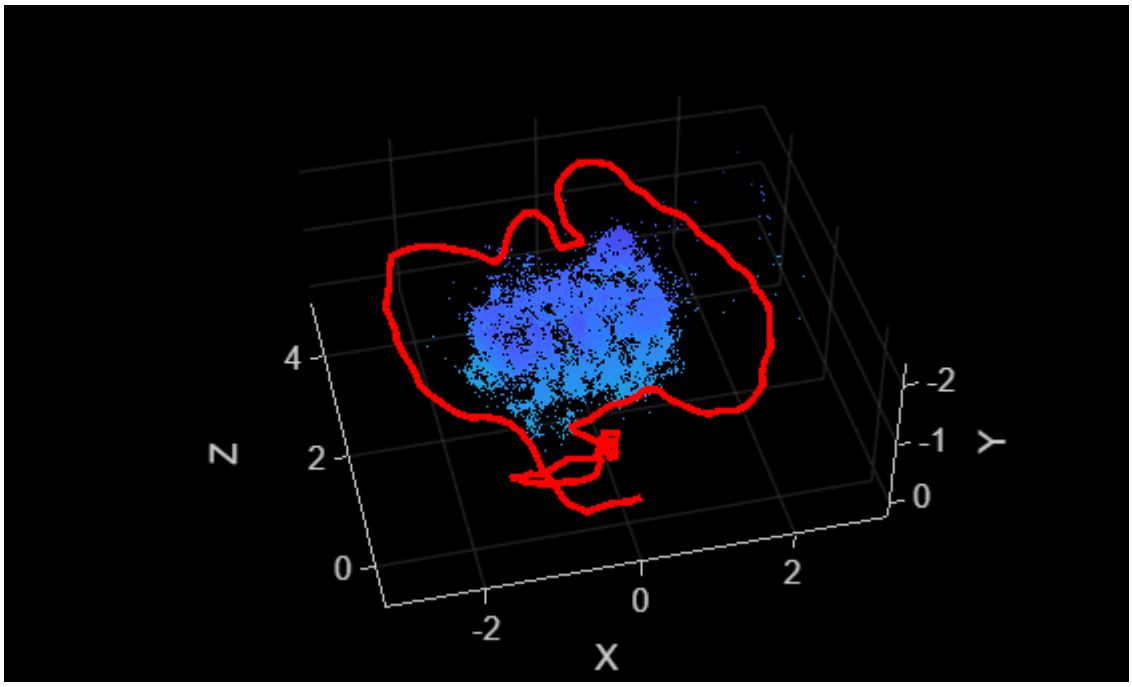
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Output Arguments

`camPoses` — Absolute camera poses of key frames

M-element array of `rigidtform3d` objects

Absolute camera poses of the key frames, returned as an *M*-element array of `rigidtform3d` objects.

`keyFrameIDs` — Key frame IDs for absolute camera poses

M-element array of integers

Key frame IDs for the absolute camera poses, returned as an *M*-element array of integers. Each element specifies the key frame ID for the camera pose in the corresponding element of `camPoses`. The IDs represent the order in which the image frames have been added to `vslam` by the `addFrame` object function.

Version History

Introduced in R2024a

See Also

Objects

`rgbdvslam`

Functions

`addFrame` | `hasNewKeyFrame` | `checkStatus` | `isDone` | `mapPoints` | `plot` | `reset`

Topics

“Simulate RGB-D Visual SLAM System with Cosimulation in Gazebo and Simulink” (ROS Toolbox)

plot

Plot 3-D map points and estimated camera trajectory in RGB-D visual SLAM

Syntax

```
plot(vslam)
plot(vslam,Name=Value)
ax = plot( ___ )
```

Description

`plot(vslam)` plots the 3-D map points and estimated camera trajectory from the RGB-D visual SLAM object `vslam`.

`plot(vslam,Name=Value)` specifies options using one or more name-value arguments. For example, `MarkerSize=10` sets the diameter of the marker size to 10 points.

`ax = plot( ___ )` returns the axes handle for the 3-D mapped points of the plot, using any combination of input arguments from previous syntaxes.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";
```

```
imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

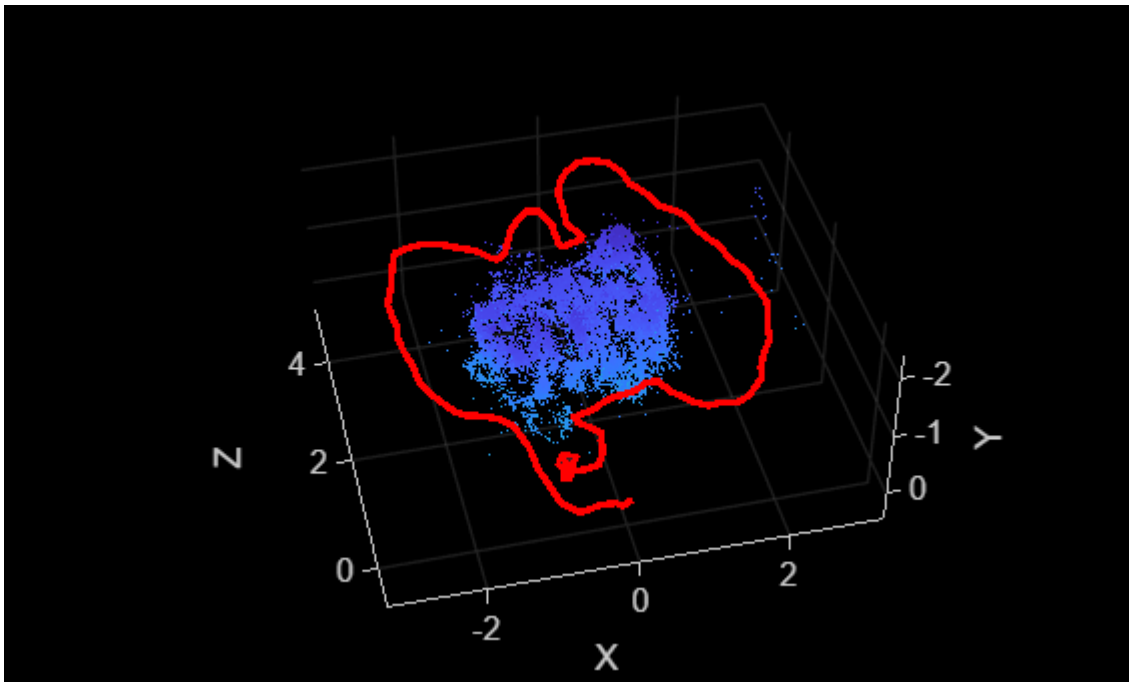
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

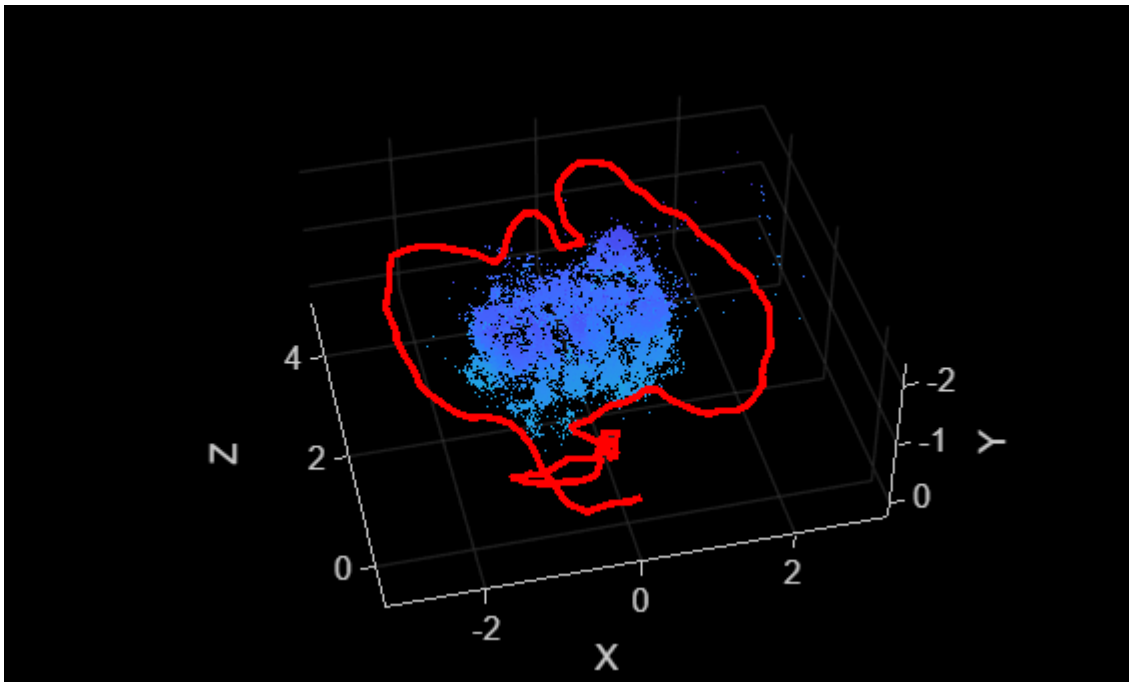
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

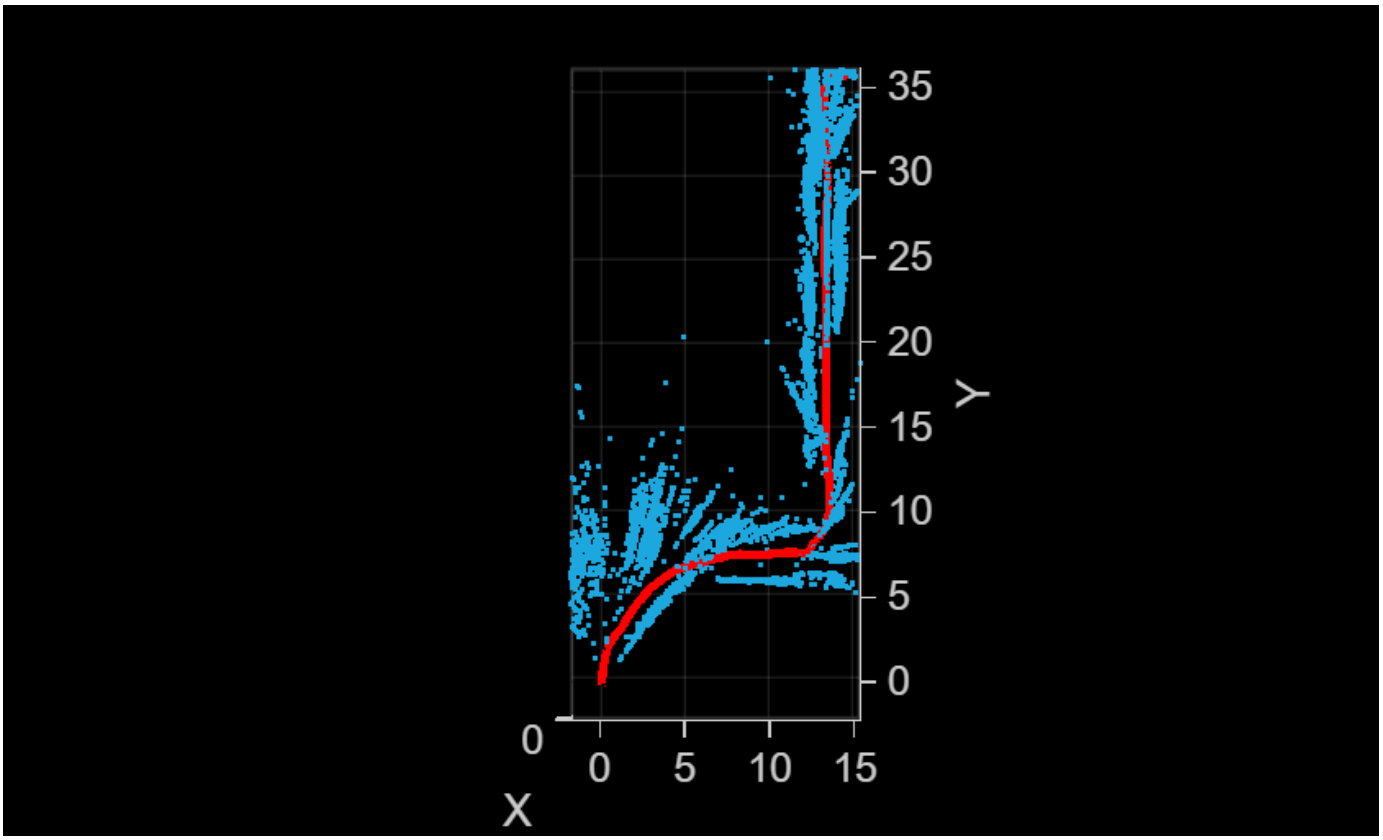
```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
```

```
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(vslam,MarkerSize=10)`, sets the diameter of the marker to 10 points.

`MarkerSize` — Diameter of markers for 3-D map points
`6` (default) | positive integer

Diameter of the markers for the 3-D map points, specified as a positive integer in points.

`MarkerColor` — Color of markers for 3-D map points
`[0 1 0]` (default) | RGB triplet | short or long color name

Color of the markers for the 3-D map points, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range `[0 1]`.

`ColorSource` — Colormap source for the markers
`"Y"` (default) | `"X"` | `"Z"` | `"MarkerColor"`

Colormap source for markers, specified as `"X"`, `"Y"`, `"Z"`, or `"MarkerColor"`.

`CameraSize` — Width of camera base
`0.1` (default) | positive scalar

Width of the camera base, specified as a positive scalar in the data units of the axes.

`CameraColor` — Color of camera
`[1 0 0]` (default) | RGB triplet | short or long color name

Color of the camera, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range `[0 1]`.

`Parent` — Axes for visualization
`Axes` object | `UIAxes` object

Axes for visualization, specified as an `Axes` graphics object or a `UIAxes` object. Use this name-value argument to visualize the plot in a UI for which you have specified `Figure` and `Axes` properties.

Output Arguments

`ax` — Axes handle
`Axes` graphics object

Axes handle, returned as an `axes` graphics object.

Version History

Introduced in R2024a

See Also

Objects
`rgbdvslam`

Functions
`addFrame` | `hasNewKeyFrame` | `checkStatus` | `isDone` | `mapPoints` | `poses` | `reset`

reset

Reset RGB-D visual SLAM object

Syntax

`reset(vslam)`

Description

`reset(vslam)` resets all internal processes and clears all data in the `rgbdvslam` object.

Examples

RGB-D Visual SLAM Using TUM RGB-D Data Set

Perform RGB-D visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://vision.in.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_o
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
colorImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
depthImageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/depth/";

imdsColor = imageDatastore(colorImageFolder);
imdsDepth = imageDatastore(depthImageFolder);
```

Select the synchronized pair of color and depth images.

```
data = load("rgbDepthPairs.mat");
imdsColor=subset(imdsColor, data.indexPairs(:, 1));
imdsDepth=subset(imdsDepth, data.indexPairs(:, 2));
```

Specify your camera intrinsic parameters, and use them to create an RGB-D visual SLAM object.

```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
depthScaleFactor = 5000;
vslam = rgbdvslam(intrinsics,depthScaleFactor);
```

Process each pair of color and depth images, and visualize the camera poses and 3-D map points.

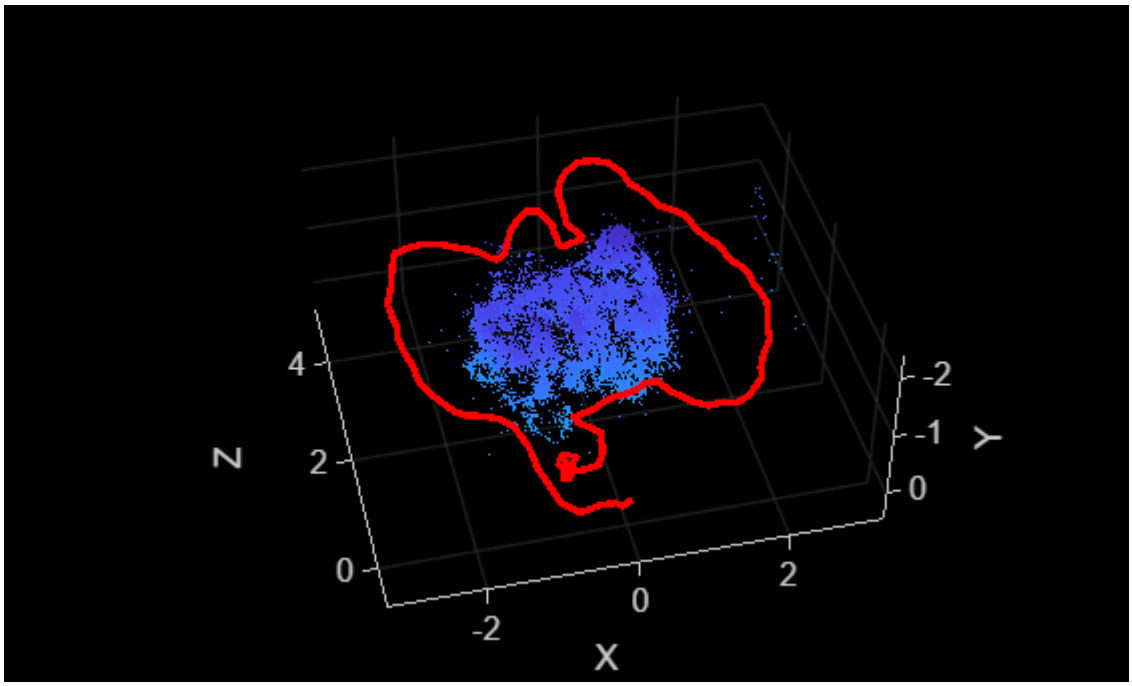
```
for i = 1:numel(imdsColor.Files)
    colorImage = readimage(imdsColor,i);
    depthImage = readimage(imdsDepth,i);
    addFrame(vslam,colorImage,depthImage);

    if hasNewKeyFrame(vslam)
        % Query 3-D map points and camera poses
        xyzPoints = mapPoints(vslam);
        [camPoses,viewIds] = poses(vslam);

        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

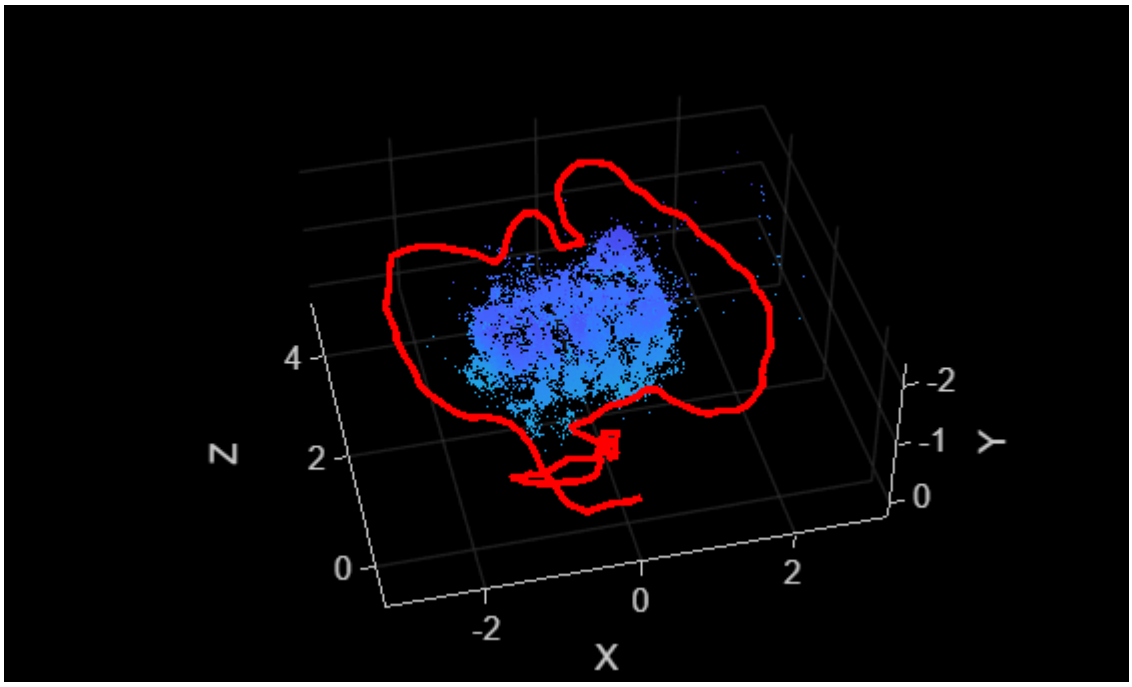
    % Get current status of system
    status = checkStatus(vslam);

    % Stop adding frames when tracking is lost
    if status == uint8(0)
        break
    end
end
```



Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    plot(vslam);
end
```




```
reset(vslam);
```

RGB-D Visual-Inertial SLAM Using OpenLORIS-Scene Data Set

Perform RGB-D visual-inertial SLAM using the data from the OpenLORIS-Scene Dataset. Download the data to a temporary directory using a web browser or by running this code:

```
dataFolder  = fullfile(tempdir,"OpenLORIS-Scene",filesep);
downloadURL = "https://ssd.mathworks.com/supportfiles/shared_nav_vision/data/OpenLORIS-Scene_cor
zipFileName = dataFolder+"corridor1-4.zip";

if ~isfolder(dataFolder)
    mkdir(dataFolder);
    disp("Downloading corridor1-4.zip (1.13 GB). This download can take a few minutes.");
    options = weboptions('Timeout', Inf);
    websave(zipFileName, downloadURL, options);
    unzip(zipFileName, dataFolder);
end
```

Create two `imageDatastore` objects. One to store the color images and the other to store the depth images.

```
imageFolder = fullfile(dataFolder,"OpenLORIS-Scene_corridor1-4");
imdsColor = imageDatastore(fullfile(imageFolder,"color"));
imdsDepth = imageDatastore(fullfile(imageFolder,"aligned_depth"));
```

Load the IMU measurements data and the camera-to-IMU transform.

```
data    = load("corridor4_IMU_data.mat");
gyro    = data.gyroDataCell;
```

```
accel   = data.accelDataCell;
cam2IMU = data.cam2IMU;
```

Specify the camera intrinsics, the IMU parameters, and use them to create an RGB-D visual-inertial SLAM object.

```
% Camera intrinsic and IMU parameters can be found in the downloaded
% sensors.yaml file
intrinsics = cameraIntrinsics([6.1145098876953125e+02, 6.1148571777343750e+02],...
    [4.3320397949218750e+02, 2.4947302246093750e+02], [480, 848]);

imuParams = factorIMUParameters(AccelerometerBiasNoise=2.499999936844688e-05*eye(3),...
       AccelerometerNoise=0.00026780980988405645*eye(3),...
       GyroscopeNoise=1.0296060281689279e-05*eye(3),...
       GyroscopeBiasNoise=2.499999993688107e-07*eye(3),...
       SampleRate=250);

depthScaleFactor = 1000;
vslam = rgbdvslam(intrinsics, depthScaleFactor, imuParams, SkipMaxFrames=10,...
    CameraToIMUTransform=cam2IMU, TrackFeatureRange = [30, 150], DepthRange= [0.1, 6.5], ...
    NumPosesThreshold=20, MaxNumPoints=1.2e3);
```

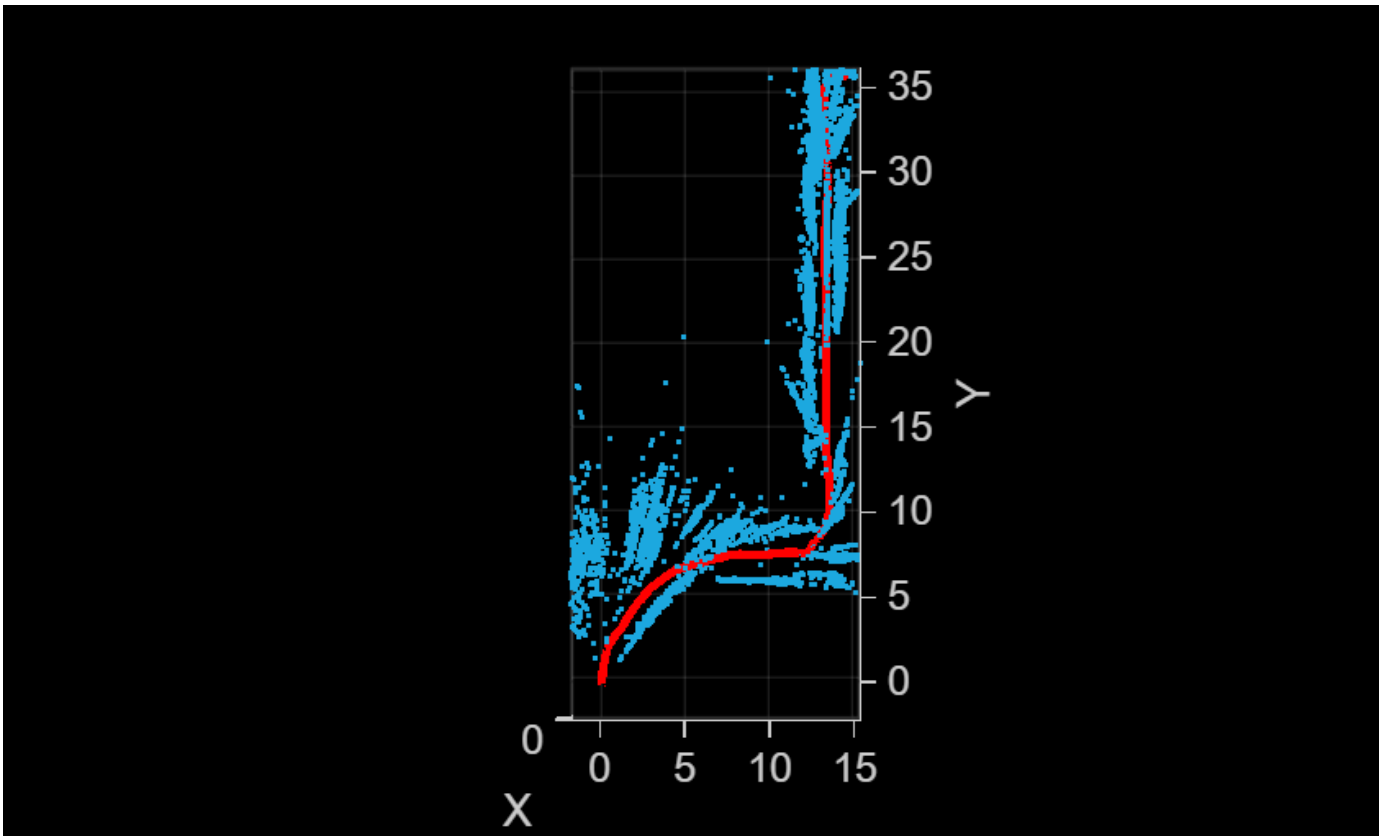
Process image data and IMU data, and visualize the camera poses and 3-D map points. The abrupt changes observed in the camera poses demonstrate that the camera trajectories have been successfully aligned with the IMU predictions, which is an essential step in visual-inertial SLAM. This process transforms the camera poses into the IMU's coordinate system.

```
for i = 1:numel(imdsColor.Files)
    colorImage  = readimage(imdsColor,i);
    depthImage  = readimage(imdsDepth,i);
    addFrame(vslam, colorImage, depthImage, gyro{i}, accel{i});

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Once all the frames have been processed, reset the system.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        ax = plot(vslam);
    end
end
view(ax, 0, 90)
```




```
reset(vslam);
```

Input Arguments

`vslam` — RGB-D visual SLAM object

`rgbdvslam` object

RGB-D visual SLAM object, specified as an `rgbdvslam` object.

Version History

Introduced in R2024a

See Also

Objects

`rgbdvslam`

Functions

`addFrame` | `hasNewKeyFrame` | `checkStatus` | `isDone` | `mapPoints` | `poses` | `plot`

monovslam

Visual simultaneous localization and mapping (vSLAM) and visual-inertial sensor fusion with monocular camera

Description

Use the `monovslam` object to perform visual simultaneous localization and mapping (vSLAM) and visual inertial SLAM (viSLAM) with a monocular camera. The object extracts Oriented FAST and Rotated BRIEF (ORB) features from incrementally read images, and then tracks those features to estimate camera poses, identify key frames, and reconstruct a 3-D environment. The vSLAM algorithm also searches for loop closures using the bag-of-features algorithm, and then optimizes the camera poses using pose graph optimization. You can enhance the accuracy and robustness of the SLAM by integrating this object with IMU data to perform visual-inertial sensor fusion. To learn more about visual SLAM, see “Implement Visual SLAM in MATLAB”.

Note The `monovslam` object requires Navigation Toolbox.

Creation

Syntax

```
vslam = monovslam(intrinsics)
vslam = monovslam(intrinsics,imuParameters)
vslam = monovslam(intrinsics,PropertyName=Value)
```

Description

`vslam = monovslam(intrinsics)` creates a monocular visual SLAM object, `vslam`, by using the camera intrinsic parameters.

The `monovslam` object does not account for lens distortion. You can use the specified `undistortImage` function to undistort images before adding them to the object.

The object represents 3-D map points and camera poses in world coordinates. The object assumes the camera pose of the first key frame is an identity `rigidtform3d` transform.

Note The `monovslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`vslam = monovslam(intrinsics,imuParameters)` creates a monocular visual SLAM object, `vslam`, by fusing the camera input intrinsics with inertial measurement unit (IMU) readings.

`vslam = monovslam(intrinsics,PropertyName=Value)` sets properties using one or more name-value arguments. For example, `MaxNumFeatures=850` sets the maximum number of ORB feature points to extract from each image to `850`.

Input Arguments

`intrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

This argument sets the `Intrinsics` property.

`imuParameters` — IMU parameters

`factorIMUParameters` object

IMU parameters, specified as a `factorIMUParameters` object. The object contains the noise, bias, and sample rate information about the inertial measurement unit (IMU).

Properties

Camera Parameters

`Intrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object

This property is read-only.

Camera intrinsic parameters, stored as a `cameraIntrinsics` object.

Use the `intrinsics` argument to set this property.

Feature Extraction

`ScaleFactor` — Scale factor for image decomposition

`1.2` (default) | scalar greater than 1

This property is read-only after object creation.

Scale factor for image decomposition, stored as a scalar greater than 1. The scale factor is also referred to as the pyramid decimation ratio. Increasing the value of `ScaleFactor` reduces the number of pyramid levels, but reduce computation time. Decreasing this value (down to just over 1), increases the number of pyramid levels, which can improve tracking performance, at the cost of computation speed. The scale value at each level of decomposition is $\texttt{ScaleFactor}^{(level\text{-}1)}$, where *level* is any value in the range [0, `Numlevels`-1]. Given the input image of size *M*-by-*N*, the image size at each level of composition is M_{level}-by-N_{level}, where:

$$M_{level} = \frac{M}{ScaleFactor^{(level\ -\ 1)}}$$

$$N_{level} = \frac{N}{ScaleFactor^{(level\ -\ 1)}}$$

`NumLevels` — Number of decomposition levels

`8` (default) | integer greater than or equal to `1`

This property is read-only after object creation.

Number of decomposition levels, specified as an integer greater than or equal to `1`. Increase this value to extract keypoints from the image at more levels of decomposition. Along with the `ScaleFactor` value, `NumLevels` controls the number of pyramid levels on which the object evaluates feature points.

The image size at each decomposition level limits the number of levels at which you can extract keypoints. The image size at a level of decomposition must be at least 63-by-63 for keypoint detection. The maximum level of decomposition is calculated as

$$level_{\max} = \text{floor}\left(\frac{\log(\min(M,N)) - \log(63)}{\log(\text{ScaleFactor})}\right) + 1$$

If either the default value or the specified value of `NumLevels` is greater than $level_{max}$, the object modifies `NumLevels` to $level_{max}$ and returns a warning.

`MaxNumPoints` — Maximum number of ORB feature points uniformly extracted from each image

`1000` | positive integer

This property is read-only after object creation.

Maximum number of ORB feature points uniformly extracted from each image, specified as a positive integer. Values are typically in the range of [800, 2000], depending on the resolution of the image. When the number of extracted features is less than the value of `MaxNumPoints`, then the object uses all feature points.

Tracking

`TrackFeatureRange` — Range for the number of feature points to identify key frame

`[30 100]` | two-element vector of positive integers

This property is read-only after object creation.

Key frame feature point range, stored as a two-element vector of positive integers in the form [*lowerLimit upperLimit*]. This property specifies the minimum and maximum numbers of tracked features points a frame must contain for the object to identify it as a key frame. The `TrackFeatureRange` and the `SkipMaxFrames` properties enable you to control the frequency at which frames in the tracking process become key frames.

The success of tracking depends on the number of tracked points in the current frame, with one of these results:

- Tracking is lost — The number of tracked feature points in the current frame is less than the *lowerLimit* set by the `TrackFeatureRange` argument. This indicates that the image does not contain enough features, or that the camera is moving too fast. To improve the tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` property and decrease the `SkipMaxFrames` property to add key frames more frequently.
- Tracking is successful — The object identifies the current frame as a key frame. The number of tracked feature points in the current frame is in the range set by `TrackFeatureRange`.
- Tracking adds key frames too frequently — The number of tracked feature points in the current frame is greater than the *upperLimit* set by the `TrackFeatureRange` property. This indicates that the camera is moving very slowly, which produces an unnecessary number of key frames. To

improve the tracking, you can reduce the frequency of adding key frames by increasing the value of the `SkipMaxFrames` property.

For more details, see the `addFrame` object function and “How to Improve Accuracy in Visual SLAM”.

`SkipMaxFrames` — Maximum number of skipped frames
`20` | positive integer

This property is read-only after object creation.

Maximum number of skipped frames, stored as a positive integer. When the number of tracked features is consistently greater than the *upperLimit* set by the `TrackFeatureRange` property, use the `SkipMaxFrames` property to control the frequency at which the object adds new key frames. The object identifies the current frame as a key frame when the number of skipped frames since the most recently added key frame equals the value of `SkipMaxFrames`. For more details, see “How to Improve Accuracy in Visual SLAM”.

`LoopClosureThreshold` — Minimum number of matched feature points between loop closure key frames
`60` | positive integer

This property is read-only after object creation.

Minimum number of matched feature points between loop closure key frames, stored as a positive integer.

`CustomBagOfFeatures` — Custom bag of features for loop detection
`[]` (default) | `bagOfFeaturesDBoW` object

This property is read-only after object creation.

Custom bag of features for loop detection, specified as a `bagOfFeaturesDBoW` object. The `bagOfFeaturesDBoW` enables you to create a custom bag of words (BoW) from feature descriptors, alongside options to utilize a built-in vocabulary or load a custom one from a specified file.

`Verbose` — Progress information display
`[]` (default) | `1` | `2` | `3`

This property is read-only after object creation.

Progress information display, specified as `[]`, `1`, `2`, or `3`. Paths to the location of log files, when they are created, is displayed on the command line. For more details, see “How to Improve Accuracy in Visual SLAM”.

Verbose value	Display description	Display location
`[]` or `false`	Display is turned off	—
`1` or `true`	Stages of vSLAM execution	Command window
`2`	Stages of vSLAM execution, with details on how the frame is processed, such as the artifacts used to initialize the map.	Log file in a temporary folder

Verbose value	Display description	Display location
3	Stages of vSLAM, artifacts used to initialize the map, poses and map points before and after bundle adjustment, and loop closure optimization data.	Log file in a temporary folder

IMU Fusion

`CameraToIMUTransform` — IMU sensor transformation
`rigidtform3d`

This property is read-only after object creation.

IMU sensor transformation, specified as a `rigidtform3d` object. The transformation describes the rotation and translation of the camera in the coordinate system of the IMU sensor.

`NumPosesThreshold` — Number of estimated camera poses to trigger camera-IMU alignment
`30` (default) | integer equal to or greater than 2

This property is read-only after object creation.

Number of estimated camera poses to trigger camera-IMU alignment, specified as an integer equal to or greater than 2. The process of aligning camera and IMU data is initiated after a specific number of camera poses have been estimated. This alignment serves two primary purposes: first, to estimate a scale factor that translates the up-to-scale results from a monocular camera into actual world units (meters), and second, to synchronize the IMU and camera frames, effectively eliminating the influence of gravity on accelerometer data. The timing for this alignment, determined by a threshold for the number of camera poses, is key to its success. A threshold set too low may not provide enough data for accurate calibration, while one set too high risks incorporating noise from measurement drift into the calibration. For a more in-depth understanding of this calibration technique, see the `estimateGravityRotationAndPoseScale` function.

`AlignmentFraction` — Subset of pose estimates
`0.8` (default) | scalar in the range (0,1]

This property is read-only after object creation.

Subset of pose estimates, specified as a scalar in the range of (0,1]. This value specifies a fraction of the number of recent pose estimates, calculated as `round(NumPosesThreshold*AlignmentFraction)`, for use in the camera-IMU alignment process. It effectively filters out initial, potentially noisy pose estimates, ensuring only the most relevant data contributes to the alignment for improved accuracy.

`IsIMUAligned` — Logical flag indicating IMU alignment completed successfully
`false` (default) | `true`

This property is read-only after object creation.

Logical flag indicating the IMU alignment completed successfully, specified as `true` or `false`. When IMU alignment completes successfully, the algorithm sets `IsIMUAligned` to `true`. Otherwise the value is `false`.

`GravityRotation` — Output rotation of IMU alignment process

`rigidtform3d` object (default)

This property is read-only after object creation.

Output rotation of IMU alignment process, specified as a `rigidtform3d` object. The rotation transforms the gravity vector from the IMU local navigation frame (NED or ENU) to the input pose reference frame.

The value of this property is automatically updated after the IMU alignment completes successfully and the `IsIMUAligned` property is `true`.

`IMUScale` — Scale of IMU alignment

1 object (default) | positive scalar

This property is read-only after object creation.

Scale of IMU alignment, specified as a positive scalar. The scale is applied as a multiplier to the input poses so that their units match the IMU measurements.

The value of this property is automatically updated after the IMU alignment completes successfully and the `isIMUAligned` property is `true`.

Object Functions

addFrame	Add image frame to visual SLAM object
hasNewKeyFrame	Check if new key frame added in visual SLAM object
checkStatus	Check status of visual SLAM object
isDone	End-of-processing status for visual SLAM object
mapPoints	Build 3-D map of world points
poses	Absolute camera poses of key frames
plot	Plot 3-D map points and estimated camera trajectory in visual SLAM
reset	Reset visual SLAM object

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_off
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);
```

```
    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

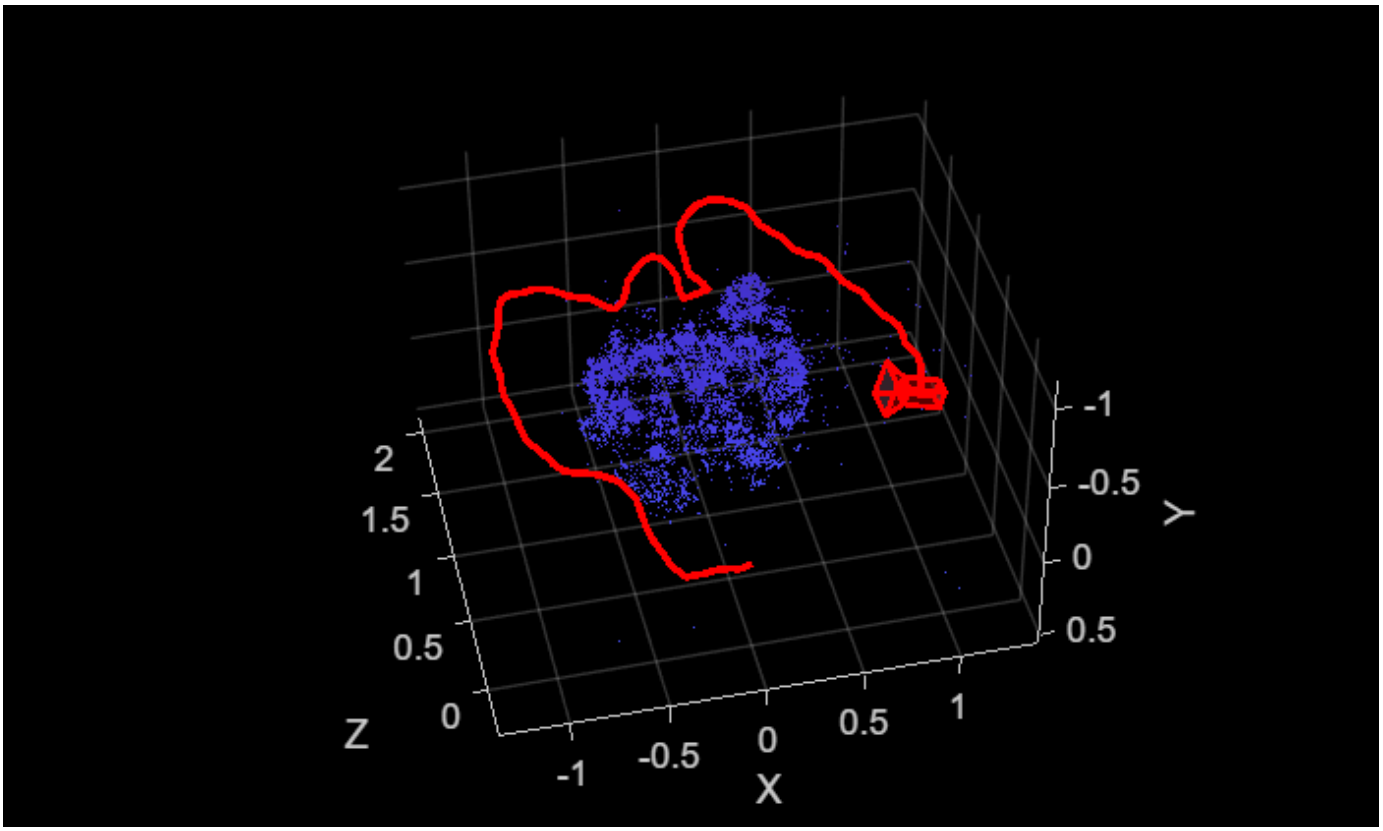
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

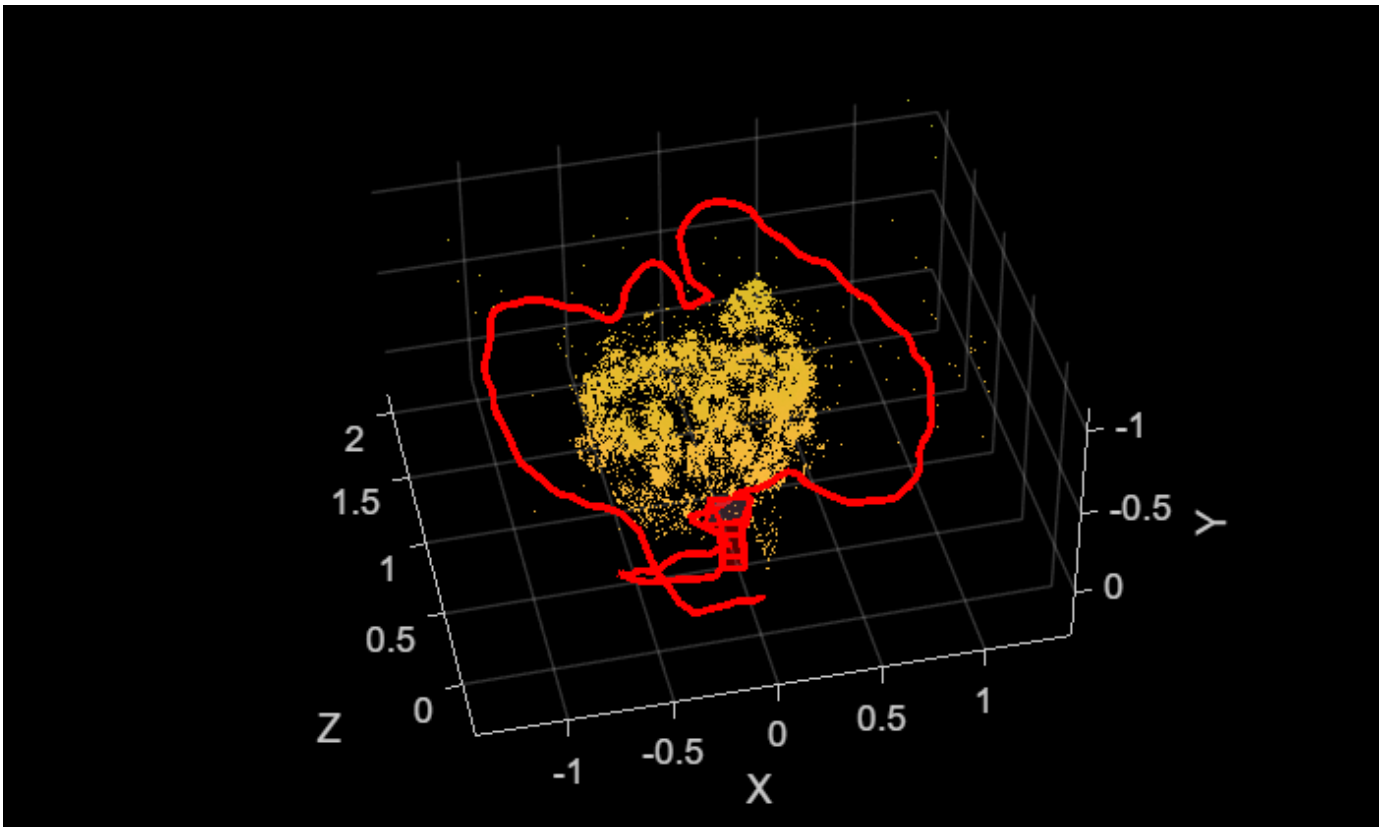
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

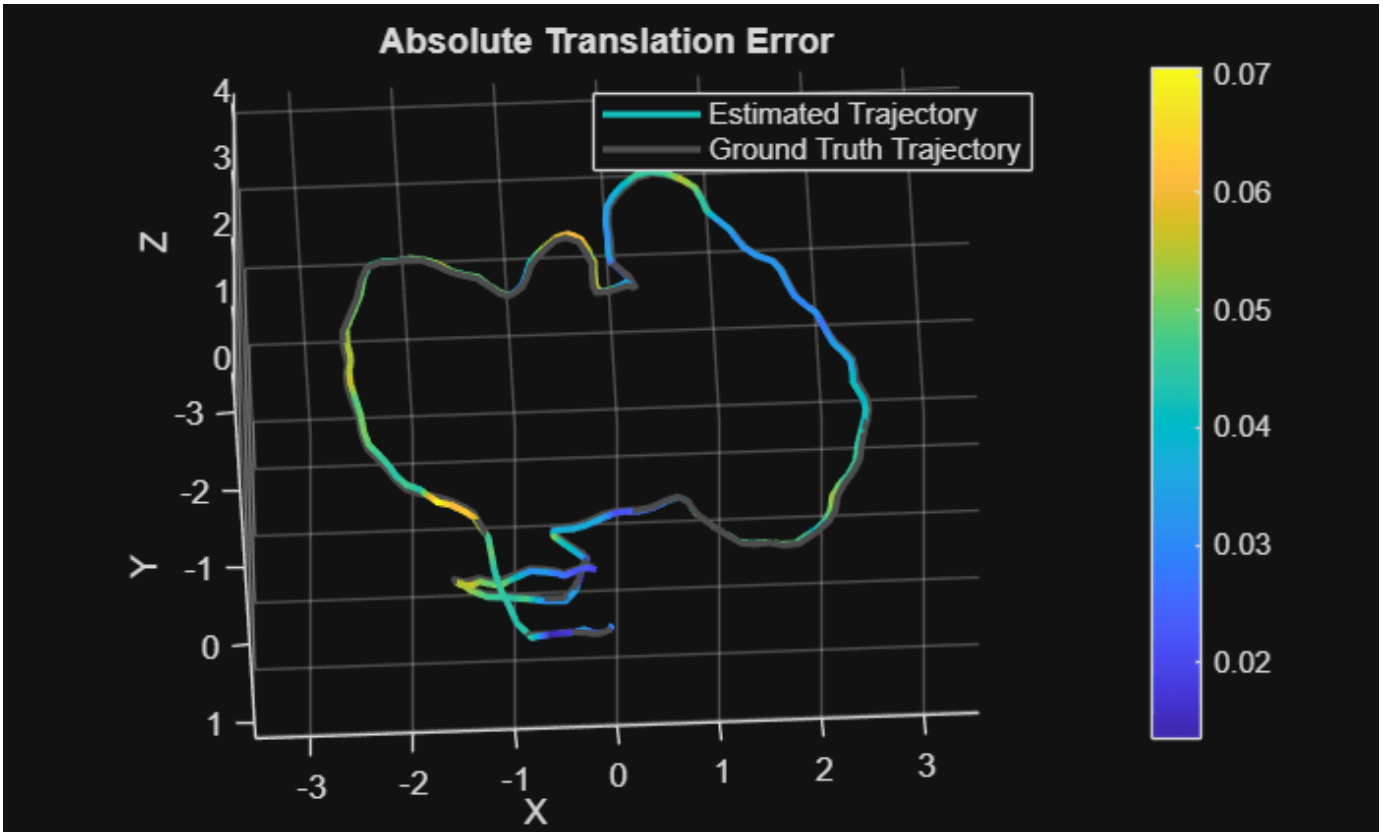
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Monocular Visual-Inertial SLAM Using the BlackBird Data Set

Perform monocular visual-inertial SLAM using the data from the Blackbird data set (NYC Subway Winter). Download the MAT file which contains the images, camera intrinsics and IMU measurements.

```
uavDataTarFile =  matlab.internal.examples.downloadSupportFile('shared_nav_vision/data','Blackbi

% Extract the file.
outputFolder = fileparts(uavDataTarFile);
if (~exist(fullfile(outputFolder,"BlackbirdVIOData"),"dir"))
    untar(uavDataTarFile,outputFolder);
end

uavData = load(fullfile(outputFolder,"BlackbirdVIOData","data.mat"));

images     = uavData.images;
timeStamps = uavData.timeStamps;
```

Set up the IMU noise parameters, camera intrinsics and the camera-IMU extrinsics transform.

```
imuParams = factorIMUParameters(SampleRate=100,GyroscopeNoise=0.1, ...
            GyroscopeBiasNoise=3e-3,AccelerometerNoise=0.3, ...
            AccelerometerBiasNoise=1e-3,ReferenceFrame="ENU");
intrinsics = uavData.intrinsics;
camera2IMU = rigidtform3d(tform(uavData.camToIMUTransform));
```

Set up the monocular visual-inertial SLAM object.

```
vslam = monovslam(intrinsics,imuParams,MaxNumPoints=2000,SkipMaxFrames=2,TrackFeatureRange=[20,1(
        CameraToIMUTransform=camera2IMU,NumPosesThreshold=40,AlignmentFraction=0.9, ...
        CustomBagOfFeatures=bagOfFeaturesDBoW('BlackBirdBoW.bin.gz'),LoopClosureThreshold=100);
```

Process all the images frames and IMU measurements available, and visualize the camera poses and 3-D map points. The sequence does not contain any movement before the 150th frame.

```
startFrameIdx = 150;

for i = startFrameIdx:length(images)

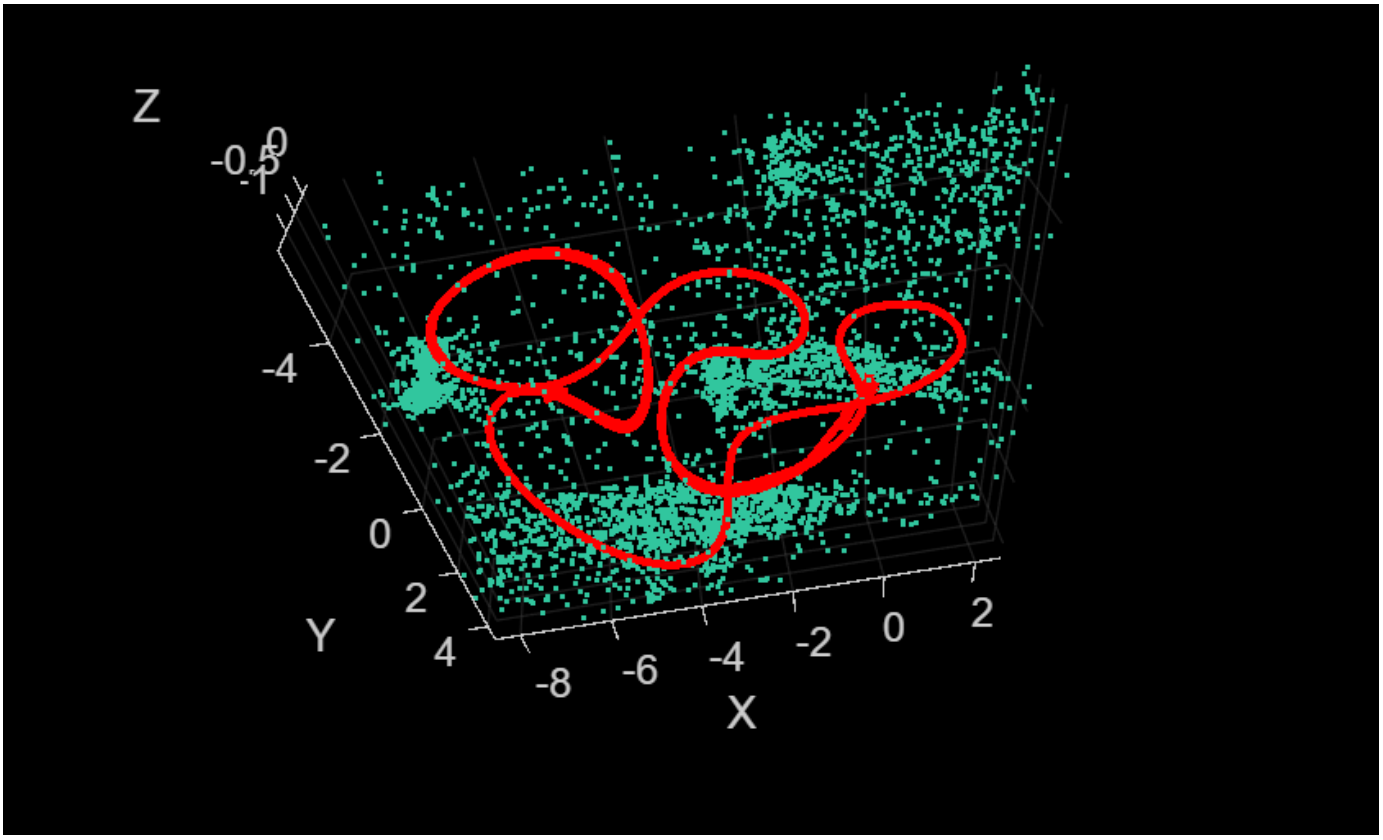
    if(i>startFrameIdx)
        imuMesurements = helperExtractIMUMeasurements(uavData, i-1, i);
    else
        imuMesurements.gyro=[];
        imuMesurements.accel=[];
    end

    addFrame(vslam, images{i}, imuMesurements.gyro, imuMesurements.accel);

    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```

Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
% Plot intermediate results and wait until all images are processed
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



The helper function below is used to extract the IMU measurements, based on the camera timestamps.

```
function imuMesurements = helperExtractIMUMeasurements(data, startFrameIdx, currFrameIdx)

    timeStamps = data.timeStamps;

    startTimeStamp = timeStamps.imageTimeStamps(startFrameIdx);
    currTimeStamp  = timeStamps.imageTimeStamps(currFrameIdx);

    [~,startIMUIdx] = min(abs(timeStamps.imuTimeStamps - startTimeStamp));
    [~,currIMUIdx]  = min(abs(timeStamps.imuTimeStamps - currTimeStamp));

    imuMesurements.accel = data.accelReadings(startIMUIdx:(currIMUIdx-1),:);
    imuMesurements.gyro  = data.gyroReadings(startIMUIdx:(currIMUIdx-1),:);

end
```

Version History

Introduced in R2023b

R2025a: `monovslam` available in Navigation Toolbox with Computer Vision Toolbox license

You can use the `monovslam` object in Navigation Toolbox if you have a Computer Vision Toolbox license.

R2025a: MATLAB Coder support added for IMU sensor support

Added C/C++ support for visual-inertial sensor fusion using the `monovslam` object. Starting in R2025a, you can generate C/C++ code using the IMU parameters along with all other object parameters.

R2024b: Fusion support for IMU sensors

Use the `imuParameters` input argument as a `factorIMUParameters` object to specify information about IMU parameters. Use the `CameraToIMUTransform`, `NumPosesThreshold`, and `AlignmentFraction` properties to tune IMU parameters. The IMU-based features of the `monovslam` object do not support C/C++ code generation.

R2024b: Add ability to specify custom bag of words

Use the `CustomBagOfFeatures` name-value argument to specify a custom bag of words vocabulary as a `bagOfFeaturesDBoW` object.

R2024b: `Verbose` property enhanced to display progress information
Behavior changed in R2024b

The `Verbose` property added values of `1`, `2`, and `3` to display levels of progress information.

References

[1] Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System." *IEEE Transactions on Robotics* 31, no. 5 (October 2015): 1147–63. https://doi.org/10.1109/TRO.2015.2463671.

[2] Galvez-López, D., and J. D. Tardos. "Bags of Binary Words for Fast Place Recognition in Image Sequences." *IEEE Transactions on Robotics*, vol. 28, no. 5, Oct. 2012, pp. 1188–97. *DOI.org (Crossref)*, https://doi.org/10.1109/TRO.2012.2197158.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

To compile the generated code for non-host target, these dependencies must be installed:

- OpenCV 4.7.0
- Ceres-solver 2.2.0 library — Download the source and build it as a shared library, on your remote device.
- eigen3 Library

When generating portable C code with a C++ compiler, you must specify hierarchical packing with non-minimal headers. For more information on packaging options, see the `packNGo` function.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`stereovslam` | `rgbdvslam` | `factorIMUParameters` | `cameraIntrinsics` | `imageDatastore` | `ORBPoints`

Functions
`detectORBFeatures`

Topics
“Implement Visual SLAM in MATLAB”
“Performant and Deployable Monocular Visual SLAM”
“Monocular Visual Simultaneous Localization and Mapping”
“Monocular Visual-Inertial SLAM”

External Websites
What is SLAM?

addFrame

Add image frame to visual SLAM object

Syntax

```
addFrame(vslam,I)
addFrame( ___ ,gyro,accel)
```

Description

`addFrame(vslam,I)` adds a grayscale or RGB image `I`, to the visual SLAM object `vslam`.

Note The `monovslam` object runs on multiple threads internally, which can delay the processing of an image frame added by using the `addFrame` function. Additionally, the object running on multiple threads means the current frame the object is processing can be different than the recently added frame.

`addFrame( ___ ,gyro,accel)` adds IMU gyroscope `gyro`, and accelerometer `accel`, measurements to the visual SLAM object.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_off
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

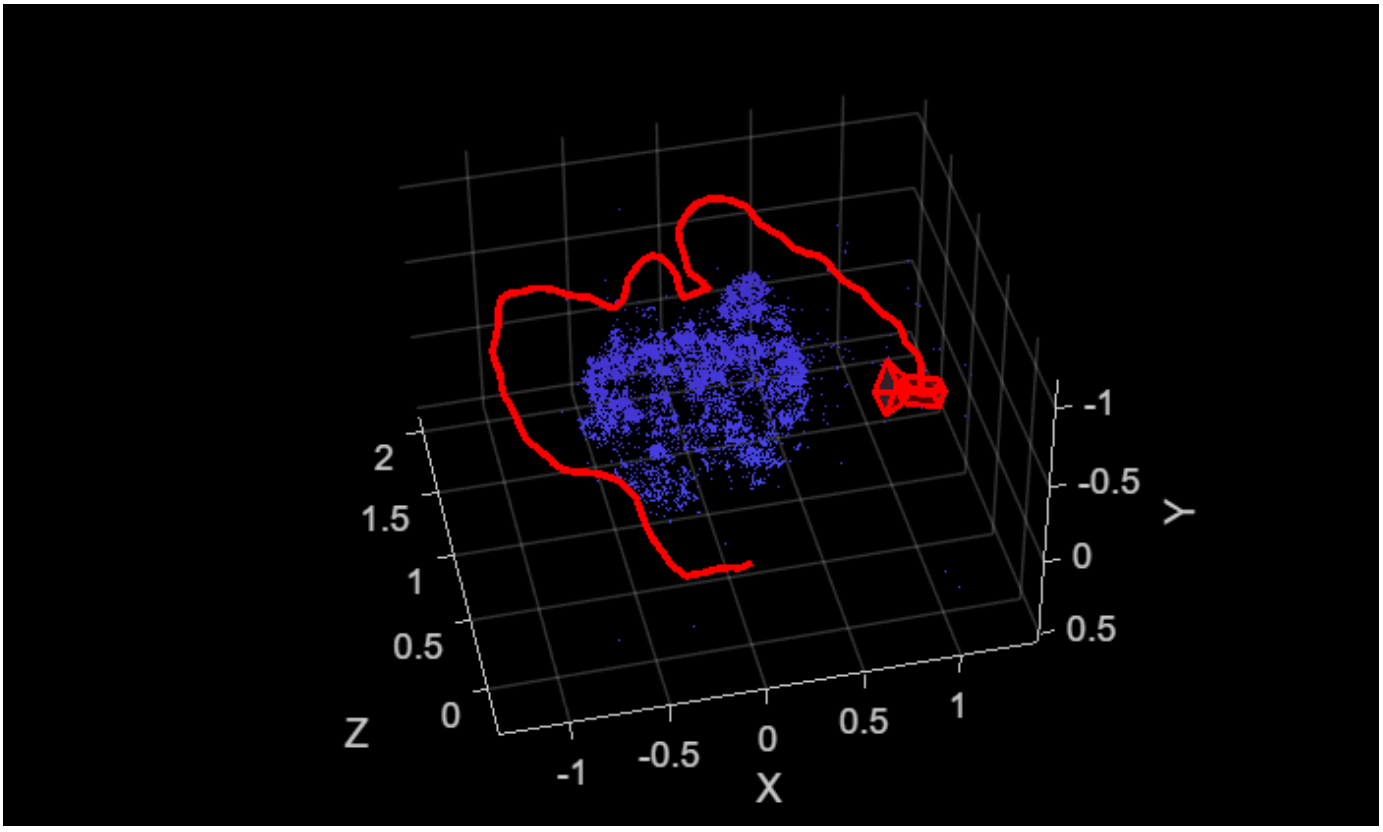
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

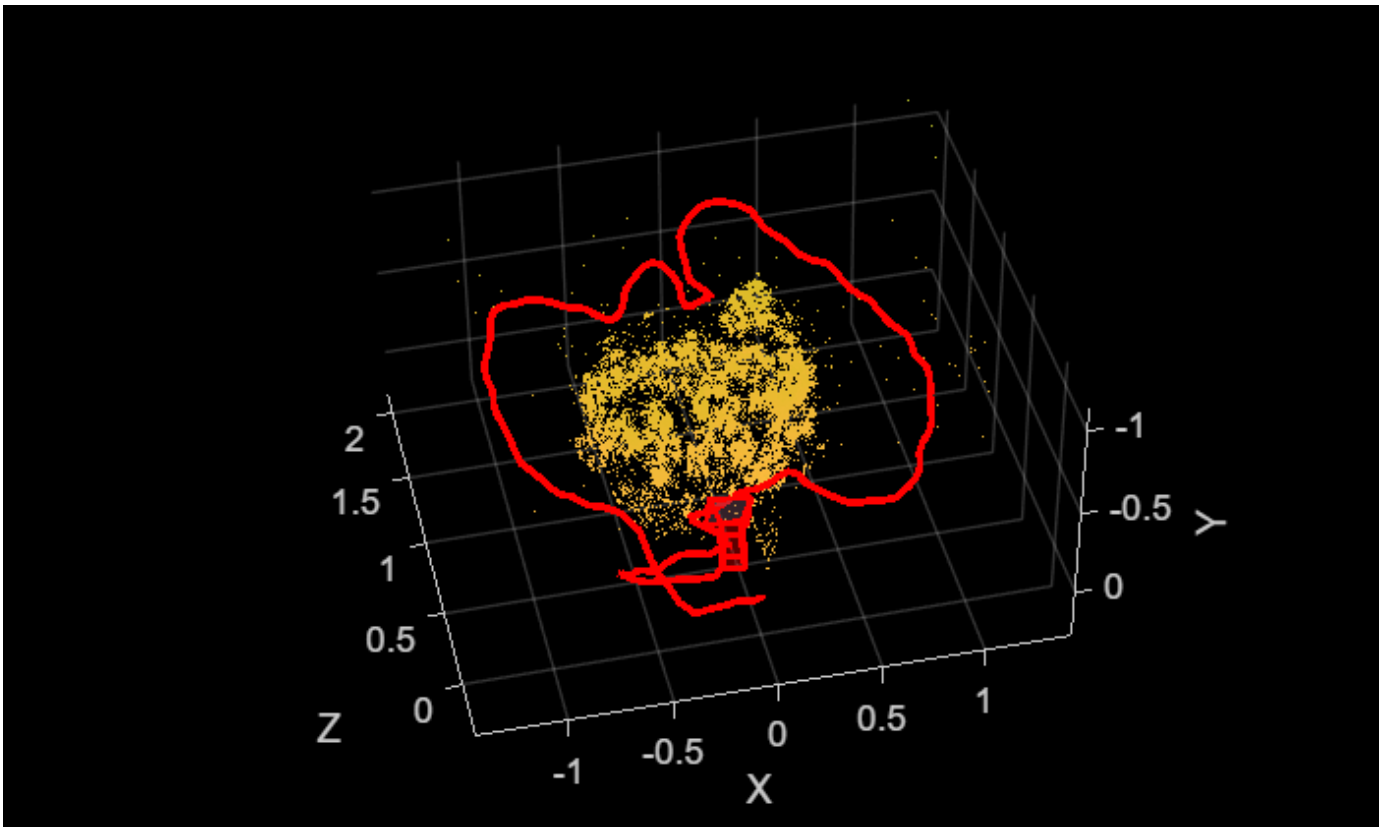
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

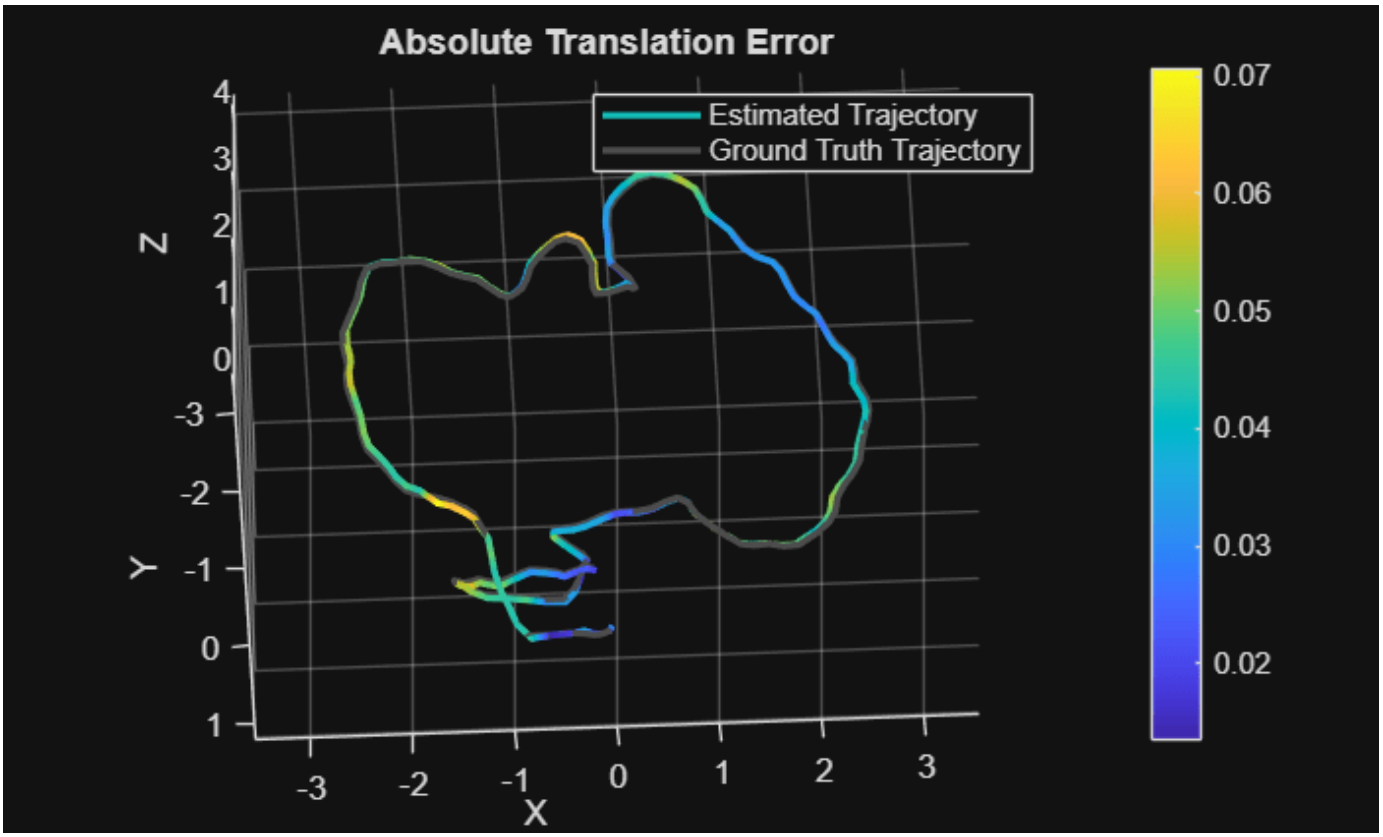
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

`I` — Image
grayscale image | RGB image

Image, specified as a grayscale or RGB image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`gyro` — Gyroscope IMU measurement
N-by-3 matrices

Gyroscope IMU measurement, specified by *N*-by-3 matrices. *N* specifies the number of IMU measurements between the previous and current camera frames. Each row of the matrix specifies a gyroscope measurement of the form *[gx, gy, gz]*.

`accel` — Acceleration IMU measurement
N-by-3 matrices

Acceleration IMU measurement, specified by *N*-by-3 matrices. *N* specifies the number of IMU measurements between the previous and current camera frames. Each row of the matrix specifies an acceleration measurement of the form *[ax, ay, az]*.

Version History

Introduced in R2023b

R2024b: Input arguments added for IMU measurements

Specify IMU measurements, gyro and acceleration, for visual-inertial fusion.

See Also

Objects
`monovslam`

Functions
`checkStatus` | `isDone` | `mapPoints` | `poses` | `plot` | `reset` | `hasNewKeyFrame`

hasNewKeyFrame

Check if new key frame added in visual SLAM object

Syntax

```
hasAdded = hasNewKeyFrame(vslam)
```

Description

`hasAdded = hasNewKeyFrame(vslam)` returns a logical value that indicates if a new key frame has been added within the visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_offi
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

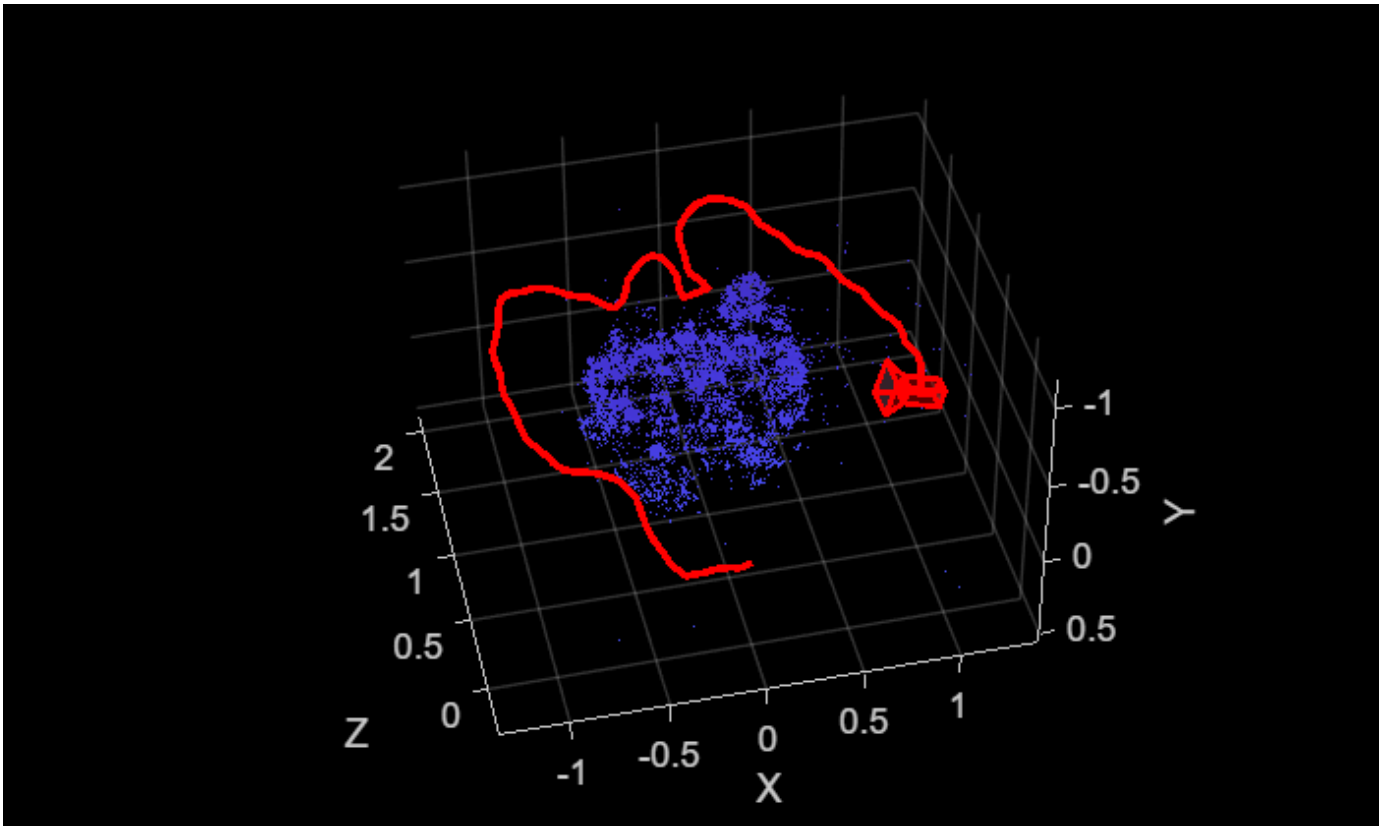
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

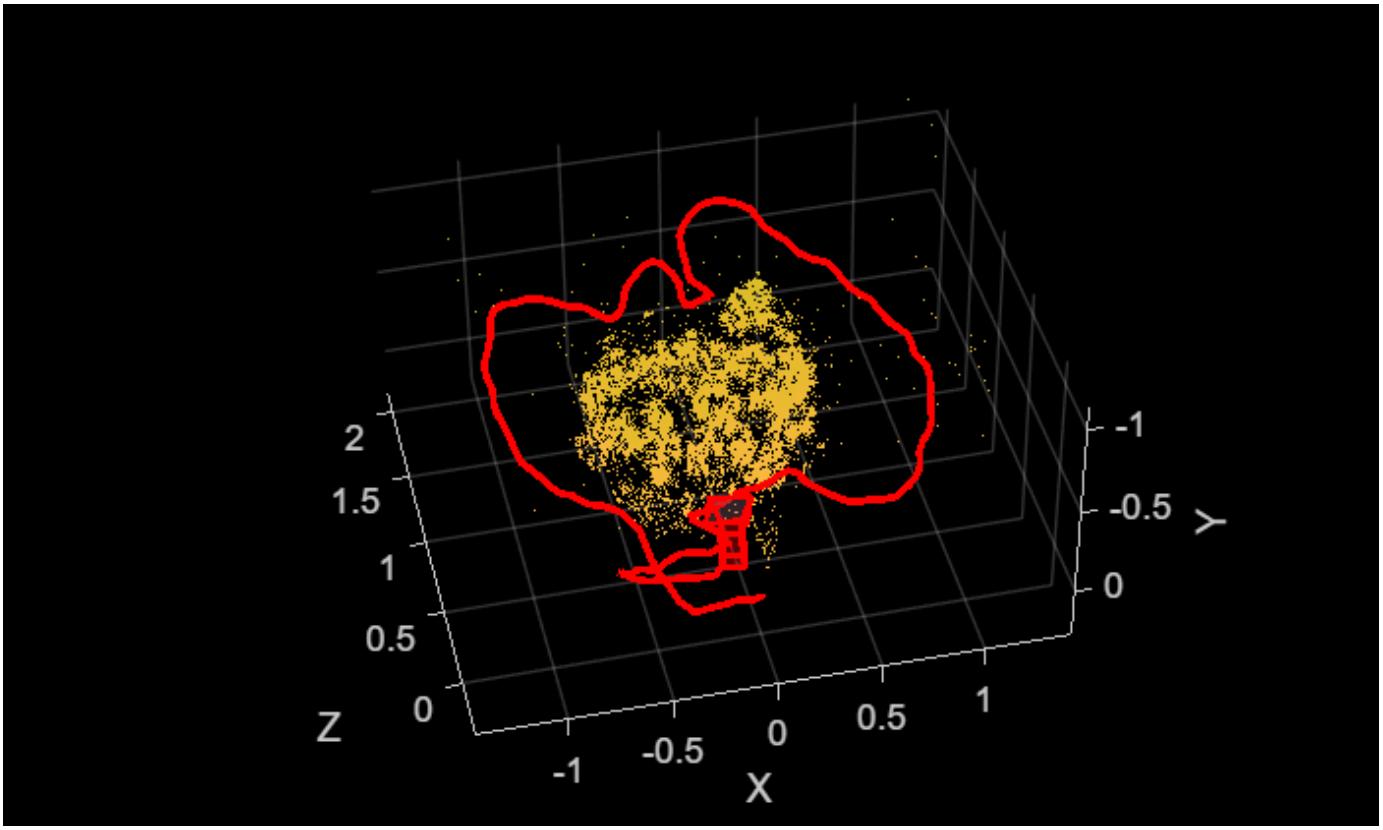
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

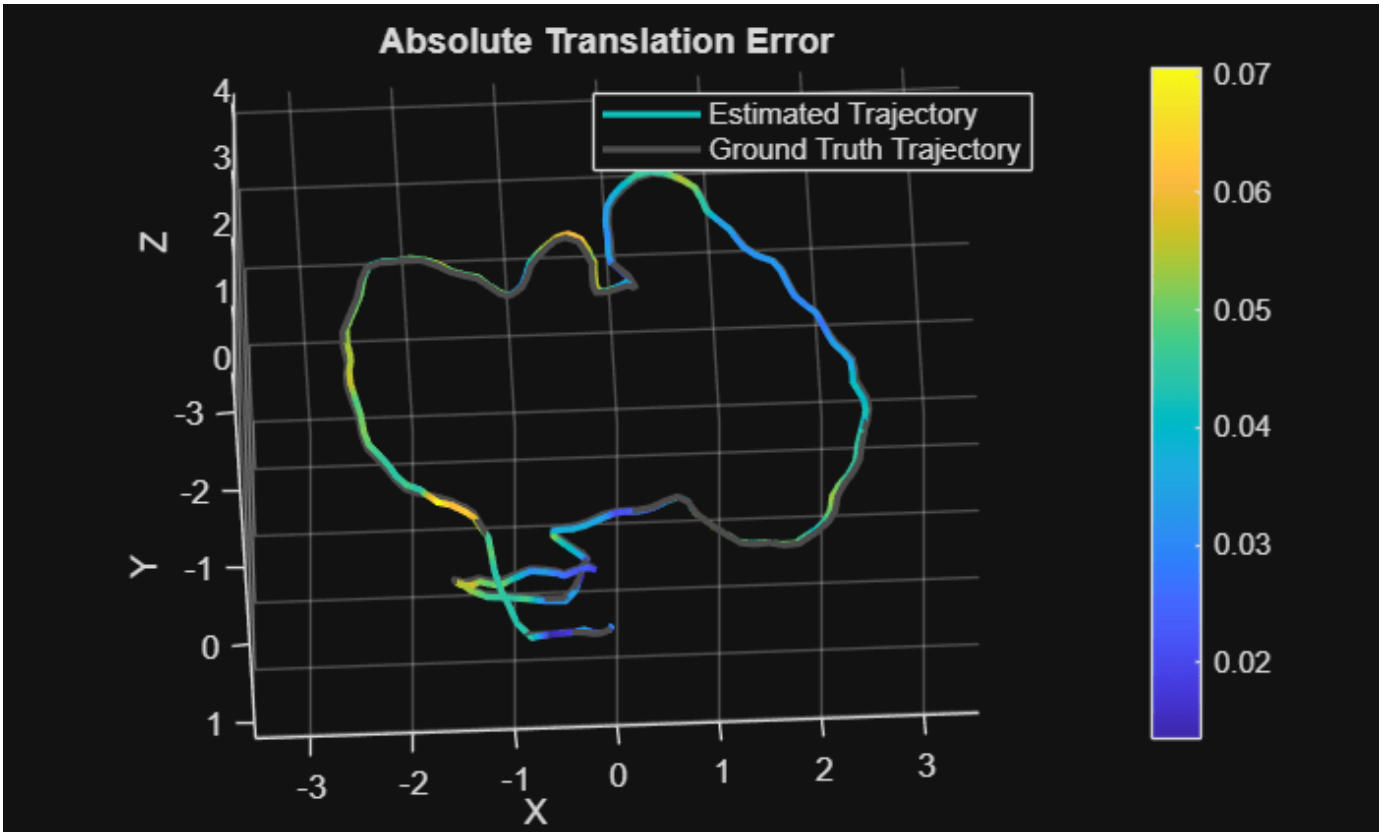
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Output Arguments

`hasAdded` — New key frame status
`false` or `0` | `true` or `1`

New key frame status, returned as `0` (`false`) or `1` (`true`). A value of `true` indicates that a new key frame has been added within the visual SLAM object `vslam`. A value of `false` indicates that no new key frame has been added. The object considers an image frame to be a key frame if the number of tracked feature points in the current frame satisfies one of these conditions:

- The number of tracked feature points is within the limits specified by the `TrackFeatureRange` property of the `monovslam` object, `vslam`.
- The number of tracked feature points is greater than the upper limit specified by the `TrackFeatureRange` property of the `monovslam` object, `vslam`, and the number of skipped frames since adding the last key frame equals the value of the `SkipMaxFrames`.

The `hasNewKeyFrame` object function returns `true` once per new key frame added. For example, if the `monovslam` object adds only one key frame, then two consecutive calls to `hasNewKeyFrame` return `true` and `false`, respectively.

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`addFrame` | `checkStatus` | `isDone` | `mapPoints` | `poses` | `plot` | `reset`

checkStatus

Check status of visual SLAM object

Syntax

```
status = checkStatus(vslam)
```

Description

`status = checkStatus(vslam)` returns the current status of the visual SLAM object. The frame the object is currently processing might be different than the most recently added frame.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_offi
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

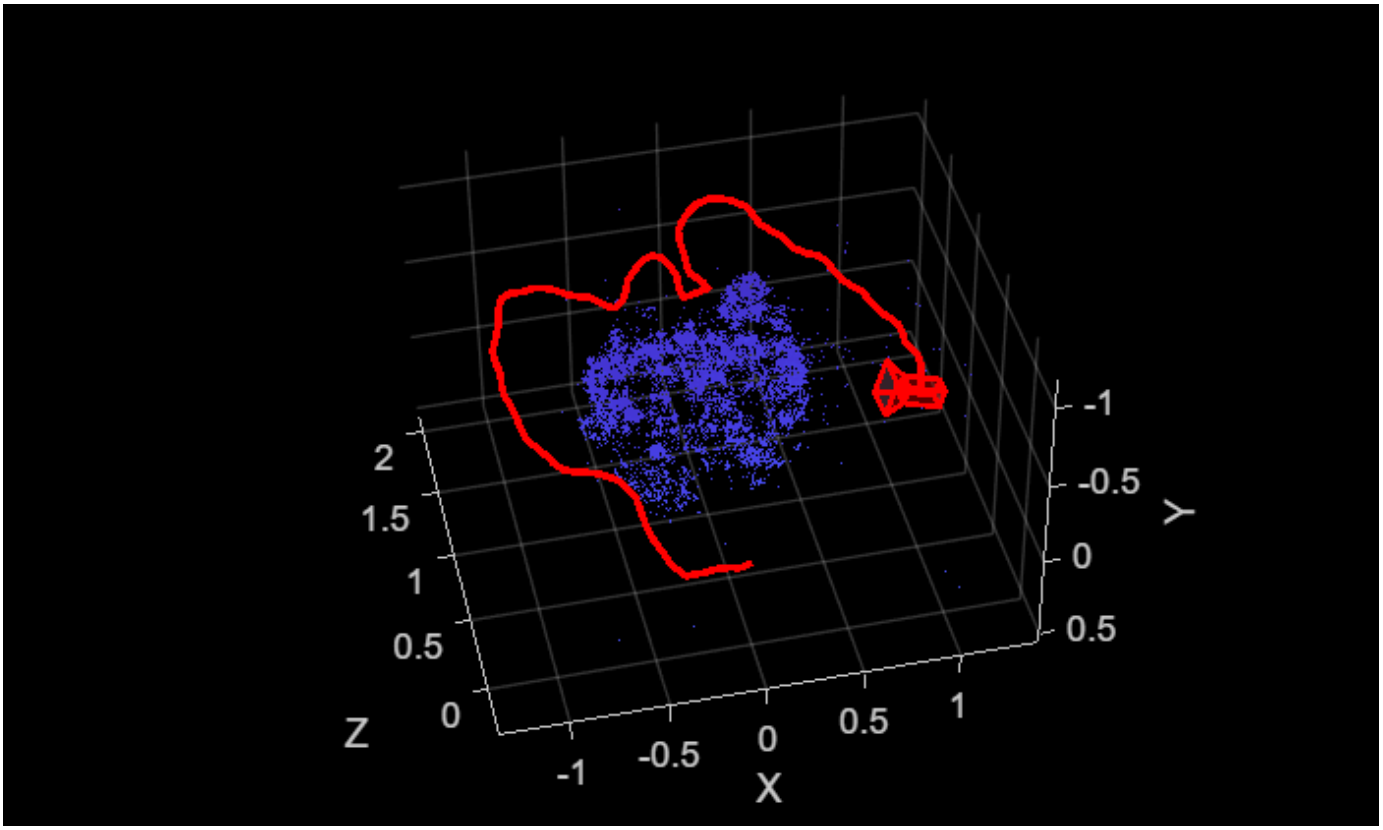
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

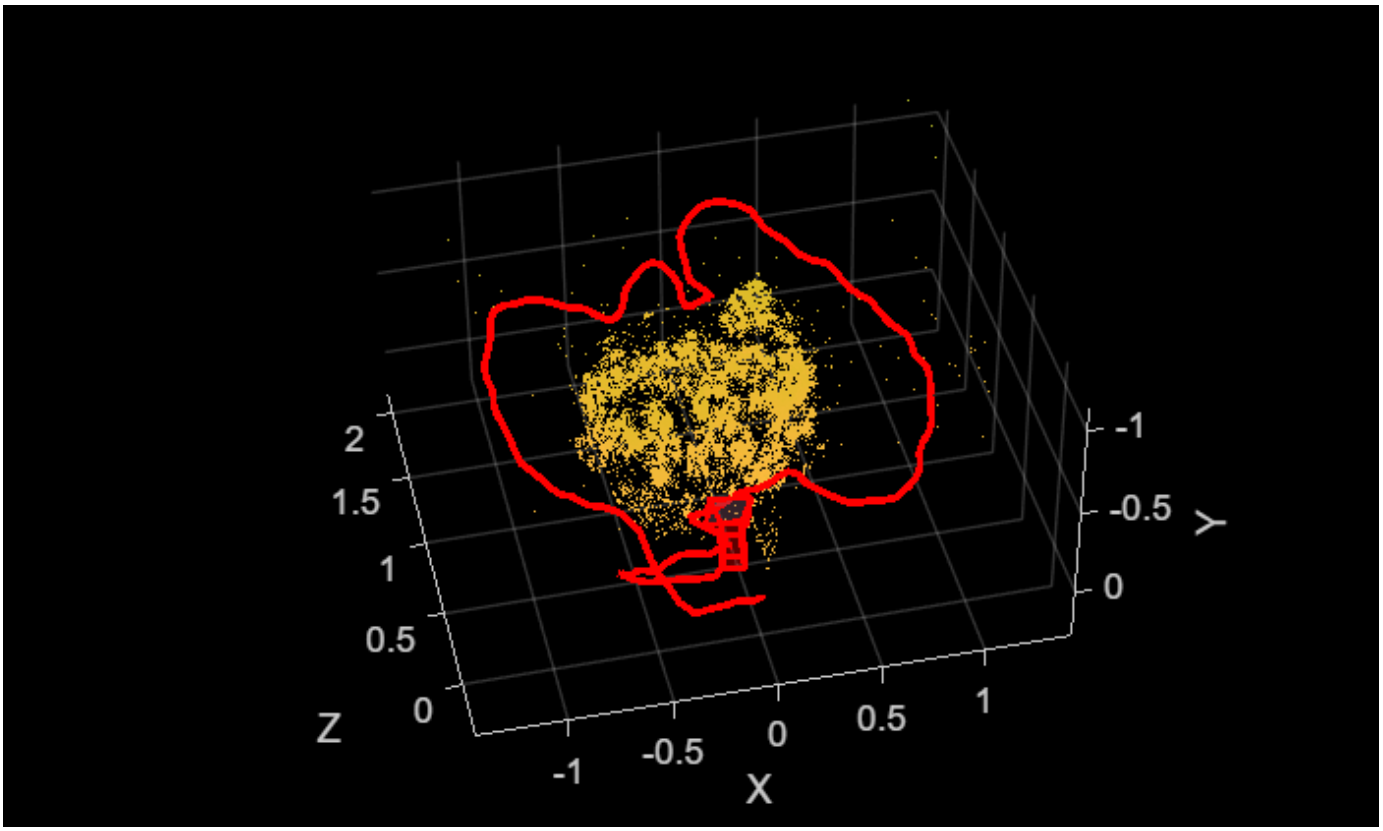
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

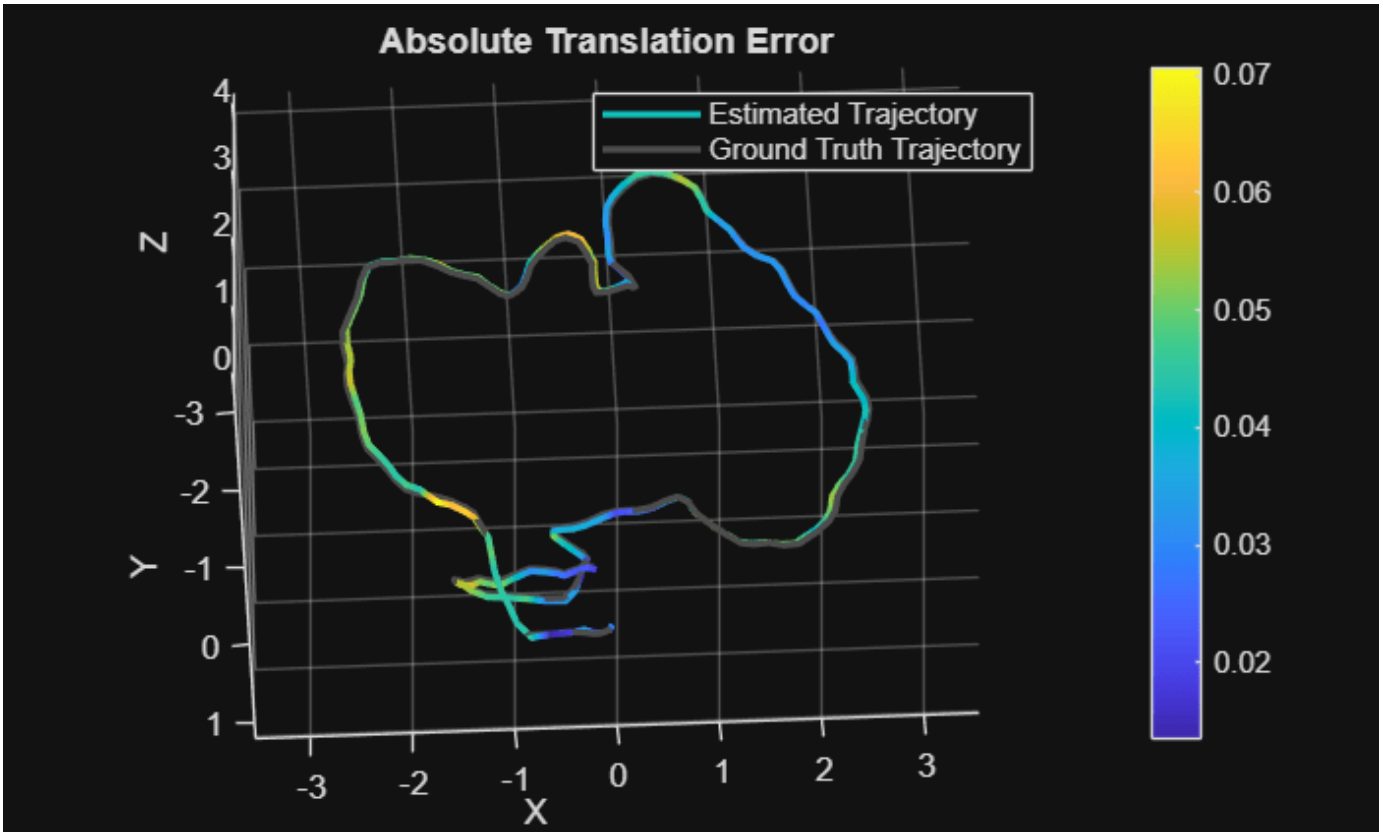
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object

`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Output Arguments

`status` — Current status of visual SLAM object

`TrackingLost` enumeration | `TrackingSuccessful` enumeration | `FrequentKeyFrames` enumeration

Current status of the visual SLAM object, returned as a `TrackingLost`, `TrackingSuccessful`, or `FrequentKeyFrames` enumeration. This table describes these enumerations.

Enumeration Value	Numeric Value	Description
`TrackingLost`	`uint8(0)`	Tracking is lost. The number of tracked feature points in the frame currently being processed is less than the lower limit of the `TrackFeatureRange` property. This indicates the image does not contain enough features, or that the camera is moving too fast. To improve the tracking, you can increase the *upperLimit* value of the `TrackFeatureRange` argument and decrease the `SkipMaxFrames` argument to add key frames more frequently.
`TrackingSuccessful`	`uint8(1)`	Tracking is successful. The number of tracked feature points in the frame currently being processed is between the lower limit and upper limit values of the `TrackFeatureRange` property.
`FrequentKeyFrames`	`uint8(2)`	Tracking adds key frames too frequently. The number of tracked feature points in the frame currently being processed is greater than the upper limit of the `TrackFeatureRange` property.

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`addFrame` | `isDone` | `mapPoints` | `poses` | `plot` | `reset` | `hasNewKeyFrame`

isDone

End-of-processing status for visual SLAM object

Syntax

```
done = isDone(vslam)
```

Description

`done = isDone(vslam)` returns a logical value indicating whether the visual SLAM object has finished processing its frames. A value of `true` indicates the object is in an idle state. A value of `false` indicates the object is processing one or more frames.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_offi
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

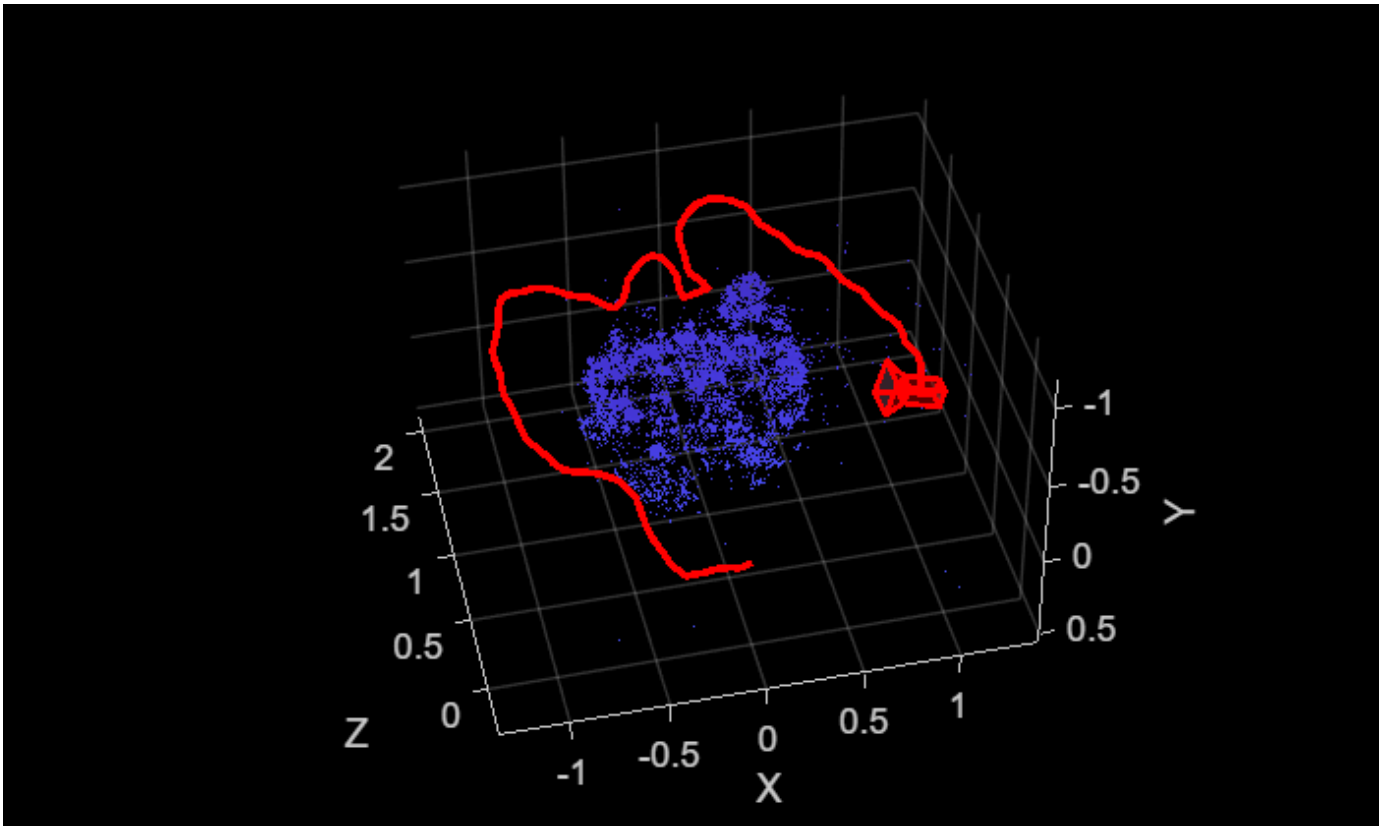
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

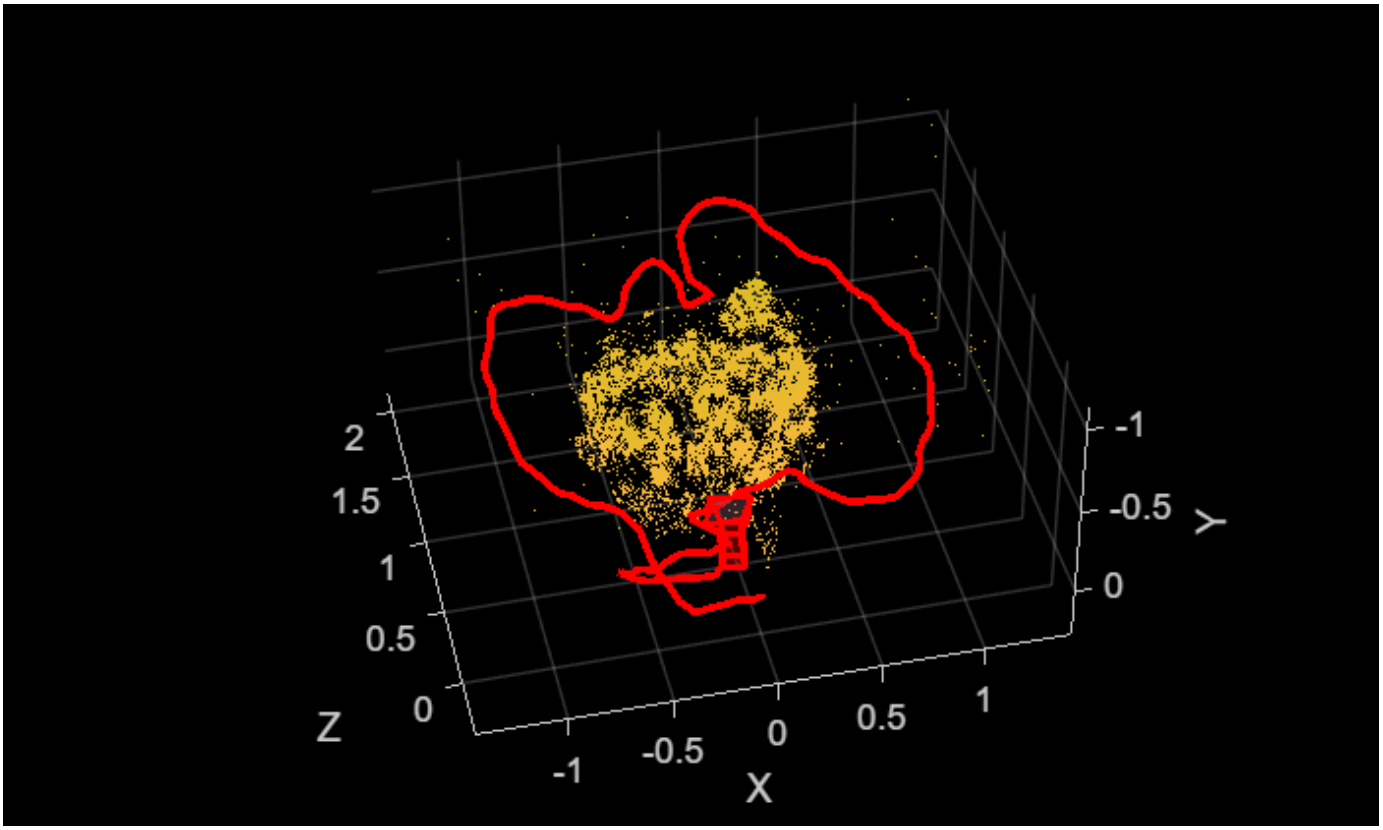
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

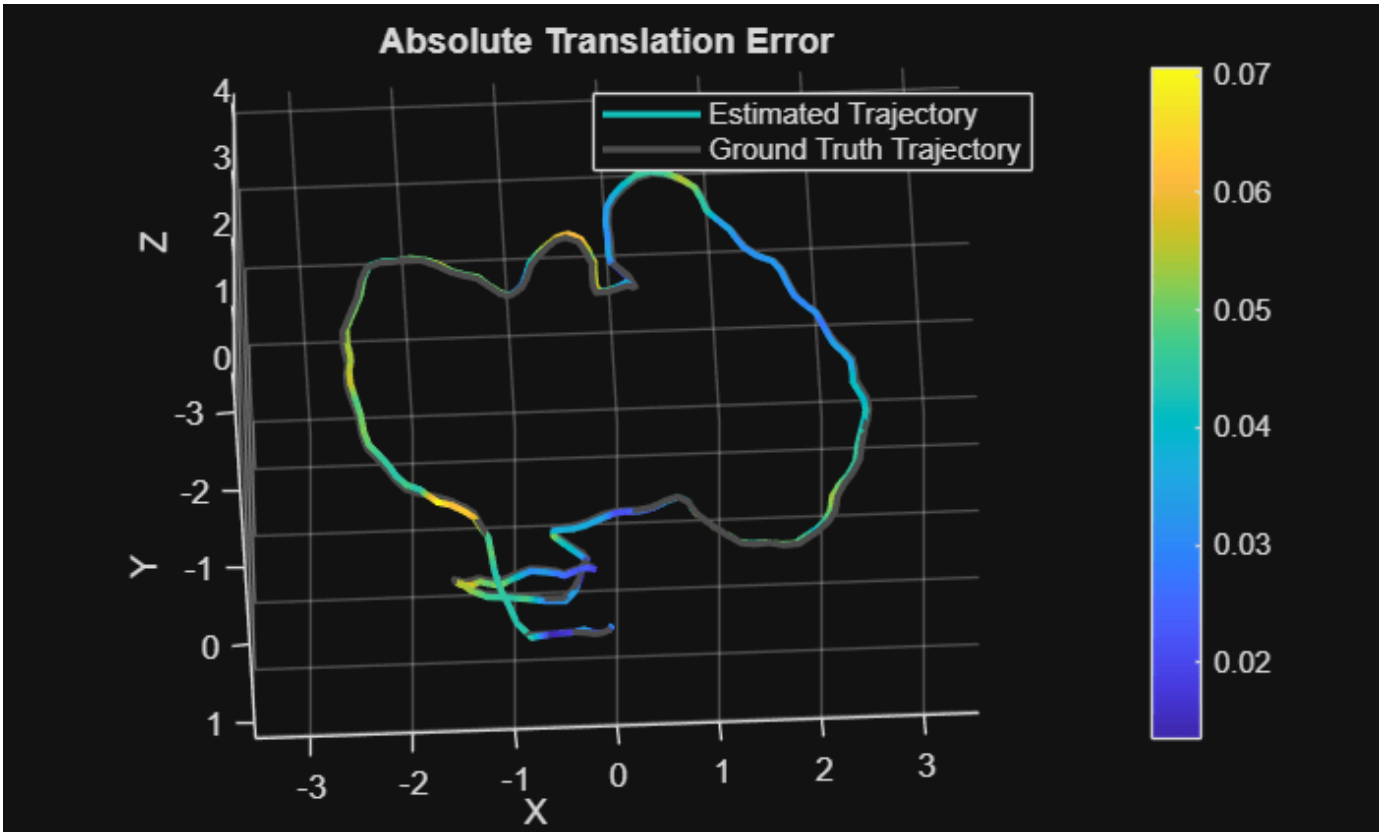
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`addFrame` | `mapPoints` | `poses` | `plot` | `reset` | `hasNewKeyFrame` | `checkStatus`

mapPoints

Build 3-D map of world points

Syntax

```
xyzPoints = mapPoints(vslam)
```

Description

`xyzPoints = mapPoints(vslam)` builds a sparse 3-D map of world points, `xyzPoints`, from the input images in the visual simultaneous localization and mapping (vSLAM) object `vslam`.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_offi
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

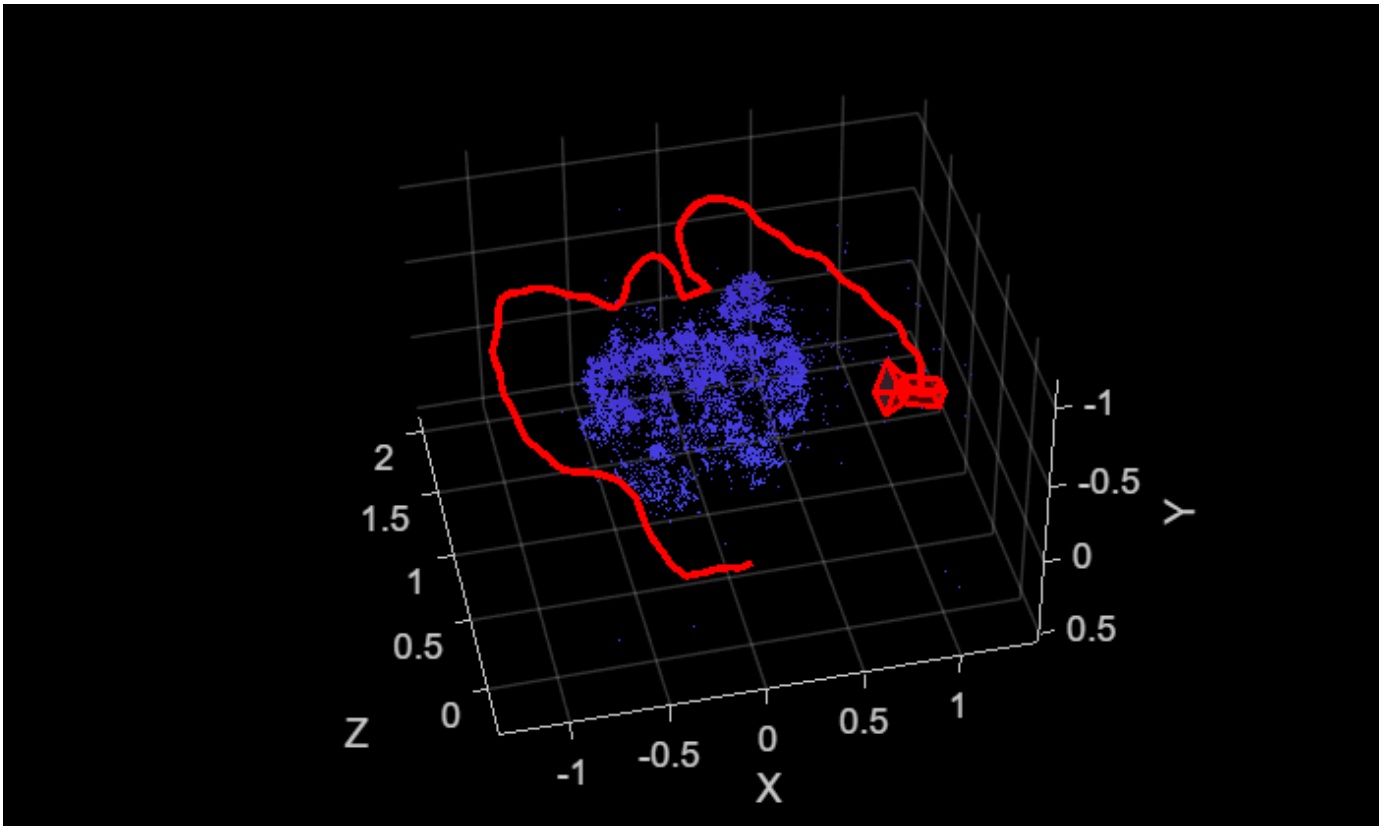
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

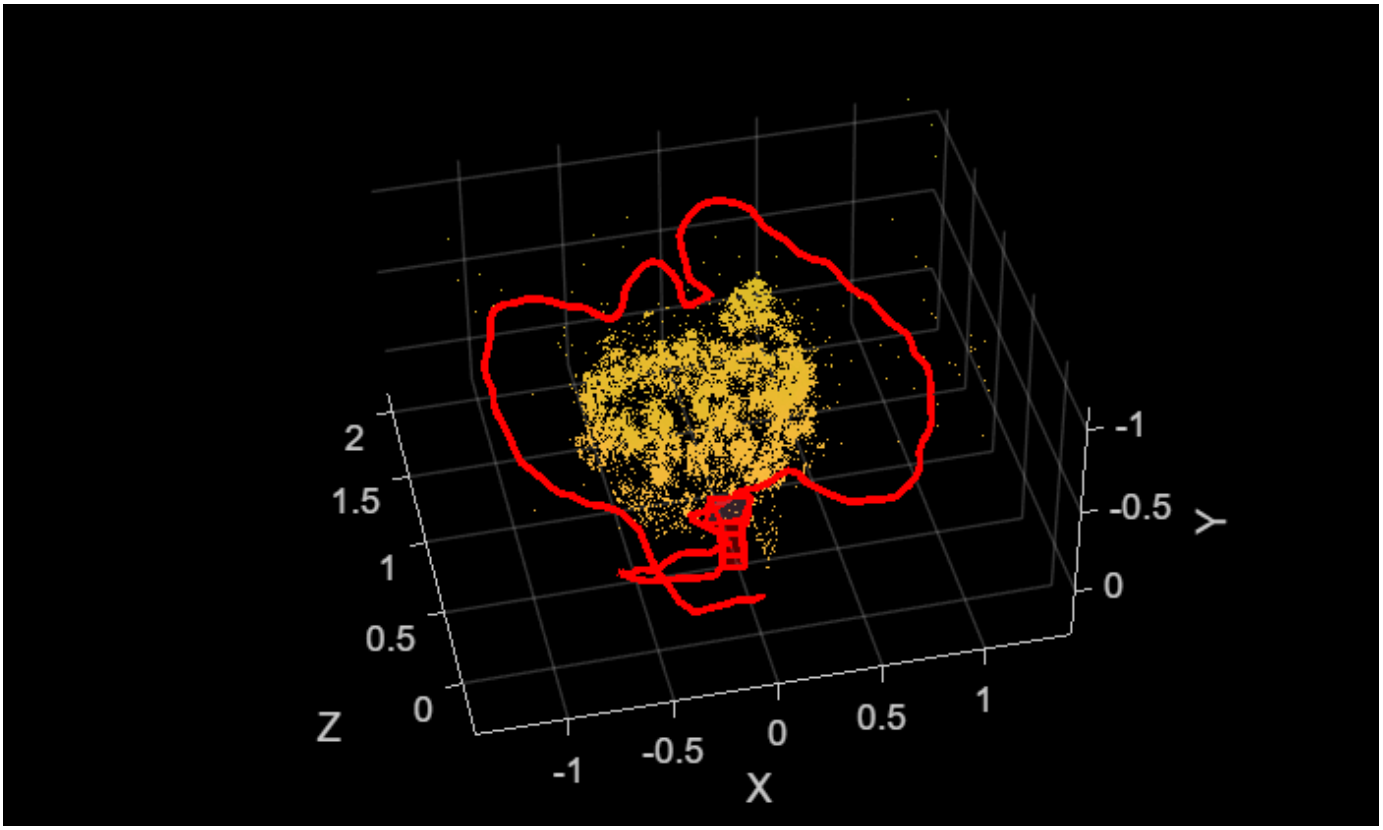
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

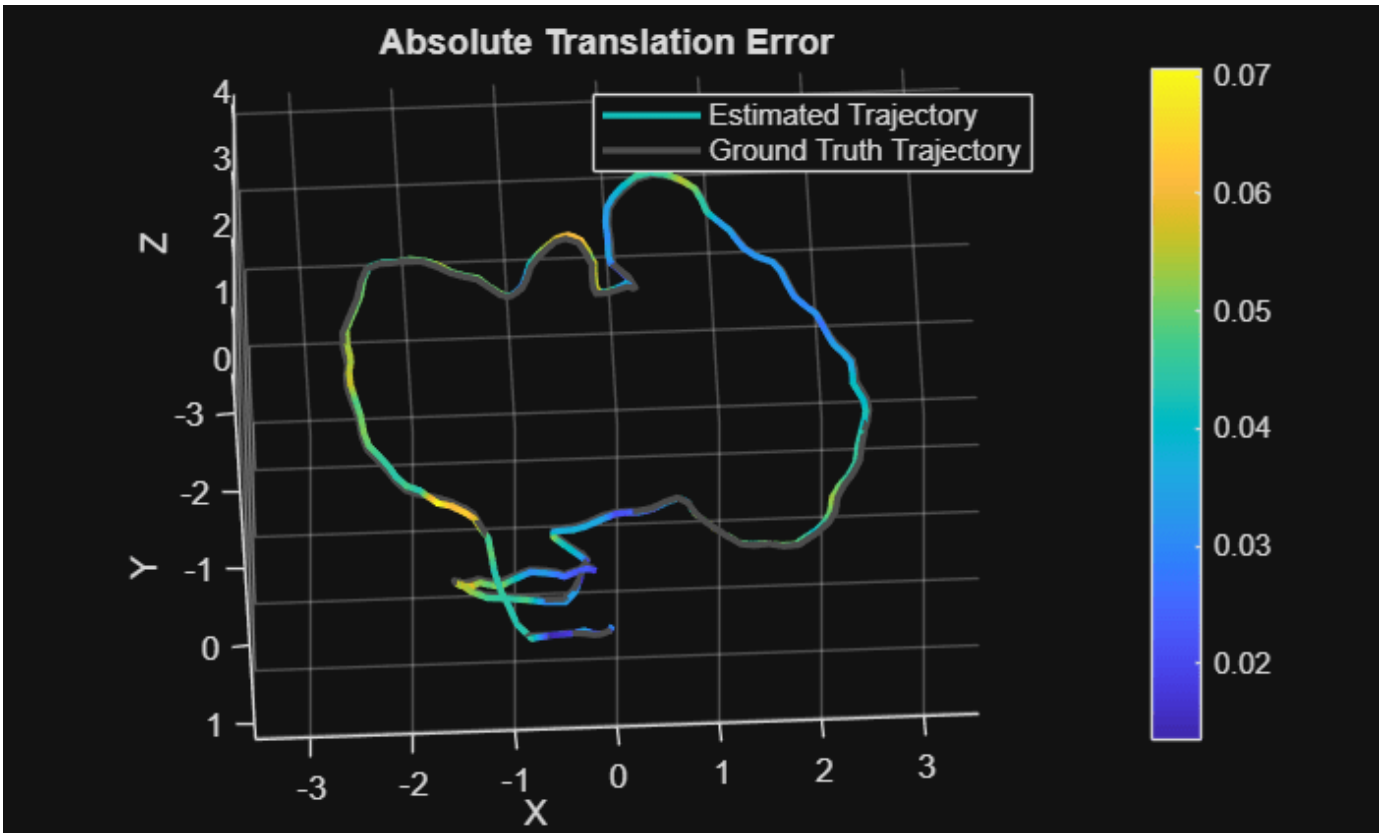
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Output Arguments

`xyzPoints` — 3-D map of world points
M-by-3 matrix

3-D map of world points, returned as an *M*-by-3 matrix. *M* is the number of points, and each point is specified in the form [*X Y Z*].

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
checkStatus | isDone | addFrame | poses | plot | reset | hasNewKeyFrame

poses

Absolute camera poses of key frames

Syntax

```
camPoses = poses(vslam)
[camPoses,keyFrameIDs] = poses(vslam)
```

Description

`camPoses = poses(vslam)` returns the absolute camera poses `camPoses` of the key frames from the visual simultaneous localization and mapping (vSLAM) object `vslam`.

`[camPoses,keyFrameIDs] = poses(vslam)` returns the IDs `keyFrameIDs` of the key frames corresponding to the camera poses. The IDs represent the order in which the image frames have been added to the `vslam` object by the `addFrame` object function.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_off
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

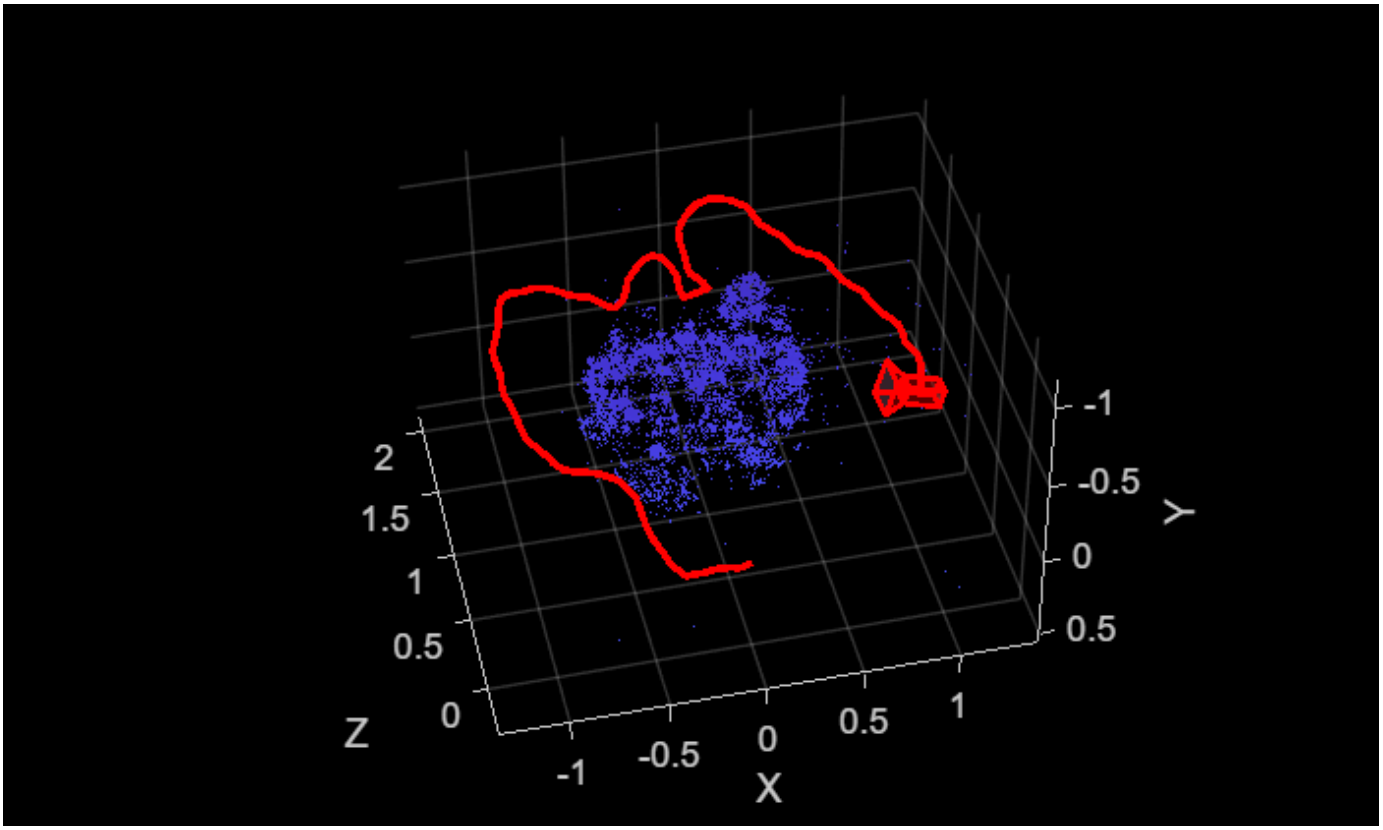
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

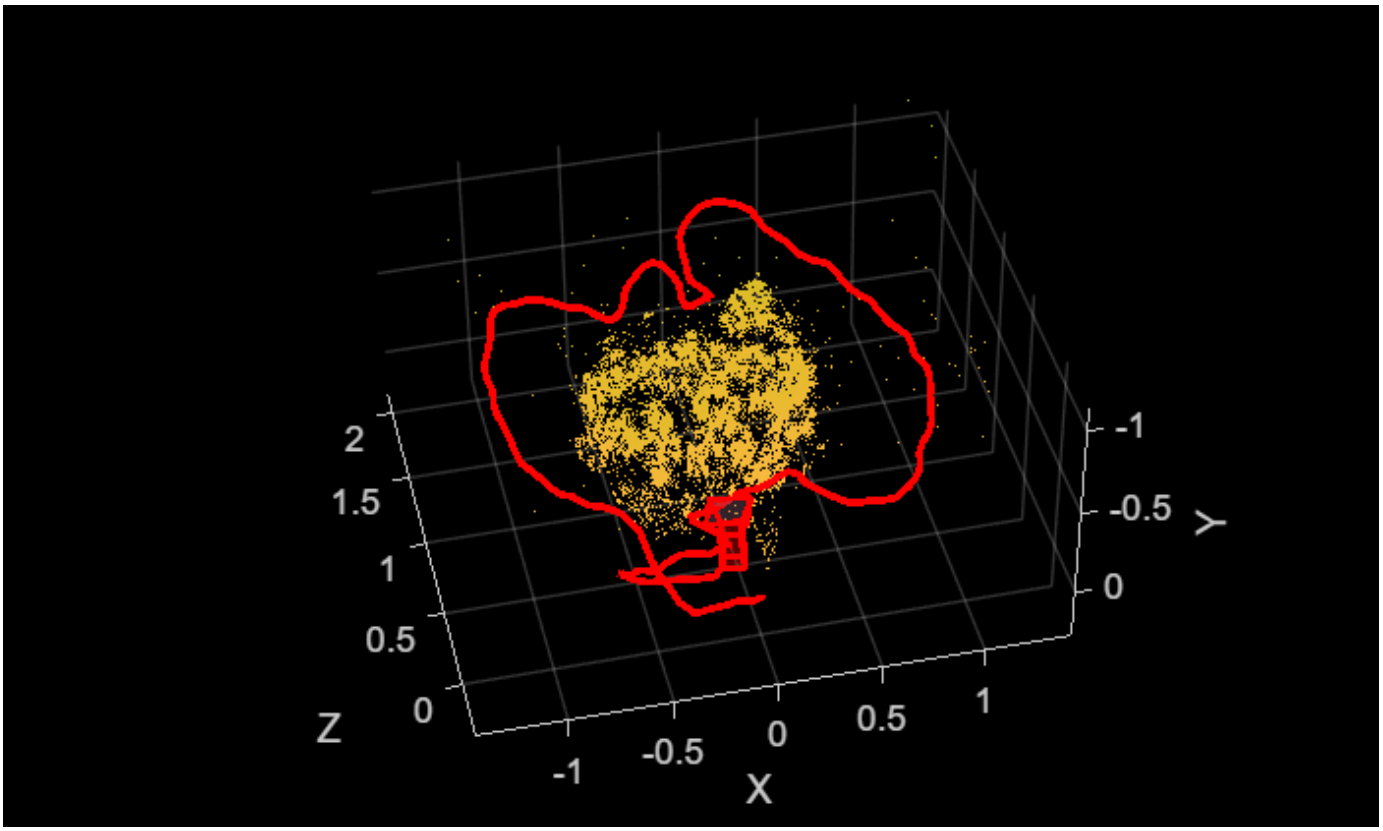
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

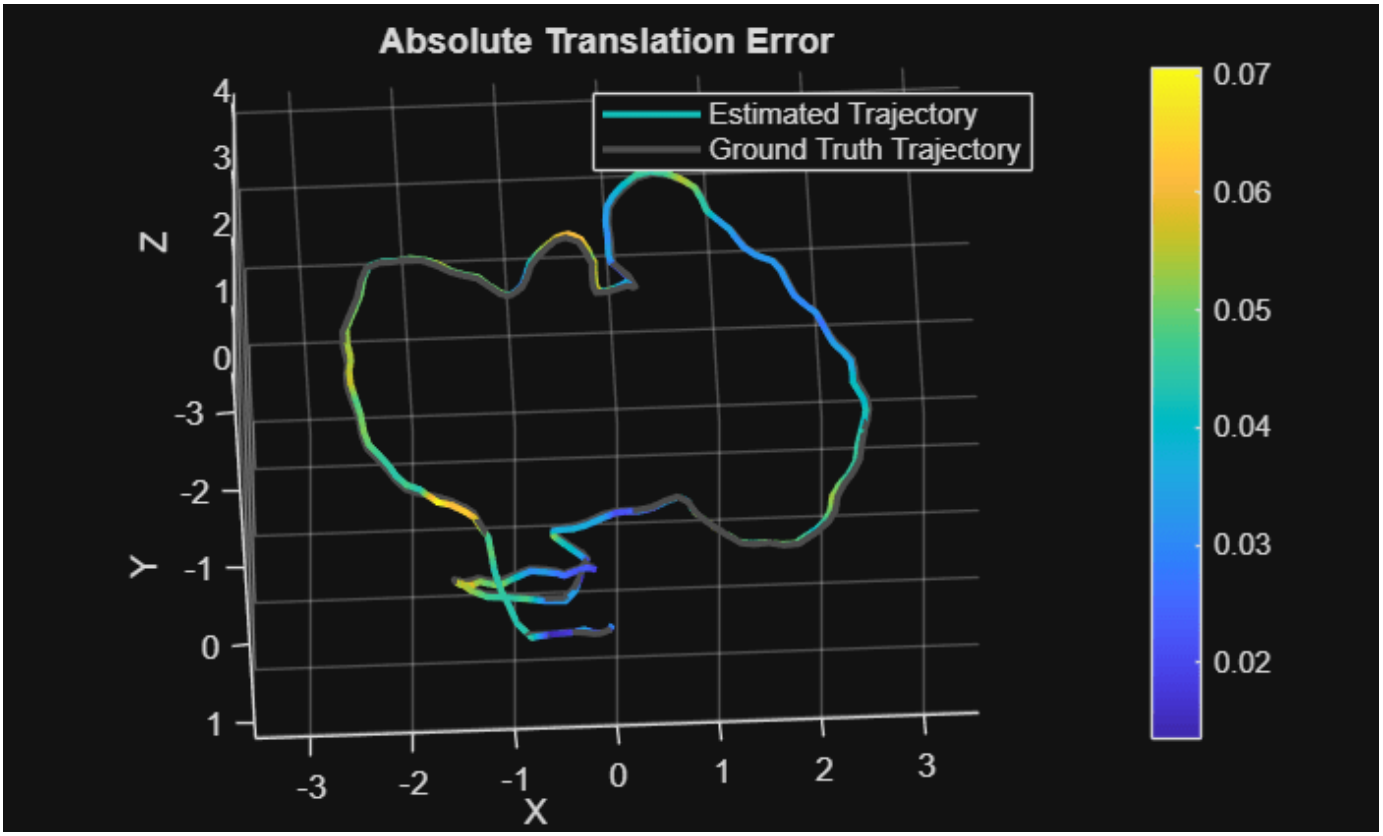
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Output Arguments

`camPoses` — Absolute camera poses of key frames
M-element array of `rigidtform3d` objects

Absolute camera poses of key frames, returned as an *M*-element array of `rigidtform3d` objects.

`keyFrameIDs` — Key frame IDs for absolute camera poses
M-element array of integers

Key frame IDs for the absolute camera poses, returned as an *M*-element array of integers. Each element specifies the key frame ID for the camera pose in the corresponding element of `camPoses`. The IDs represent the order in which the image frames have been added to the `vslam` object by the `addFrame` object function.

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`checkStatus` | `isDone` | `addFrame` | `mapPoints` | `plot` | `reset` | `hasNewKeyFrame`

plot

Plot 3-D map points and estimated camera trajectory in visual SLAM

Syntax

```
plot(vslam)
plot(vslam,Name=Value)
ax = plot( ___ )
```

Description

`plot(vslam)` plots the 3-D map points and estimated camera trajectory from the visual SLAM object `vslam`.

`plot(vslam,Name=Value)` specifies options using one or more name-value arguments. For example, `MarkerSize=10` sets the diameter of the marker size to 10 points.

`ax = plot( ___ )` returns the axes handle for the 3-D mapped points of the plot, using any combination of input arguments from previous syntaxes.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_off
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

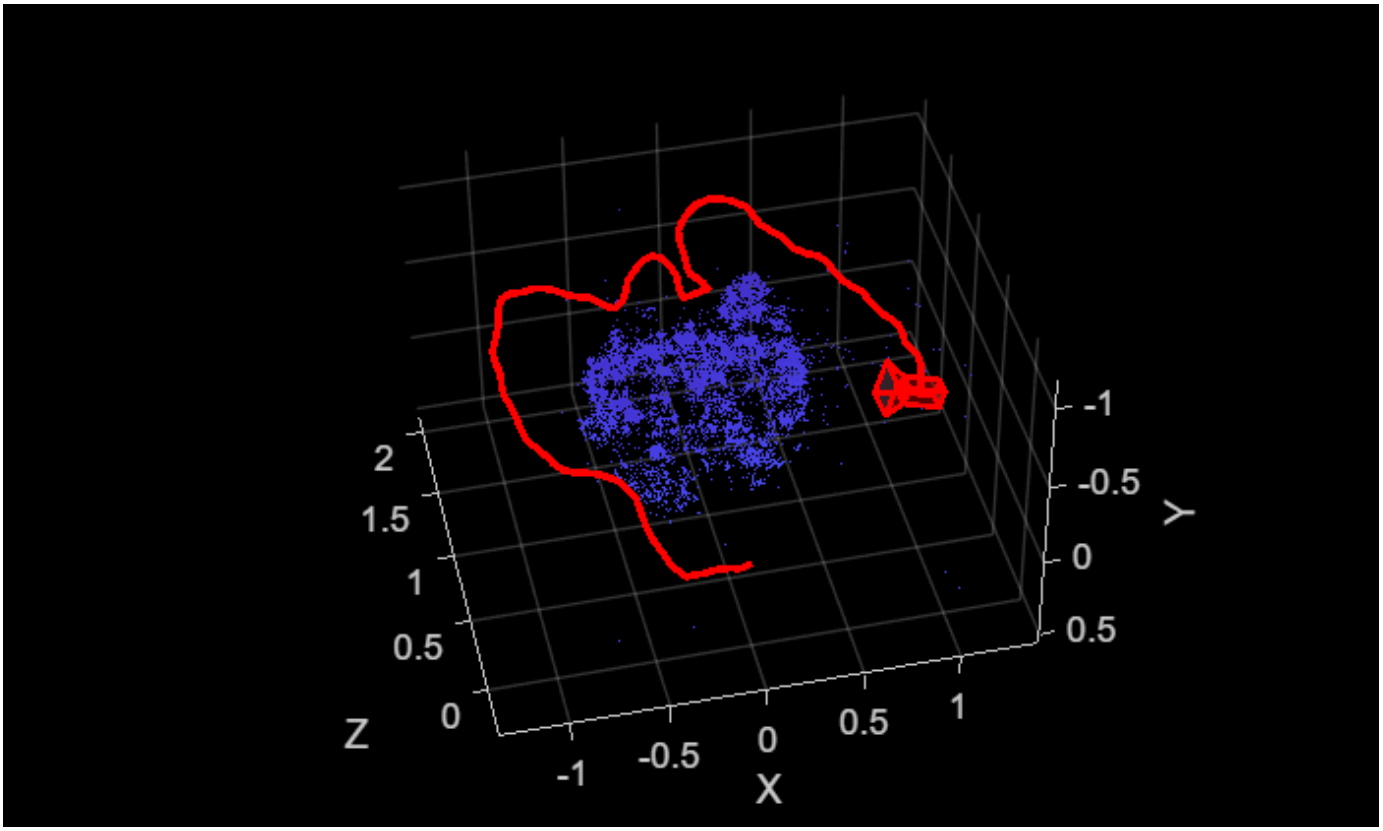
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))

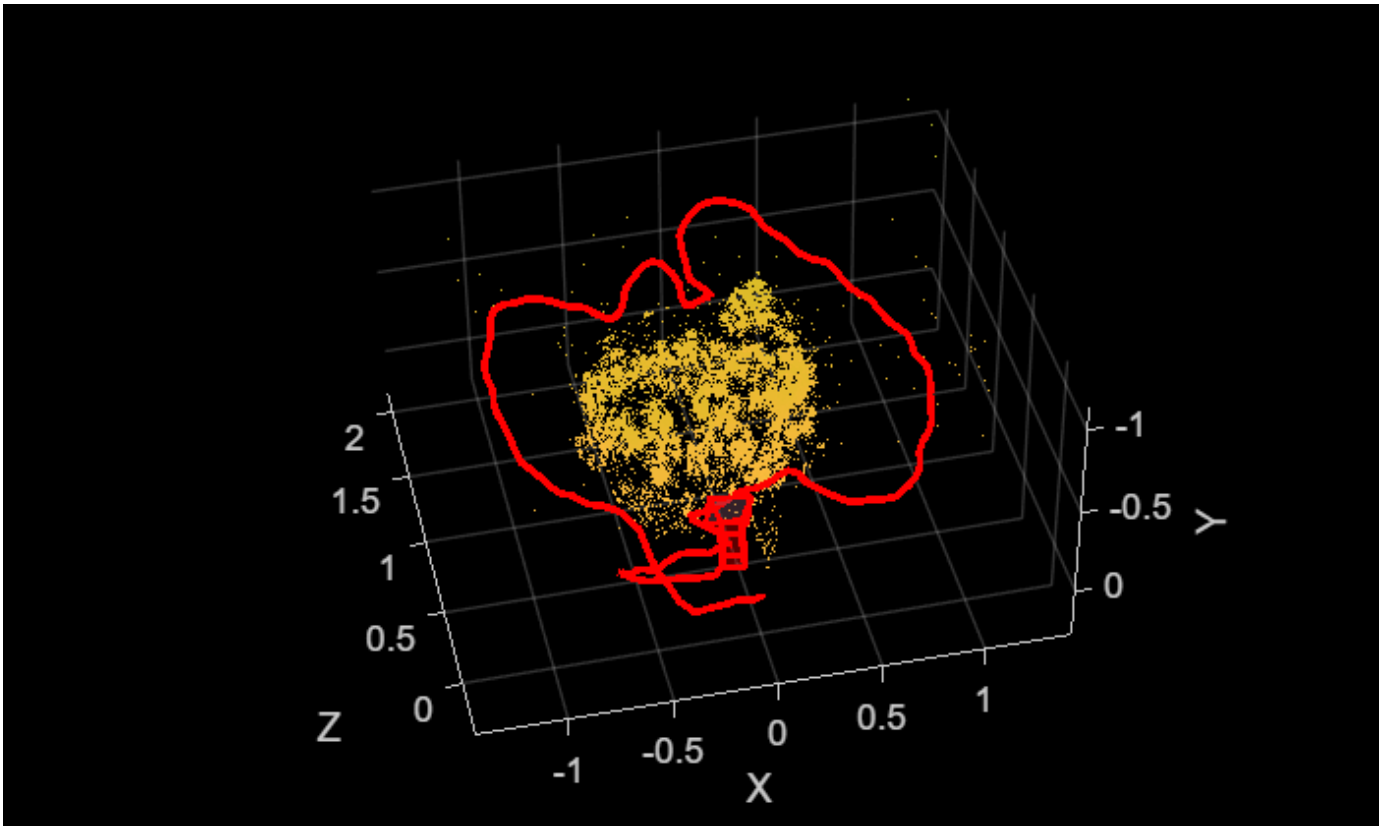
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

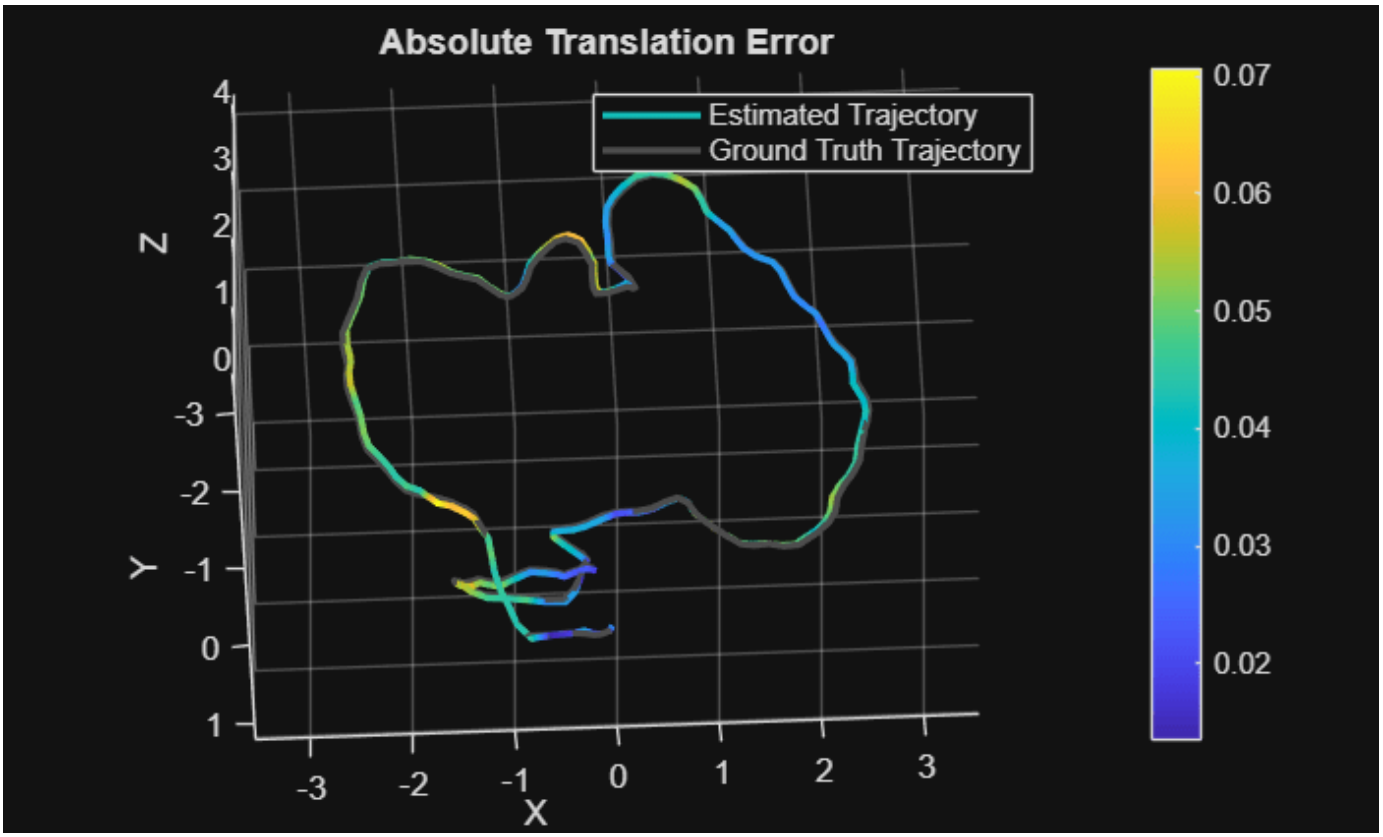
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object

`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(vslam,MarkerSize=10)`, sets the diameter of the marker to 10 points.

`MarkerSize` — Diameter of markers for 3-D map points

`6` (default) | positive integer

Diameter of markers for 3-D map points, specified as a positive integer in points.

`MarkerColor` — Color of markers for 3-D map points

`[0,1,0]` (default) | RGB triplet | short or long color name

Color of markers for 3-D map points, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range `[0, 1]`

`ColorSource` — Colormap source for markers
`"Y"` (default) | `"X"` | `"Z"` | `"MarkerColor"`

Colormap source for markers, specified as `"X"`, `"Y"`, `"Z"`, or `"MarkerColor"`.

`CameraSize` — Width of camera base
`0.1` (default) | scalar

Width of the camera base, specified as a scalar in the data units of the axes.

`CameraColor` — Color of camera
`[1,0,0]` (default) | RGB triplet | short or long color name

Color of the camera, specified as an RGB triplet or a short or long color name. Each value within an RGB triplet must be in the range [`0`, `1`].

`Parent` — Axes for visualization
`Axes` object | `UIAxes` object

Axes for visualization, specified as an `Axes` object or a `UIAxes` object. Use this name-value argument to visualize the plot in a UI for which you have specified `Figure` and `Axes` properties.

Output Arguments

`ax` — Axes handle
`Axes` graphics object

Axes handle, returned as an `axes` graphics object

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`checkStatus` | `isDone` | `addFrame` | `poses` | `mapPoints` | `reset` | `hasNewKeyFrame`

reset

Reset visual SLAM object

Syntax

`reset(vslam)`

Description

`reset(vslam)` resets all internal processes and clears all data in the object.

Examples

Monocular Visual SLAM Using TUM RGB-D Data Set

Perform monocular visual simultaneous localization and mapping (vSLAM) using the data from the TUM RGB-D Benchmark. You can download the data to a temporary directory using a web browser or by running this code:

```
baseDownloadURL = "https://cvg.cit.tum.de/rgbd/dataset/freiburg3/rgbd_dataset_freiburg3_long_offi
dataFolder = fullfile(tempdir,"tum_rgbd_dataset",filesep);
options = weboptions(Timeout=Inf);
tgzFileName = dataFolder+"fr3_office.tgz";
folderExists = exist(dataFolder,"dir");

% Create a folder in a temporary directory to save the downloaded file
if ~folderExists
    mkdir(dataFolder)
    disp("Downloading fr3_office.tgz (1.38 GB). This download can take a few minutes.")
    websave(tgzFileName,baseDownloadURL,options);

    % Extract contents of the downloaded file
    disp("Extracting fr3_office.tgz (1.38 GB) ...")
    untar(tgzFileName,dataFolder);
end
```

Create an `imageDatastore` object to store all the RGB images.

```
imageFolder = dataFolder+"rgbd_dataset_freiburg3_long_office_household/rgb/";
imds = imageDatastore(imageFolder);
```

Specify your camera intrinsic parameters, and use them to create a monocular visual SLAM object.

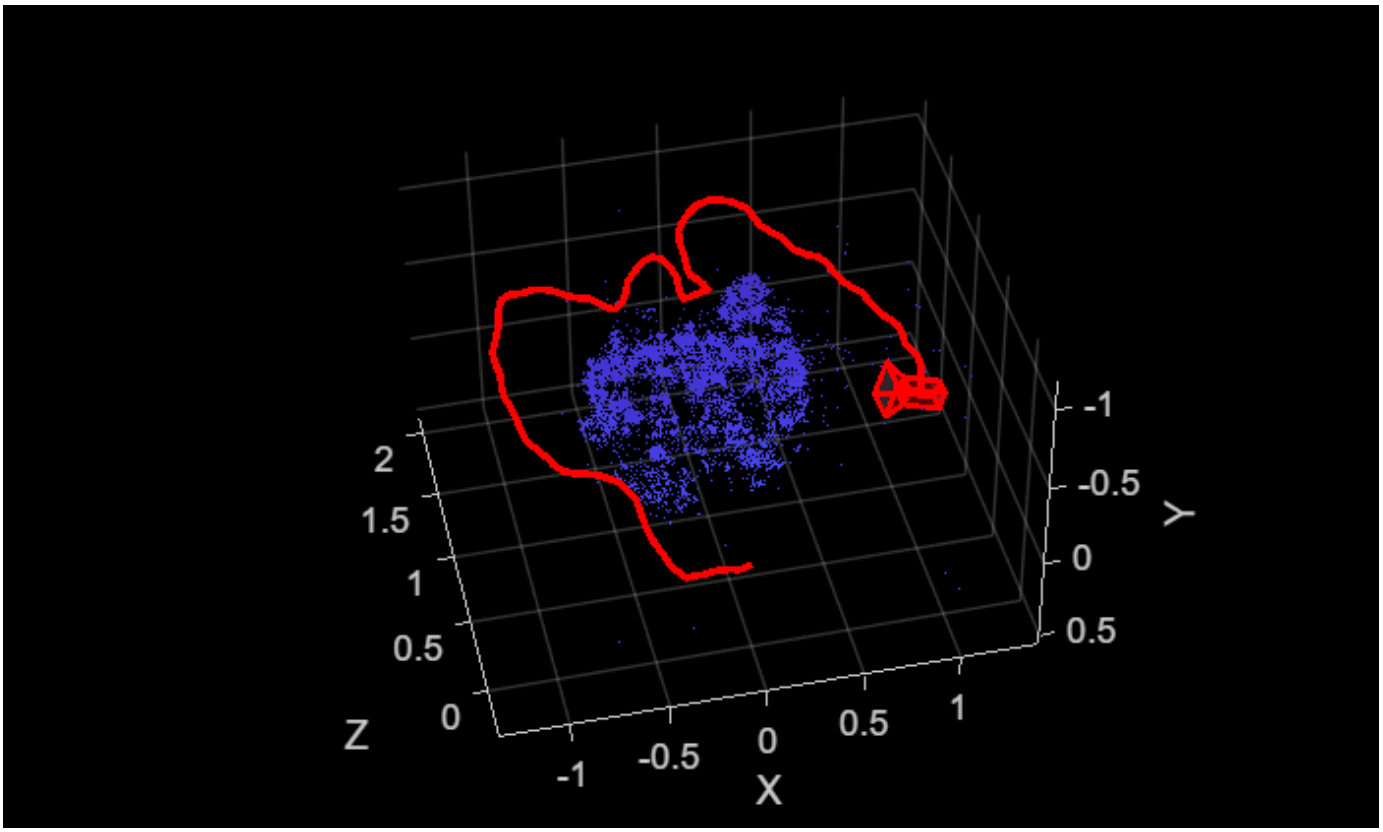
```
intrinsics = cameraIntrinsics([535.4 539.2],[320.1 247.6],[480 640]);
vslam = monovslam(intrinsics,SkipMaxFrames=15);
```

Process each image frame, and visualize the camera poses and 3-D map points. Note that the `monovslam` object runs several algorithm parts on separate threads, which can introduce a latency in processing of an image frame added by using the `addFrame` function.

```
for i = 1:numel(imds.Files)
    addFrame(vslam,readimage(imds,i))
```

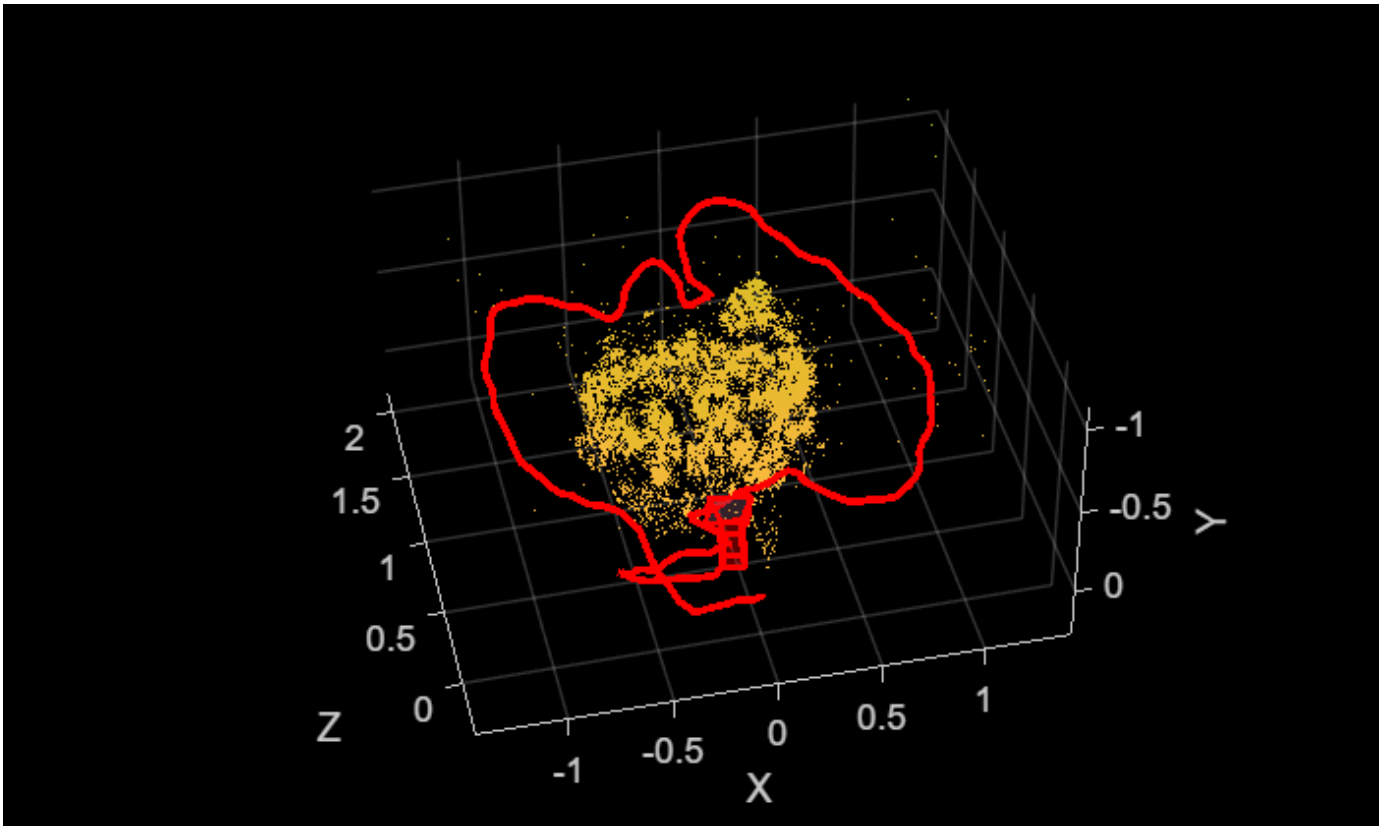
```
    if hasNewKeyFrame(vslam)
        % Display 3-D map points and camera trajectory
        plot(vslam);
    end

    % Get current status of system
    status = checkStatus(vslam);
end
```



Plot intermediate results and wait until all images are processed.

```
while ~isDone(vslam)
    if hasNewKeyFrame(vslam)
        plot(vslam);
    end
end
```



After all the images are processed, you can collect the final 3-D map points and camera poses for further analysis.

```
xyzPoints = mapPoints(vslam);
[camPoses,addedFramesIdx] = poses(vslam);

% Reset the system
reset(vslam)
```

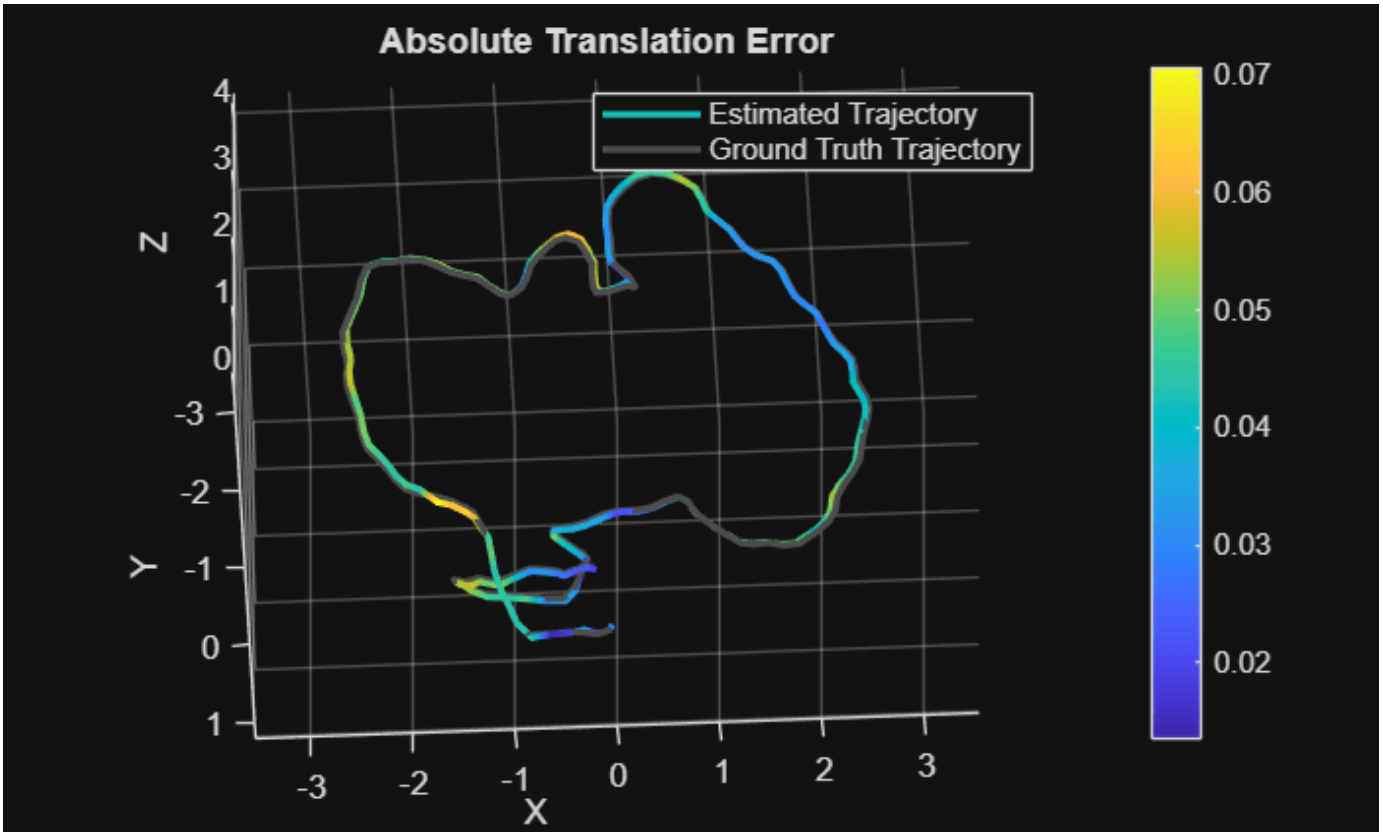
Compare the estimated camera trajectory with the ground truth to evaluate the accuracy.

```
% Load ground truth
gTruthData = load("orbslamGroundTruth.mat");
gTruth     = gTruthData.gTruth;

% Evaluate tracking accuracy
mtrics = compareTrajectories(camPoses, gTruth(addedFramesIdx), AlignmentType="similarity");
disp(['Absolute RMSE for key frame location (m): ', num2str(mtrics.AbsoluteRMSE(2))]);
```

```
Absolute RMSE for key frame location (m): 0.043641
```

```
% Plot the absolute translation error at each key frame
figure
ax = plot(mtrics, "absolute-translation");
view(ax, [2.70 -49.20]);
```



Input Arguments

`vslam` — Visual SLAM object
`monovslam` object

Visual SLAM object, specified as a `monovslam` object.

Version History

Introduced in R2023b

See Also

Objects
`monovslam`

Functions
`checkStatus` | `isDone` | `addFrame` | `poses` | `mapPoints` | `plot` | `hasNewKeyFrame`

ocrTrainingOptions

Options for training OCR model

Description

The `ocrTrainingOptions` object specifies training options for an optical character recognition (OCR) model.

Creation

Syntax

```
ocrOptions = ocrTrainingOptions(Name=Value)
```

Description

`ocrOptions = ocrTrainingOptions(Name=Value)` creates an `ocrTrainingOptions` object. Specify one or more properties using name-value arguments to configure a set of training options, and then use them with the `trainOCR` function to train an OCR model. For example, `CharacterSetSource="base-model"` sets the source of the character set to use during OCR training to the base model.

Properties

`OutputLocation` — Folder in which to store the output model file
`pwd` (default) | string scalar | character vector

Folder in which to store the output model file, specified as a string scalar or character vector. If the folder already contains a file with the name specified by the `modelName` argument of the `trainOCR` function, then the `trainOCR` function overwrites it during training.

`CharacterSetSource` — Source of character set for text recognition training
`"auto"` (default) | `"base-model"` | `"ground-truth-data"`

Source of the character set for text recognition training, specified as `"auto"`, `"base-model"`, or `"ground-truth-data"`.

- `"auto"` — Training uses the character set of the base model when resuming from a checkpoint. Otherwise, training selects a character set as defined by the ground truth data.
- `"base-model"` — Training uses the character set of the base model specified in the `trainOCR` function. Use this option to fine-tune the base model used on the ground truth data.
- `"ground-truth-data"` — To use the character set defined by the ground truth data specified in the `trainOCR` function.

`MaxEpochs` — Maximum number of epochs used for training
5 (default) | positive integer

Maximum number of epochs used for training, specified as a positive integer.

`InitialLearnRate` — Initial learning rate used for training
1.0000e-03 (default) | numeric scalar

Initial learning rate used for training, specified as a numeric scalar. Decreasing the initial learning rate can increase training time. However, increasing the rate too much can result in the training being unable to converge.

`Verbose` — Progress information display
`true` or `1` (default) | `false` or `0`

Progress information display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

`VerboseFrequency` — Number of iterations between progress displays
`50` (default) | positive integer

Number of iterations between progress displays, specified as a positive integer. This argument only applies when you set `Verbose` to `true`.

`CheckpointPath` — Path to save check point models
`""` (default) | character vector | string

Path to save check point models, specified as a character vector or a string. When you set `CheckpointPath` to `""`, the `trainOCR` function does not save checkpoints.

`CheckpointFrequency` — Frequency for saving check point models
`1` (default) | positive integer

Frequency for saving check point models, specified as a positive integer. This argument only applies when you specify a path by using `CheckpointPath`.

`CheckpointFrequencyUnit` — Unit of check point frequency
`"epoch"` (default)

Unit of check point frequency, specified as `"epoch"` or `"iteration"`. When you set this value to `"epoch"`, the `trainOCR` function saves a checkpoint every `CheckpointFrequecy` epochs. When you set this value to `"iteration"`, the `trainOCR` function saves a checkpoint every `CheckpointFrequecy` iterations.

`ValidationData` — Data to use for validation during training
`[]` (default) | datastore object

Data to use for validation during training, specified as a datastore object with the same format as the `trainingData` input argument of the `trainOCR` function.

`ValidationFrequency` — Number of iterations between evaluations of validation metrics
`50` (default) | positive integer

Number of iterations between evaluations of validation metrics, specified as a positive integer.

`ValidationPatience` — Validation loss tolerance
`Inf` (default) | positive integer

Validation loss tolerance, specified as `Inf` or a positive integer. This value specifies the consecutive number of iterations for which the validation loss can be greater than or equal to the previous smallest loss before the network stops training.

`OutputNetwork` — Network returned when training ends
`"auto"` (default) | `"best-training-loss"` | `"best-validation-loss"` | `"last-iteration"`

Network returned when training ends, specified as `"auto"`, `"best-training-loss"`, `"best-validation-loss"`, or `"last-iteration"`.

- `"auto"` — Returns `"best-validation-loss"` when you specify the `ValidationData` property. Otherwise, returns `"best-training-loss"`.
- `"best-training-loss"` — Returns the network at the iteration corresponding to the best training loss.
- `"best-validation-loss"` — Returns the network at the iteration corresponding to the best validation loss. To use this option, you must specify the `ValidationData` property.
- `"last-iteration"` — Returns the network corresponding to the last training iteration.

`SolverName` — Name of solver to use for training
`"adam"` (default) | `"sgdm"`

Name of the solver to use for training, specified as `"adam"` or `"sgdm"`.

`GradientDecayFactor` — Exponential decay rate for gradient moving average in the `adam` solver
`0.5` (default) | scalar in range `[0, 1)`

Exponential decay rate for the gradient moving average in the `adam` solver, specified as a scalar in the range `[0, 1)`.

The `GradientDecayFactor` corresponds to β_1, as defined by Kingma and Bar [1].

`SquareGradientDecayFactor` — Exponential decay rate for squared gradient moving average in `adam` solver
`0.999` (default) | scalar in the range `[0, 1)`

Exponential decay rate for the squared gradient moving average in the `adam` solver, specified by a scalar in the range `[0, 1)`.

The `GradientDecayFactor` corresponds to β_1, as defined by Kingma and Bar [1].

`Momentum` — Contribution of gradient step from previous iteration to current iteration of training
`0.5` (default) | scalar in range `[0, 1]`

Contribution of the gradient step from the previous iteration to the current iteration of training, specified as a scalar in the range `[0, 1]`. This argument applies only when using the `sgdm` solver.

`Shuffle` — Control shuffling of training data
`"once"` (default) | `"never"`

Control shuffling of training data, specified as `"once"` or `"never"`.

Examples

Specify Options For OCR Training

Create a directory to save the trained OCR model.

```
outputDir = "OCRModel";
if ~exist(outputDir, "dir")
    mkdir(outputDir);
end
```

Create a directory to save checkpoints.

```
checkpointsDir = "Checkpoints";
if ~exist(checkpointsDir, "dir")
    mkdir(checkpointsDir);
end
```

Create a set of options for training an OCR model.

- Set the gradient decay factor for ADAM optimization to 0.9.
- Use an initial learning rate of 20e-4.
- Set the maximum number of epochs for training to 15.
- Set the verbose frequency to 100 iterations.
- Specify the output directory.
- Specify the checkpoint path to enable saving checkpoints.

```
ocrOptions = ocrTrainingOptions(GradientDecayFactor=0.9,...
    InitialLearnRate=20e-4,...
    MaxEpochs=15,...
    VerboseFrequency=100,...
    OutputLocation=outputDir,...
    CheckpointPath=checkpointsDir)
```

```
ocrOptions =
  ocrTrainingOptions with properties:

               OutputLocation: "OCRModel"
                    MaxEpochs: 15
             InitialLearnRate: 0.0020
                      Verbose: 1
             VerboseFrequency: 100
               CheckpointPath: "Checkpoints"
          CheckpointFrequency: 1
      CheckpointFrequencyUnit: "epoch"
               ValidationData: []
          ValidationFrequency: 50
           ValidationPatience: Inf
                OutputNetwork: "auto"
                   SolverName: "adam"
          GradientDecayFactor: 0.9000
    SquareGradientDecayFactor: 0.9990
                     Momentum: 0.5000
                      Shuffle: "once"
```

```
CharacterSetSource: "auto"
```

Version History

Introduced in R2023a

References

[1] Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization". arXiv, January 29, 2017. https://doi.org/10.48550/arXiv.1412.6980.

See Also

Apps
Image Labeler

Functions
`trainOCR` | `ocrTrainingData` | `ocr` | `quantizeOCR` | `evaluateOCR`

Objects
`ocrText`

Topics
“Getting Started with OCR”
“Automate Ground Truth Labeling for OCR”
“Train an OCR Model to Recognize Seven-Segment Digits”

ocrMetrics

Store OCR quality metrics

Description

The `ocrMetrics` object stores optical character recognition (OCR) quality metrics, consisting of the character and word error rates, for a set of images. The `evaluateOCR` function returns the `ocrMetrics` object.

Creation

Create an OCR quality metrics object by using the `evaluateOCR` function.

Properties

`DataSetMetrics` — Data set metrics
table

Data set metrics, specified as a table with one row. The table contains OCR metrics aggregated over the data set. The table contains one column for each metric specified by the `Metrics` name-value that correspond to the metrics of the `evaluateOCR` function.

`ImageMetrics` — OCR metrics
table with *N* rows

OCR metrics, specified as a table with *N* rows, where *N* is the number of images in the data set. The table contains one column for each metric specified by the `Metrics` name-value of the `evaluateOCR` function. Each image metric returns a vector with one element for each image in the data set. The order of the rows matches the order of the images in the data set.

Examples

Evaluate Accuracy of OCR Model

This example shows how to evaluate the accuracy of an OCR model that can recognize seven-segment numerals on a dataset. The evaluation dataset contain images of energy meter displays that have seven-segment numerals in them.

Download and extract dataset.

```
datasetURL = "https://ssd.mathworks.com/supportfiles/vision/data/7SegmentImages.zip";
datasetZip = "7SegmentImages.zip";
if ~exist(datasetZip,"file")
    disp("Downloading evaluation dataset (" + datasetZip + " - 96 MB) ...");
    websave(datasetZip,datasetURL);
end
```

```
datasetFiles = unzip(datasetZip);
```

Load the evaluation ground truth.

```
ld = load("7SegmentGtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Digits";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
```

Combine the datastores.

```
cds = combine(imds,boxds,txtds);
```

Run OCR on the evaluation dataset.

```
results = ocr(cds, Model="seven-segment");
```

Evaluate the OCR results against the ground truth.

```
metrics = evaluateOCR(results,cds);
```

```
Evaluating ocr results
----------------------
* Selected metrics: character error rate, word error rate.
* Processed 119 images.
* Finalizing... Done.
* Data set metrics:

    CharacterErrorRate    WordErrorRate
    __________________    _____________

         0.082195             0.19958
```

Display accuracy of the OCR model.

```
modelAccuracy = 100*(1-metrics.DataSetMetrics.CharacterErrorRate);
disp("Accuracy of the OCR model= " + modelAccuracy + "%")
```

```
Accuracy of the OCR model= 91.7805%
```

Algorithms

- Error rates are a percentage of characters (words) in the input `groundTruthTxt` that have been incorrectly predicted in `resultTxt`.
- To compute the number of incorrect predictions to use in the error rate calculation, the function uses the Levenshtein distance, which is defined as the minimum number of edits (such as insertions, deletions, or substitutions) required to change one word (or sentence) into another one.

  Error rate = $(S + D + I)/N$ where,

- S — Number of substitutions
- D — Number of deletions
- I — Number of insertions
- N — Maximum number of characters (words) between `groundTruthTxt` or `resultTxt`

Version History

Introduced in R2023a

See Also

Apps
Image Labeler

Functions
`trainOCR` | `ocrTrainingData` | `ocr` | `evaluateOCR`

Objects
`ocrText` | `ocrTrainingOptions`

Topics
“Getting Started with OCR”
“Automate Ground Truth Labeling for OCR”
“Train an OCR Model to Recognize Seven-Segment Digits”

pcviewer

Visualize and inspect large 3-D point cloud

Description

Use the `pcviewer` object to view and inspect large 3-D point clouds. You can interact with the point cloud, and set camera and viewer properties. Point cloud interactions include navigating through the point cloud, modifying the size of points or enlarging their display, and selecting viewing planes. Camera and viewer properties enable you to select between first-person and orbital camera navigation, select a perspective or orthographic camera projection, and set camera location properties.

Creation

Syntax

```
viewer = pcviewer(ptCloud)
viewer = pcviewer(filename)
viewer = pcviewer( ___ ,color)
viewer = pcviewer( ___ ,Name=Value)
```

Description

`viewer = pcviewer(ptCloud)` creates a point cloud viewer that visualizes points stored in the object specified by `ptCloud`. You can access and modify point cloud viewer properties using the returned `pcviewer` object viewer.

`viewer = pcviewer(filename)` visualizes points stored in the file specified by `filename`. `filename` must be a PLY, PCD, LAS, or LAZ file type. LAS/LAZ files support requires the Lidar Toolbox™.

`viewer = pcviewer( ___ ,color)` specifies the colors with which to visualize the points of the point cloud, in addition to any combination of input arguments from previous syntaxes.

`viewer = pcviewer( ___ ,Name=Value)` specifies options using one or more name-value arguments. For example, `VerticalAxis="XUp"` sets the *x*-axis as the vertical axis in the viewer, with up as positive.

Input Arguments

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object. The viewer uses the `Location` property of the input point cloud object to render the position of the points. By default, the viewer sets the colors of the points from the `Color` property of the input `pointCloud` object or by applying a colormap based on the *z*-values from the `Location` property of the object.

`filename` — Filename

string scalar | character vector

Filename, specified as a string scalar or a character vector. The file must be provided as a PLY, PCD, LAS, or LAZ file type. formatted file.

LAS/LAZ files support requires the Lidar Toolbox. LAZ files returned by the `lasFileWriter` object, which facilitates writing point cloud data to LAS/LAZ formats, are optimized for display. The viewer is also optimized for LAZ files stored in the Cloud Optimized Point Cloud (COPC) format. For more details, see the COPC website.

`color` — Color for points in point cloud

RGB triplet | short color name | long color name | *M*-by-3 matrix | *M*-by-*N*-by-3 array | string array | cell array of character vectors

Color for points in the point cloud, specified as an RGB triplet, an *M*-by-3 matrix, or an *M*-by-*N*-by-3 array for an organized point cloud, a short color name, a long color name, array of color names, or a cell array of color names. The `color` input sets the `Color` property. For details on color values, see Color Values on page 2-290 table.

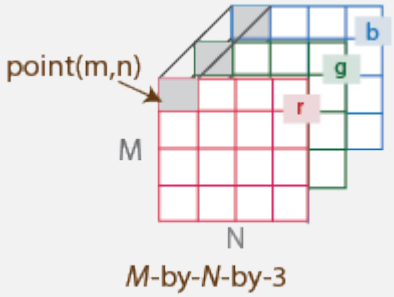
- If the data type of the point cloud `Location` property is `single` or `double`, each `color` RGB value must be in the range [0, 1].
- If the data type of the point cloud `Location` property is `uint8`, each `color` RGB value must be in the range [0, 255].
- The `Color` property of the `pcviewer` object stores the `color` value as an RGB triplet, an *M*-by-3 matrix, or an *M*-by-*N*-by-3 array. If you specify the color one or more color names, the function converts them to their corresponding RGB triplets.

Color	Format	Example

Specify one color for all points	Short or long color name.	`"r"`
		
		`"red"`

	1-by-3 vector (RGB triplet).	 `[255 0 0]`
Specify a color for each point	String array, or cell array of character vectors.	`["red","yellow","blue"]`

	M-by-3 matrix for an unorganized point cloud, in which each row is an RGB triplet. *M* is the number of rows in the unorganized point cloud.	`255 0 0`
		`255 0 0`
		`0 255 255`
		

Color	Format	Example
	M-by-*N*-by-3 array for an organized point cloud, in which each page specifies the red, green, or blue value for the corresponding point in the point cloud. *M*-by-*N* is the size of the organized point cloud.	

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `viewer = pcviewer(ptCloud,ViewPlane="XY")` visualizes the point cloud from the *xy*-plane.

`CameraProjection` — Camera projection for 3-D views

`"perspective"` (default) | `"orthographic"`

Camera projection for 3-D views, specified as one of these values:

- `"perspective"` — Projects the viewing volume as the frustum of a pyramid (a pyramid whose apex has been cut off parallel to the base). Objects further from the camera appear smaller. Distance causes foreshortening, which enables you to perceive the depth of 3-D objects. This projection type is useful when you want to display realistic views of real objects. Perspective projection does not preserve the relative dimensions of objects. Instead, it displays a distant line segment smaller than a nearer line segment of the same length. Lines that are parallel in the data might not appear parallel in the scene.
- `"orthographic"` — This projection type maintains the correct relative dimensions of graphics objects regarding the distance of a given point from the viewer. Relative distance from the camera does not affect the size of objects. Lines that are parallel in the data parallel appear parallel on the screen. This projection type is useful when it is important to maintain the actual size of objects and the angles between objects.

`ViewPlane` — Axes plane to display of point cloud

`"auto"` (default) | `"XY"` | `"YX"` | `"XZ"` | `"ZX"` | `"YZ"` | `"ZY"`

Axes plane to display of point cloud, specified as `"auto"`, `"XY"`, `"YX"`, `"XZ"`, `"ZX"`, `"YZ"`, or `"ZY"`.

`VerticalAxis` — Vertical axis and direction

`"ZUp"` (default) | `"ZDown"` | `"XUp"` | `"XDown"` | `"YUp"` | `"YDown"`

Vertical axis and direction, specified as `"ZUp"`, `"ZDown"`, `"XUp"`, `"XDown"`, `"YUp"`, or `"YDown"`. This argument sets the selected axis of the point cloud as the vertical axis, positive in the specified direction..

`ColorSource` — Source of colormap data to use for the viewer

`"auto"` (default) | `"X"` | `"Y"` | `"Z"` | `"Intensity"` | `"Color"` | `"Row"` | `"Column"`

Source of colormap data to use for the viewer, specified as one of:

- `"X"` — Viewer maps the *x*-coordinates in the `Location` property of the input point cloud.
- `"Y"` — Viewer maps the *y*-coordinates in the `Location` property of the input point cloud.
- `"Z"` — Viewer maps the *z*-coordinates in the `Location` property of the input point cloud.
- `"Intensity"` — Viewer maps the `Intensity` property of the input point cloud.
- `"Color"` — Viewer maps the `Color` property of the input point cloud.
- `"Row"` — Viewer maps the row values of an organized point cloud.
- `"Column"` — Viewer maps the column values of an organized point cloud.
- `"auto""`— If the `Color` property of the input point cloud is available, the object uses those color values. Otherwise, the object assigns the colormap based on the *z*-coordinates in the `Location` property of the input point cloud.

If you provided the `color` input, the `ColorSource` name-value argument is not used.

`BackgroundColor` — Viewer background color
`[0 0 0]` (default) | RGB triplet | hexadecimal color code | short color name | long color name

Viewer background color, specified as an RGB triplet, hexadecimal color code, or a character vector for a short or long color name. For details on color values, see the Color Value on page 2-290 table.

`PointSize` — Size of point cloud points
2 (default) | positive scalar

Size of point cloud points, specified as a positive scalar, in pixels.

`AxesVisibility` — State of axes visibility
`"off"` (default) | `"on"` | `0` or `false` | `1` or `true`

State of axes visibility, specified as `"on"` or `"off"`, or as a numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true`, and `"off"` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `"on"` — Display the axes and its children.
- `"off"` — Hide the axes without deleting it. You still can access the properties of an invisible axes object. (Child objects such as lines remain visible.)

`OptimizeForDisplay` — Logical to enable point cloud optimization for display performance
`1` (`true`) (default) | `0` (`false`)

Logical to enable point cloud optimization for display performance, specified as `1` (`true`) or `0` (`false`). When you set this argument to `true`, point clouds with more than 10 million points are converted into an octree structure. This structure renders a subset of points visible from the current viewpoint and adapts as the viewpoint changes. Enabling this optimization enhances interaction performance.

When you set this argument to `false`, all points are displayed. Displaying all points can decrease interaction performance, depending on your graphics hardware. For more details about graphics rendering in MATLAB, see “System Requirements for Graphics”

Creating an octree structure requires the Lidar Toolbox. If the Lidar Toolbox is not available, all points will be displayed, which can decrease interaction performance.

Properties

`PointSize` — Size of points
2 (default) | positive scalar

Sizes of points, specified as a positive scalar, in pixels.

`MagnifyPointDisplay` — Point display magnification
`"off"` (default) | `"on"` | `0` or `false` | `1` or `true`

Point display magnification, specified as `"on"` or `"off"`, or as a numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true` and shows the points magnified. A value of `"off"` is equivalent to `false` and shows points unmagnified, as they would appear from the position of the camera.

`CameraPosition` — Camera position viewpoint
`[1 1 1]` (default) | three-element vector

Camera position viewpoint, specified as a three-element vector of the form [*x y z*]. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. Changing the `CameraPosition` property changes the point from which you view the volume. For an illustration, see “Camera Graphics Terminology”.

`CameraTarget` — Camera target
`[0 0 0]` (default) | three-element vector

Camera target, specified as a three-element vector of the form [*x y z*]. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. For an illustration, see “Camera Graphics Terminology”.

`CameraUpVector` — Upwards direction
`[0 0 1]` (default) | three-element vector

Upwards direction for the camera, specified as a three-element vector of the form [*x y z*]. For example, the default orientation of the *z*-axis is the up direction (`[0 0 1]`). For an illustration, see “Camera Graphics Terminology”.

`CameraZoom` — Camera zoom level
`1` (default) | positive number

Camera zoom level, specified as a positive number.

`CameraProjection` — Type of camera view projection
`"projection"` | `"orthographic"`

Type of camera view projection, specified as a `"projection"` or `"orthographic"`.

`CameraViewAngle` — Vertical field of view
`60` (default) | positive scalar

Vertical field of view (FOV) for the perspective camera, specified as a positive scalar, in degrees. This property applies only when you set the `CameraProjection` property to `"perspective"`.

`BackgroundColor` — Viewer background color
`[0 0 0]` (default) | RGB triplet | hexadecimal color code | short color name | long color name

Viewer background color, specified as an RGB triplet, hexadecimal color code, or a short or long color name. For details on color values, see the Color Value on page 2-290 table.

`ShowOrientationAxes` — Orientation axes visibility
`"off"` (default) | `"on"` | `0` or `false` | `1` or `true`

Orientation axes visibility, specified as `"on"` or `"off"`, or as a numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true`, and `"off"` is equivalent to `false`. Thus, you can use the value of this property as a logical value.

`ShowPointCloudAxes` — Gridded axes visibility
`"off"` or `0` (`false`) (default) | `"on"` or `1` (`true`)

Gridded axes visibility, specified as `"on"` or `"off"`, or as a numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true`, and `"off"` is equivalent to `false`. Thus, you can use the value of this property as a logical value.

Examples

Plot Spherical Point Cloud with Texture Mapping

Create a sphere with 2500 faces. This generates a point cloud with approximately 6.25 million points.

```
numFaces = 2500;
[x,y,z] = sphere(numFaces);
```

Load an image into the workspace, then map it to the coordinates of the point cloud points.

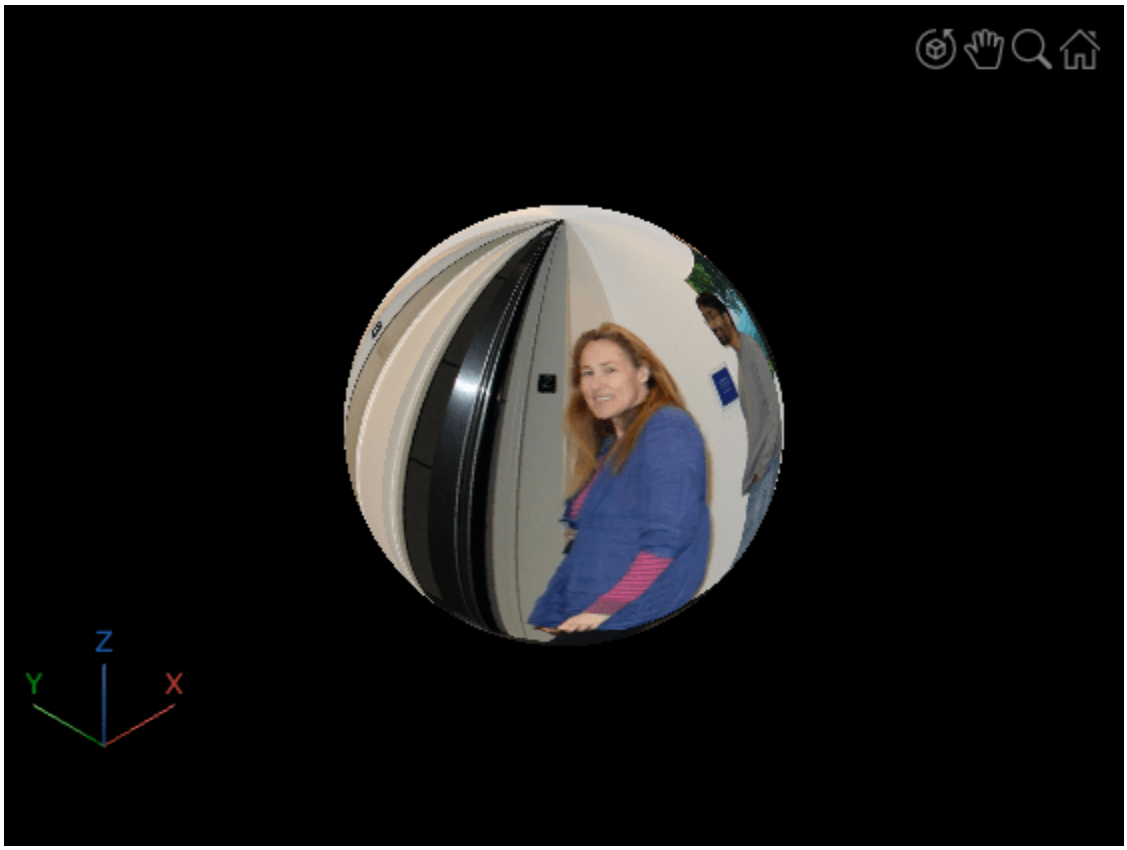
```
I = imresize(imread("visionteam1.jpg"),size(x));
color = flipud(im2double(I));
color = reshape(color,[],3);
```

Create a point cloud object with the location of the sphere and the color from the image.

```
ptCloud = pointCloud([x(:) y(:) z(:)], Color=color);
```

Visualize the point cloud, and optionally interact with it using the viewer controls.

```
pcviewer(ptCloud);
```



More About

Color Values

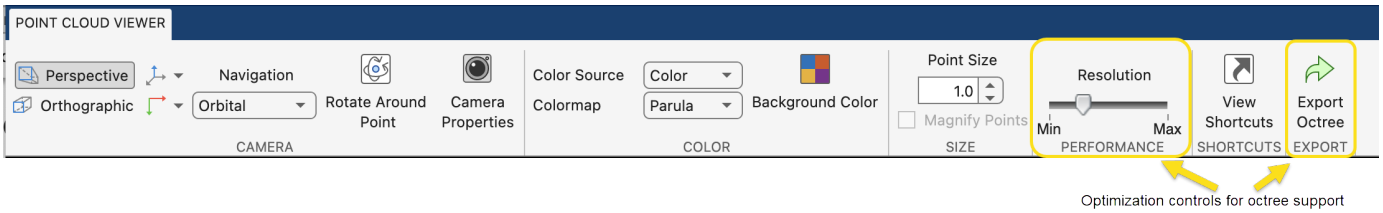
This table shows the valid long color names, their short name and RGB triplet equivalents, and how they appear in MATLAB.

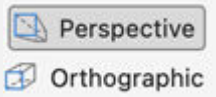
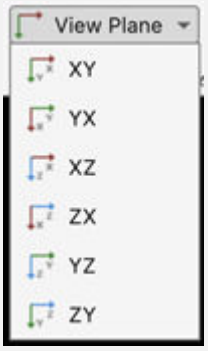
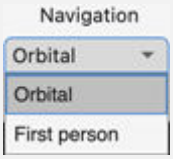
Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	
"green"	"g"	[0 1 0]	
"blue"	"b"	[0 0 1]	
"cyan"	"c"	[0 1 1]	
"magenta"	"m"	[1 0 1]	
"yellow"	"y"	[1 1 0]	
"black"	"k"	[0 0 0]	

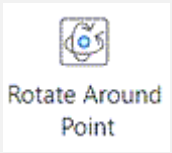
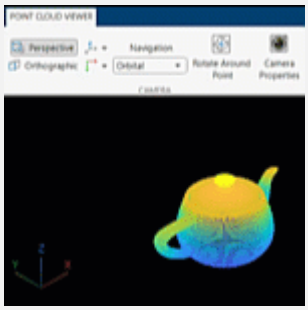
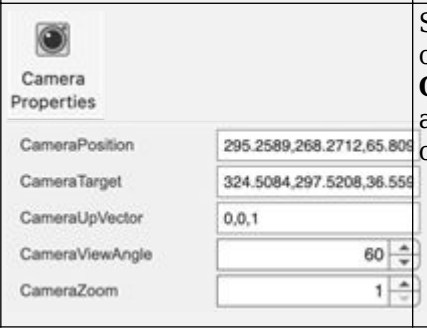
Color Name	Short Name	RGB Triplet	Appearance
"white"	"w"	[1 1 1]	

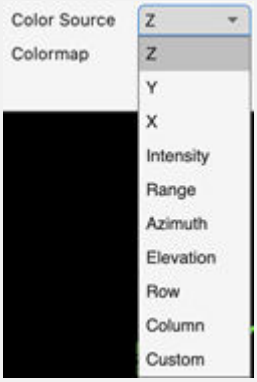
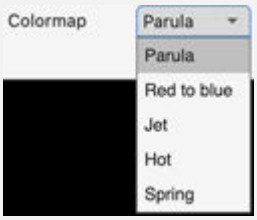
Point Cloud Viewer Toolstrip and Interactions

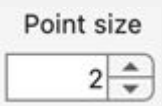
You can expand the viewer toolstrip by clicking the **Point Cloud Viewer** tab. By default, the toolstrip is collapsed.



Feature	Viewer Interaction	Description
Camera Projection		Projection style of the camera.
Vertical Axis		Vertical axis and direction, specified by `ZUp`, `ZDown`, `XUp`, `XDown`, `YUp`, or `YDown`. This value sets the selected axis of the point cloud as the vertical axis, positive in the direction specified.
View Plane		Axes plane to display point cloud, specified as `XY`, `YX`, `XZ`, `ZX`, `YZ`, or `ZY`.
Navigation		View projection used to display the 3-D point cloud, specified as `Orbital` or `First person`. For more details, see “Camera Navigation Controls” on page 2-295

Feature	Viewer Interaction	Description
Rotate Around Point		Select any point as the center of rotation. When the **Rotate Around Point** is disabled, the center of rotation is the axes center. The **Rotate Around Point** mode can only be used with the `Orbital` Navigation mode.
Camera Properties		Set camera location and orientation. The **CameraViewAngle** value applies only for the **Perspective** camera projection.

Feature	Viewer Interaction	Description
Color Source		Color source from which to render color values for the points. The available options depend on the data in the input point cloud. • `X`, `Y`, `Z` — Available for all point clouds • `Color` — Point cloud must have a non-empty color property. • `Intensity` — Point cloud must have non- empty `Intensity` property. • `Range`, `Azimuth`, and `Elevation` — Point cloud must come from a `velodyneFileReader` or a Pandar file reader. • `Row`, `Column` — Organized point clouds. • `Custom` — Color set when launching the `pcviewer` with the `color` argument, or by setting the `Color` property of the `pcviewer` object.
Colormap		Colormap used to render color.
Background Color		Viewer background color.

Feature	Viewer Interaction	Description
Point Size		Size of points, in pixels.
Magnify Points		Magnify the display of points. Enabling this option attenuates the size of the points based on the camera position. If the camera is positioned closer to the points, the points appear bigger than when the camera is far away. Magnify Points applies only for the perspective camera projection and the first person controls.
Resolution		Controls the number of points rendered in the viewer. This control is only visible if the point cloud data was optimized using an octree conversion. Increasing the resolution provides greater detail but can decrease interaction performance.
View Shortcuts		View the point cloud viewer keyboard shortcuts.
Export Octree		Export optimized point cloud. This file can be reused to visualize the point cloud without needing to regenerate the octree conversion.

Mouse interactions:

Behavior	Mouse Interaction
Rotate	Left-click and drag
Pan	Hold **Ctrl**, then left-click and drag
Zoom	Scroll the wheel

Axes toolbar:

Behavior	Mouse Interaction or Interaction after Selection
Rotate	Left-click and drag
Pan	Left-click and drag
Zoom buttons	Scroll wheel
Default	Return to view from launch of viewer instance

Camera Navigation Controls

The point cloud viewer provides orbital and first-person navigation controls. These controls describe the behavior of the observer or first person, represented as the camera, with respect to the object to visualize or orbital, represented as the target.

- `Orbital` — Camera target is fixed. Only the camera position can change.
- `First person` — Both the camera position and the target position can change. Enables first-person navigation for the perspective camera projection.

This table lists the mouse interactions for the `Orbital` and `First person` camera controls:

Interaction	Orbital	First person
Rotate	Rotate the object around the observer	Observer rotates. (Looks around the scene.)
Pan	Pan the object with respect to the observer	Observer moves or "walks" through the scene in the left, right, up, or down directions.
Zoom	Zoom object toward or away from observer. (Make the object appear larger or smaller).	Observer moves or "walks" in the forward or backward directions.

This table shows the orbital keyboard shortcuts:

Behavior	Keyboard shortcut
Rotate by *Z*-axis	Left and right arrows
Rotate by *X*-axis	Up and down arrows
Rotate by *Y*-axis	**Q** and **E**

This table shows the first-person keyboard shortcuts:

Behavior	Keyboard Shortcut
Move forward	**W** — Forward **S** — Backward
Move left or right	**A** — Left **D** — Right

Behavior	Keyboard Shortcut
Move up or down	**R** — Up **F** — Down
Look around the scene (Pitch/Yaw rotation of camera)	Up arrow — Pitch up Down arrow — Pitch down Left arrow — Yaw left Right arrow — Yaw right
Roll the scene (Roll rotation of camera)	**Q** — Clockwise roll **E** — Counter-clockwise roll

Version History

Introduced in R2023a

R2025a: Support for LAZ files

LAZ files can now store point clouds in an octree structure optimized for visualization. To enable this optimization, set the `LazVersion` name-value argument to `"1.4"` of the `lasFileWriter` object. This optimized storage format for LAZ files is based on the Cloud Optimized Point Cloud (COPC) format. For details, see the COPC website.

See Also

Objects
`pointCloud` | `pcplayer` | `planeModel`

Functions
`pcshow` | `showShape` | `pcshowpair` | `plot3` | `pcwrite` | `pcread` | `pcmerge` | `scatter3` | `reconstructScene` | `triangulate` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp` | `lasFileWriter`

Topics
“Coordinate Systems”
Axes Properties

External Websites
COPC

inflated3dVideoClassifier

Inflated-3D (I3D) video classifier. Requires Computer Vision Toolbox Model for Inflated-3D Video Classification

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for Inflated-3D Video Classification add-on.

The `inflated3dVideoClassifier` object is an Inflated-3D (I3D) video classifier pretrained on the Kinetics-400 data set. You can use the pretrained video classifier to classify 400 human actions, such as running, walking, and shaking hands. The I3D classifier model contains two subnetworks: the video network and the optical flow network. Both of these networks are trained on Kinetics-400 with RGB data and optical flow data respectively.

Creation

Syntax

```
i3d = inflated3dVideoClassifier
i3d = inflated3dVideoClassifier(classifierName,classes)
i3d = inflated3dVideoClassifier( ___ ,Name=Value)
```

Description

`i3d = inflated3dVideoClassifier` returns the I3D video classifier pretrained on the Kinetics-400 dataset.

`i3d = inflated3dVideoClassifier(classifierName,classes)` configures the pretrained Inflated 3D (I3D) video classifier for transfer learning on a new set of classes, `classes`, using one of two pretrained classifiers, specified by `classifierName`.

`i3d = inflated3dVideoClassifier( ___ ,Name=Value)` sets properties using name-value arguments in addition to the input arguments from the previous syntax. For example, `i3d = inflated3dVideoClassifier("googlenet-video","wavingHello","clapping",InputSize=[224,224,3,64])` sets the input size of the network to 64 frames of 224-by-224 pixels with 3 channels. You can specify multiple name-value arguments.

Note This object requires the Computer Vision Toolbox Model for Inflated-3D Video Classification. You can install the Computer Vision Toolbox Model for Inflated-3D Video Classification from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To use this object, you must have a license for the Deep Learning Toolbox.

Input Arguments

`classifierName` — Classifier name

`"googlenet-video"` | `"googlenet-video-flow"`

Classifier name, specified as `"googlenet-video"` or `"googlenet-video-flow"`.

Classifier	Description
`"googlenet-video"`	GoogLeNet-based I3D model pretrained on the Kinetics-400 video data for transfer learning.
`"googlenet-video-flow"`	GoogLeNet-based I3D model pretrained on the Kinetics-400 video and optical flow data for transfer learning. During training and inference, both video and optical flow data are used for classification.

Properties

Configure Classifier Properties

`InputSize` — Size of network

`[224,224,3,64]` (default) | four-element row vector

This property is read-only.

Size of the video classifier network, specified as a four-element row vector in the form [*H*,*W*,*C*,*T*], where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames for the video subnetwork.

The input size of the flow subnetwork is equal in height, width, and number of frames, but the number of channels is fixed to 2.

Typical values for the number of frames are 8, 16, 32, or 64. Increase the number of frames to capture the temporal nature of activities when training the classifier. When you are using optical flow data, the number of channels must equal 2, which correspond to the *x*- and *y*-components of velocity.

`InputNormalizationStatistics` — Normalization statistics for the video and optical flow data

structure

This property is read-only.

Normalization statistics for the video and optical flow data, specified as a structure with field names `Video` and `OpticalFlow`, which are also structures with field names, `Min`, `Max`, `Mean`, and `StandardDeviation`. The `Min` and `Max` field values define the minimum and maximum values for rescaling the video and optical flow data. The `Mean`, and `StandardDeviation` values define the mean and standard deviation for input normalization. All field values must be specified as a row vector of size equal to the number of channels for the video input data. When you are using optical flow data, the number of channels must equal 2, which correspond to the *x*- and *y* components of velocity.

The default structure contains:

- A `Video` field, which contains the field `Min` set to `[0,0,0]`, and the field `Max` set to `[255,255,255]`.
- Empty `OpticalFlow`, `Mean`, and `StandardDeviation` field values.

For a video input, the data is rescaled between `-1` and `1` using the `Min` and `Max` field values. For an optical flow input, the data is rescaled between `-1` and `1` using computed minimum and maximum values from the input data.

Note When the `Min` and `Max` field values are not empty, the object first rescales the input data between `-1` and `1`. Then, if the `Mean`, and `StandardDeviation` field values are not empty, the object normalizes the rescaled values by subtracting the mean and dividing by the standard deviation.

An example using this property:

```
stats.Video = struct(Min=[0,0,0],Max=[255,255,255], ...
Mean=[],StandardDeviation=[]);
stats.OpticalFlow = struct(Min=[-20,-20],Max=[20,20] ,...
Mean=[],StandardDeviation=[]);
i3d = inflated3dVideoClassifier('googlenet-video-flow',["waving","clapping"],InputNormalizationS
```

`ModelName` — Name of trained video classifier

string scalar

Name of the trained video classifier, specified as a string scalar.

`Classes` — Classes that the video classifier is configured to train or classify

vector of strings | cell array of character vectors

This property is read-only.

Classes that the video classifier is configured to train or classify, specified as a vector of strings or a cell array of character vectors. For example:

```
classes = ['kiss','laugh','pick','pour','pushup'];
```

Training Properties

`VideoLearnables` — Learnable parameters for video subnetwork of I3D video classifier

table with three columns

Learnable parameters for the video subnetwork of the I3D video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

`VideoState` — State of the nonlearnable parameters for the video subnetwork of the I3D video classifier

table with three columns

State of the nonlearnable parameters for the video subnetwork of the I3D video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of LSTM and batch normalization layers. During training or inference, you can update the network state using the output of the forward and predict functions.

`OpticalLearnables` — Learnable parameters for the optical flow subnetwork of the I3D video classifier

table with three columns

Learnable parameters for the optical flow subnetwork of the I3D video classifier, specified as a table with three columns. Network learnable parameters, specified as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

`OpticalFlowState` — State of the nonlearnable parameters for the optical flow subnetwork of the I3D video classifier

table with three columns

State of the nonlearnable parameters for the video subnetwork of the I3D video classifier, specified as a table with three columns. Network learnable parameters, specified as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

Streaming Video Classification Properties

`VideoSequence` — Video sequence used for streaming classification

4-D numeric array

This property is read-only.

Video sequence used to update and classify sequences for streaming classification, specified as a 4-D numeric array. Each vector in the array is of the form [H,W,C,T], where H and W represent the height and width respectively, C represents the number of channels, and T represents the number of frames, for the video subnetwork. The `updateSequence` and `classifySequence` object functions use the video sequence specified by the `VideoSequence` property.

`OpticalFLowSequence` — Optical flow sequence used for streaming classification

4-D numeric array

This property is read-only.

Optical flow sequence used to update and classify sequences for streaming classification, specified as a 4-D numeric array. Each vector in the array is of the form (*H*,*W*,*C*,*T*), where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames, for the optical flow subnetwork. The `updateSequence` and `classifySequence` object functions use the optical flow sequence specified by the `OpticalFlowSequence`.

Object Functions

Video Classification

classifyVideoFile	Classify a video file
resetSequence	Reset video and optical flow sequence properties for streaming video classification
updateSequence	Update video or optical flow sequence for classification
classifySequence	Classify video and optical flow sequence

Custom Training and Inference

forward	Compute video classifier outputs for training
predict	Compute video classifier predictions

Examples

Classify Video File Using Inflated 3D Video Classifier

This example shows how to use `classifyVideoFile` to classify a video using an Inflated 3D video classifier.

Load a pretrained Inflated-3D video network.

```
i3d = inflated3dVideoClassifier();
```

Specify the video file name to classify.

```
videoFilename = 'visiontraffic.avi';
```

Classify the video using the video classifier.

```
label = classifyVideoFile(i3d, videoFilename);
```

Note that the classifier is not fine-tuned to compute the correct predictions for visiontraffic.avi, therefore, the predicted label will not be correct. You must train the classifier for optimal performance on your video data.

Version History

Introduced in R2021b

See Also

Apps
Video Labeler

Functions
`sceneTimeRanges` | `writeVideoScenes`

Objects
`dlnetwork` | `r2plus1dVideoClassifier` | `slowFastVideoClassifier`

Topics
“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”
“Activity Recognition from Video and Optical Flow Data Using Deep Learning”

slowFastVideoClassifier

SlowFast video classifier. Requires Computer Vision Toolbox Model for SlowFast Video Classification

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for SlowFast Video Classification add-on.

The `slowFastVideoClassifier` object is a SlowFast video classifier pretrained on the Kinetics-400 data set with a ResNet-50 3-D convolutional neural network (CNN). You can use the pretrained video classifier to classify 400 human actions such as running, walking, and shaking hands.

Creation

Syntax

```
sf = slowFastVideoClassifier
sf = slowFastVideoClassifier("resnet50-3d",classes)
sf = slowFastVideoClassifier( ___ ,Name=Value)
```

Description

`sf = slowFastVideoClassifier` returns a SlowFast video classifier pretrained on the Kinetics-400 data set.

`sf = slowFastVideoClassifier("resnet50-3d",classes)` configures the pretrained SlowFast video classifier for transfer learning on a new set of classes, `classes`.

`sf = slowFastVideoClassifier( ___ ,Name=Value)` sets properties using name-value arguments in addition to the input arguments from the previous syntax. For example, `sf = slowFastVideoClassifier("resnet50-3d",classes,InputSize=[256,256,3,32])` sets the input size of the network. You can specify multiple name-value arguments.

Note This function requires the Computer Vision Toolbox Model for SlowFast Video Classification. You can install Computer Vision Toolbox Model for SlowFast Video Classification from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To use this object, you must have a license for the Deep Learning Toolbox.

Properties

Configure Classifier Properties

`InputSize` — Size of the network
`[256,256,3,32]` (default) | four-element row vector

This property is read-only.

Size of the video classifier network, specified as a four-element row vector in the form [*H*,*W*,*C*,*T*], where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames for the video subnetwork.

Typical values for the number of frames are 8, 16, 32, or 64. Increase the number of frames to capture the temporal nature of activities when training the classifier.

`InputNormalizationStatistics` — Normalization statistics for the video data
`structure` (default)

This property is read-only.

Normalization statistics for the video data, specified as a structure with field names `Min`, `Max`, `Mean`, and `StandardDeviation`. The `Min` and `Max` field values define the minimum and maximum values for rescaling the video data. The `Mean`, and `StandardDeviation` values define the mean and standard deviation for input normalization. All field values must be specified as a row vector of size equal to the number of channels for the video input data.

The default structure contains the fields, `Min`, `Max`, `Mean` and `StandardDeviation` with values `[0,0,0]`, `[255,255,255]`, `[0.45,0.45,0.45]`, and `[0.225,0.225,0.225]`, respectively. You must calculate the statistics values from the dataset for which you are training the video classifier. To rescale the data using minimum and maximum values precomputed from your dataset, specify both `Min` and `Max`. Otherwise, the minimum and maximum values are calculated from each input sequence when using `updateSequence` or `classifyVideoFile`.

Note The object normalizes the data by rescaling it between `0` and `1`, and then the rescaled data is standardized by subtracting the mean and dividing by the standard deviation. The rescaled data is standardized if the `Mean` and `StandardDeviation` fields are non-empty. The input is automatically normalized when using `updateSequence` or `classifyVideoFile` object functions. The data must be manually normalized when using the `forward` or `predict` object functions.

`ModelName` — Name of trained video classifier
string scalar

Name of the trained video classifier, specified as a string scalar.

`Classes` — Classes that the video classifier is configured to train or classify
vector of strings | cell array of character vectors

This property is read-only.

Classes that the video classifier is configured to train or classify, specified as a vector of strings or a cell array of character vectors. For example:

```
classes = ['kiss','laugh','pick','pour','pushup'];
```

Training Properties

`Learnables` — Learnable parameters for the SlowFast video classifier
table with three columns

Learnable parameters for the SlowFast video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of long short term networks (LSTM) and batch normalization layers. During training or inference, you can update the network state using the output of the `forward` and `predict` object functions.

`State` — State of the nonlearnable parameters of the SlowFast video classifier
table with three columns

State of the nonlearnable parameters of the SlowFast video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

Streaming Video Classification Properties

`VideoSequence` — Video sequence used for streaming classification
4-D numeric array

This property is read-only.

Video sequence used to update and classify sequences for streaming classification, specified as a 4-D numeric array. Each vector in the array is of the form [*H*,*W*,*C*,*T*], where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames, for the video subnetwork. The `updateSequence` and `classifySequence` object functions use the video sequence specified by the `VideoSequence` property.

Object Functions

Video Classification

classifyVideoFile	Classify a video file
classifySequence	Classify video sequence
resetSequence	Reset video sequence properties for streaming video classification
updateSequence	Update video sequence for classification

Custom Training and Inference

forward	Compute video classifier outputs for training
predict	Compute video classifier predictions

Examples

Classify Video File Using Video Classifier

This example requires the Computer Vision Toolbox™ Model for SlowFast Video Classification. You can install the Computer Vision Toolbox Model for SlowFast Video Classification from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”.

Load a slowfast video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the file name of the video to classify.

```
videoFilename = "washingHands.avi";
```

For video classification, set the number of randomly selected video sequences to 15.

```
numSequences = 15;
```

Classify the video using the `classifyVideoFile` function.

```
[label,score] = classifyVideoFile(sf,videoFilename,NumSequences=numSequences)
```

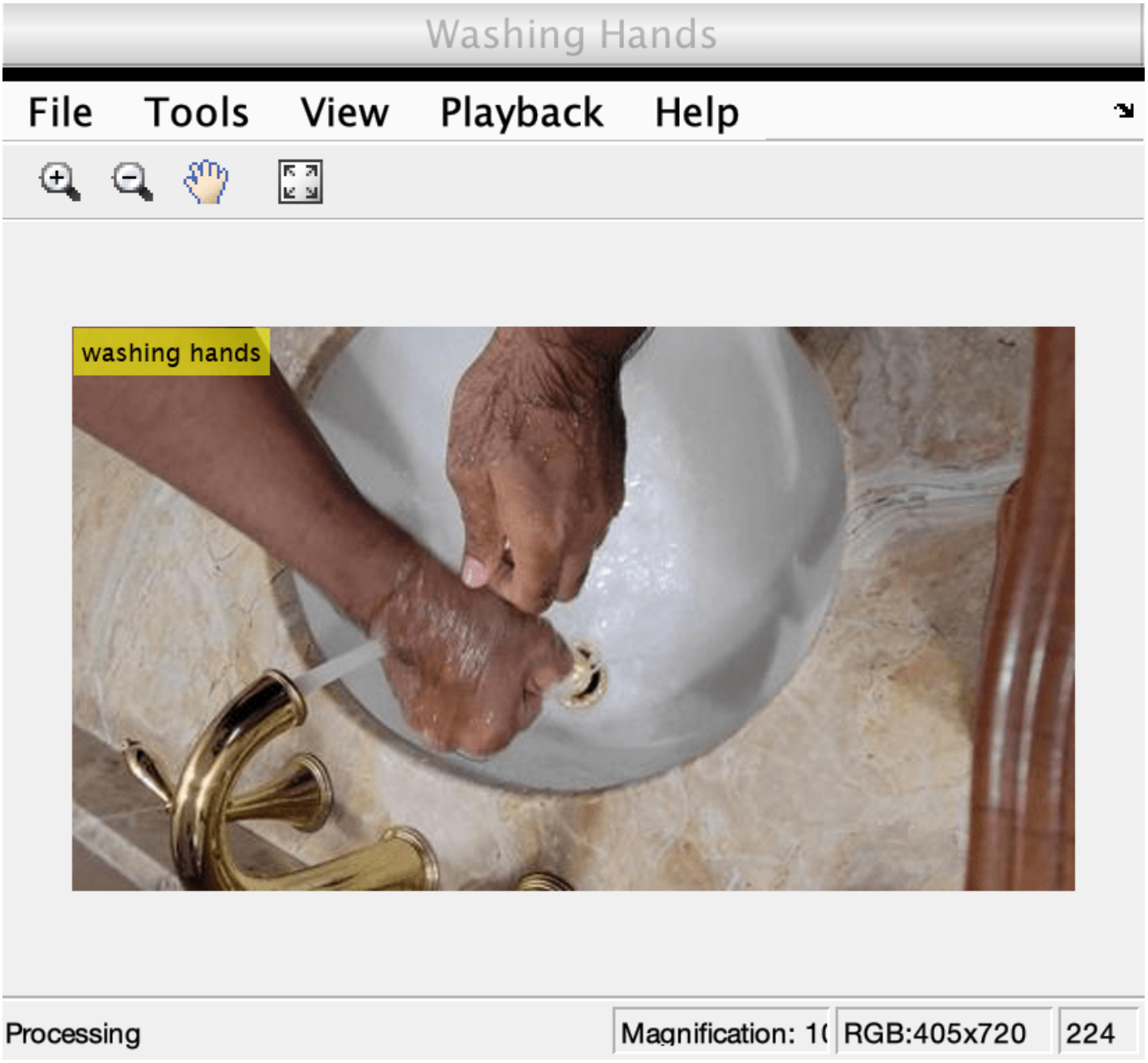
```
label = categorical
     washing hands


score = single

0.0034
```

Display the classified label using a `vision.VideoPlayer`.

```
player = vision.VideoPlayer('Name','Washing Hands');
reader = VideoReader(videoFilename);
while hasFrame(reader)
    frame = readFrame(reader);
    % Resize the frame by 1.5 times for display
    frame = imresize(frame,1.5);
    frame = insertText(frame,[2,2], string(label),'FontSize',18);
    step(player,frame);
end
```



Version History

Introduced in R2021b

See Also

Apps
Video Labeler

Functions
`sceneTimeRanges` | `writeVideoScenes`

Objects
`dlnetwork` | `inflated3dVideoClassifier` | `r2plus1dVideoClassifier`

Topics
“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”
“Gesture Recognition using Videos and Deep Learning”

r2plus1dVideoClassifier

R(2+1)D video classifier. Requires Computer Vision Toolbox Model for R(2+1)D Video Classification

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for R(2+1)D Video Classification add-on.

The `r2plus1dVideoClassifier` object returns an R(2+1)D video classifier pretrained on the Kinetics-400 data set. You can use the pretrained video classifier to classify 400 human actions, such as running, walking, and shaking hands.

Creation

Syntax

```
rd = r2plus1dVideoClassifier
rd = r2plus1dVideoClassifier("resnet-3d-18",classes)
rd = r2plus1dVideoClassifier( ___ ,Name=Value)
```

Description

`rd = r2plus1dVideoClassifier` returns a R(2+1)D video classifier pretrained on the Kinetics-400 dataset.

`rd = r2plus1dVideoClassifier("resnet-3d-18",classes)` configures the pretrained R(2+1)D video classifier for transfer learning on a new set of classes, `classes`. The video classifier is pretrained on the Kinetics-400 dataset with a ResNet3D convolutional neural network(CNN) with 18 spatio-temporal layers.

`rd = r2plus1dVideoClassifier( ___ ,Name=Value)` sets properties using name-value arguments in addition to the input arguments from the previous syntax. For example, `rd = r2plus1dVideoClassifier("resnet-3d-18",classes,InputSize=[112,112,3,32])` sets the input size of the network. You can specify multiple name-value arguments.

Note This function requires the Computer Vision Toolbox Model for R(2+1)D Video Classification. You can install the Computer Vision Toolbox Model for R(2+1)D Video Classification from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To use this object, you must have a license for the Deep Learning Toolbox.

Properties

Configure Classifier Properties

`InputSize` — Size of the network

`[112,112,3,32]` (default) | four-element row vector

This property is read-only.

Size of the video classifier network, specified as a four-element row vector in the form [*H*,*W*,*C*,*T*], where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames for the video subnetwork.

Typical values for the number of frames are 8, 16, 32, or 64. Increase the number of frames to capture the temporal nature of activities when training the classifier.

`InputNormalizationStatistics` — Normalization statistics for the video data
`structure` (default)

This property is read-only.

Normalization statistics for the video data, specified as a structure with field names `Min`, `Max`, `Mean`, and `StandardDeviation`. The `Min` and `Max` field values define the minimum and maximum values for rescaling the video data. The `Mean`, and `StandardDeviation` values define the mean and standard deviation for input normalization. All field values must be specified as a row vector of size equal to the number of channels for the video input data.

The default structure contains the fields, `Min`, `Max`, `Mean` and `StandardDeviation` with values `[0,0,0]`, `[255,255,255]`, `[0.45,0.45,0.45]`, and `[0.225,0.225,0.225]`, respectively. You must calculate the statistics values from the dataset for which you are training the video classifier. To rescale the data using minimum and maximum values precomputed from your dataset, specify both `Min` and `Max`. Otherwise, the minimum and maximum values are calculated from each input sequence when using `updateSequence` or `classifyVideoFile`.

Note The object normalizes the data by rescaling it between `0` and `1`, and then the rescaled data is standardized by subtracting the mean and dividing by the standard deviation. The rescaled data is standardized if the `Mean` and `StandardDeviation` fields are non-empty. The input is automatically normalized when using `updateSequence` or `classifyVideoFile` object functions. The data must be manually normalized when using the `forward` or `predict` object functions.

`ModelName` — Name of trained video classifier
string scalar

Name of the trained video classifier, specified as a string scalar.

`Classes` — Classes that the video classifier is configured to train or classify
vector of strings | cell array of character vectors

This property is read-only.

Classes that the video classifier is configured to train or classify, specified as a vector of strings or a cell array of character vectors. For example:

```
classes = ['kiss','laugh','pick','pour','pushup'];
```

Training Properties

`Learnables` — Learnable parameters for the ResNet (2+1)D video classifier
table with three columns

Learnable parameters for the ResNet (2+1)D video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

`State` — State of the nonlearnable parameters of the R(2+1)D video classifier
table with three columns

State of the nonlearnable parameters for the ResNet (2+1)D video classifier, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Parameter value, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of long short term networks (LSTM) and batch normalization layers. During training or inference, you can update the network state using the output of the `forward` and `predict` object functions.

Streaming Video Classification Properties

`VideoSequence` — Video sequence used for streaming classification
4-D numeric array

This property is read-only.

Video sequence used to update and classify sequences for streaming classification, specified as a 4-D numeric array. Each vector in the array is of the form [*H*,*W*,*C*,*T*], where *H* and *W* represent the height and width respectively, *C* represents the number of channels, and *T* represents the number of frames, for the video subnetwork. The `updateSequence` and `classifySequence` object functions use the video sequence specified by the `VideoSequence` property.

Object Functions

Video Classification

classifyVideoFile	Classify a video file
classifySequence	Classify video sequence
resetSequence	Reset video sequence properties for streaming video classification
updateSequence	Update video sequence for classification

Custom Training and Inference

forward	Compute video classifier outputs for training
predict	Compute video classifier predictions

Examples

Classify Video Stream Using R(2+1)D Video Classifier

This example shows how to classify a video stream using a pretrained R(2+1)D video classifier.

Load a pretrained R(2+1)D video classifier.

```
rd = r2plus1dVideoClassifier();
```

Create a VideoReader to read a video frame by frame.

```
videoFilename = "visiontraffic.avi";
reader = VideoReader(videoFilename);
```

Create a video player to visualize the video data and update the player position to match the size of the video.

```
player = vision.VideoPlayer;
player.Position(:,3:4) = [reader.Width reader.Height];
```

Specify the frequency at which the streaming video frames will be classified as 10. The classifier will be applied to a sequence of video frames every 10 frames to balance runtime performance against classification performance.

```
classificationFrequency = 10;
```

Specify the sequence length required by the classifier. This is based on the input size of the video classifier. You can begin to classify the sequence only after the sequence length reaches the required length.

```
sequenceLength = rd.InputSize(4);
```

Read through the video frame by frame, update the sequence with each new frame using `updateSequence`, and then classify the collected frames using `classifySequence`.

```
numFrames = 0;
text = "";

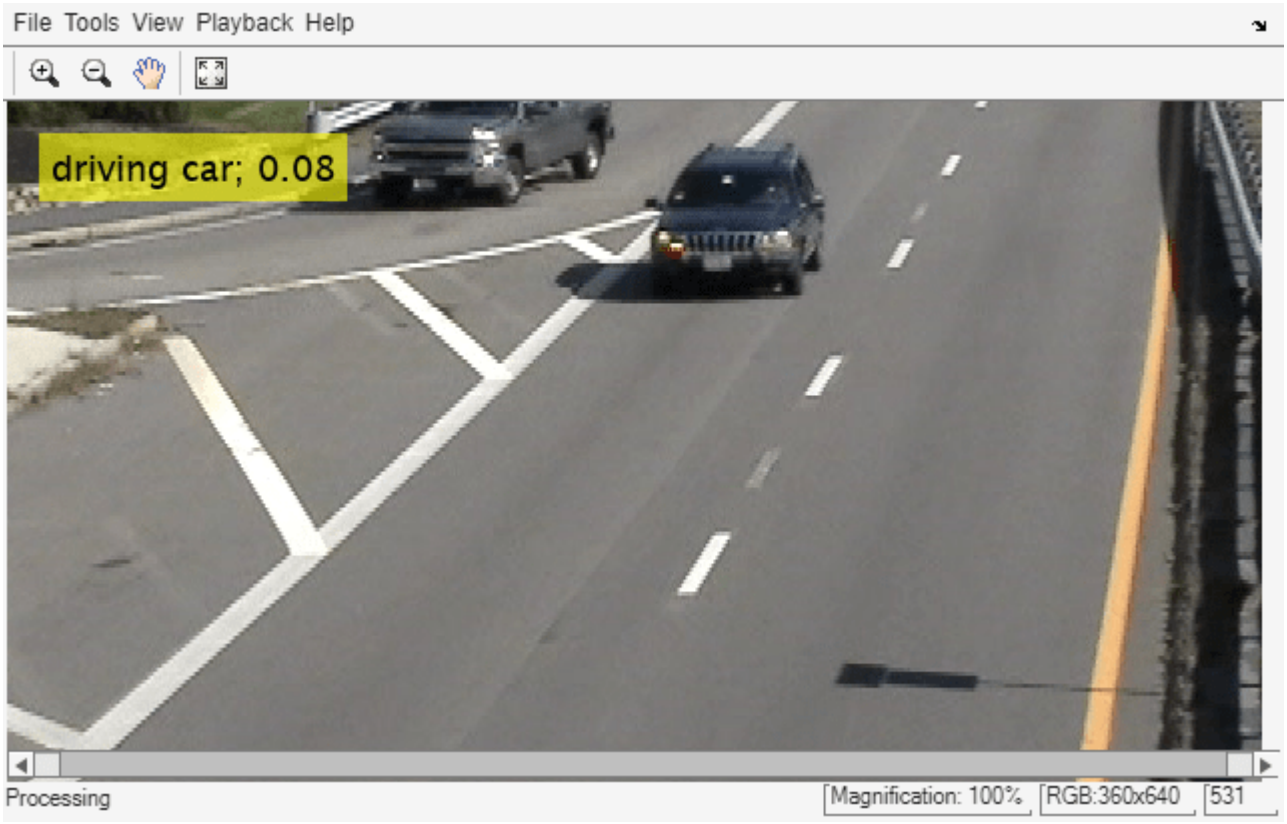
while hasFrame(reader)
    frame = readFrame(reader);
    numFrames = numFrames + 1;

    % Update the sequence with the next video frame.
    rd = updateSequence(rd,frame);

    % Classify the sequence based on the classificationFrequency.
    if mod(numFrames, classificationFrequency) == 0 && numFrames >= sequenceLength
        [label,score] = classifySequence(rd);
        text = string(label) + "; " + num2str(score, "%0.2f");
    end

    % Insert the predicted label into the video frame.
    frame = insertText(frame,[30,30],text,'FontSize',18);

    % Display the video and label.
    step(player,frame);
end
```



Version History

Introduced in R2021b

See Also

Apps
Video Labeler

Functions
`sceneTimeRanges` | `writeVideoScenes`

Objects
`dlnetwork` | `inflated3dVideoClassifier` | `slowFastVideoClassifier`

Topics
“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Human Activity Recognition Using R(2+1)D Video Classification”
“Gesture Recognition using Videos and Deep Learning”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”

classifySequence

Classify video sequence

Syntax

```
label = classifySequence(classifier)
[label,score] = classifySequence(classifier)
[___] = classifySequence(classifier,ExecutionEnvironment=env)
```

Description

`label = classifySequence(classifier)` classifies a video sequence using the video classifier `classifier`. The function returns `label`, a scalar categorical that specifies the classification of the video or optical flow sequence. `label` is one of the values of the `Classes` property of the video classifier object.

`[label,score] = classifySequence(classifier)` additionally returns the classification score associated with the label. The score represents the confidence of the predicted class label, and contains values between `0` and `1`.

`[___] = classifySequence(classifier,ExecutionEnvironment=env)` specifies the hardware resources for running the classifier in addition to any combination of arguments from previous syntaxes, as one of these options:

- `"auto"` — Sets the execution environment to the GPU, if available. Otherwise the function sets it to the CPU.
- `"gpu"` — Sets the execution environment to the GPU. Usage of the GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Sets the execution environment to the CPU.

Examples

Classify Video Sequences in a Video File

This example shows how to classify video sequences in a video file using a SlowFast Video Classifier pretrained on the Kinetics-400 video activity recognition dataset. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

```
sf = slowFastVideoClassifier();
```

Set Up Video Player and Video Reader

Specify the video file name to stream video frames.

```
videoFilename = "pushup.mp4";
```

Create a VideoReader to read video.

```
reader = VideoReader(videoFilename);
```

Set up a video player.

```
player = vision.VideoPlayer;
```

Classify Video Sequences

Specify how frequently the classifier should be applied to incoming video frames.

```
classifyInterval = 10;
```

A value of 10 balances runtime performance against classification performance. Increase this value to improve runtime performance at the cost of missing actions from the video file.

Obtain the sequence length of the SlowFast Video Classifier. Classify only after capturing at least `sequenceLength` number of frames from the video file.

```
sequenceLength = sf.InputSize(4);
```

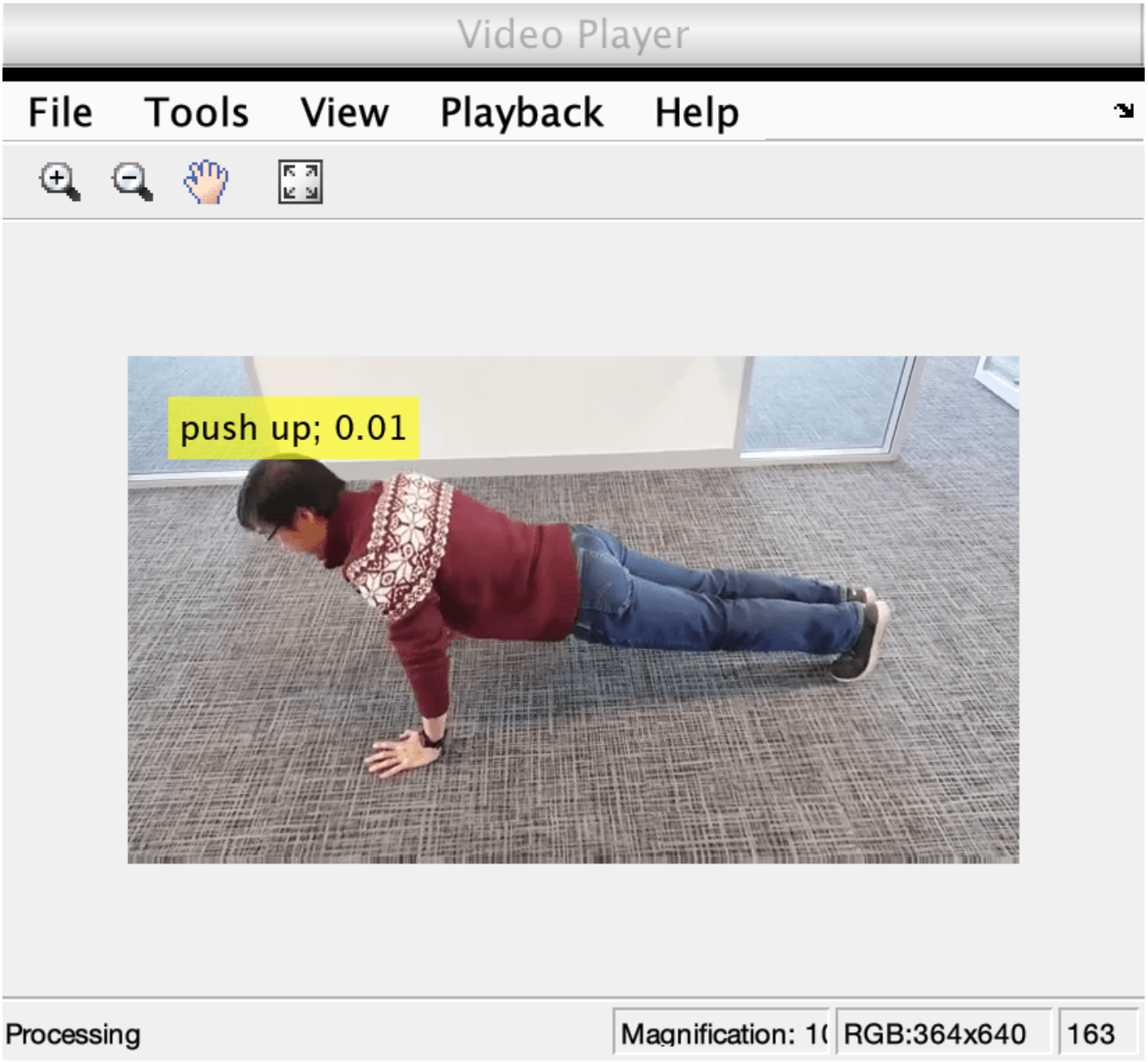
Read video frames using the `hasFrame` and `readFrame` functions of the `VideoReader`. Using the `updateSequence` function update the video classifier's sequence. Using the `classifySequence` function classify the updated sequence.

```
numFrames = 0;
text = "";

while hasFrame(reader)
    frame = readFrame(reader);
    numFrames = numFrames + 1;

    % Update the sequence with the next video frame.
    sf = updateSequence(sf,frame);

    % Classify the sequence only at every classifyInterval number of frames.
    if mod(numFrames, classifyInterval) == 0 && numFrames >= sequenceLength
        [label,score] = classifySequence(sf);
        text = string(label) + "; " + num2str(score, "%0.2f");
    end
    frame = insertText(frame,[30,30],text,'FontSize',24);
    step(player,frame);
end
```



Classify Streaming Webcam Video Using SlowFast Video Classifier

This example shows how to classify a streaming video from a webcam using a pretrained SlowFast Video Classifier. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Download Pretrained Video Classifier

Download the pretrained SlowFast video classifier.

```
downloadFolder = fullfile(tempdir,"gesture");
zipFile = "slowFastPretrained_fourClasses.zip";
if ~isfile(fullfile(downloadFolder,zipFile))
    disp("Downloading the pretrained network...");
    downloadURL = "https://ssd.mathworks.com/supportfiles/vision/data/" + zipFile;
    zipFile = fullfile(downloadFolder,zipFile);
    websave(zipFile,downloadURL);
    unzip(zipFile,downloadFolder);
end
```

Load the pretrained SlowFast video classifier.

```
pretrainedDataFile = fullfile(downloadFolder,"slowFastPretrained_fourClasses.mat");
pretrained = load(pretrainedDataFile);
slowFastClassifier = pretrained.data.slowFast;
```

Display the class label names of the pretrained video classifier. Any gesture such as "clapping" and "wavingHello" on to the webcam will be recognized by the SlowFast Video Classifier.

```
classes = slowFastClassifier.Classes
```

```
classes = 4×1 categorical array
    "clapping"
    "noAction"
    "somethingElse"
    "wavingHello"
```

Set Up the Webcam and the Video Player

In this example, a webcam object is used to capture streaming video. A Video Player is used to display the streaming video along with the predicted class.

Create a webcam object using the `webcam` function.

```
cam = webcam;
```

Create a Video Player using `vision.VideoPlayer` function. Make sure to place the Video Player in a position where you can clearly see the streaming video when running the classification.

```
player = vision.VideoPlayer;
```

Classify the Webcam Streaming Video

Specify how frequently the classifier should be applied to incoming video frames.

```
classifyInterval = 10;
```

A value of 10 balances runtime performance against classification performance. Increase this value to improve runtime performance at the cost of missing gestures from the live video stream.

Obtain the sequence length of the SlowFast Video Classifier. Classify only after capturing at least `sequenceLength` number of frames from the webcam.

```
sequenceLength = slowFastClassifier.InputSize(4);
```

Specify the maximum number of frames to capture in a loop using the `maxNumFrames` variable. Make sure you wave one of your hands to recognize `"wavingHello"` label, and clap using both your hands for the classifier to recognize `"clapping"` label.

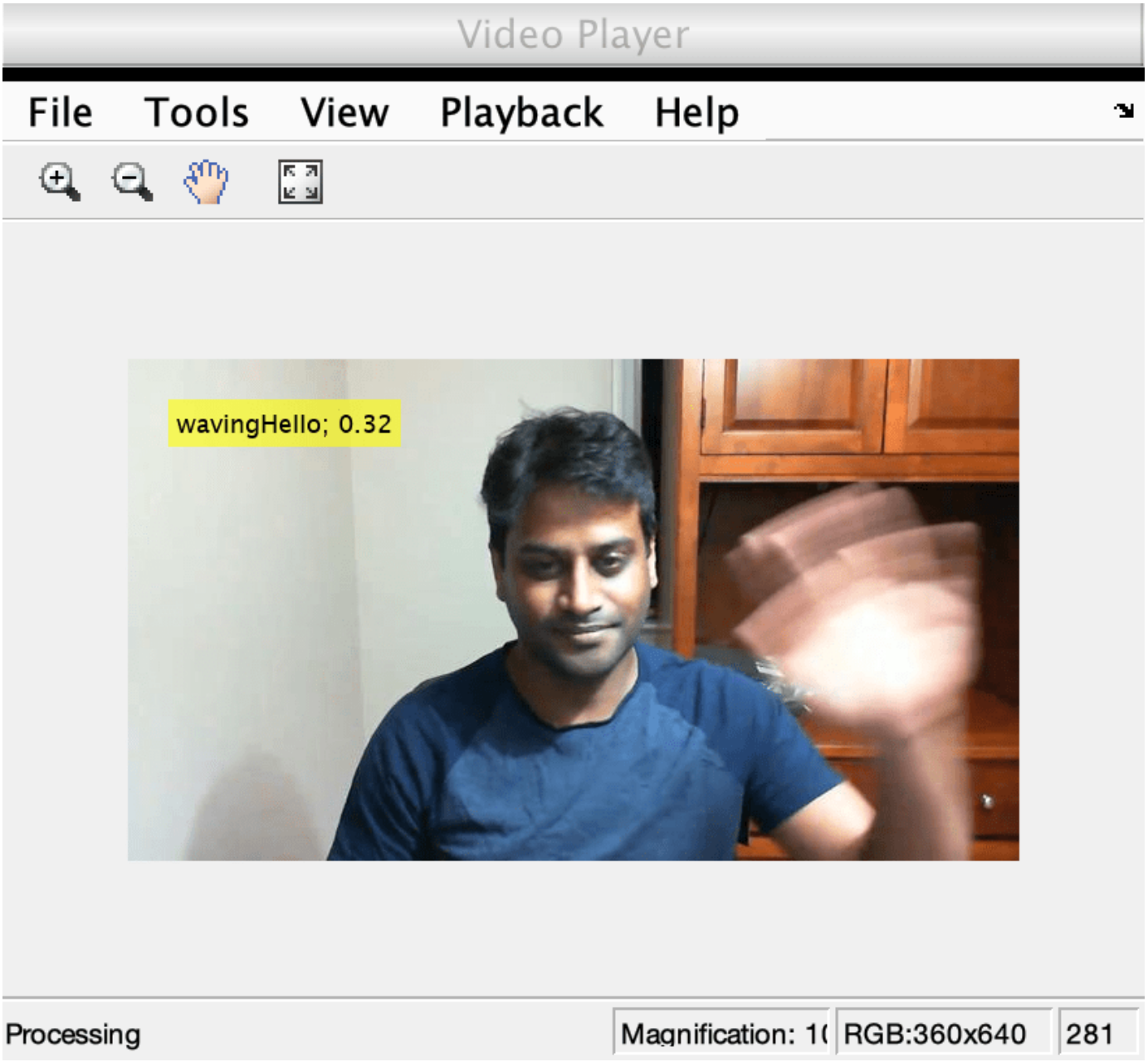
```
maxNumFrames = 280;
```

Capture the webcam snapshot in a loop. Update the streaming video sequence of the classifier using the `updateSequence` method, and classify the streaming sequence using the `classifySequence` method.

```
numFrames = 0;
text = "";

while numFrames <= maxNumFrames
    frame = snapshot(cam);

    numFrames = numFrames + 1;
    slowFastClassifier = updateSequence(slowFastClassifier,frame);
    if mod(numFrames, classifyInterval) == 0 && numFrames >= sequenceLength
        [label,scores] = classifySequence(slowFastClassifier);
        if ~isempty(label)
            text = string(label) + "; " + num2str(max(scores), "%0.2f");
        end
    end
    frame = insertText(frame,[30,30],text,'FontSize',18);
    step(player,frame);
end
```



Input Arguments

`classifier` — Classifier

`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier, specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

Output Arguments

`label` — Classification of video or optical flow sequence

categorical scalar

Classification of the video or optical flow sequence, returned as a categorical scalar.

`score` — Classification score associated with the label
scalar value between `0` and `1`

Classification score associated with the label, returned as a scalar value between `0` and `1`. The score represents the confidence of the predicted class label.

Version History

Introduced in R2021b

See Also

Functions
`sceneTimeRanges` | `writeVideoScenes` | `updateSequence` | `classifyVideoFile` | `predict` | `forward` | `resetSequence`

Objects
`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics
“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”

classifySequence

Classify video and optical flow sequence

Syntax

```
label = classifySequence(i3d)
[label,score] = classifySequence(i3d)
[___] = classifySequence(i3d,ExecutionEnvironment=env)
```

Description

`label = classifySequence(i3d)` classifies a video and optical flow sequence using the Inflated-3D (I3D) video classifier `i3d`. The function returns `label`, a scalar categorical that specifies the classification of the video or optical flow sequence. `label` is one of the values of the `Classes` property of the video classifier object.

`[label,score] = classifySequence(i3d)` additionally returns the classification score associated with the label. The score represents the confidence of the predicted class label, and contains values between `0` and `1`.

`[___] = classifySequence(i3d,ExecutionEnvironment=env)` specifies the hardware resources for running the classifier in addition to any combination of arguments from previous syntaxes, as one of these options:

- `"auto"` — Sets the execution environment to the GPU, if available. Otherwise the function sets it to the CPU.
- `"gpu"` — Sets the execution environment to the GPU. Usage of the GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Sets the execution environment to the CPU.

Examples

Classify Video Sequences in a Video File

This example shows how to classify video sequences in a video file using a SlowFast Video Classifier pretrained on the Kinetics-400 video activity recognition dataset. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

```
sf = slowFastVideoClassifier();
```

Set Up Video Player and Video Reader

Specify the video file name to stream video frames.

```
videoFilename = "pushup.mp4";
```

Create a VideoReader to read video.

```
reader = VideoReader(videoFilename);
```

Set up a video player.

```
player = vision.VideoPlayer;
```

Classify Video Sequences

Specify how frequently the classifier should be applied to incoming video frames.

```
classifyInterval = 10;
```

A value of 10 balances runtime performance against classification performance. Increase this value to improve runtime performance at the cost of missing actions from the video file.

Obtain the sequence length of the SlowFast Video Classifier. Classify only after capturing at least `sequenceLength` number of frames from the video file.

```
sequenceLength = sf.InputSize(4);
```

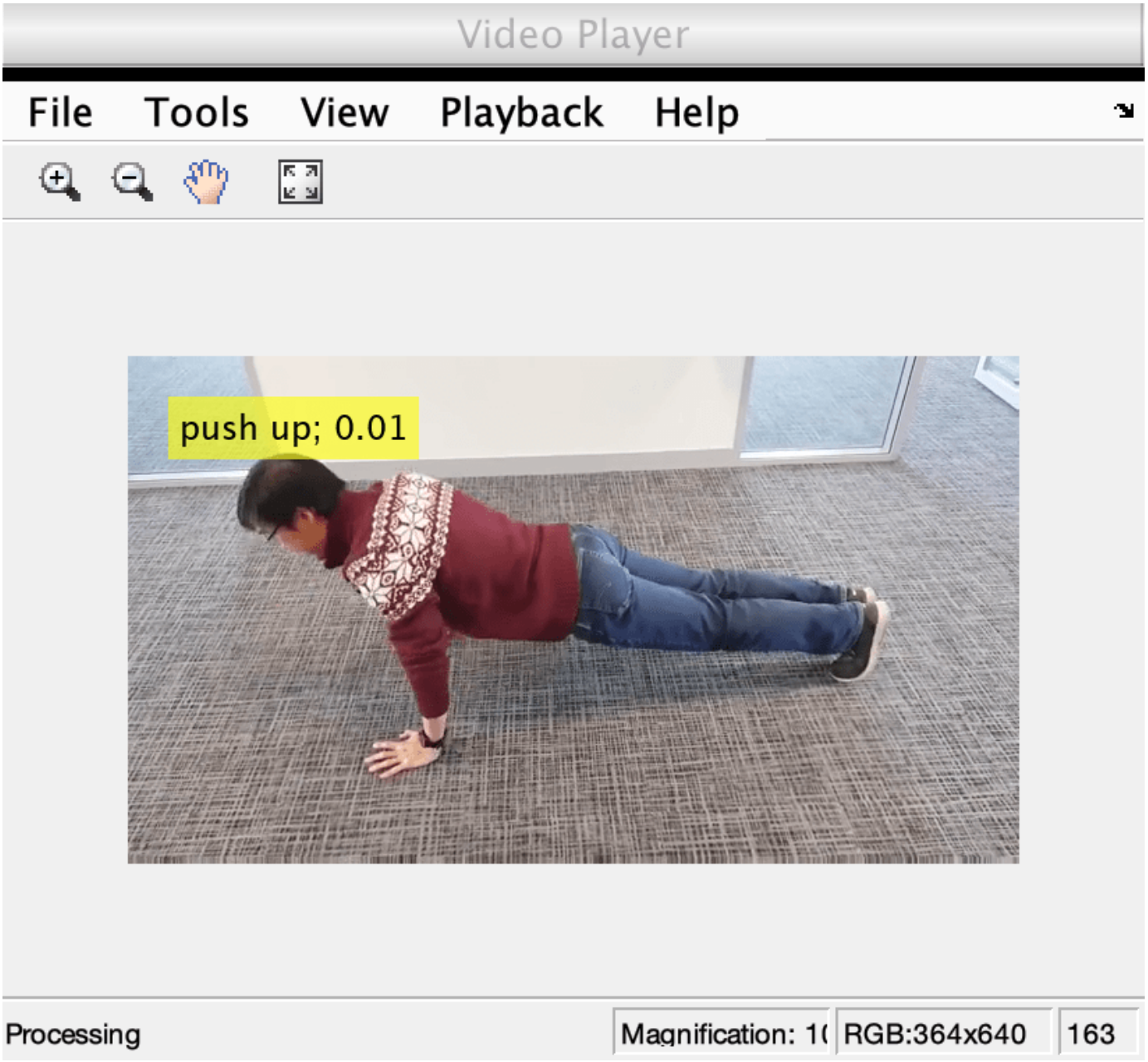
Read video frames using the `hasFrame` and `readFrame` functions of the `VideoReader`. Using the `updateSequence` function update the video classifier's sequence. Using the `classifySequence` function classify the updated sequence.

```
numFrames = 0;
text = "";

while hasFrame(reader)
    frame = readFrame(reader);
    numFrames = numFrames + 1;

    % Update the sequence with the next video frame.
    sf = updateSequence(sf,frame);

    % Classify the sequence only at every classifyInterval number of frames.
    if mod(numFrames, classifyInterval) == 0 && numFrames >= sequenceLength
        [label,score] = classifySequence(sf);
        text = string(label) + "; " + num2str(score, "%0.2f");
    end
    frame = insertText(frame,[30,30],text,'FontSize',24);
    step(player,frame);
end
```



Input Arguments

`i3d` — Classifier
`inflated3dVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier` object.

Output Arguments

`label` — Classification of video or optical flow sequence
categorical scalar

Classification of the video or optical flow sequence, returned as a categorical scalar.

`score` — Classification score associated with the label
scalar value between `0` and `1`

Classification score associated with the label, returned as a scalar value between `0` and `1`. The score represents the confidence of the predicted class label.

Version History

Introduced in R2021b

See Also

Functions
`sceneTimeRanges` | `writeVideoScenes` | `classifyVideoFile` | `resetSequence` | `updateSequence` | `predict` | `forward`

Objects
`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics
“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”

classifyVideoFile

Classify a video file

Syntax

```
label = classifyVideoFile(classifier,videoFilename)
[label,score] = classifyVideoFile( ___ )
[ ___ ] = classifySequence( ___ ,Name=Value)
```

Description

`label = classifyVideoFile(classifier,videoFilename)` predicts the class label for the video file, specified by `videoFilename` using the specified classifier `classifier`. The video file must be readable by the `VideoReader` object.

Use the `classifyVideoFile` object function for applications where the entire input video can be classified with one label. For example, you can use this object function to evaluate the performance of the classifier on a collection of ground truth video files. If the video contains sequences with different class labels, use the `classifySequence` object function.

This function selects sequences of frames from the video file using uniform random sampling to select the start of time of each sequence. The number of frames is set by the `InputSize` property of the classifier object. The function selects the most frequently occurring label in the video file as the classification label for the file. The video file must contain at least the number of frames set by the `InputSize` property.

`[label,score] = classifyVideoFile( ___ )` additionally returns the classification score associated with the label using the combination of input arguments from the previous syntax.. The score represents the confidence of the predicted class label and contains values between `0` and `1`.

`[ ___ ] = classifySequence( ___ ,Name=Value)` specifies options using name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `label = classifySequence(i3d,ExecutionEnvironment="cpu")` specifies for the classifier to run on the CPU.

Examples

Classify Video File Using Video Classifier

This example requires the Computer Vision Toolbox™ Model for SlowFast Video Classification. You can install the Computer Vision Toolbox Model for SlowFast Video Classification from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”.

Load a slowfast video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the file name of the video to classify.

```
videoFilename = "washingHands.avi";
```

For video classification, set the number of randomly selected video sequences to 15.

```
numSequences = 15;
```

Classify the video using the `classifyVideoFile` function.

```
[label,score] = classifyVideoFile(sf,videoFilename,NumSequences=numSequences)
```

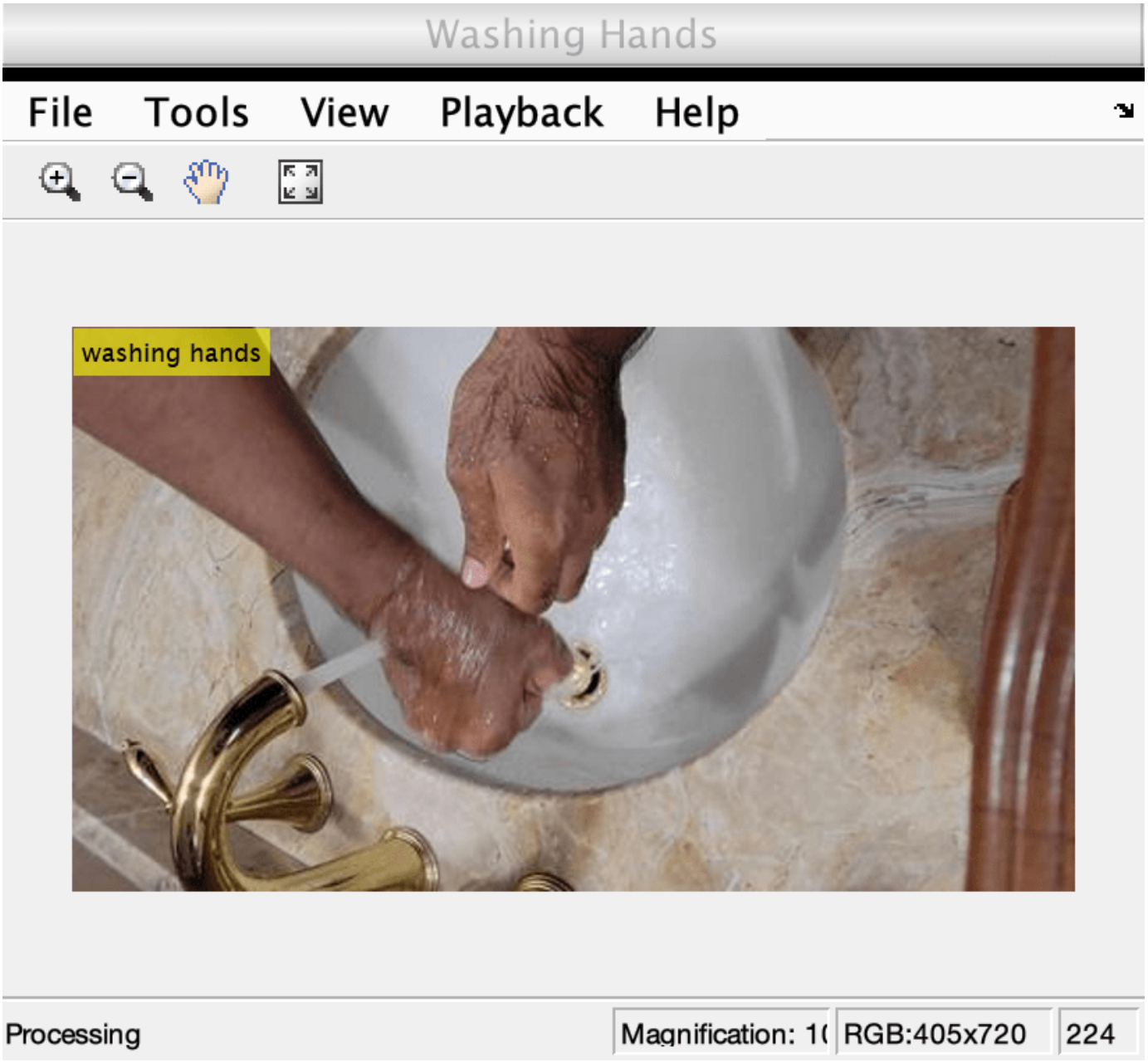
```
label = categorical
     washing hands


score = single

0.0034
```

Display the classified label using a `vision.VideoPlayer`.

```
player = vision.VideoPlayer('Name','Washing Hands');
reader = VideoReader(videoFilename);
while hasFrame(reader)
    frame = readFrame(reader);
    % Resize the frame by 1.5 times for display
    frame = imresize(frame,1.5);
    frame = insertText(frame,[2,2], string(label),'FontSize',18);
    step(player,frame);
end
```



Input Arguments

`classifier` — Classifier
`inflated3dVideoClassifier` object | `r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier`, `r2plus1dVideoClassifier`, or `slowFastVideoClassifier` object.

`videoFilename` — Video file name
character vector | string scalar

Video file name, specified as a character vector or a string scalar. The video file must be readable by the `VideoReader` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `label = classifySequence(i3d,ExecutionEnvironment="cpu")` specifies for the classifier to run on the CPU.

`ExecutionEnvironment` — Hardware execution environment

`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware execution environment on which to run the classifier, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Sets the execution environment to the GPU, if available. Otherwise the function sets it to the CPU.
- `"gpu"` — Sets the execution environment to the GPU. Usage of the GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Sets the execution environment to the CPU.

`NumSequences` — Maximum number of sequences to sample from video file

`"auto"` (default) | positive integer

Maximum number of sequences to sample from the video file, specified as `"auto"` or a positive integer. When you set this value to `"auto"`, the function sets the number of sequences to twice the number of sequences that can be selected from the video file based on the number of frames available. The `"auto"` setting provides a uniform coverage of the entire video. Increasing the value of `NumSequences` increases the uniform coverage of the video.

`MiniBatchSize` — Mini-batch size for processing a large video

16 | positive integer

Mini-batch size used for processing a large video, specified as a positive integer. Video sequences are grouped and processed in mini-batches to improve computational efficiency. Increasing the value of `MiniBatchSize` increases processing speed, but uses more memory.

Output Arguments

`label` — Classification of video

categorical scalar

Classification of video, returned as a categorical scalar.

`score` — Classification score associated with the label

scalar value between `0` and `1`

Classification score associated with the label, returned as a scalar value between `0` and `1`. The score represents the confidence of the predicted class label.

Version History

Introduced in R2021b

See Also

Objects

`inflated3dVideoClassifier` | `slowFastVideoClassifier` | `r2plus1dVideoClassifier`

Topics

“Getting Started with Video Classification Using Deep Learning”
“Evaluate a Video Classifier”
“Classify Streaming Webcam Video Using SlowFast Video Classifier”

forward

Compute video classifier outputs for training

Syntax

```
dLYVideo = forward(classifier,dlXVideo)
[dLYVideo,stateVideo] = forward(classifier,dlXVideo)
[dLYVideo,dlYFlow] = forward(classifier,dlXVideo,dlXFlow)
[dLYVideo,dlYFlow,stateVideo,stateFlow] = forward(classifier,dlXVideo,
dlXFlow)
```

Description

`dLYVideo = forward(classifier,dlXVideo)` computes the video classifier outputs for training. You can use this function with `dlfeval` to automatically compute gradients for updating the learnable parameters of the video classifier. `classifier` is specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` classifier object.

`[dLYVideo,stateVideo] = forward(classifier,dlXVideo)` also returns the updated video network state. The output, `stateVideo`, contains information maintained by the classifier between training iterations. For example, the state of batch normalization operation.

`[dLYVideo,dlYFlow] = forward(classifier,dlXVideo,dlXFlow)` also returns the optical flow outputs from the classifier for training. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"Farneback"`.

`[dLYVideo,dlYFlow,stateVideo,stateFlow] = forward(classifier,dlXVideo, dlXFlow)` also returns the updated video network and the optical flow network states.

Examples

Compute Video Classifier Outputs for Training

This example shows how to compute video classifier outputs for training. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load a video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the video file name.

```
videoFilename = "washingHands.avi";
```

Create a `VideoReader` to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read the required number of video frames corresponding to the video classifier network from the beginning of the video file. The required number of frames is defined by the value of the 4th element of the `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
sequenceRange = [1,sequenceLength];
videoFrames = read(reader,sequenceRange);
```

Resize the video frames for training. The required height and width are defined by the first two elements of the `InputSize` property of the video classifier.

```
heightWidth = sf.InputSize(1:2);
resized = imresize(videoFrames,heightWidth);
```

Convert the input to type `single`.

```
resized = single(resized);
```

Rescale the input between 0 and 1.

```
minValue = sf.InputNormalizationStatistics.Min;
maxValue = sf.InputNormalizationStatistics.Max;
minValue = reshape(minValue,1,1,3);
maxValue = reshape(maxValue,1,1,3);
resized = rescale(resized,0,1,InputMin=minValue,InputMax=maxValue);
```

Normalize the video data using the mean and standard deviation.

```
meanValue = sf.InputNormalizationStatistics.Mean;
stdValue = sf.InputNormalizationStatistics.StandardDeviation;
meanValue = reshape(meanValue,1,1,3);
stdValue = reshape(stdValue,1,1,3);
resized = resized - meanValue;
resized = resized./stdValue;
```

Convert the input to a `dlarray` object.

```
dlVideo = dlarray(resized,"SSCTB");
trainingActivations = forward(sf,dlVideo);
```

Find the class label corresponding to the maximum score.

```
[score,idx] = max(trainingActivations);
label = sf.Classes(idx)
```

```
label = categorical
     washing hands
```

```
score
```

```
score =
  1(S) × 1(S) × 1(S) × 1(C) × 1(B) single dlarray

    0.0026
```

Display the class label.

```
frame = videoFrames(:,:,:,end);
frame = insertText(frame,[2,2],string(label),FontSize=24);
imshow(frame)
```



Input Arguments

`classifier` — Classifier object

`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier object, specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

`dlXVideo` — Video input

H-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object

Video input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

`dlXFlow` — Video and optical flow input

`SSCTB` formatted `dlarray` object

Video and optical flow input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video and optical flow input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

Output Arguments

`dLYVideo` — Activations of video network
`dlarray` object

Activations of the video network, returned as a formatted `dlarray` object.

`stateVideo` — Updated video network state
table

Updated video network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of the LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

`dlYFlow` — Activations of optical flow network
`dlarray` object

Activations of the optical flow network, returned as a formatted `dlarray` object.

`stateFlow` — Updated optical flow network state
table

Updated optical flow network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifySequence` | `updateSequence` | `classifyVideoFile` | `predict`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics

“Gesture Recognition using Videos and Deep Learning”

forward

Compute video classifier outputs for training

Syntax

```
dLYVideo = forward(i3d,dlXVideo)
[dLYVideo,stateVideo] = forward(i3d,dlXVideo)
[dLYVideo,dlYFlow] = forward(i3d,dlXVideo,dlXFlow)
[dLYVideo,dlYFlow,stateVideo,stateFlow] = forward(i3d,dlXVideo,dlXFlow)
```

Description

`dLYVideo = forward(i3d,dlXVideo)` computes the video classifier outputs for training. You can use this function with `dlfeval` to automatically compute gradients for updating the learnable parameters of the video classifier. `i3d` is specified as an `inflated3dVideoClassifier` classifier object. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"none"`.

`[dLYVideo,stateVideo] = forward(i3d,dlXVideo)` also returns the updated video network state. The output, `stateVideo`, contains information maintained by the classifier between training iterations. For example, the state of batch normalization operation.

`[dLYVideo,dlYFlow] = forward(i3d,dlXVideo,dlXFlow)` also returns the optical flow outputs from the classifier for training. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"Farneback"`.

`[dLYVideo,dlYFlow,stateVideo,stateFlow] = forward(i3d,dlXVideo,dlXFlow)` also returns the updated video network and the optical flow network states.

Examples

Compute Video Classifier Outputs for Training

This example shows how to compute video classifier outputs for training. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load a video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the video file name.

```
videoFilename = "washingHands.avi";
```

Create a `VideoReader` to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read the required number of video frames corresponding to the video classifier network from the beginning of the video file. The required number of frames is defined by the value of the 4th element of the `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
sequenceRange = [1,sequenceLength];
videoFrames = read(reader,sequenceRange);
```

Resize the video frames for training. The required height and width are defined by the first two elements of the `InputSize` property of the video classifier.

```
heightWidth = sf.InputSize(1:2);
resized = imresize(videoFrames,heightWidth);
```

Convert the input to type `single`.

```
resized = single(resized);
```

Rescale the input between 0 and 1.

```
minValue = sf.InputNormalizationStatistics.Min;
maxValue = sf.InputNormalizationStatistics.Max;
minValue = reshape(minValue,1,1,3);
maxValue = reshape(maxValue,1,1,3);
resized = rescale(resized,0,1,InputMin=minValue,InputMax=maxValue);
```

Normalize the video data using the mean and standard deviation.

```
meanValue = sf.InputNormalizationStatistics.Mean;
stdValue = sf.InputNormalizationStatistics.StandardDeviation;
meanValue = reshape(meanValue,1,1,3);
stdValue = reshape(stdValue,1,1,3);
resized = resized - meanValue;
resized = resized./stdValue;
```

Convert the input to a `dlarray` object.

```
dlVideo = dlarray(resized,"SSCTB");
trainingActivations = forward(sf,dlVideo);
```

Find the class label corresponding to the maximum score.

```
[score,idx] = max(trainingActivations);
label = sf.Classes(idx)
```

```
label = categorical
     washing hands
```

```
score
```

```
score =
  1(S) × 1(S) × 1(S) × 1(C) × 1(B) single dlarray

    0.0026
```

Display the class label.

```
frame = videoFrames(:,:,:,end);
frame = insertText(frame,[2,2],string(label),FontSize=24);
imshow(frame)
```



Input Arguments

`i3d` — Classifier

`inflated3dVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier` object.

`dlXVideo` — Video input

H-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object

Video input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

`dlXFlow` — Video and optical flow input

`SSCTB` formatted `dlarray` object

Video and optical flow input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video and optical flow input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

Output Arguments

`dLYVideo` — Activations of video network
`dlarray` object

Activations of the video network, returned as a formatted `dlarray` object.

`stateVideo` — Updated video network state
table

Updated video network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of the LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

`dlYFlow` — Activations of optical flow network
`dlarray` object

Activations of the optical flow network, returned as a formatted `dlarray` object.

`stateFlow` — Updated optical flow network state
table

Updated optical flow network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifySequence` | `updateSequence` | `classifyVideoFile` | `predict` | `resetSequence`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

predict

Compute video classifier predictions

Syntax

```
dLYVideo = predict(classifier,dlXVideo)
[dLYVideo,stateVideo] = predict(classifier,dlXVideo)
[dLYVideo,dlYFlow] = predict(classifier,dlXVideo,dlXFlow)
[dLYVideo,dlYFlow,stateVideo,stateFlow] = predict(classifier,dlXVideo,
dlXFlow)
```

Description

`dLYVideo = predict(classifier,dlXVideo)` computes the predictions of the video classifier. `classifier` is specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` classifier object.

`[dLYVideo,stateVideo] = predict(classifier,dlXVideo)` also returns the updated video network state. The output, `stateVideo`, contains information between training iterations maintained by the classifier. For example, the state of batch normalization operation.

`[dLYVideo,dlYFlow] = predict(classifier,dlXVideo,dlXFlow)` also returns the optical flow predictions from the classifier for training. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"Farneback"`.

`[dLYVideo,dlYFlow,stateVideo,stateFlow] = predict(classifier,dlXVideo,dlXFlow)` also returns the updated video network and the optical flow network states.

Examples

Compute Predictions for Video Using Video Classifier

This example shows how to compute predictions for a video using a video classifier. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load a video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the video file name.

```
videoFilename = "pushup.mp4";
```

Create a `VideoReader` to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read the required number of video frames corresponding to the video classifier network, from the beginning of the video file. The required number of frames is defined by the value of the 4th element of the `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
sequenceRange = [1, sequenceLength];
videoFrames = read(reader,sequenceRange);
```

Resize video frames for prediction. The required height and width are defined by the first two elements of the `InputSize` property of the video classifier.

```
heightWidth = sf.InputSize(1:2);
resized = imresize(videoFrames,heightWidth);
```

Convert the input to type `single`.

```
resized = single(resized);
```

Rescale the input between 0 and 1.

```
minValue = sf.InputNormalizationStatistics.Min;
maxValue = sf.InputNormalizationStatistics.Max;
minValue = reshape(minValue,1,1,3);
maxValue = reshape(maxValue,1,1,3);
resized = rescale(resized,0,1,InputMin=minValue,InputMax=maxValue);
```

Normalize the video data using the mean and standard deviation.

```
meanValue = sf.InputNormalizationStatistics.Mean;
stdValue = sf.InputNormalizationStatistics.StandardDeviation;
meanValue = reshape(meanValue,1,1,3);
stdValue = reshape(stdValue,1,1,3);
resized = resized - meanValue;
resized = resized./stdValue;
```

Convert the input to dlarray object.

```
dlVideo = dlarray(resized,"SSCTB");
predictionScores = predict(sf,dlVideo);
```

Find the class label corresponding to the maximum score.

```
[score,idx] = max(predictionScores);
label = sf.Classes(idx)
```

```
label = categorical
     push up
```

Display the predicted class label and the score.

```
text = string(label) + "; " + num2str(score,"%0.2f");
frame = videoFrames(:,:,:,end);
frame = insertText(frame,[30,30],text,FontSize=24);

imshow(frame)
```



Input Arguments

`classifier` — Classifier object

`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier object, specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

`dlXVideo` — Video input

H-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object

Video input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

`dlXFlow` — Video and optical flow input

`SSCTB` formatted `dlarray` object

Video and optical flow input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video and optical flow input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

Output Arguments

`dLYVideo` — Activations of video network
`dlarray` object

Activations of the video network, returned as a formatted `dlarray` object.

`stateVideo` — Updated video network state
table

Updated video network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of the LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

`dlYFlow` — Activations of optical flow network
`dlarray` object

Activations of the optical flow network, returned as a formatted `dlarray` object.

`stateFlow` — Updated optical flow network state
table

Updated optical flow network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifySequence`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics

“Gesture Recognition using Videos and Deep Learning”

predict

Compute video classifier predictions

Syntax

```
dLYVideo = predict(i3d,dlXVideo)
[dLYVideo,stateVideo] = predict(i3d,dlXVideo)
[dLYVideo,dlYFlow] = predict(i3d,dlXVideo,dlXFlow)
[dLYVideo,dlYFlow,stateVideo,stateFlow] = predict(i3d,dlXVideo,dlXFlow)
```

Description

`dLYVideo = predict(i3d,dlXVideo)` computes the predictions of the video classifier. `i3d` is specified as an `inflated3dVideoClassifier`, `r2plus1dVideoClassifier`, or `slowFastVideoClassifier` classifier object. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"none"`.

`[dLYVideo,stateVideo] = predict(i3d,dlXVideo)` also returns the updated video network state. The output, `stateVideo`, contains information between training iterations maintained by the classifier. For example, the state of batch normalization operation.

`[dLYVideo,dlYFlow] = predict(i3d,dlXVideo,dlXFlow)` also returns the optical flow predictions from the classifier for training. Use this syntax when you set the `OpticalFlowMethod` property of the classifier object to `"Farneback"`.

`[dLYVideo,dlYFlow,stateVideo,stateFlow] = predict(i3d,dlXVideo,dlXFlow)` also returns the updated video network and the optical flow network states.

Examples

Compute Predictions for Video Using Video Classifier

This example shows how to compute predictions for a video using a video classifier. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load a video classifier pretrained on the Kinetics-400 data set.

```
sf = slowFastVideoClassifier;
```

Specify the video file name.

```
videoFilename = "pushup.mp4";
```

Create a `VideoReader` to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read the required number of video frames corresponding to the video classifier network, from the beginning of the video file. The required number of frames is defined by the value of the 4th element of the `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
sequenceRange = [1, sequenceLength];
videoFrames = read(reader,sequenceRange);
```

Resize video frames for prediction. The required height and width are defined by the first two elements of the `InputSize` property of the video classifier.

```
heightWidth = sf.InputSize(1:2);
resized = imresize(videoFrames,heightWidth);
```

Convert the input to type `single`.

```
resized = single(resized);
```

Rescale the input between 0 and 1.

```
minValue = sf.InputNormalizationStatistics.Min;
maxValue = sf.InputNormalizationStatistics.Max;
minValue = reshape(minValue,1,1,3);
maxValue = reshape(maxValue,1,1,3);
resized = rescale(resized,0,1,InputMin=minValue,InputMax=maxValue);
```

Normalize the video data using the mean and standard deviation.

```
meanValue = sf.InputNormalizationStatistics.Mean;
stdValue = sf.InputNormalizationStatistics.StandardDeviation;
meanValue = reshape(meanValue,1,1,3);
stdValue = reshape(stdValue,1,1,3);
resized = resized - meanValue;
resized = resized./stdValue;
```

Convert the input to dlarray object.

```
dlVideo = dlarray(resized,"SSCTB");
predictionScores = predict(sf,dlVideo);
```

Find the class label corresponding to the maximum score.

```
[score,idx] = max(predictionScores);
label = sf.Classes(idx)
```

```
label = categorical
     push up
```

Display the predicted class label and the score.

```
text = string(label) + "; " + num2str(score,"%0.2f");
frame = videoFrames(:,:,:,end);
frame = insertText(frame,[30,30],text,FontSize=24);

imshow(frame)
```



Input Arguments

`i3d` — Classifier

`inflated3dVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier` object.

`dlXVideo` — Video input

H-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object

Video input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

`dlXFlow` — Video and optical flow input

`SSCTB` formatted `dlarray` object

Video and optical flow input, specified as an *H*-by-*W*-by-*C*-by-*T*-by-*B* `SSCTB` formatted `dlarray` object that corresponds to the video and optical flow input of the classifier.

- *H* — Height.
- *W* — Width.
- *C* — Number of channels. The number of channels must be equal to the channels value of the `InputSize` property of the classifier object.
- *T* — Number of frames. The number of frames must be equal to the frames value of the `InputSize` property of the classifier object.
- *B* — Batch size.

Output Arguments

`dLYVideo` — Activations of video network
`dlarray` object

Activations of the video network, returned as a formatted `dlarray` object.

`stateVideo` — Updated video network state
table

Updated video network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of the LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

`dlYFlow` — Activations of optical flow network
`dlarray` object

Activations of the optical flow network, returned as a formatted `dlarray` object.

`stateFlow` — Updated optical flow network state
table

Updated optical flow network state, returned as a table with three columns:

- `Layer` — Layer name, specified as a string scalar.
- `Parameter` — Parameter name, specified as a string scalar.
- `Value` — Value of the parameter, specified as a `dlarray` object.

The network state contains information remembered by the network between iterations. For example, the state of LSTM and batch normalization layers.

During training or inference, you can update the network state using the output of the `forward` and `predict` functions.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifySequence` | `forward` | `resetSequence`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics

“Gesture Recognition using Videos and Deep Learning”

resetSequence

Reset video sequence properties for streaming video classification

Syntax

```
classifierReset = resetSequence(classifier)
```

Description

`classifierReset = resetSequence(classifier)` resets the video sequences for classification. `classifier` is the classifier, specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

Examples

Classify Two Videos Using a Video Classifier

This example shows how to classify two different video sequences one after the other by reseting the video sequences of the video classifier using the `resetSequence` function. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

Load the SlowFast Video Classifier pretrained on the Kinetics-400 dataset.

```
sf = slowFastVideoClassifier();
```

Classify First Video Sequence

Specify the video file name.

```
videoFilename = "washingHands.avi";
```

Create a VideoReader to read the video frames.

```
reader = VideoReader(videoFilename);
```

Update the video classifier sequence with video frames before using `classifySequence`. The required number of frames is defined by the value of the 4th element of `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
```

Read a video sequence starting from the first frame.

```
startSequenceIndex = 1;
sequenceRange = [startSequenceIndex, startSequenceIndex + sequenceLength - 1];
```

Update the video classifier sequence with video frames.

```
for ii = sequenceRange
    videoFrame = read(reader, ii);
    sf = updateSequence(sf,videoFrame);
end
```

Classify the video sequence updated so far.

```
[label1, score1] = classifySequence(sf)
```

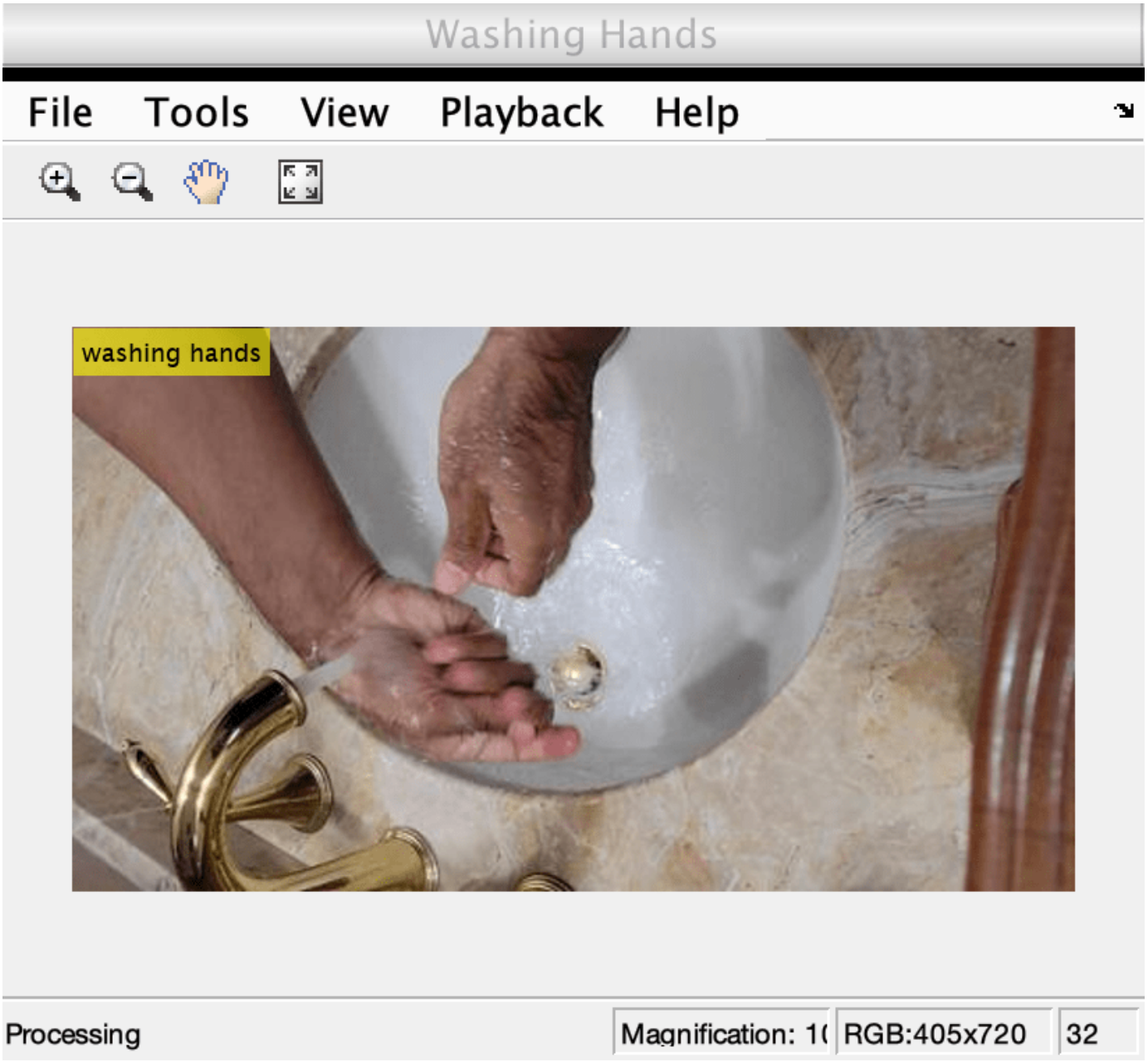
```
label1 = categorical
     washing hands


score1 = single

0.0031
```

Display the classified label.

```
player = vision.VideoPlayer('Name','Washing Hands');
for ii = sequenceRange
    frame = read(reader,ii);
    % Resize the frame by 1.5 times for display
    frame = imresize(frame,1.5);
    frame = insertText(frame,[2,2], string(label1),'FontSize',18);
    step(player,frame);
end
```



Classify Second Video Sequence

Reset the video sequence of the video classifier without creating a new video classifier.

```
sf = resetSequence(sf);
```

Specify another video file to classify.

```
videoFilename = "vipmen.avi";
```

Create a VideoReader to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read a video sequence starting from the 131st frame, where the `"shaking hands"` action begins.

```
startSequenceIndex = 131;
sequenceRange = [startSequenceIndex, startSequenceIndex + sequenceLength - 1];
```

Classify the second video sequence.

```
for ii = sequenceRange
    videoFrame = read(reader,ii);
    sf = updateSequence(sf,frame);
end

[label2, score2] = classifySequence(sf)
```

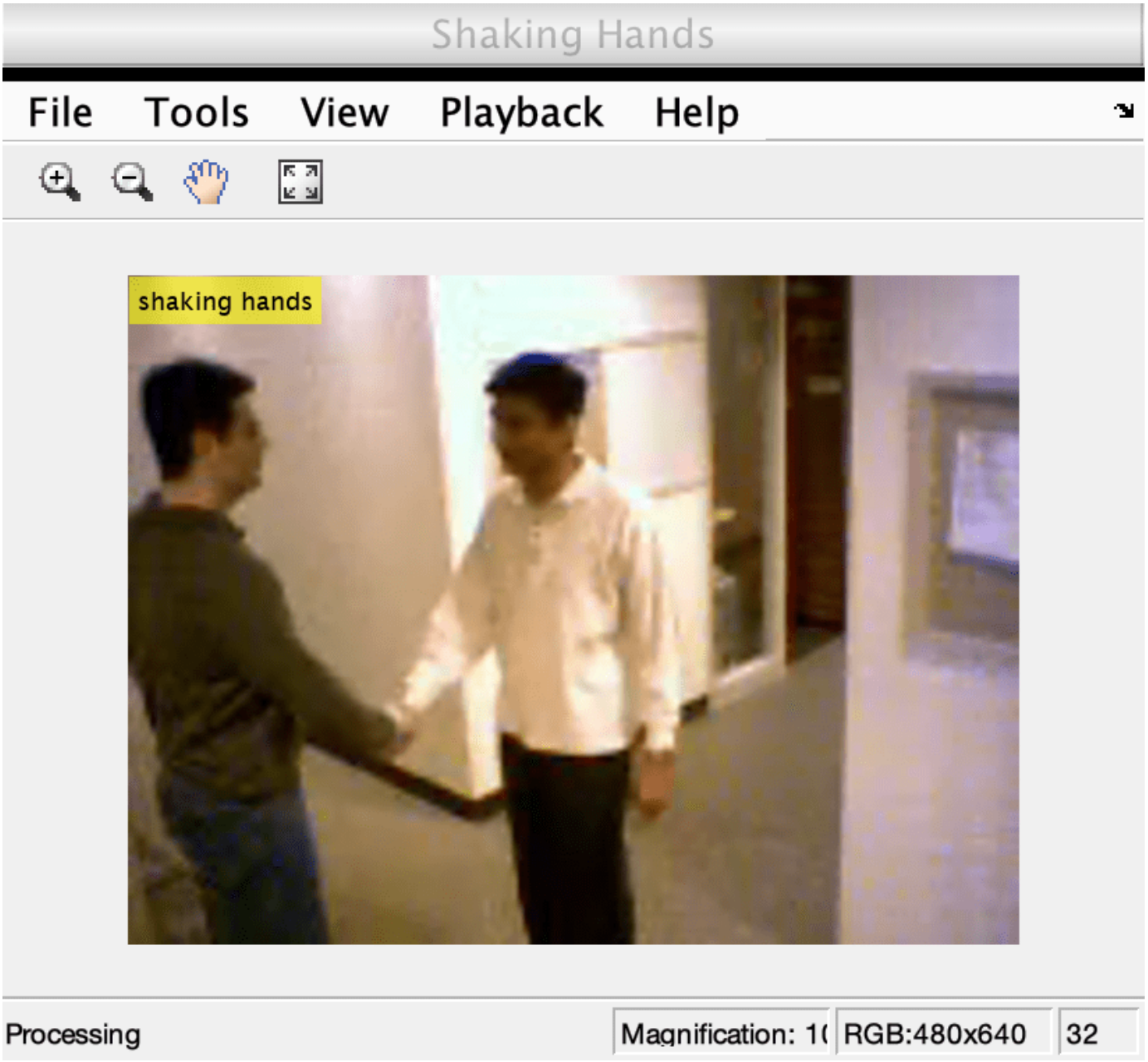
```
label2 = categorical
     shaking hands


score2 = single

0.0039
```

Display the classified label.

```
player = vision.VideoPlayer('Name','Shaking Hands');
for ii = sequenceRange
    frame = read(reader,ii);
    % Resize the frame by 4 times for display
    frame = imresize(frame, 4);
    frame = insertText(frame,[2,2], string(label2),'FontSize',18);
    step(player,frame);
end
```



Input Arguments

`classifier` — Classifier
`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier, specified as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

Output Arguments

`classifierReset` — Reset video classifier
`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Reset video classifier, returned as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifySequence` | `updateSequence` | `classifyVideoFile` | `predict` | `forward`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics

“Classify Streaming Webcam Video Using SlowFast Video Classifier”

resetSequence

Reset video and optical flow sequence properties for streaming video classification

Syntax

```
i3dReset = resetSequence(i3d)
```

Description

`i3dReset = resetSequence(i3d)` resets the video and optical flow sequences for classification. `i3d` is the classifier, specified as an `inflated3dVideoClassifier` object.

Examples

Classify Two Videos Using a Video Classifier

This example shows how to classify two different video sequences one after the other by reseting the video sequences of the video classifier using the `resetSequence` function. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

Load the SlowFast Video Classifier pretrained on the Kinetics-400 dataset.

```
sf = slowFastVideoClassifier();
```

Classify First Video Sequence

Specify the video file name.

```
videoFilename = "washingHands.avi";
```

Create a VideoReader to read the video frames.

```
reader = VideoReader(videoFilename);
```

Update the video classifier sequence with video frames before using `classifySequence`. The required number of frames is defined by the value of the 4th element of `InputSize` property of the video classifier.

```
sequenceLength = sf.InputSize(4);
```

Read a video sequence starting from the first frame.

```
startSequenceIndex = 1;
sequenceRange = [startSequenceIndex, startSequenceIndex + sequenceLength - 1];
```

Update the video classifier sequence with video frames.

```
for ii = sequenceRange
    videoFrame = read(reader, ii);
```

```
    sf = updateSequence(sf,videoFrame);
end
```

Classify the video sequence updated so far.

```
[label1, score1] = classifySequence(sf)
```

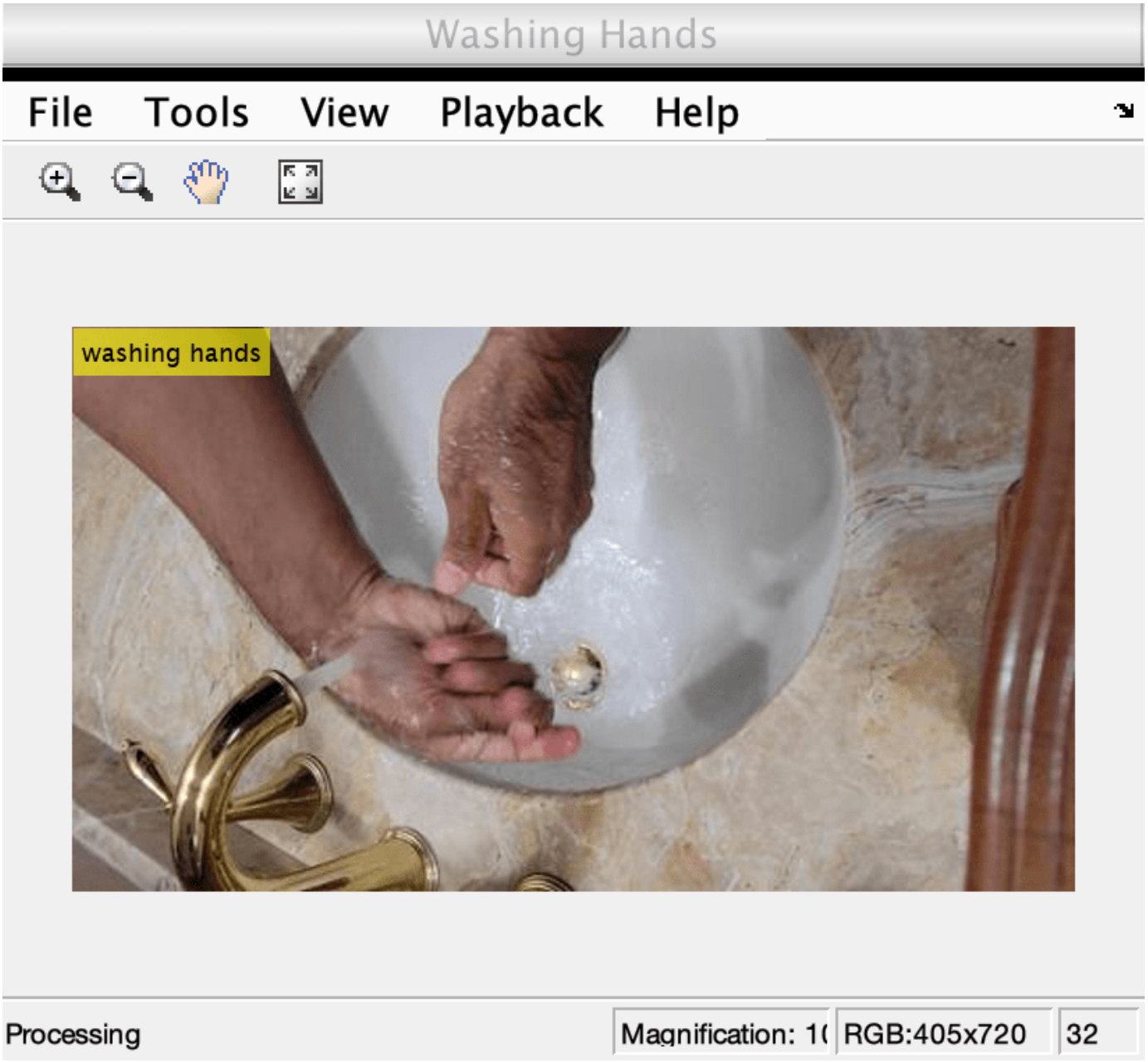
```
label1 = categorical
     washing hands


score1 = single

0.0031
```

Display the classified label.

```
player = vision.VideoPlayer('Name','Washing Hands');
for ii = sequenceRange
    frame = read(reader,ii);
    % Resize the frame by 1.5 times for display
    frame = imresize(frame,1.5);
    frame = insertText(frame,[2,2], string(label1),'FontSize',18);
    step(player,frame);
end
```



Classify Second Video Sequence

Reset the video sequence of the video classifier without creating a new video classifier.

```
sf = resetSequence(sf);
```

Specify another video file to classify.

```
videoFilename = "vipmen.avi";
```

Create a VideoReader to read the video frames.

```
reader = VideoReader(videoFilename);
```

Read a video sequence starting from the 131^{st} frame, where the `"shaking hands"` action begins.

```
startSequenceIndex = 131;
sequenceRange = [startSequenceIndex, startSequenceIndex + sequenceLength - 1];
```

Classify the second video sequence.

```
for ii = sequenceRange
    videoFrame = read(reader,ii);
    sf = updateSequence(sf,frame);
end

[label2, score2] = classifySequence(sf)
```

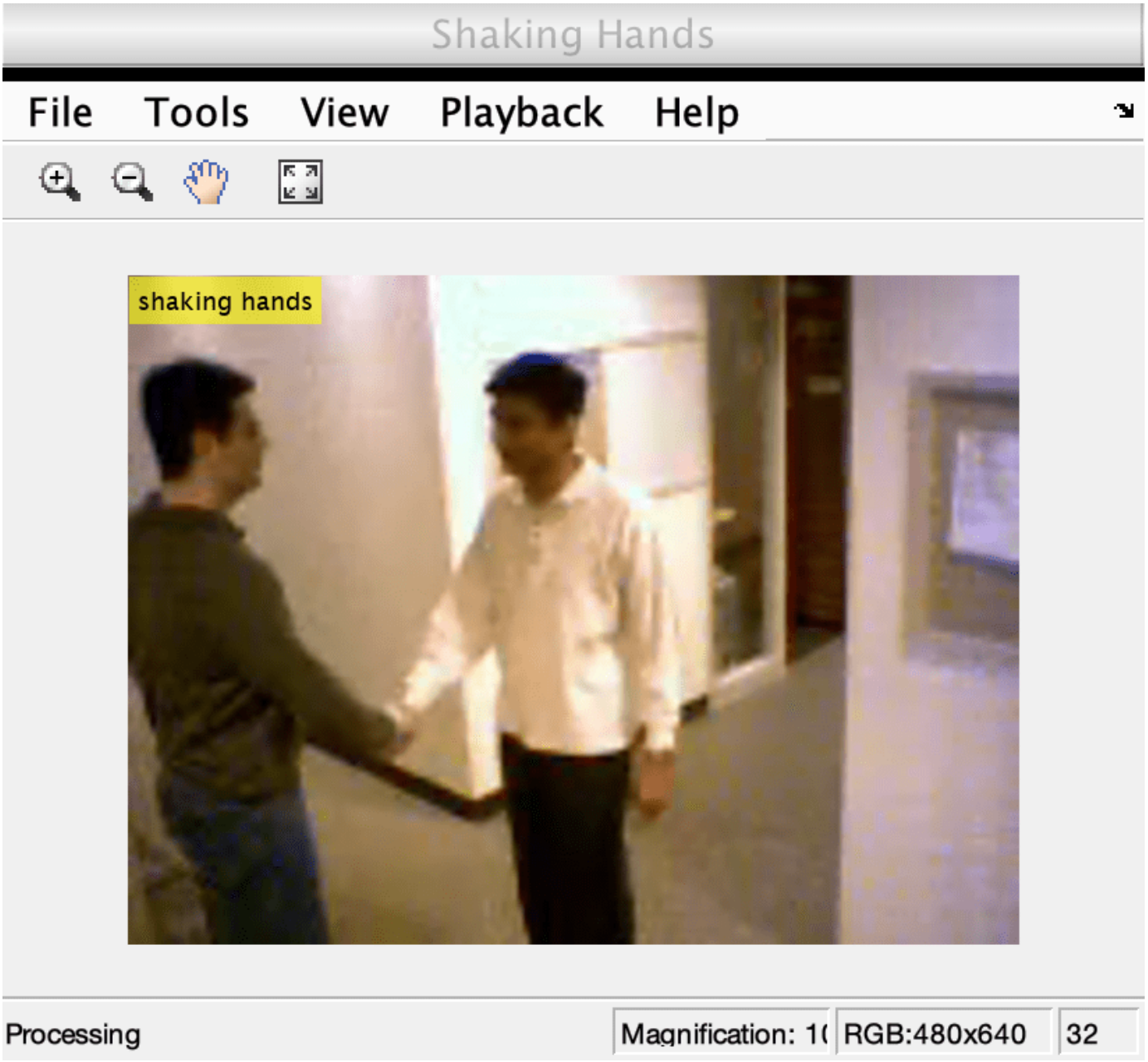
```
label2 = categorical
     shaking hands


score2 = single

0.0039
```

Display the classified label.

```
player = vision.VideoPlayer('Name','Shaking Hands');
for ii = sequenceRange
    frame = read(reader,ii);
    % Resize the frame by 4 times for display
    frame = imresize(frame, 4);
    frame = insertText(frame,[2,2], string(label2),'FontSize',18);
    step(player,frame);
end
```



Input Arguments

`i3d` — Classifier
`inflated3dVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier` object.

Output Arguments

`i3dReset` — Reset video classifier
`inflated3dVideoClassifier` object

Reset video classifier, returned as an `inflated3dVideoClassifier` object.

Version History

Introduced in R2021b

See Also

Functions

`sceneTimeRanges` | `writeVideoScenes` | `classifyVideoFile` | `classifySequence` | `updateSequence` | `forward` | `predict`

Objects

`r2plus1dVideoClassifier` | `slowFastVideoClassifier` | `inflated3dVideoClassifier`

Topics

“Classify Streaming Webcam Video Using SlowFast Video Classifier”

updateSequence, updateSequence

Update video sequence for classification

Syntax

```
classifierUpdated = updateSequence(classifier,videoFrame)
classifierUpdated = updateSequence(classifier,
videoFrame,ExecutionEnvironment=env)
```

Description

`classifierUpdated = updateSequence(classifier,videoFrame)` updates the video sequence with the video frame `videoFrame`, and outputs an updated classifier. The output contains the updated classifier object.

To classify the updated sequence, use the `classifySequence` object function.

The video classifier object maintains the sequence as a first-in first-out (FIFO) queue. The `InputSize` property of the classifier object specifies the number of frames in the queue.

`classifierUpdated = updateSequence(classifier,
videoFrame,ExecutionEnvironment=env)` specifies the hardware resources for running the classifier in addition to any combination of arguments from previous syntaxes, as one of these options:

- `"auto"` — Sets the execution environment to the GPU, if available. Otherwise the function sets it to the CPU.
- `"gpu"` — Sets the execution environment to the GPU. Usage of the GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Sets the execution environment to the CPU.

Examples

Classify Video Sequences in a Video File

This example shows how to classify video sequences in a video file using a SlowFast Video Classifier pretrained on the Kinetics-400 video activity recognition dataset. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

```
sf = slowFastVideoClassifier();
```

Set Up Video Player and Video Reader

Specify the video file name to stream video frames.

```
videoFilename = "pushup.mp4";
```

Create a VideoReader to read video.

```
reader = VideoReader(videoFilename);
```

Set up a video player.

```
player = vision.VideoPlayer;
```

Classify Video Sequences

Specify how frequently the classifier should be applied to incoming video frames.

```
classifyInterval = 10;
```

A value of 10 balances runtime performance against classification performance. Increase this value to improve runtime performance at the cost of missing actions from the video file.

Obtain the sequence length of the SlowFast Video Classifier. Classify only after capturing at least `sequenceLength` number of frames from the video file.

```
sequenceLength = sf.InputSize(4);
```

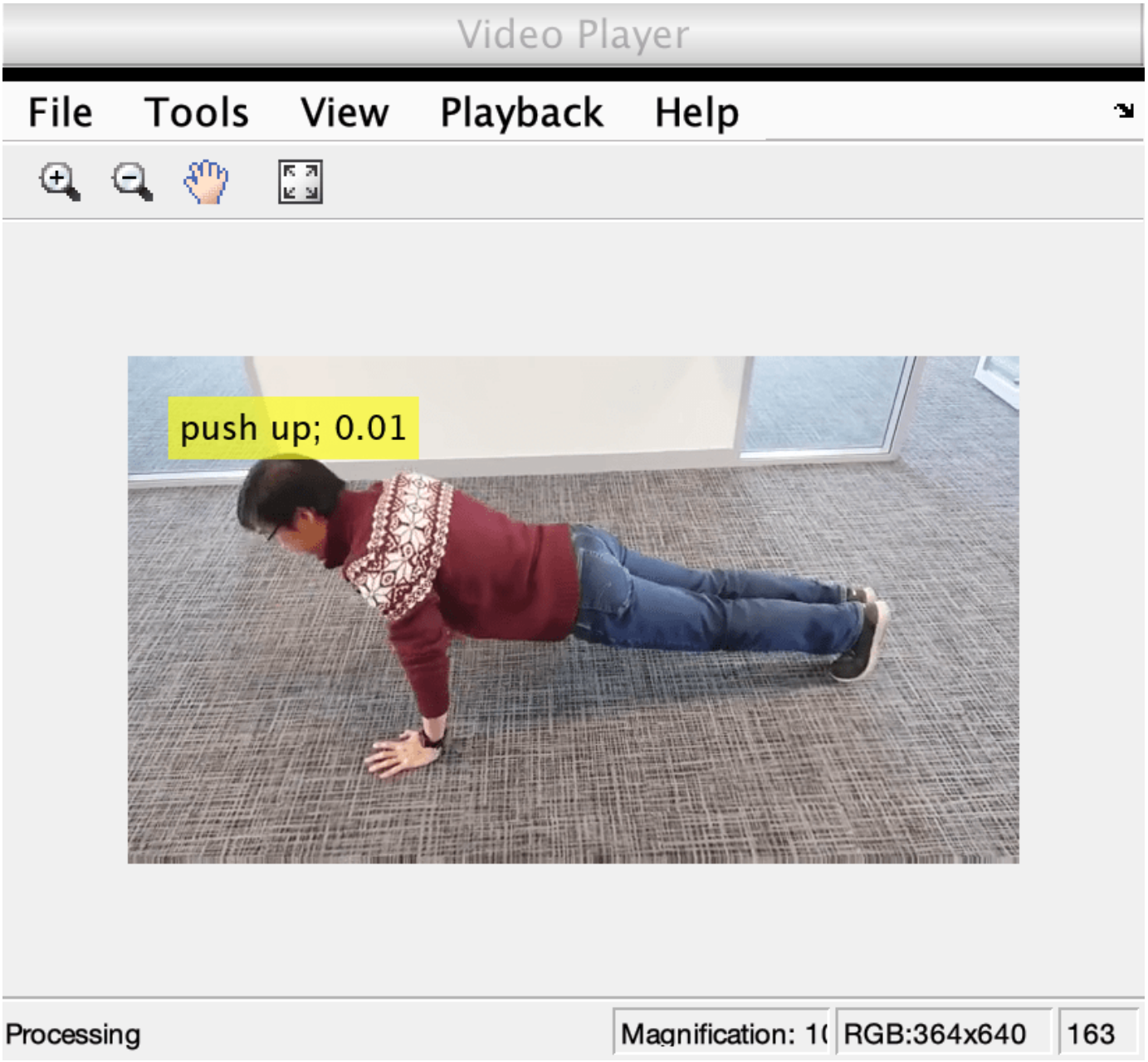
Read video frames using the `hasFrame` and `readFrame` functions of the `VideoReader`. Using the `updateSequence` function update the video classifier's sequence. Using the `classifySequence` function classify the updated sequence.

```
numFrames = 0;
text = "";

while hasFrame(reader)
    frame = readFrame(reader);
    numFrames = numFrames + 1;

    % Update the sequence with the next video frame.
    sf = updateSequence(sf,frame);

    % Classify the sequence only at every classifyInterval number of frames.
    if mod(numFrames, classifyInterval) == 0 && numFrames >= sequenceLength
        [label,score] = classifySequence(sf);
        text = string(label) + "; " + num2str(score, "%0.2f");
    end
    frame = insertText(frame,[30,30],text,'FontSize',24);
    step(player,frame);
end
```



Input Arguments

`classifier` — Classifier object
`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Classifier object, specified as `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

`videoFrame` — Video frame
numeric array

Video frame for updating the classifier, specified as an *H*-by-*W*-by-*C* numeric array. *H*, *W*, and *C* represent the height, width, and number of channels, respectively. The number of channels must match the number of channels set by the `InputSize` property of the classifier object.

Output Arguments

`classifierUpdated` — Updated video classifier
`r2plus1dVideoClassifier` object | `slowFastVideoClassifier` object

Updated video classifier, returned as a `r2plus1dVideoClassifier` or `slowFastVideoClassifier` object.

Version History

Introduced in R2021b

See Also

Objects
`inflated3dVideoClassifier` | `slowFastVideoClassifier` | `r2plus1dVideoClassifier`

Functions
`classifySequence` | `resetSequence` | `classifyVideoFile` | `forward` | `predict`

Topics
“Classify Streaming Webcam Video Using SlowFast Video Classifier”

updateSequence

Update video or optical flow sequence for classification

Syntax

```
i3dUpdated = updateSequence(i3d,videoFrame)
i3dUpdated = udpateSequence(i3d,videoFrame,ExecutionEnvironment=env)
```

Description

`i3dUpdated = updateSequence(i3d,videoFrame)` updates the video or optical flow sequence with the video frame `videoFrame`, and outputs the updated Inflated-3D (I3D) video classifier `i3d` classifier. The output contains the updated classifier object.

To classify the updated sequence, use the `classifySequence` object function.

The video classifier object maintains the sequence as a first-in first-out (FIFO) queue. The `InputSize` property of the classifier object specifies the number of frames in the queue.

`i3dUpdated = udpateSequence(i3d,videoFrame,ExecutionEnvironment=env)` specifies the hardware resources for running the classifier in addition to any combination of arguments from previous syntaxes, as one of these options:

- `"auto"` — Sets the execution environment to the GPU, if available. Otherwise the function sets it to the CPU.
- `"gpu"` — Sets the execution environment to the GPU. Usage of the GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Sets the execution environment to the CPU.

Examples

Classify Video Sequences in a Video File

This example shows how to classify video sequences in a video file using a SlowFast Video Classifier pretrained on the Kinetics-400 video activity recognition dataset. To learn more about how to train a video classifier network for your dataset, see “Gesture Recognition using Videos and Deep Learning”.

Load SlowFast Video Classifier

```
sf = slowFastVideoClassifier();
```

Set Up Video Player and Video Reader

Specify the video file name to stream video frames.

```
videoFilename = "pushup.mp4";
```

Create a VideoReader to read video.

```
reader = VideoReader(videoFilename);
```

Set up a video player.

```
player = vision.VideoPlayer;
```

Classify Video Sequences

Specify how frequently the classifier should be applied to incoming video frames.

```
classifyInterval = 10;
```

A value of 10 balances runtime performance against classification performance. Increase this value to improve runtime performance at the cost of missing actions from the video file.

Obtain the sequence length of the SlowFast Video Classifier. Classify only after capturing at least `sequenceLength` number of frames from the video file.

```
sequenceLength = sf.InputSize(4);
```

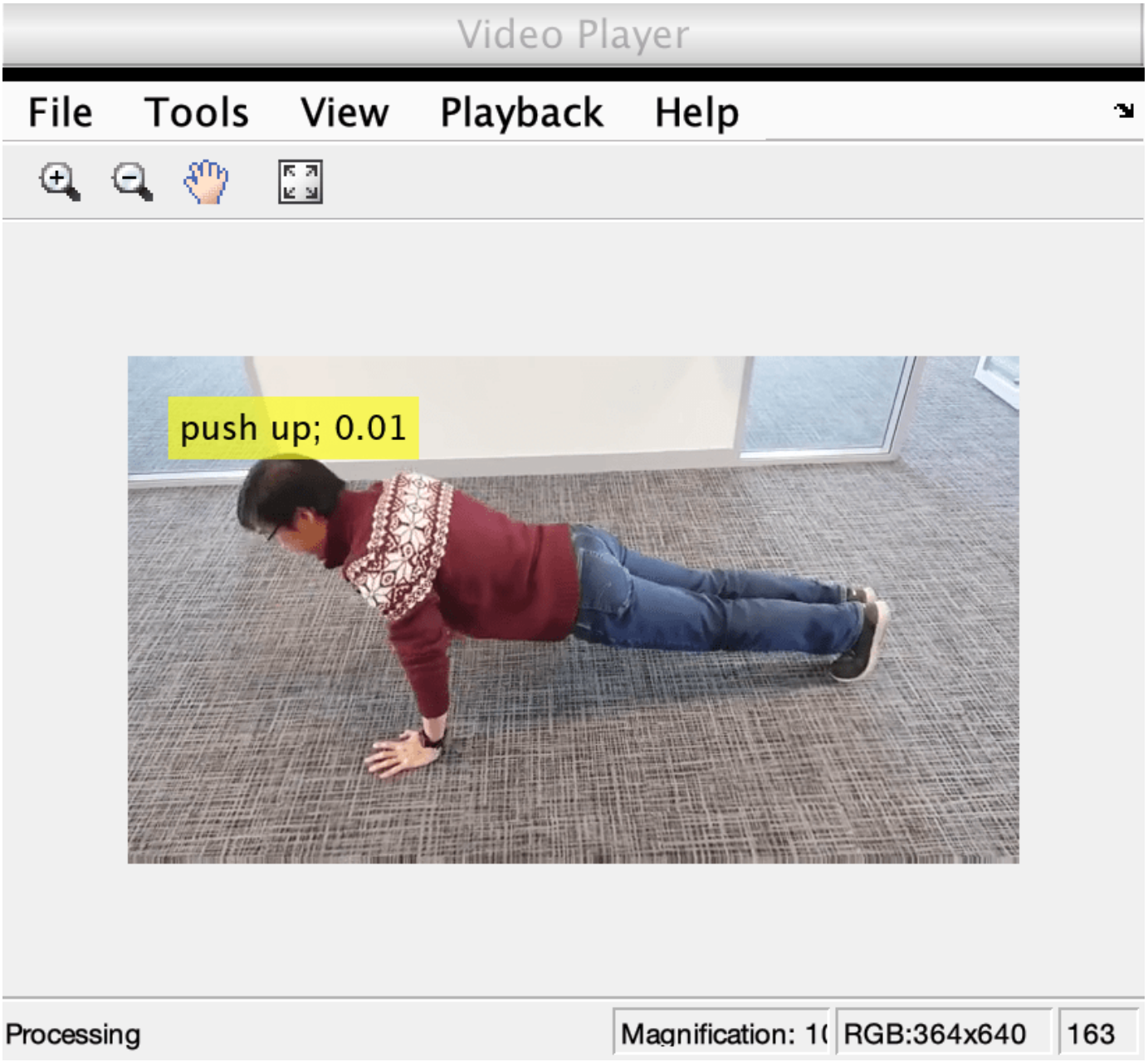
Read video frames using the `hasFrame` and `readFrame` functions of the `VideoReader`. Using the `updateSequence` function update the video classifier's sequence. Using the `classifySequence` function classify the updated sequence.

```
numFrames = 0;
text = "";

while hasFrame(reader)
    frame = readFrame(reader);
    numFrames = numFrames + 1;

    % Update the sequence with the next video frame.
    sf = updateSequence(sf,frame);

    % Classify the sequence only at every classifyInterval number of frames.
    if mod(numFrames, classifyInterval) == 0 && numFrames >= sequenceLength
        [label,score] = classifySequence(sf);
        text = string(label) + "; " + num2str(score, "%0.2f");
    end
    frame = insertText(frame,[30,30],text,'FontSize',24);
    step(player,frame);
end
```



Input Arguments

`i3d` — Classifier
`inflated3dVideoClassifier` object

Classifier, specified as an `inflated3dVideoClassifier` object.

`videoFrame` — Video frame
numeric array

Video frame for updating the classifier, specified as an *H*-by-*W*-by-*C* numeric array. *H*, *W*, and *C* represent the height, width, and number of channels, respectively. The number of channels must match the number of channels set by the `InputSize` property of the classifier object.

Output Arguments

`i3dUpdated` — Updated video classifier
`inflated3dVideoClassifier` object

Updated video classifier, returned as an `inflated3dVideoClassifier` object.

Version History

Introduced in R2021b

See Also

Objects
`inflated3dVideoClassifier` | `slowFastVideoClassifier` | `r2plus1dVideoClassifier`

Functions
`classifyVideoFile` | `forward` | `predict` | `resetSequence` | `classifySequence`

vision.calibration.PatternDetector Class

Namespace: `vision.calibration`

Interface for defining custom planar pattern detectors

Description

The `vision.calibration.PatternDetector` class specifies the interface for defining custom planar pattern detectors for the Camera Calibrator and Stereo Camera Calibrator apps, as well as on the MATLAB command line.

To define a custom pattern detector, you must construct a class that inherits from the `vision.OpenPatternDetectorTemplate` class. This is an abstract class that defines the methods and properties for detecting calibration pattern keypoints in the images, invoked during the calibration process, and to generate corresponding points in the world coordinates.

You can define a custom pattern detector for single camera calibration, starting with a template, by typing this command at the MATLAB prompt:

```
vision.OpenPatternDetectorTemplate("monocular")
```

You can define a custom pattern detector for stereo camera calibration, starting with a template, by typing the following at the MATLAB prompt:

```
vision.OpenPatternDetectorTemplate("stereo")
```

The `vision.calibration.PatternDetector` class is a `handle` class.

Class Attributes

`Abstract`	`true`

For information on class attributes, see “Class Attributes”.

Properties

Clients of the `PatternDetector` class specify these predefined properties.

`Name` — Pattern detector name
character vector | string scalar

Pattern detector name, specified as a character vector or a string scalar.

`WorldUnits` — World point units
character vector | string scalar

World point units, specified as a character vector or a string scalar. `WorldUnits` specifies the unit of measure for the pattern keypoints in world coordinates.

`Panel` — Properties panel user-defined UI components
`uipanel` object

Properties panel user-defined UI components for displaying and specifying detector properties, specified as a `uipanel` object. To set this property, use the `propertiesPanel` function with a `uipanel` object, which the `PatternDetector` invokes.

Methods

Public Methods

Clients of a `PatternDetector` implement these user-defined functions to define execution of the algorithm. For more information on creating a custom detection pattern, see “Camera Calibration Using Custom Planar Calibration Patterns”.

generateWorldPoints	Generate world coordinates for keypoints of custom planar pattern
propertiesPanel	Define properties panel for custom pattern detector
drawImageAxesLabels	Display location and orientation of axes labels
detectPatternPoints	Detect calibration pattern keypoints in images containing custom patterns

Version History

Introduced in R2021b

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`generateCircleGridPoints` | `detectCircleGridPoints` | `estimateCameraParameters` | `detectCheckerboardPoints` | `generateCheckerboardPoints`

Objects

`cameraParameters` | `stereoParameters`

Topics

“Camera Calibration Using Custom Planar Calibration Patterns”

propertiesPanel

Namespace: `vision.calibration`

Define properties panel for custom pattern detector

Syntax

```
propertiesPanel(detectorObj,panel)
```

Description

`propertiesPanel(detectorObj,panel)` displays properties contained in the detector object `detecterObj` in the parent user interface (UI) panel `panel`.

The Camera Calibrator or the Stereo Camera Calibrator app invokes the `propertiesPanel` object function when you select a custom pattern from one of the available patterns. Use this object function to define UI elements in the **Properties** section of the Image and Pattern Properties dialog box of the Camera Calibrator or the **Load Stereo Images** dialog box of the Stereo Camera Calibrator app.

Examples

Custom Pattern Properties Panel for Camera Calibrator

The `propertiesPanel` function enables you to customize the properties panel of a calibrator app by creating UI elements and defining their behavior. This `propertiesPanel` code snippet creates a properties panel related to a checkerboard detection:

```
function propertiesPanel(this,panel)
  %-------------------------------------------------------------
  % UI components for square size selector
  %-------------------------------------------------------------
  this.Panel = panel;

  % Label
      position = [90, 40, 185, 20];
      labelText = 'Square Size (in millimeters): ';
      uicontrol('Parent',this.Panel,'Style','text','FontUnits', ...
                'pixels',FontSize', 12,'Position',position, ...
                'HorizontalAlignment','left','String',labelText);

  % Editbox
      position = [275, 37, 50, 25];
      initSquareSize = 25;
      squareSizeEditBox = uicontrol('Parent',this.Panel, ...
                 'Style','edit','FontUnits','pixels','FontSize',15, ...
                 'String',initSquareSize,'Position', position, ...
                 'Callback',@(~, ~) doSquareSizeChanged(this));

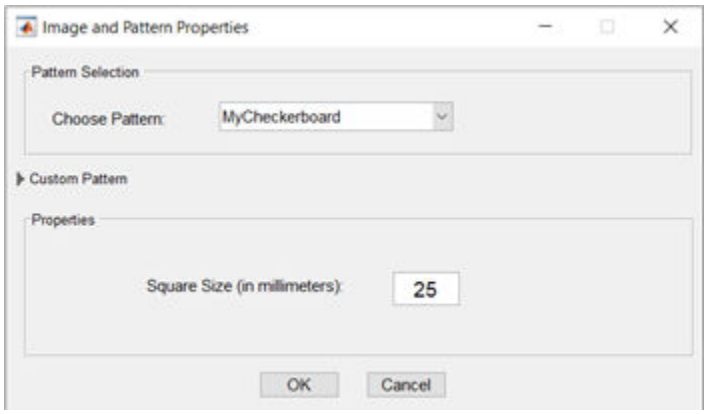
  % Initialize property values
      this.SquareSize = str2double(get(squareSizeEditBox,'String'));
```

```
end

%-------------------------------------------------------------------------
function doSquareSizeChanged(this)
   this.SquareSize = str2double(get(this.SquareSizeEditBox,'String'));

   if this.SquareSize <= 0 || isnan(this.SquareSize)
      errordlg('Square size must be a numeric value greater than zero.');
   end
end
```

This function produces this UI pane:



Input Arguments

`detectorObj` — Detector object

`vision.calibration.PatternDetector` object

Detector object, specified as a single or stereo `vision.calibration.PatternDetector` object.

`panel` — User-defined UI components

`uipanel` object

User-defined UI components for displaying and specifying detector properties, specified as a `uipanel` object. When you invoke the `propertiesPanel` function with a `uipanel` object.

Version History

Introduced in R2021b

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`generateWorldPoints` | `drawImageAxesLabels` | `detectPatternPoints`

Objects
`vision.calibration.PatternDetector`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”

drawImageAxesLabels

Namespace: `vision.calibration`

Display location and orientation of axes labels

Syntax

```
[originLabel,xLabel,yLabel] = drawImageAxesLabels(detectorObj,imagePoints)
```

Description

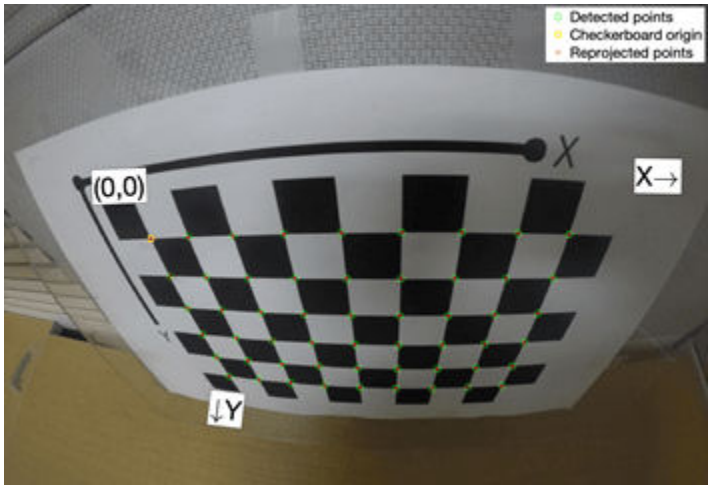
`[originLabel,xLabel,yLabel] = drawImageAxesLabels(detectorObj,imagePoints)` displays the orientation and location of the origin `originLabel` and the *X*- and *Y*-axis labels `xLabel` and `yLabel` of the circle grid pattern detected in the images rendered in the Camera Calibrator and Stereo Camera Calibrator apps.

If you do not implement this function, or if the returned values are empty, then the function does not render origin or axes labels in the image.

Examples

Label Image in Calibrator App

Use the `drawImageAxesLabels` function to identify and label the orientation and location for the origin and the *X*- and *Y*-axis labels of a detected pattern grid in an image. For example:



Calculate Orientation and Location for Pattern Origin

Use this function template, which includes the use of the `drawImageAxesLabels` object function, to calculate the orientation and location of the origin of the detected calibration pattern in an image.

The function omits the *X*- and *Y*-labels. The function includes both programmatic and app workflows for detecting the points.

```
function [originLabel,xLabel,yLabel] = drawImageAxesLabels(this,imagePoints)

    numBoardRows = this.BoardSize(1) - 1;
    numBoardCols = this.BoardSize(2) - 1;

    % Reshape checkerboard corners to boardSize-shaped array
    boardCoordsX = reshape(imagePoints(:,1),[numBoardRows numBoardCols]);
    boardCoordsY = reshape(imagePoints(:,2),[numBoardRows numBoardCols]);
    boardCoords = cat(3,boardCoordsX,boardCoordsY);

    % Origin label (check if the origin location is inside the image)
    if ~isnan(boardCoordsX(1,1))
        p1 = boardCoords(1,1,:);

        refPointIdx = find(~isnan(boardCoordsX(:,1)),2);
        p2 = boardCoords(refPointIdx(2),1,:);

        refPointIdx = find(~isnan(boardCoordsX(1,:)),2);
        p3 = boardCoords(1,refPointIdx(2),:);

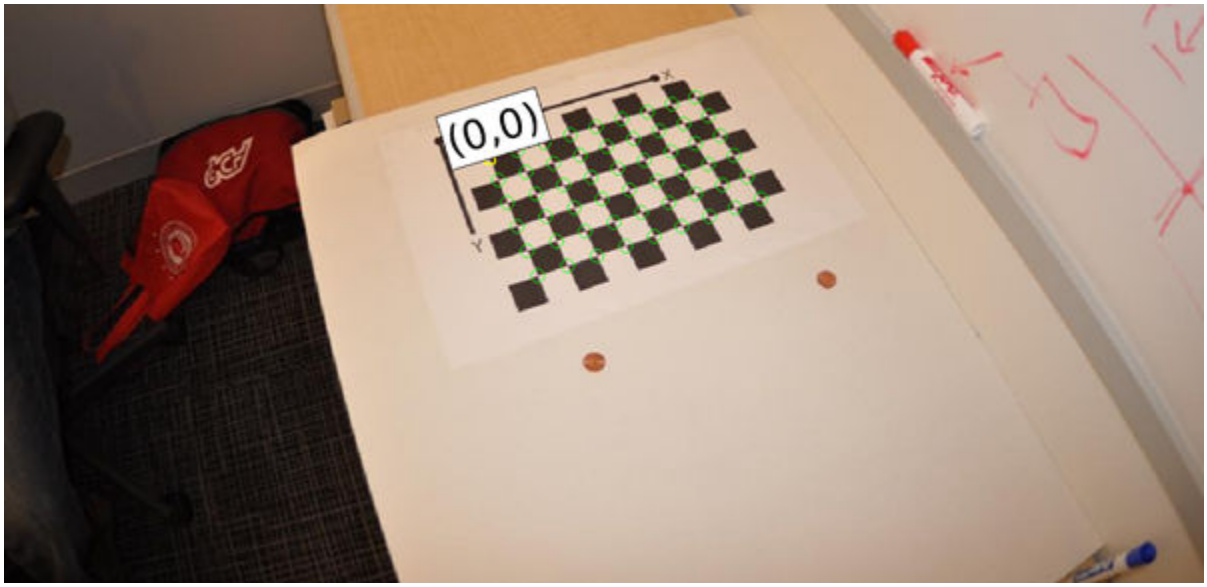
        [loc, theta] = getAxesLabelPosition(p1,p2,p3);

        originLabel.Location = loc;
        originLabel.Orientation = theta;
    else
        originLabel = struct;
    end

    % X-axis and Y-axis labels
    xLabel = struct('Orientation',[],'Location',[]);
    yLabel = struct('Orientation',[],'Location',[]);

    %-------------------------------------------------------------
    % p1+v
    %  \
    %   \     v1
    %    p1 ------ p2
    %    |
    % v2 |
    %    |
    %    p3
    function [loc, theta] = getAxesLabelPosition(p1,p2,p3)
        v1 = p3 - p1;
        theta = -atan2d(v1(2),v1(1));

        v2 = p2 - p1;
        v = -v1 - v2;
        d = hypot(v(1),v(2));
        minDist = 40;
        if d < minDist
            v = (v/d) * minDist;
        end
        loc = p1 + v;
    end
end
```



Input Arguments

`detectorObj` — Detector object
`vision.calibration.PatternDetector` object

Detector object, specified as a single or stereo `vision.calibration.PatternDetector` object.

`imagePoints` — Image coordinates of detected circle grid
M-by-2 array

Image coordinates of the detected circle grid, returned as an *M*-by-2 array of *x*,*y* coordinates.

Output Arguments

`originLabel` — Orientation and location for pattern origin label
`structure`

Orientation and location of the pattern origin label, returned as a structure with two fields. The `Orientation` field contains the rotation of the origin label in the image frame, specified in degrees. The `Location` field contains the *x*,*y* location of the origin label in the image frame.

`xLabel` — Orientation and location of *X*-axis label
`structure`

Orientation and location of the *X*-axis label, returned as a structure with two fields. The `Orientation` field contains the rotation of the axis label in the image frame, specified in degrees. The `Location` field contains the *x*,*y* location of the axis label in the image frame.

`yLabel` — Orientation and location for *Y*- axis label
`structure`

Orientation and location of the *Y*-axis label, returned as a structure with two fields. The `Orientation` field contains the rotation of the axis label in the image frame, specified in degrees. The `Location` field contains the *x*,*y* location of the axis label in the image frame.

Version History

Introduced in R2021b

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`generateWorldPoints` | `propertiesPanel` | `detectPatternPoints`

Objects

`cameraParameters` | `stereoParameters` | `vision.calibration.PatternDetector`

Topics

“Camera Calibration Using Custom Planar Calibration Patterns”

generateWorldPoints

Namespace: `vision.calibration`

Generate world coordinates for keypoints of custom planar pattern

Syntax

```
worldPoints = generateWorldPoints(detectorObj)
worldPoints = generateWorldPoints(detectorObj,varargin)
```

Description

`worldPoints = generateWorldPoints(detectorObj)` generates world coordinates for the detected keypoints of the custom planar calibration pattern.

`worldPoints = generateWorldPoints(detectorObj,varargin)` uses the `varargin` input to parse any additional parameters or options necessary to generate world points. For example, you can provide the parameters or options obtained from the properties pane of your calibrator app.

Examples

Generate World Points for Checkerboard Pattern

Use this function template, which includes the use of the `generateCheckerboardPoints` object function, to generate world points for a checkerboard pattern generator. The function includes both programmatic and app workflows for generating the world (image) points. The function uses name-value arguments for board and square size to set the corresponding fields in the **Properties** pane of the calibrator app.

```
function worldPoints = generateWorldPoints(this,varargin)

    if nargin > 1 % Command Line workflow
        parser = inputParser;
        parser.addParameter('SquareSize',25,@checkSquareSize);
        parser.addParameter('BoardSize',this.BoardSize,@checkBoardSize);
        parser.parse(varargin{:});

        boardSize = parser.Results.BoardSize;
        squareSize = parser.Results.SquareSize;
    else % Calibrator App workflow
        boardSize = this.BoardSize;
        squareSize = this.SquareSize;
    end

    worldPoints = generateCheckerboardPoints(boardSize,squareSize);

    %-----------------------------------------------------------------
    % Validation functions for command line workflow
    %-----------------------------------------------------------------
    function tf = checkSquareSize(squareSize)
```

```
            validateattributes(squareSize, {'numeric'}, ...
                {'scalar','positive','finite','nonsparse'},mfilename,'SquareSize');
            tf = true;
        end

        %-------------------------------------------------------------
        function tf = checkBoardSize(boardSize)
            validateattributes(boardSize,{'numeric'},...
                {'nonempty','vector','numel',2,'integer','positive','>=', 3},...
                mfilename,'BoardSize');
            tf = true;
        end
    end
```

Input Arguments

`detectorObj` — Pattern detector

`vision.calibration.PatternDetector` object.

Pattern detector, specified as a single or stereo `vision.calibration.PatternDetector` object.

`varargin` — Variable number of inputs

1-by-*N* cell array

Variable number of inputs, specified as a 1-by-*N* cell array, where *N* is the number of inputs to the function after `detectorObj`. For more details about using this input, see `varargin`.

Output Arguments

`worldPoints` — Image coordinates of detected key points

M-by-2 array

Image coordinates of detected keypoints in the calibration pattern, returned as an *M*-by-2 array of *x*,*y* coordinates. *M* is the number of pattern keypoints detected in the calibration images by using the `detectPatternPoints` function.

Version History

Introduced in R2021b

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`generateCircleGridPoints` | `detectPatternPoints` | `detectCheckerboardPoints` | `propertiesPanel` | `drawImageAxesLabels` | `detectPatternPoints`

Objects

`cameraParameters` | `stereoParameters` | `vision.calibration.PatternDetector`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”

detectPatternPoints

Namespace: `vision.calibration`

Detect calibration pattern keypoints in images containing custom patterns

Syntax

```
[imagePoints,imagesUsed] = detectPatternPoints(detectorObj,imageFileNames)
[imagePoints,pairsUsed] = detectPatternPoints(detectorObj,imageFileNames1,
imageFileNames2)

[___] = detectPatternPoints(___,varargin)
```

Description

`[imagePoints,imagesUsed] = detectPatternPoints(detectorObj,imageFileNames)` detects the calibration pattern keypoints in the image files, specified by `imageFileNames`. The function also indicates in which of the images it detects the calibration pattern.

`[imagePoints,pairsUsed] = detectPatternPoints(detectorObj,imageFileNames1, imageFileNames2)` detects the calibration pattern keypoints in the stereo pairs of image files specified in `imageFileNames1` and `imageFileNames2`. The function also indicates in which of the image pairs it detects the calibration pattern.

`[___] = detectPatternPoints(___,varargin)` parses any parameters or options needed for detection. For example, you can provide the parameters or options obtained from the properties panel of your calibrator app while using it from the command line.

Examples

Detect Pattern Points for a Checkerboard Pattern

Use this function template, which includes the use of the `detectPatternPoints` object function, to detect the keypoints of a calibration pattern. The function includes both programmatic and app workflows for detecting the points.

```
function [imagePoints,imagesUsed] = detectPatternPoints(this,imageFileNames,varargin)
    parser = inputParser;
    parser.addParameter('HighDistortion',false,@checkHighDistortion);
    parser.parse(varargin{:});
    highDistortion = parser.Results.HighDistortion;

    % Use 'IsDistortionHigh' member value when the default value is
    % being used for HighDistortion
    if ismember('HighDistortion',parser.UsingDefaults)
       highDistortion = this.IsDistortionHigh;
    end

    [imagePoints,boardSize,imagesUsed] = detectCheckerboardPoints( ...
        imageFileNames,HighDistortion=highDistortion);
```

```
    this.BoardSize = boardSize;

    %-------------------------------------------------------------
    function tf = checkHighDistortion(highDistortion)
        validateattributes(highDistortion,{'logical','numeric'}, ...
            {'scalar','binary'},mfilename,'HighDistortion');
        tf = true;
    end
end
```

Input Arguments

`detectorObj` — Detector object

`vision.calibration.PatternDetector` object

Detector object, specified as a single or stereo `vision.calibration.PatternDetector` object.

`imageFileNames` — Image file names

cell array of character vectors | vector of strings

Image file names, specified as a cell array of character vectors or a vector of strings.

`imageFileNames1` — Stereo image file names for camera one

cell array of character vectors | array of string vectors

Stereo image file names for camera one, specified as a cell array of character vectors or an array of strings.

`imageFileNames2` — Stereo image file names for camera two

cell array of character vectors | array of string vectors

Stereo image file names for camera two, specified as a cell array of character vectors or an array of strings.

`varargin` — Variable number of inputs

1-by-*N* cell array

Variable number of inputs, specified as a 1-by-*N* cell array, where *N* is the number of inputs to the function receives after the explicitly declared inputs. For more details about using this input, see `varargin`.

Output Arguments

`imagePoints` — Image coordinates of detected pattern keypoints

M-by-2- by-*numImages* | *M*-by-2-by-*numPairs*-by-2 array

Image *x*-*y* coordinates of detected pattern keypoints, returned as an *M*-by-2-by-*numImages* array for single camera calibration or an *M*-by-2-by-*numPairs*-by-2 array for stereo camera calibration. *numImages* is the number of images in which a circle grid is detected. *numPairs* is the number of image pairs in which a circle grid is detected. For stereo images, `imagePoints(:,:,:,1)` returns the pattern keypoints from the first set of images, and `imagePoints(:,:,:,2)` returns the pattern keypoints from the second set of images. If the function cannot detect the complete pattern, then it returns a partially detected pattern with `[NaN,NaN]` for those *x*-*y* coordinates that could not be detected.

`imagesUsed` — Pattern detection flag
logical vector

Pattern detection flag, returned as a logical vector. A value of `true` for an element of the `imagesUsed` vector indicates that the function has detected the pattern in the corresponding image.

`pairsUsed` — Stereo pair pattern detection flag
logical vector

Stereo pair pattern detection flag, returned as a logical vector. A value of `true` for an element of the `pairsUsed` vector indicates that the function has detected the pattern in the corresponding image pair.

Version History

Introduced in R2021b

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`estimateCameraParameters` | `detectCheckerboardPoints` | `generateWorldPoints` | `propertiesPanel` | `detectCircleGridPoints` | `readAprilTag`

Objects
`cameraParameters` | `stereoParameters` | `vision.calibration.PatternDetector`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”

scanContextLoopDetector

Detect loop closures using scan context descriptors

Description

`scanContextLoopDetector` detects loop closures in point cloud data using global scan context descriptors. A scan context descriptor encodes a point cloud view into a 2-D descriptor. `scanContextLoopDetector` stores the descriptors with a corresponding view identifier to associate the detected loop closures to a view in a `pcviewset` for map building.

Creation

Syntax

```
loopDetector = scanContextLoopDetector()
```

Description

`loopDetector = scanContextLoopDetector()` creates a default `scanContextLoopDetector` object. Use the `addDescriptor` object function to add descriptors to the loop detector. Use the `detectLoop` object function to detect loop closures.

Properties

`ViewIds` — View identifiers
P-element vector

This property is read-only.

View identifiers, specified as a *P*-element vector of unique positive integers, where *P* is the number of descriptors added to the detector.

`Descriptors` — Scan context descriptors
M-by-*N*-by-*P* array

This property is read-only.

Scan context descriptors, specified as an *M*-by-*N*-by-*P* array. *M* is the number of radial bins and *N* is the number of azimuthal bins in the descriptor. *P* is the number of descriptors in the detector.

Object Functions

addDescriptor	Add descriptor to the loop closure detector
deleteDescriptor	Delete descriptors from the loop closure detector
detectLoop	Detect loop closures

Examples

Detect Loop Closure Against Recently Added Descriptor

Create a loop closure detector.

```
loopDetector = scanContextLoopDetector;
```

Create a Velodyne® PCAP file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the point clouds and extract the scan context descriptor from each point cloud scan. Add the descriptors to the detector.

```
for viewId = 1:10
    ptCloud = readFrame(veloReader,viewId);
    descriptor = scanContextDescriptor(ptCloud);
    addDescriptor(loopDetector,viewId,descriptor);
end
```

Check if the next point cloud can be classified as a loop closure detection without excluding any recently added descriptors.

```
viewId = viewId + 1;
ptCloud = readFrame(veloReader,viewId);
descriptor = scanContextDescriptor(ptCloud);
[loopViewId,dists] = detectLoop(loopDetector,descriptor,'NumExcludedDescriptors',0)
```

```
loopViewId = uint32

10

dists = single

0.0858
```

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pointCloud` | `pcviewset` | `pcmapndt`

Functions
`scanContextDescriptor` | `scanContextDistance` | `pcregisterndt`

Topics
“Implement Point Cloud SLAM in MATLAB”
“Build a Map from Lidar Data Using SLAM”

addDescriptor

Add descriptor to the loop closure detector

Syntax

```
addDescriptor(loopDetector,viewID,descriptor)
```

Description

`addDescriptor(loopDetector,viewID,descriptor)` adds the scan context descriptor `descriptor` that corresponds to the view identifier `viewID` to the loop closure detector `loopDetector`.

Compute descriptors to add to the loop closure detector over the same radial range, with the same number of radial and azimuthal bins. Use the `viewID` input to associate descriptors with a view in `pcviewset`.

Examples

Add Descriptor to Loop Closure Detector

Create a loop closure detector.

```
loopDetector = scanContextLoopDetector;
```

Load point cloud data into the workspace.

```
data = load("drivingLidarPoints.mat");
ptCloud = data.ptCloud;
```

Extract a scan context descriptor from the point cloud data.

```
descriptor = scanContextDescriptor(ptCloud);
```

Add the descriptor to the loop closure detector.

```
viewId = 1;
addDescriptor(loopDetector,viewId,descriptor);
```

Input Arguments

`loopDetector` — Loop closure detector
`scanContextLoopDetector` object

Loop closure detector, specified as a `scanContextLoopDetector` object.

`viewID` — View identifier
positive integer

View identifier, specified as a positive integer. You can use the same `viewID` to associate descriptors with a view in a `pcviewset`.

`descriptor` — Scan context descriptor
M-by-*N* matrix

Scan context descriptor, specified as an *M*-by-*N* matrix, where *M* is the number of radial bins and *N* is the number of azimuthal bins.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pointCloud` | `scanContextLoopDetector`

Functions
`scanContextDescriptor` | `scanContextDistance` | `detectLoop` | `deleteDescriptor`

Topics
“Implement Point Cloud SLAM in MATLAB”
“Build a Map from Lidar Data Using SLAM”

deleteDescriptor

Delete descriptors from the loop closure detector

Syntax

```
deleteDescriptor(loopDetector,viewIDs)
```

Description

`deleteDescriptor(loopDetector,viewIDs)` deletes the descriptors that correspond to the view identifiers `viewIDs` from the loop closure detector `loopDetector`. `viewIDs` is a vector of positive integers.

Examples

Delete Descriptor from Loop Closure Detector

Create a loop closure detector.

```
loopDetector = scanContextLoopDetector;
```

Create a Velodyne® PCAP file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the first and second point cloud scans.

```
ptCloud1 = readFrame(veloReader,1);
ptCloud2 = readFrame(veloReader,2);
```

Extract the descriptor for the first point cloud and add it to the loop closure detector.

```
viewId1 = 1;
descriptor1 = scanContextDescriptor(ptCloud1);
addDescriptor(loopDetector,viewId1,descriptor1);
```

Extract the second descriptor and add it to the loop closure detector.

```
viewId2 = 2;
descriptor2 = scanContextDescriptor(ptCloud2);
addDescriptor(loopDetector,viewId2,descriptor2);
```

Delete the descriptor that corresponds to view identifier `2`, from the detector.

```
deleteDescriptor(loopDetector,viewId2);
```

Input Arguments

`loopDetector` — Loop closure detector
`scanContextLoopDetector` object

Loop closure detector, specified as a `scanContextLoopDetector` object.

`viewIDs` — View identifiers
vector of integers

View identifiers, specified as a vector of positive integers.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pointCloud` | `scanContextLoopDetector`

Functions
`scanContextDescriptor` | `addDescriptor` | `detectLoop`

Topics
“Implement Point Cloud SLAM in MATLAB”

detectLoop

Detect loop closures

Syntax

```
loopViewIds = detectLoop(loopDetector)
loopViewIds = detectLoop(loopDetector,descriptor)
[loopViewIds,dists] = detectLoop( ___ )
[ ___ ] = detectLoop( ___ ,Name=Value)
```

Description

`loopViewIds = detectLoop(loopDetector)` detects loop closures for the last added descriptor and returns the view identifiers `loopViewIds` that correspond to loop closures. If the function detects no loop closures, `loopViewIds` is empty. The loop closure detector ignores the number of most recently added descriptors, set by the `NumExcludedDescriptors` name-value argument, to avoid detecting loop closures against recent descriptors.

`loopViewIds = detectLoop(loopDetector,descriptor)` detects loop closures using the scan context descriptor `descriptor`.

`[loopViewIds,dists] = detectLoop( ___ )` returns the scan context distances `dists` between the loop descriptor and the query descriptors, using any combination of input arguments from previous syntaxes. The function computes distance between scan context descriptors, normalized to the range [0,1], using a modified cosine distance.

`[ ___ ] = detectLoop( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from the previous syntaxes. For example, `detectLoop(loopDetector,NumExcludedDescriptors=15)` detects loop closures for the last added descriptor while ignoring only the 15 most recently added loop descriptors.

Examples

Detect Loop Closure Against Recently Added Descriptor

Create a loop closure detector.

```
loopDetector = scanContextLoopDetector;
```

Create a Velodyne® PCAP file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the point clouds and extract the scan context descriptor from each point cloud scan. Add the descriptors to the detector.

```
for viewId = 1:10
    ptCloud = readFrame(veloReader,viewId);
    descriptor = scanContextDescriptor(ptCloud);
```

```
    addDescriptor(loopDetector,viewId,descriptor);
end
```

Check if the next point cloud can be classified as a loop closure detection without excluding any recently added descriptors.

```
viewId = viewId + 1;
ptCloud = readFrame(veloReader,viewId);
descriptor = scanContextDescriptor(ptCloud);
[loopViewId,dists] = detectLoop(loopDetector,descriptor,'NumExcludedDescriptors',0)
```

```
loopViewId = uint32

10

dists = single

0.0858
```

Input Arguments

`loopDetector` — Loop closure detector

`scanContextLoopDetector` object

Loop closure detector, specified as a `scanContextLoopDetector` object.

`descriptor` — Scan context descriptor

M-by-*N* matrix

Scan context descriptor, specified as an *M*-by-*N* matrix, where *M* is the number of radial bins and *N* is the number of azimuthal bins.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectLoop(loopDetector,NumExcludedDescriptors=15)` detects loop closures for the last added descriptor while ignoring only the 15 most recently added loop descriptors.

`DistanceThreshold` — Scan context distance threshold

`0.1` (default) | positive scalar

Scan distance threshold to classify a view as a loop closure, specified as a positive scalar. Increasing this value can return more loop closures, but this can also increase false positives.

`NumExcludedDescriptors` — Number of most recently added descriptors to exclude

`30` (default) | nonnegative integer

Number of most recently added descriptors to exclude as loop closure candidates, specified as a nonnegative integer. Exclude the most recently added descriptors to avoid detecting loop closures for the most recently added descriptor against recent descriptors. Increase `NumExcludedDescriptors` if many consecutive descriptors correspond to the same area.

`SearchRadius` — Search radius
`0.3` (default) | positive scalar

Search radius in a subdescriptor space, specified as a positive scalar. The function computes the scan context distance for only those descriptors within the search radius. Increasing this value can return more loop closures, but can also increase false positives. Typical values range between `0.2` and `0.4`.

`MaxDetections` — Maximum number of strongest loop closure detections returned
3 (default) | positive integer

Maximum number of strongest loop closure detections returned, specified as a positive integer. Increase this value to increase the potential number of loop closure detections returned. However, increasing this value can decrease computation speed. Set this value to `Inf` to return all loop closure detections.

Output Arguments

`loopViewIds` — Loop closure view identifiers
P-element vector

Loop closure view identifiers corresponding to loop closures, returned as a *P*-element vector of integer values. If the function finds no loop closures, it returns `loopViewIds` as an empty vector. The loop closure detector ignores the last “NumExcludedDescriptors” on page 2-0 descriptors to avoid detecting loop closures against recent descriptors.

`dists` — Scan context distance
P-element vector

Scan context distances, returned as a *P*-element vector of positive values. The distances represent the scan context distance between the loop descriptor and the corresponding query descriptor. The function computes the distance between scan context descriptors, normalized to the range [0,1], using a modified cosine distance.

Algorithms

The ring key descriptor is a subdescriptor extracted from a scan context descriptor. It is the occupancy ratio of each azimuthal bin that makes it rotation-invariant.

The scan context loop closure detector is a two-phase algorithm. It first uses the ring key descriptor for a nearest neighbor search to find candidate loop closures. Then, it computes the scan context distance using the scan context descriptors, and thresholds it to identify the best loop closure detections.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pointCloud` | `scanContextLoopDetector` | `pcviewset`

Functions
`scanContextDescriptor` | `addDescriptor` | `deleteDescriptor` | `scanContextDistance`

Topics
“Implement Point Cloud SLAM in MATLAB”
“Build a Map from Lidar Data Using SLAM”

SIFTPoints

Object for storing SIFT interest points

Description

The `SIFTPoints` object enables you to pass data between the `detectSIFTFeatures` and `extractFeatures` functions. You can also use it to manipulate and plot the data returned by these functions. You can use the object to fill interest points interactively.

Creation

Syntax

```
points = SIFTPoints(location)
```

Description

`points = SIFTPoints(location)` constructs a `SIFTPoints` object from an *M*-by-2 matrix of [*x y*] point coordinates in `location`.

Input Arguments

`location` — Point coordinates
M-by-2 matrix of [*x y*] point coordinates.

Point coordinates, specified as an *M*-by-2 matrix of [*x y*] point coordinates. *M* is the number of points.

Properties

`Scale` — Scale
`1.6` (default) | scalar

Scale at which the interest points are detected, specified as a value greater than `0`.

`Metric` — Strength of detected feature
`0.0` (default) | numeric scalar

Strength of the detected feature, specified as a value in the range `[0 1]`. The SIFT algorithm uses the contrast threshold to determine strong features.

`Orientation` — Orientation
`0.0` (default) | angle in radians

Orientation of the detected feature, specified as an angle, in radians. The angle is measured counterclockwise from the *X*-axis with the origin specified by the `location` input. Do not set this property manually. Instead, use the call to `extractFeatures` to fill in this value. The `extractFeatures` function modifies the default value of `0.0`.The `Orientation` is mainly useful for visualization purposes.

`Octave` — Index of the Gaussian pyramid octave layer

`0` (default) | non-negative integer

Index of the Gaussian pyramid octave layer that the keypoint is extracted from, specified as an integer. An octave is a collection of difference-of-Gaussian pyramid layers, and is used for scale-space.

`Layer` — Layer within the octave

`0` (default) | integer

Layer within the octave from which the keypoint was extracted, specified as an integer.

Object Functions

plot	Plot detected feature points
isempty	Determine if points object is empty
length	Number of stored points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation

Examples

Detect SIFT Features in Image

Read an image.

```
I = imread("cameraman.tif");
```

Detect SIFT features in the image.

```
points = detectSIFTFeatures(I)
```

```
points =
  245×1 SIFTPoints array with properties:

          Scale: [245×1 single]
    Orientation: [245×1 single]
         Octave: [245×1 int32]
          Layer: [245×1 int32]
       Location: [245×2 single]
         Metric: [245×1 single]
          Count: 245
```

Display the location and scale of the 10 strongest points in the image.

```
strongest = points.selectStrongest(10);
imshow(I)
hold on
plot(strongest)
```



Display the [*x* *y*] coordinates for the selected points.

```
strongest.Location
```

```
ans = 10×2 single matrix

  183.0139  205.4790
  112.5054  206.6145
  135.2954  125.0968
  101.1768  173.8070
  131.3702  113.8461
  167.1152   77.5036
  110.9539  155.9264
  135.0020   92.3701
  112.1106  173.5276
  156.8862   77.8692
```

Detect SIFT Features and Display Points

Read an image into the workspace.

```
I = imread("cameraman.tif");
```

Detect SIFT features in the image.

```
points = detectSIFTFeatures(I);
```

Display the last 5 detected points.

```
imshow(I)
hold on
plot(points(end-4:end))
```



Tips

- Although `SIFTPoints` may hold many points, it is a scalar object. Therefore, `numel(SIFTPoints)` always returns `1`. This value may differ from `length(SIFTPoints)`, which returns the true number of points held by the object.
- Properties can be specified as a scalar or a vector whose length matches the number of coordinates in `location`.

Version History

Introduced in R2021b

References

[1] Lowe, David G.. "Distinctive Image Features from Scale-Invariant Keypoints." *Int. J. Comput. Vision* 60 , no. 2 (2004): 91--110.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To index locations with this object, use the syntax: `points.Location(idx,:)`, for `points` object. See visionRecovertformCodeGeneration_kernel.m, which is used in the “Introduction to Code Generation with Feature Matching and Registration” example.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`MSERRegions` | `BRISKPoints` | `cornerPoints` | `detectORBFeatures` | `ORBPoints`

Functions
`detectSIFTFeatures` | `detectBRISKFeatures` | `detectKAZEFeatures` | `detectFASTFeatures` | `detectORBFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectSURFFeatures` | `extractFeatures` | `matchFeatures`

Topics
“Create Panorama”
“Detect SURF Interest Points in a Grayscale Image” on page 3-615
“Display MSER Feature Regions from the MSERRegions Object” on page 2-827
“Find MSER Regions in an Image” on page 3-603
“Detect MSER Features in an Image” on page 2-826

pcmapndt

Localization map based on normal distributions transform (NDT)

Description

The `pcmapndt` object creates a normal distributions transform (NDT) map from a prebuilt point cloud map of the environment. The NDT map is a compressed, memory-efficient representation suitable for localization. The object converts the point cloud map into a set of voxels (3-D boxes), each represented by a 3-D normal distribution. Use the `selectSubmap` object function to select a submap within the map from a coarse position estimate. Use the `findPose` object function to localize the pose of the sensor based on the assembled map.

Creation

Syntax

```
ndtMap = pcmapndt(ptCloudMap,voxelSize)
```

Description

`ndtMap = pcmapndt(ptCloudMap,voxelSize)` returns an NDT map from a point cloud map, `ptCloudMap`.

Input Arguments

`ptCloudMap` — Point cloud map
`pointCloud` object

Point cloud map, specified as a `pointCloud` object. The `pcmapndt` object creates a normal distributions transform (NDT) map from the prebuilt map of the environment, specified by `ptCloudMap`.

`voxelSize` — Size of voxels
scalar

This property is read-only.

Size of the voxels, specified as a scalar value in world units.

Properties

`SelectedSubmap` — Currently selected submap
entire map (default) | 6-element vector

This property is read-only.

Currently selected submap, specified as a 6-element vector of the form [*xmin*,*xmax ymin ymax zmin zmax*] that describes the range of the submap along each axis. The elements of the vector describe the region of interest represented by the submap.

`VoxelSize` — Size of voxels
scalar

This property is read-only.

Size of the voxels, specified as a scalar value in world units.

`XLimits` — Range of map along *x*-axis
2-element vector

This property is read-only.

Range of the map along the *x*-axis, specified as a 2-element vector of the form [*xmin xmax*].

`YLimits` — Range of map along the *Y*-axis
2-element vector

This property is read-only.

Range of the map along the *Y*-axis, specified as a 2-element vector of the form [*ymin ymax*].

`ZLimits` — Range of map along the *z*-axis
2-element vector

This property is read-only.

Range of the map along the *z*-axis, specified as a 2-element vector of the form [*zmin zmax*].

`VoxelMean` — Mean value of each voxel
M-by-3 matrix

This property is read-only.

Mean value of each voxel, specified as an *M*-by-3 matrix. Each row of the matrix contains the [*x y z*] values for a voxel. *M* is the number of voxels.

`VoxelCovariance` — Covariance of each voxel
3-by-3-by-*M* array

This property is read-only.

Covariance of each voxel, specified as a 3-by-3-by-*M* array for *M* voxels.

`VoxelNumPoints` — Number of points in each voxel
M-by-1 vector

This property is read-only.

Number of points in each voxel, specified as an *M*-by-1 vector for *M* voxels.

Object Functions

selectSubmap	Select submap within map
isInsideSubmap	Check if query position is inside selected submap
findPose	Localize point cloud within map using normal distributions transform (NDT) algorithm
show	Visualize normal distributions transform (NDT) map

Examples

Create an NDT Map Representation

Load a point cloud view set, which was saved from a `pcviewset` object.

```
data = load('vSetPointClouds.mat');
vSet = data.vSet;
```

Extract point clouds and absolute poses to build a map.

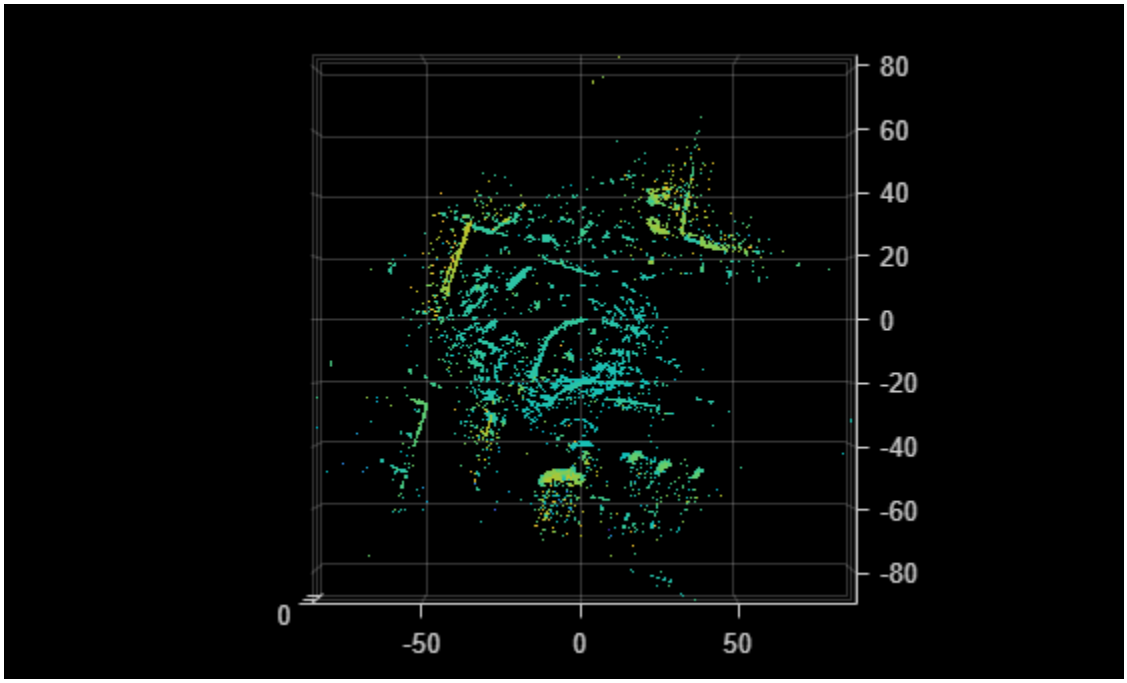
```
ptClouds = vSet.Views.PointCloud;
tforms   = vSet.Views.AbsolutePose;
```

Create a point cloud map by aligning the point cloud scans using the absolute poses.

```
ptCloudMap = pcalign(ptClouds,tforms);
```

Create and visualize an NDT map from a point cloud map.

```
voxelSize = 1;
ndtMap = pcmapndt(ptCloudMap,voxelSize);
figure
show(ndtMap)
view(2)     % Change viewing angle to top-view
```



Lidar Localization Using NDT

Load a normal distributions transform (NDT) map from a MAT file.

```
data = load('ndtMapParkingLot.mat');
ndtMap = data.ndtMapParkingLot;
```

Load point cloud scans and pose estimates from a second MAT file.

```
data = load('parkingLotData.mat');
ptCloudScans = data.parkingLotData.ptCloudScans;
initPoseEsts = data.parkingLotData.initPoseEsts;
```

Display the NDT map.

```
show(ndtMap)
```

Change the viewing angle to top-view.

```
view(2)
```

Select the submap centered around the first estimate.

```
center = initPoseEsts(1).Translation;
sz = [70 50 20];
ndtMap = selectSubmap(ndtMap,center,sz);
```

Set the radius for visualization of the current location and the distance threshold to update the submap.

```
radius = 0.5;
distThresh = 15;
```

Loop over the point clouds, localize them in the map, and update the selected submap as needed.

```
numScans = numel(ptCloudScans);

for n = 1:numScans
    ptCloud = ptCloudScans(n);
    initPose = initPoseEsts(n);

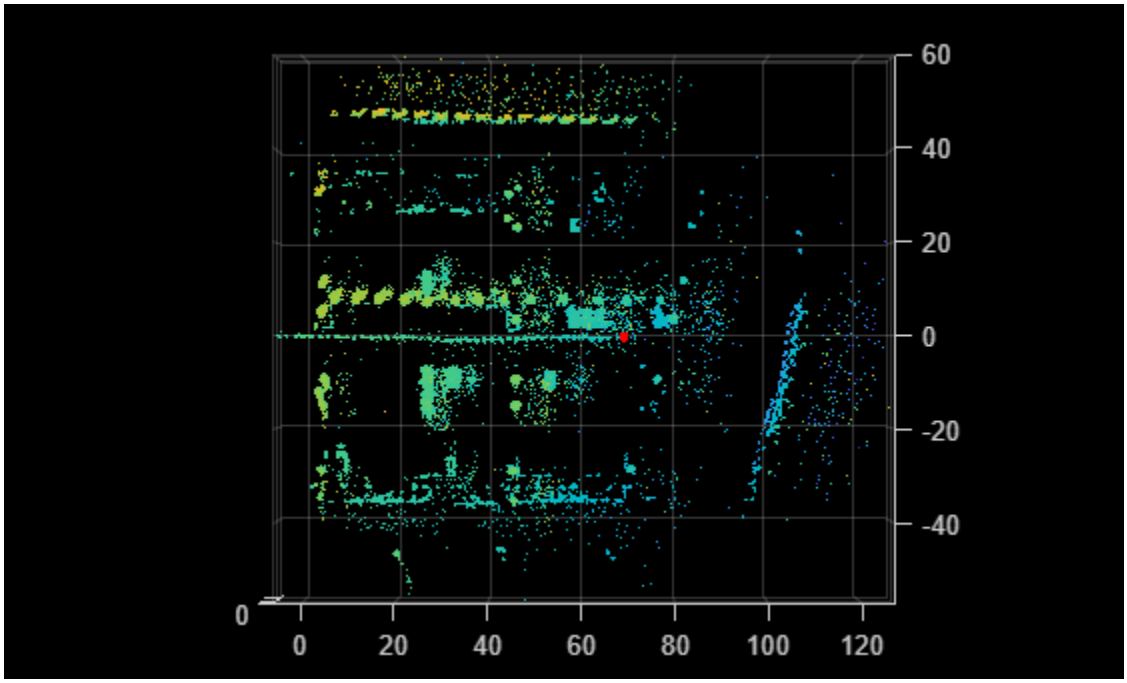
    poseTranslation = initPose.Translation;
    [isInside,distToEdge] = isInsideSubmap(ndtMap,poseTranslation);
        submapNeedsUpdate = ~isInside ...       % Current pose is outside submap
        || any(distToEdge(1:2) < distThresh);   % Current pose is close to submap edge

if submapNeedsUpdate
    ndtMap = selectSubmap(ndtMap,poseTranslation,sz);
end

% Localize the point cloud scan in the map.
currPose = findPose(ndtMap,ptCloud,initPose);

% Display the position of the estimate as a circle.
pos = [currPose.Translation(1:2) radius];
showShape('circle',pos,'Color','r');
```

```
% Pause to view the change.
pause(0.05)
end
```



Version History

Introduced in R2021a

References

Biber, P., and W. Strasser. “The Normal Distributions Transform: A New Approach to Laser Scan Matching.” In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)* Vol. 3, 2743–48. Las Vegas, Nevada, USA: IEEE, 2003. https://doi.org/10.1109/IROS.2003.1249285.

[1] Magnusson, Martin. "The Three-Dimensional Normal-Distributions Transform: An Efficient Representation for Registration, Surface Analysis, and Loop Detection." PhD thesis, Örebro universitet, 2009. http://urn.kb.se/resolve?urn=urn:nbn:se:oru:diva-8458 urn:nbn:se:oru:diva-8458.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Because of architectural differences between the CPU and the GPU, there may be minor differences in results from simulation and code generation.
- In certain cases, there may be differences in the location of points in a voxel between simulation and code generation. However, the total number of points remains the same.

See Also

Functions

`selectSubmap` | `findPose` | `pcregisterndt` | `pcalign`

selectSubmap

Select submap within map

Syntax

```
ndtMapOut = selectSubmap(ndtMapIn,roi)
ndtMapOut = selectSubmap(ndtMapIn,center,sz)
```

Description

`ndtMapOut = selectSubmap(ndtMapIn,roi)` selects a submap within the NDT map `ndtMapIn` using the specified region of interest `roi`.

Use this function to confine the search space for localization when using coarse position estimates.

`ndtMapOut = selectSubmap(ndtMapIn,center,sz)` selects a submap specified by the center `center` and size `sz` of the submap.

Examples

Lidar Localization Using NDT

Load a normal distributions transform (NDT) map from a MAT file.

```
data = load('ndtMapParkingLot.mat');
ndtMap = data.ndtMapParkingLot;
```

Load point cloud scans and pose estimates from a second MAT file.

```
data = load('parkingLotData.mat');
ptCloudScans = data.parkingLotData.ptCloudScans;
initPoseEsts = data.parkingLotData.initPoseEsts;
```

Display the NDT map.

```
show(ndtMap)
```

Change the viewing angle to top-view.

```
view(2)
```

Select the submap centered around the first estimate.

```
center = initPoseEsts(1).Translation;
sz = [70 50 20];
ndtMap = selectSubmap(ndtMap,center,sz);
```

Set the radius for visualization of the current location and the distance threshold to update the submap.

```
radius = 0.5;
distThresh = 15;
```

Loop over the point clouds, localize them in the map, and update the selected submap as needed.

```
numScans = numel(ptCloudScans);

for n = 1:numScans
    ptCloud = ptCloudScans(n);
    initPose = initPoseEsts(n);

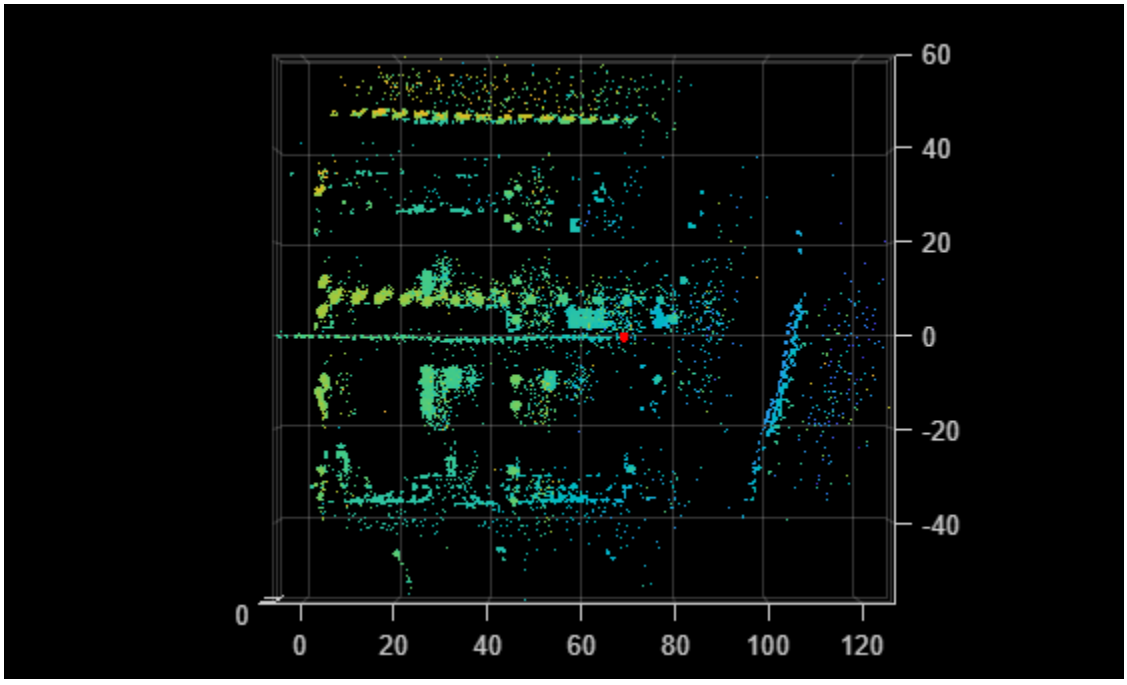
    poseTranslation = initPose.Translation;
    [isInside,distToEdge] = isInsideSubmap(ndtMap,poseTranslation);
        submapNeedsUpdate = ~isInside ...       % Current pose is outside submap
        || any(distToEdge(1:2) < distThresh);   % Current pose is close to submap edge

if submapNeedsUpdate
    ndtMap = selectSubmap(ndtMap,poseTranslation,sz);
end

% Localize the point cloud scan in the map.
currPose = findPose(ndtMap,ptCloud,initPose);

% Display the position of the estimate as a circle.
pos = [currPose.Translation(1:2) radius];
showShape('circle',pos,'Color','r');

% Pause to view the change.
pause(0.05)
end
```



Input Arguments

ndtMapIn — NDT map
`pcmapndt` object

NDT map, specified as a `pcmapndt` object.

`roi` — Region of interest
6-element vector

Region of interest, specified as a 6-element vector of the form [*xmin xmax ymin ymax zmin zmax*].

`center` — Center of submap
3-element vector

Center of the submap, specified as 3-element vector of the form [*xc yc zc*].

`sz` — Size of submap along each axis
3-element vector

Size of the submap along each axis, specified as 3-element vector of the form [*xsz ysz zsz*].

Output Arguments

`ndtMapOut` — NDT map
`pcmapndt` object

NDT map, returned as a `pcmapndt` object with an updated `SelectedSubmap` property.

If a region of the selected submap is outside of the limits of the map, the selected submap is constrained to the map limits as described by the `XLimits`, `YLimits`, and `ZLimits` properties of the `pcmapndt` object.

Tips

- Use a submap size large enough to include the uncertainty of the position estimates and the range of the sensor used with `findPose`. A larger submap can increase computation time during each call to the `findPose` function, but it can also reduce the frequency of submap updates.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pcmapndt`

Functions
`isInsideSubmap` | `show` | `showShape`

isInsideSubmap

Check if query position is inside selected submap

Syntax

```
isInside = isInsideSubmap(ndtMap pos)
[isInside,distToEdge] = isInsideSubmap(ndtMap,pos)
```

Description

`isInside = isInsideSubmap(ndtMap pos)` check if the query position `pos`, is inside the selected submap of the NDT map `ndtMap`.

`[isInside,distToEdge] = isInsideSubmap(ndtMap,pos)` also returns the distance from the query position to the closest edge of the submap along the *X*-,*Y*-, and *Z*-axes respectively.

Examples

Check If Query Position Is in Submap

Load an NDT map from a MAT file.

```
data = load('ndtMapParkingLot.mat');
ndtMap = data.ndtMapParkingLot;
```

Select a submap with a specified center and set size.

```
center = [40 0 0];
sz = [50 50 20];
ndtMap = selectSubmap(ndtMap,center,sz);
```

Display the extent of the submap.

```
disp(ndtMap.SelectedSubmap)
```

```
   15.0000   65.0000  -25.0000   25.0000   -9.1840    5.4975
```

Check if a query position is inside the submap.

```
pos1 = [40 0 0]; % near center
[isInside1,distToEdge1] = isInsideSubmap(ndtMap,pos1)
```

```
isInside1 = logical
   1


distToEdge1 = 1×3

   25.0000   25.0000    5.4975
```

```
pos2 = [66 0 0]; % completely outside
[isInside2,distToEdge2] = isInsideSubmap(ndtMap,pos2)

isInside2 = logical
   0


distToEdge2 = 1×3

    1.0000   25.0000    5.4975


pos3 = [60 0 0]; % inside, 5 meters from edge in x direction
[isInside3,distToEdge3] = isInsideSubmap(ndtMap,pos3)

isInside3 = logical
   1


distToEdge3 = 1×3

    5.0000   25.0000    5.4975
```

Input Arguments

`ndtMap` — NDT map

`pcmapndt` object

NDT map, specified as a `pcmapndt` object.

`pos` — Query position

3-element vector

Query position, specified as a 3-element vector of the form [*x y z*].

Output Arguments

`isInside` — Indication of position inside submap

`true` | `false`

Indication of position inside submap, returned as a logical `true` or `false`.

`distToEdge` — Distance from the query position to closest edge of the submap

3-element vector

Distance from the query position to the closest edge of the submap in the *X*-, *Y*-, and *Z*-axes respectively, returned as a 3-element vector.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`selectSubmap` | `findPose`

Objects

`pcmapndt`

findPose

Localize point cloud within map using normal distributions transform (NDT) algorithm

Syntax

```
currPose = findPose(ndtMap,ptCloud,initPose)
currPose = findPose( ___ ,Name,Value)
```

Description

`currPose = findPose(ndtMap,ptCloud,initPose)` localizes the pose of the point cloud `ptCloud` within the NDT map `ndtMap` using the NDT algorithm. The function confines the search space to the submap, specified by the `SelectedSubmap` property of the `ndtMap` object.

`currPose = findPose( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntax. For example, `"MaxIterations",30` sets the maximum number of iterations before the function stops the NDT algorithm.

Examples

Lidar Localization Using NDT

Load a normal distributions transform (NDT) map from a MAT file.

```
data = load('ndtMapParkingLot.mat');
ndtMap = data.ndtMapParkingLot;
```

Load point cloud scans and pose estimates from a second MAT file.

```
data = load('parkingLotData.mat');
ptCloudScans = data.parkingLotData.ptCloudScans;
initPoseEsts = data.parkingLotData.initPoseEsts;
```

Display the NDT map.

```
show(ndtMap)
```

Change the viewing angle to top-view.

```
view(2)
```

Select the submap centered around the first estimate.

```
center = initPoseEsts(1).Translation;
sz = [70 50 20];
ndtMap = selectSubmap(ndtMap,center,sz);
```

Set the radius for visualization of the current location and the distance threshold to update the submap.

```
radius = 0.5;
distThresh = 15;
```

Loop over the point clouds, localize them in the map, and update the selected submap as needed.

```
numScans = numel(ptCloudScans);

for n = 1:numScans
    ptCloud = ptCloudScans(n);
    initPose = initPoseEsts(n);

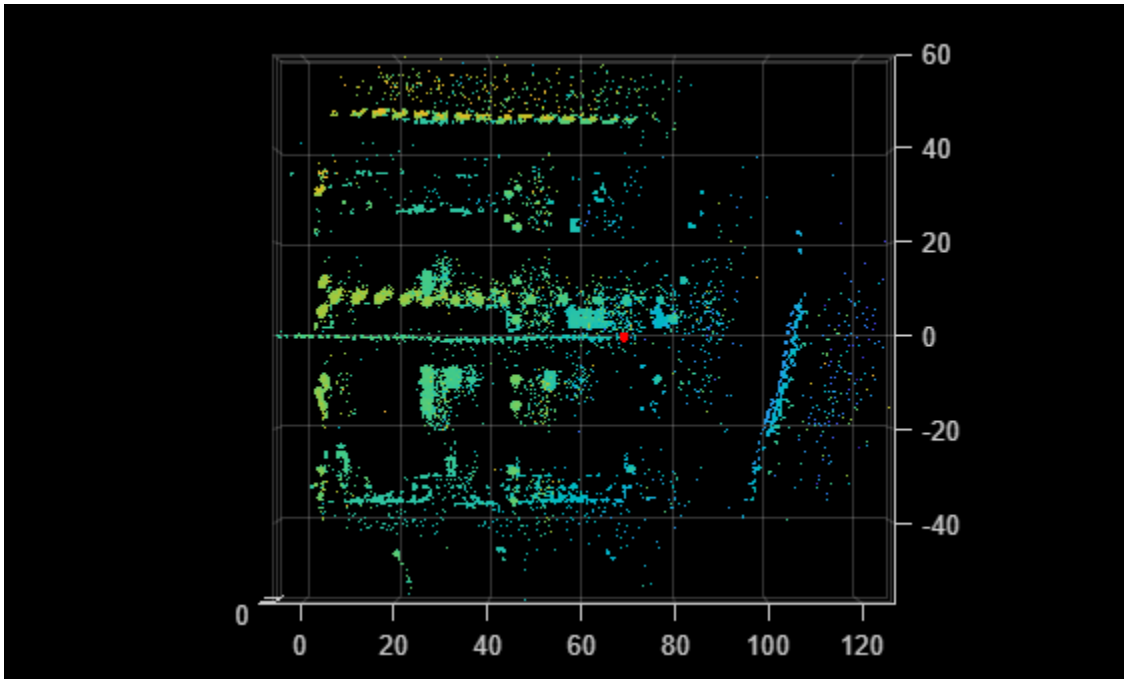
    poseTranslation = initPose.Translation;
    [isInside,distToEdge] = isInsideSubmap(ndtMap,poseTranslation);
        submapNeedsUpdate = ~isInside ...       % Current pose is outside submap
        || any(distToEdge(1:2) < distThresh);   % Current pose is close to submap edge

if submapNeedsUpdate
    ndtMap = selectSubmap(ndtMap,poseTranslation,sz);
end

% Localize the point cloud scan in the map.
currPose = findPose(ndtMap,ptCloud,initPose);

% Display the position of the estimate as a circle.
pos = [currPose.Translation(1:2) radius];
showShape('circle',pos,'Color','r');

% Pause to view the change.
pause(0.05)
end
```



Input Arguments

`ndtMap` — NDT map
`pcmapndt` object

NDT map, specified as a `pcmapndt` object.

`ptCloud` — Point cloud
`pointCloud` object

Point cloud in the sensor coordinate system, specified as a `pointCloud` object.

`initPose` — Initial estimate for the pose of the sensor in the map
`rigidtform3d` object

Initial estimate for the pose of the sensor in the map, specified as a `rigidtform3d` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"MaxIterations",30` stops the NDT algorithm after 30 iterations.

`OutlierRatio` — Expected percentage of outliers
`0.55` (default) | scalar in the range [0, 1)

Expected percentage of outliers with respect to a normal distribution, specified as a scalar in the range [0, 1). The NDT algorithm assumes a point is generated by the mixture of a normal distribution for inliers and a uniform distribution for outliers. A larger value of `OutlierRatio` reduces the influence of outliers.

Data Types: `single` | `double`

`MaxIterations` — Maximum number of iterations
`30` (default) | nonnegative integer

Maximum number of iterations before the NDT algorithm stops, specified as a nonnegative integer.

Data Types: `single` | `double`

`Tolerance` — Tolerance between consecutive NDT iterations
`[0.01 0.5]` (default) | 2-element vector

Tolerance between consecutive NDT iterations, specified as a 2-element vector with nonnegative values. The vector, [*Tdiff Rdiff*], represents the tolerance of absolute difference in translation and rotation, respectively, estimated in consecutive NDT iterations. *Tdiff* measures the Euclidean distance between two translation vectors. *Rdiff* measures the angular difference in degrees. The algorithm stops when the difference between estimated rigid transformations in the most recent consecutive iterations falls below the specified tolerance values.

Data Types: `single` | `double`

`Verbose` — Display progress information
`false` or `0` (default) | `true` or `1`

Display progress information, specified as a logical `0` (`false`) or `1` (`true`). Set `Verbose` to `true` to display progress information.

Data Types: `logical`

Output Arguments

`currPose` — Pose of sensor in map
`rigidtform3d` object

Pose of the sensor in the map, returned as a `rigidtform3d` object. The function confines the search space to the submap, specified by the `SelectedSubmap` property of `ndtMap`.

Tips

- To improve the accuracy and efficiency of localization, consider downsampling the point cloud using `pcdownsample` before using this function.

Version History

Introduced in R2021a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `findpose` function now supports specifying `initPose` as a `rigidtform3d` object, which uses the premultiply convention. Although you can still specify `initPose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. The `findpose` function returns `currPose` as a geometric transformation object of the same type as `initPose`.

You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Biber, P., and W. Strasser. “The Normal Distributions Transform: A New Approach to Laser Scan Matching.” In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)* Vol. 3, 2743–48. Las Vegas, Nevada, USA: IEEE, 2003. https://doi.org/10.1109/IROS.2003.1249285.

[2] Magnusson, Martin. "The Three-Dimensional Normal-Distributions Transform: An Efficient Representation for Registration, Surface Analysis, and Loop Detection." PhD thesis, Örebro universitet, 2009. http://urn.kb.se/resolve?urn=urn:nbn:se:oru:diva-8458 urn:nbn:se:oru:diva-8458.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcmapndt` | `pointCloud` | `rigidtform3d`

Functions

`selectSubmap` | `show` | `pcregisterndt` | `isInsideSubmap` | `pcdownsample`

show

Visualize normal distributions transform (NDT) map

Syntax

```
show(ndtMap)
show(ndtMap,spatialExtent)
show(___,Name,Value)

ax = show(___)
```

Description

`show(ndtMap)` visualizes the NDT map as a point cloud. The function uses voxel means, covariances, and point counts to generate points for the display. These points are not the points used to build the map. The NDT map is a compressed representation of the point cloud map that stores the voxels and their 3-D normal distributions.

`show(ndtMap,spatialExtent)` displays points within the spatial map or submap specified by `spatialExtent`.

`show(___,Name,Value)` specifies options using one or more name-value arguments in addition to any combination of input arguments in previous syntaxes. For example, `'MarkerSize',6` sets the marker size to 6 points.

`ax = show(___)` returns the axes used to plot the NDT map.

Examples

Select and Visualize Submap

Load an NDT map from a MAT file.

```
data = load('ndtMapParkingLot.mat');
ndtMap = data.ndtMapParkingLot;
```

Select a submap from within the loaded NDT map at the specified location and size.

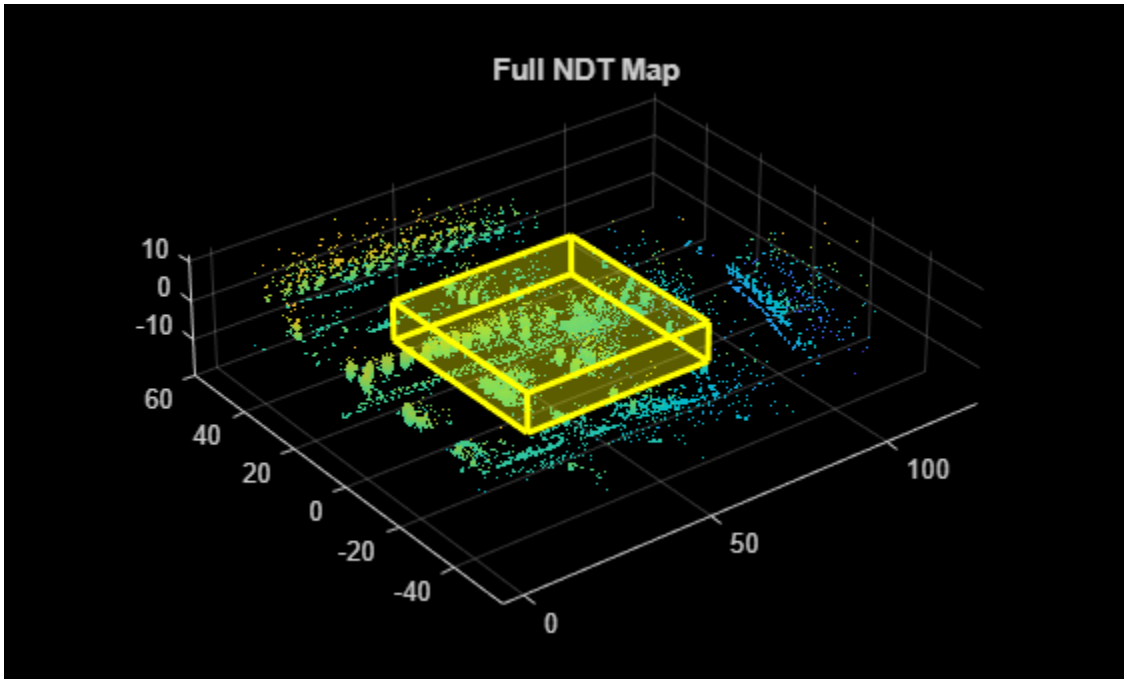
```
center  = [50  0  0];
sz      = [50 50 10];
ndtMap = selectSubmap(ndtMap,center,sz);
```

Visualize the full NDT map.

```
figure
show(ndtMap)
title('Full NDT Map')
```

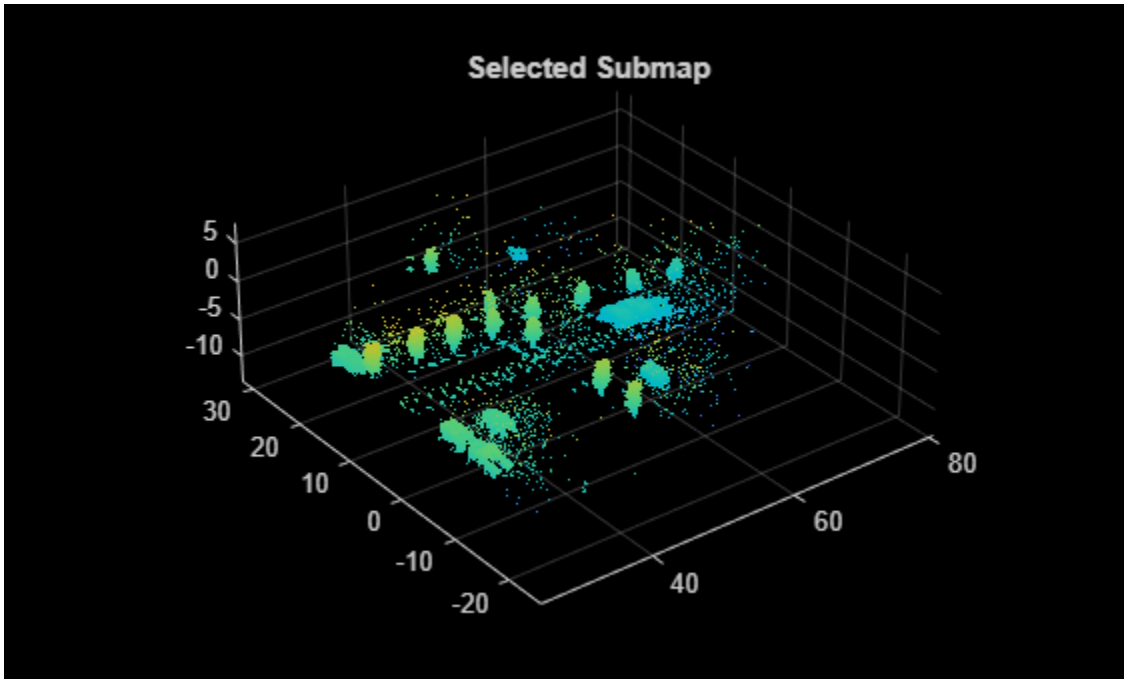
Highlight the selected submap on the full NDT map.

```
pos = [center,sz,zeros(1, 3)];
showShape('cuboid',pos,'Color','y','Opacity',0.2);
```



Visualize the selected submap.

```
figure
show(ndtMap,'submap')
title('Selected Submap')
```



Input Arguments

`ndtMap` — NDT map

`pcmapndt` object

NDT map, specified as a `pcmapndt` object.

`spatialExtent` — Spatial extent

`'map'` | `'submap'`

Spatial extent, specified as `'map'` or `'submap'`. When you specify `'submap'`, only points within the current submap are displayed.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MarkerSize',6` sets the marker size to 6 points.

`MarkerSize` — Diameter of marker

`6` (default) | positive scalar

Diameter of marker, specified as a positive scalar. This value specifies the approximate diameter of the point marker. Units are in points. A marker size larger than six can reduce rendering performance.

`Parent` — Axes on which to display visualization

`Axes` object

Axes on which to display the visualization, specified as an `Axes` object. To create an `Axes` object, use the `axes` function. To display the visualization in a new figure, leave `'Parent'` unspecified.

Output Arguments

`ax` — Plot axes

`Axes` object

Plot axes, returned as an `axes` graphics object.

Version History

Introduced in R2021a

See Also

Objects

`pcmapndt`

Functions
`pcshow` | `selectSubmap`

worldpointset

Manage 3-D to 2-D point correspondences

Description

The `worldpointset` object stores correspondences between 3-D world points and 2-D image points across camera views. You can use a `worldpointset` object with an `imageviewset` object to manage image and map data for structure-from-motion, visual odometry, and visual simultaneous localization and mapping (SLAM).

Creation

Syntax

```
wpSet = worldpointset
```

Description

`wpSet = worldpointset` creates a worldpointset object with default properties. Use the object functions to perform actions such as adding, modifying or removing correspondences, finding points in a view, and finding views of points.

Properties

`WorldPoints` — 3-D world points
[ ] (default) | *M*-by-3 matrix

This property is read-only.

3-D world points, specified as an *M*-by-3 matrix with rows containing [*x y z*] world points. *M* is the number of 3-D world points.

`ViewIds` — Identifiers for views
[ ] (default) | *N*-element row vector of integers

This property is read-only.

Identifiers for views associated with world points, specified as an *N*-element row vector of integers.

`PointIds` — Identifiers for 3-D points
[ ] (default) | *M*-element row vector of integers

This property is read-only.

Identifiers for 3-D points in `WorldPoints`, specified as an *M*-element row vector of integers.

`Correspondences` — 3-D to 2-D point correspondences
empty three-column table (default) | three-column table

This property is read-only.

3-D to 2-D point correspondences, specified as a three-column table.

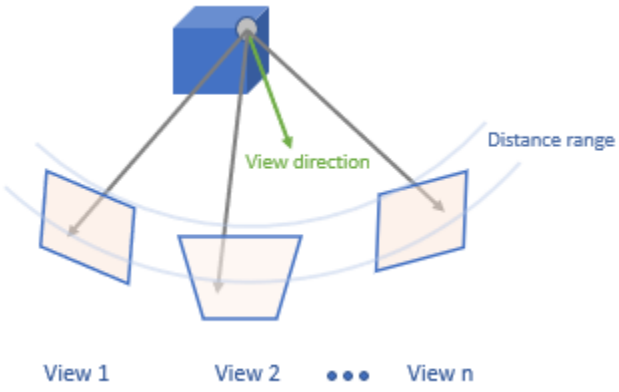
Column	Description
`PointIndex`	Each row contains the linear index of a world point.
`ViewId`	Each row contains a 1-by-*N* vector specifying the IDs of the views associated with the corresponding world points. *N* is the number of views associated with the world point.
`FeatureIndex`	Each row contains a 1-by-*N* vector specifying the indices of the feature points that correspond to the world point. Each element is the index of the feature point in the view specified by the corresponding element in the `ViewID` cell.

`ViewingDirection` — Mean viewing direction of each world point

empty matrix (default) | *M*-by-3 matrix

This property is read-only.

Mean viewing direction of each world point, specified as an *M*-by-3 matrix, where *M* is the number of points. The viewing direction provides an estimate for a view angle from which a 3-D point can be observed. The mean viewing direction is the average of all the unit vectors pointing from the world point to the camera centers of the associated views. When a new camera view is introduced into the system, the 3-D world points that can potentially be observed in this view can be predicted based on the range set by the distance limits and the `ViewingDirection`.



`DistanceLimits` — Distance limits for 3-D point observation

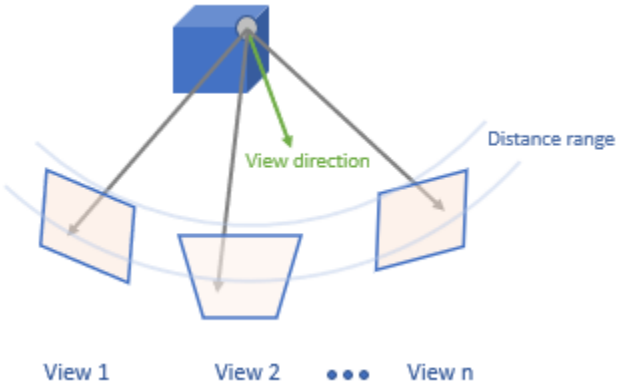
empty vector (default) | *M*-by-2 vector

This property is read-only.

An representing the minimum and maximum distances from which each world point is observed.

Distance limits for 3-D point observation, specified as an *M*-by-2 vector that represents the minimum and maximum distance, where *M* is the number of points. The range indicates how far the 3-D point

can be observed. When a new camera view is introduced into the system, the 3-D world points that can potentially be observed in this view can be predicted based on the range set by the distance limits and the `ViewingDirection`.



`RepresentativeViewId` — View IDs of the representative views
`empty vector` (default) | *M*-element column vector

This property is read-only.

View IDs of the representative views, specified as an *M*-element column vector, where *M* is the number of points. The representative view corresponds to the view that contains the representative feature in the representative view for each world point. A representative feature is the medoid of all the feature descriptors associated with the world point.

`RepresentativeFeatureIndex` — Representative feature descriptor index
`empty vector` (default) | *M*-element column vector

This property is read-only.

Representative feature descriptor index, specified as an *M*-element column vector, where *M* is the number of points. The index value corresponds to the view that contains the representative feature descriptor for each world point.

`Count` — Number of 3-D world points
`0` (default) | scalar

This property is read-only.

Number of 3-D world points, specified as a scalar.

Object Functions

addWorldPoints	Add world points to world point set
removeWorldPoints	Remove world points from world point set
updateWorldPoints	Update world points in world point set
selectWorldPoints	Select world points from world point set
addCorrespondences	Update world points in a world point set
removeCorrespondences	Remove 3-D to 2-D correspondences from world point set
updateCorrespondences	Update 3-D to 2-D correspondences in world point set

updateLimitsAndDirection	Update distance limits and viewing direction
updateRepresentativeView	Update representative view ID and corresponding feature index
findViewsOfWorldPoint	Find views that observe a world point
findWorldPointsInTracks	Find world points that correspond to point tracks
findWorldPointsInView	Find world points observed in view

Examples

Triangulate Stereo Images

Load a `MAT` file containing stereo parameters into the workspace.

```
load('webcamsSceneReconstruction.mat');
```

Read a stereo pair of images into the workspace.

```
I1 = imread('sceneReconstructionLeft.jpg');
I2 = imread('sceneReconstructionRight.jpg');
```

Undistort the images.

```
I1 = undistortImage(I1,stereoParams.CameraParameters1);
I2 = undistortImage(I2,stereoParams.CameraParameters2);
```

Define a rectangular region of interest (ROI), in the format [*x y width height*].

```
roi = [30 30 size(I1,2)-30 size(I1,1)-30];
```

Detect and extract SIFT features from both images using the ROI.

```
imagePoints1 = detectSIFTFeatures(im2gray(I1),ROI=roi);
imagePoints2 = detectSIFTFeatures(im2gray(I2),ROI=roi);

[feature1,validPoints1] = extractFeatures(im2gray(I1),imagePoints1);
[feature2,validPoints2] = extractFeatures(im2gray(I2),imagePoints2);
```

Match the extracted features to each other.

```
indexPairs = matchFeatures(feature1,feature2,MatchThreshold=10,Unique=true);
```

Compute the 3-D world points.

```
matchedPoints1 = validPoints1(indexPairs(:,1));
matchedPoints2 = validPoints2(indexPairs(:,2));
worldPoints    = triangulate(matchedPoints1,matchedPoints2,stereoParams);
```

Create a `worldpointset` object to manage correspondences.

```
wpSet = worldpointset;
```

Add the world points to the `worldpointset`.

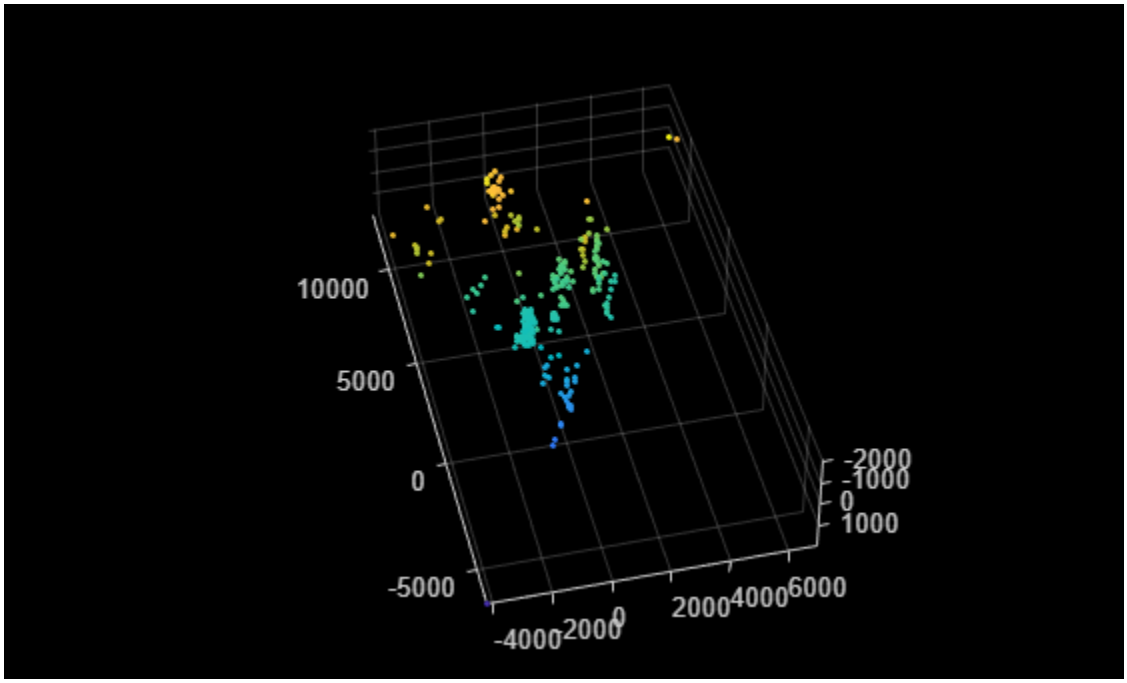
```
[wpSet,newPointIndices] = addWorldPoints(wpSet,worldPoints);
```

Add the 3-D to 2-D point correspondences to the `worldpointset`.

```
wpSet = addCorrespondences(wpSet,1,newPointIndices,indexPairs(:,1));
wpSet = addCorrespondences(wpSet,2,newPointIndices,indexPairs(:,2));
```

Display the world points.

```
pcshow(wpSet.WorldPoints,VerticalAxis="y",VerticalAxisDir="down",MarkerSize=45)
```



Version History

Introduced in R2020b

R2022b: Visual SLAM: Track Visibility Relationship Between 3-D World Points and Camera Views

- Added `ViewingDirection`, `DistanceLimits`, `RepresentativeViewId`, and `RepresentativeFeatureIndex` properties.
- Added `updateLimitsAndDirection` and `updateRepresentativeView` object functions.

R2024a: Use unique point identifiers to specify world points

Use the new `selectWorldPoints` object function to select world points from a `worldpointset` object. Also, you can now specify unique point IDs to manage world points. In prior releases if you did not specify unique identifiers when points were added, then points were saved with sequential identifiers.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`pointTrack`

Objects

`imageviewset`

Topics

“Monocular Visual Simultaneous Localization and Mapping”
“Monocular Visual-Inertial SLAM”

selectWorldPoints

Select world points from world point set

Syntax

```
wpSetSelected = selectWorldPoints(wpSet,pointIds)
```

Description

`wpSetSelected = selectWorldPoints(wpSet,pointIds)` selects world points from `wpSet` using the identifiers in `pointIds`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Select World Points from World Point Set Using Point IDs

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate point IDs.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset
```

```
wpSet =
  worldpointset with properties:

                  WorldPoints: [0×3 single]
                      ViewIds: [1×0 uint32]
                     PointIds: [1×0 uint32]
             ViewingDirection: [0×3 single]
               DistanceLimits: [0×2 single]
         RepresentativeViewId: [0×1 double]
    RepresentativeFeatureIndex: [0×1 double]
                        Count: 0
              Correspondences: [0×3 table]
```

Add the world points to the world point set.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Select the first 50 points in the point set.

```
points = selectWorldPoints(wpSet,pointIds(1:50))
```

```
points = 50×3 single matrix
```

```
    0.1577    0.6827    0.6699
    0.1665    0.2842    0.4921
    0.9152    0.5038    0.7163
    0.1393    0.0580    0.1907
    0.6703    0.1710    0.7205
    0.5399    0.2495    0.9283
    0.8973    0.5022    0.7917
    0.9272    0.5898    0.0153
    0.2026    0.0748    0.2160
    0.1765    0.1055    0.2241
      ⋮
```

Input Arguments

`wpSet` — World point set
`worldpointset` object

World point set, specified as a `worldpointset` object.

`pointIds` — Identifiers for 3-D points
M-element row vector of integers

Identifiers for 3-D points in `wpSet`, specified as an *M*-element row vector of integers.

Output Arguments

`wpSetSelected` — Selected world point set
`worldpointset` object

Selected world point set, returned as a `worldpointset` object.

Version History

Introduced in R2024a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`worldpointset`

Functions
`removeWorldPoints` | `updateWorldPoints`

updateRepresentativeView

Update representative view ID and corresponding feature index

Syntax

```
wpSet = updateRepresentativeView(wpSet,pointIndices,viewTable)
wpSet = updateRepresentativeView(wpSet,pointIds,viewTable)
```

Description

`wpSet = updateRepresentativeView(wpSet,pointIndices,viewTable)` updates representative view ID and corresponding feature index of world points in the world point set object, `wpSet`. The representative view is based on a table of views, `viewTable`. `pointIndices` is the index to the world point related to the update. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = updateRepresentativeView(wpSet,pointIds,viewTable)` updates representative view ID and corresponding feature point identifier of world points in the world point set object, `wpSet`. The representative view is based on a table of views, `viewTable`. `pointIds` is the point identifier to the world point related to the update. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Update Representative View of World Point Set

Create an `imageviewset` object.

```
vSet = imageviewset;
```

Add three views related to binary features.

```
viewId1 = 1;
features1 = binaryFeatures(randi(255,[10, 32],"uint8"));
vSet = addView(vSet,viewId1,Features=features1);
viewId2 = 2;
features2 = binaryFeatures(randi(255,[10, 32],"uint8"));
vSet = addView(vSet,viewId2,Features=features2);
viewId3 = 3;
features3 = binaryFeatures(randi(255,[10, 32],"uint8"));
vSet = addView(vSet,viewId3,Features=features3);
```

Create a `worldpointset` object and add world points

```
wpSet = worldpointset;
worldPoints = rand(10, 3); % Add world points
wpSet = addWorldPoints(wpSet, worldPoints);
```

Add correspondences for three views.

```
pointIndices= 1:10;
featureIndices1 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId1,pointIndices,featureIndices1);
featureIndices2 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId2,pointIndices,featureIndices2);
featureIndices3 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId3,pointIndices,featureIndices3);
```

Update representative view.

```
viewTable = findView(vSet, [viewId1,viewId2,viewId3]);
wpSet = updateRepresentativeView(wpSet,pointIndices,viewTable);
```

Check representative view ID and feature index.

```
wpSet.RepresentativeViewId
```

```
ans = 10×1

     1
     3
     1
     2
     3
     3
     1
     3
     1
     2
```

```
wpSet.RepresentativeFeatureIndex
```

```
ans = 10×1

     9
     1
     8
     2
     6
     2
     2
     4
    10
     3
```

Update Representative View of World Point Set

Create an `imageviewset` object.

```
vSet = imageviewset;
```

Add three views related to binary features.

```
viewId1 = 1;
features1 = binaryFeatures(randi(255,[10, 32],"uint8"));
```

```
vSet = addView(vSet,viewId1,Features=features1);
viewId2 = 2;
features2 = binaryFeatures(randi(255,[10, 32],"uint8"));
vSet = addView(vSet,viewId2,Features=features2);
viewId3 = 3;
features3 = binaryFeatures(randi(255,[10, 32],"uint8"));
vSet = addView(vSet,viewId3,Features=features3);
```

Create a `worldpointset` object and add world points

```
wpSet = worldpointset;
worldPoints = rand(10, 3); % Add world points
pointIds = randperm(10);
wpSet = addWorldPoints(wpSet, worldPoints, pointIds);
```

Add correspondences for three views.

```
featureIndices1 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId1,pointIds,featureIndices1);
featureIndices2 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId2,pointIds,featureIndices2);
featureIndices3 = randperm(10);
wpSet  = addCorrespondences(wpSet,viewId3,pointIds,featureIndices3);
```

Update representative view.

```
viewTable = findView(vSet, [viewId1,viewId2,viewId3]);
wpSet = updateRepresentativeView(wpSet,pointIds,viewTable);
```

Check representative view ID and feature index.

```
wpSet.RepresentativeViewId
```

```
ans = 10×1

     1
     2
     2
     2
     2
     1
     1
     2
     1
     3
```

```
wpSet.RepresentativeFeatureIndex
```

```
ans = 10×1

     9
     1
     9
     3
     6
    10
     6
     4
```

```
5
7
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`pointIndices` — Point index

M-element column vector of integers

Point index, specified as an *M*-element column vector of integers, where *M* is the number of world points. The function updates the representative view ID and corresponding feature index of the point that corresponds to the specified index.

`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, returned as an *M*-element vector of integers. *M* is the number of world points. The function updates the representative view ID and corresponding feature point identifier of the point that corresponds to the specified index. Use this input when points were added using `addWorldPoints`.

`viewTable` — One or more views

four-column table

One or more connected or indirectly connected views, specified as a four-column table. The table must contain the columns as described in this table.

Column	Description
`ViewId`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Features`	Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures`.
`Points`	Image points, specified as one of these options: • *M*-by-2 matrix of coordinates in the format [*x*,*y*] • *M*-element feature point array. For more details, see “Point Feature Types”.

Output Arguments

`wpSet` — World point set

`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`updateLimitsAndDirection`

Objects

`imageviewset` | `worldpointset`

updateLimitsAndDirection

Update distance limits and viewing direction

Syntax

```
wpSet = updateLimitsAndDirection(wpSet,pointIndices,viewTable)
wpSet = updateLimitsAndDirection(wpSet,pointIds,viewTable)
```

Description

`wpSet = updateLimitsAndDirection(wpSet,pointIndices,viewTable)` updates the distance limits and viewing direction of world points in the world point set object, `wpSet`. The viewing direction is based on a table of views, `viewTable`. `pointIndices` is the index to the world point related to the update. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = updateLimitsAndDirection(wpSet,pointIds,viewTable)` updates the distance limits and viewing direction of world points in the world point set object, `wpSet`. The viewing direction is based on a table of views, `viewTable`. `pointIds` is the point identifier to the world point related to the update. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Update Distance Limits and Viewing Direction of World Points

Create a `worldpointset` object and add world points and correspondences for view 1 and view 2.

```
wpSet = worldpointset;
worldPoints = rand(10,3);
wpSet = addWorldPoints(wpSet,worldPoints);
pointIndices = 1:10;
featureIndices = 1:10;
viewId1 = 1;
wpSet = addCorrespondences(wpSet,viewId1,pointIndices,featureIndices);
viewId2 = 2;
wpSet = addCorrespondences(wpSet,viewId1,pointIndices,featureIndices);
```

Create an `imageviewset` object and add views and poses to the set. Add a translation in the *x*-axis for pose 2.

```
vSet  = imageviewset;
pose1 = rigidtform3d;
vSet  = addView(vSet, viewId1, pose1);
pose2 = rigidtform3d([0 0 0],[1 0 0]);
vSet  = addView(vSet,viewId2,pose2);
```

Update the distance limit and viewing direction.

```
viewTable = findView(vSet,[viewId1,viewId2]);
wpSet = updateLimitsAndDirection(wpSet,pointIndices,viewTable);
```

Check distance limits and viewing direction.

```
wpSet.ViewingDirection
```

```
ans = 10×3 single matrix

    0.7703    0.1490    0.6200
    0.6820    0.7308    0.0269
    0.0988    0.7444    0.6604
    0.6554    0.3483    0.6702
    0.5161    0.6532    0.5540
    0.1255    0.1826    0.9751
    0.3099    0.4693    0.8269
    0.4812    0.8058    0.3451
    0.6815    0.5638    0.4665
    0.7035    0.6996    0.1248
```

```
wpSet.DistanceLimits
```

```
ans = 10×2 single matrix

    1.0576    1.0576
    1.3281    1.3281
    1.2858    1.2858
    1.3936    1.3936
    1.2252    1.2252
    0.7771    0.7771
    0.8987    0.8987
    1.1364    1.1364
    1.4050    1.4050
    1.3715    1.3715
```

Update Distance Limits and Viewing Direction of World Points

Create a `worldpointset` object and add world points and correspondences for view 1 and view 2.

```
wpSet = worldpointset;
worldPoints = rand(10,3);
pointIds = randperm(10);
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
featureIndices = 1:10;
viewId1 = 1;
wpSet = addCorrespondences(wpSet,viewId1,pointIds(1:10),featureIndices);
viewId2 = 2;
wpSet = addCorrespondences(wpSet,viewId1,pointIds(1:10),featureIndices);
```

Create an `imageviewset` object and add views and poses to the set. Add a translation in the *x*-axis for pose 2.

```
vSet  = imageviewset;
pose1 = rigidtform3d;
vSet  = addView(vSet, viewId1, pose1);
pose2 = rigidtform3d([0 0 0],[1 0 0]);
vSet  = addView(vSet,viewId2,pose2);
```

Update the distance limit and viewing direction.

```
viewTable = findView(vSet,[viewId1,viewId2]);
wpSet = updateLimitsAndDirection(wpSet,pointIds(1:10),viewTable);
```

Check distance limits and viewing direction.

```
wpSet.ViewingDirection
```

```
ans = 10×3 single matrix

    0.7703    0.1490    0.6200
    0.6820    0.7308    0.0269
    0.0988    0.7444    0.6604
    0.6554    0.3483    0.6702
    0.5161    0.6532    0.5540
    0.1255    0.1826    0.9751
    0.3099    0.4693    0.8269
    0.4812    0.8058    0.3451
    0.6815    0.5638    0.4665
    0.7035    0.6996    0.1248
```

```
wpSet.DistanceLimits
```

```
ans = 10×2 single matrix

    1.0576    1.0576
    1.3281    1.3281
    1.2858    1.2858
    1.3936    1.3936
    1.2252    1.2252
    0.7771    0.7771
    0.8987    0.8987
    1.1364    1.1364
    1.4050    1.4050
    1.3715    1.3715
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`pointIndices` — Point index

M-element column vector of integers

Point index, specified as an *M*-element column vector of integers, where *M* is the number of world points. The function updates the distance limits and viewing direction of the point that corresponds to the specified index. Use this input when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, returned as an *M*-element vector of integers. *M* is the number of world points. The function updates the distance limits and viewing direction of the point that corresponds to the specified point identifier. Use this input when points were added using `addWorldPoints`.

`viewTable` — One or more views
four-column table

One or more connected or indirectly connected views, specified as a four-column table. The table must contain the columns as described in this table.

Column	Description
`ViewId`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Features`	Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures`.
`Points`	Image points, specified as one of these options: • *M*-by-2 matrix of coordinates in the format [*x*,*y*] • *M*-element feature point array. For more details, see “Point Feature Types”.

Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`updateRepresentativeView`

Objects
`imageviewset` | `worldpointset`

Topics
“Point Feature Types”

addCorrespondences

Update world points in a world point set

Syntax

```
wpSet = addCorrespondences(wpSet,viewId,pointIndices,featureIndices)
wpSet = addCorrespondences(wpSet,viewId,pointIds,featureIndices)
```

Description

`wpSet = addCorrespondences(wpSet,viewId,pointIndices,featureIndices)` adds correspondences between 3-D world points and 2-D feature points to the world point set `wpSet`, using the point indices `pointIndices` and the feature indices `featureIndices` for the specified view `viewId`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = addCorrespondences(wpSet,viewId,pointIds,featureIndices)` adds correspondences between 3-D world points and 2-D feature points to the world point set `wpSet`, using the point identifiers `pointIds` and the feature indices `featureIndices` for the specified view `viewId`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Add 3-D to 2-D Correspondences to World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Add correspondences to View 1 for the first 10 points.

```
viewId = 1;
pointIndices   = 1:10;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIndices,featureIndices);
```

Add 3-D to 2-D Correspondences to World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate pointIds.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Add correspondences to View 1 for the first 10 points.

```
viewId = 1;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIds(1:10),featureIndices);
```

Input Arguments

`wpSet` — World point set
`worldpointset` object

World point set, specified as a `worldpointset` object.

`viewId` — View identifier
scalar

View identifier, specified as a scalar.

`pointIds` — Identifiers for 3-D points
[ ] (default) | *M*-element row vector of integers

Identifiers for 3-D points in `wpSet`, specified as an *M*-element row vector of integers.

`pointIndices` — World point indices
M-element column vector of integers

World point indices, specified as an *M*-element column vector of integers. *M* is the number of world points for which you are adding correspondences.

`featureIndices` — 2-D feature point indices
M-element column vector of integers

Feature point indices, specified as an *M*-element column vector of integers. *M* is the number of world points for which you are adding correspondences.

Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`updateCorrespondences` | `removeCorrespondences`

Objects

`imageviewset` | `worldpointset`

addWorldPoints

Add world points to world point set

Syntax

```
wpSet = addWorldPoints(wpSet,worldPoints)
wpSet = addWorldPoints(wpSet,worldPoints,pointIds)
[wpSet,newPointIndices] = addWorldPoints( ___ )
```

Description

`wpSet = addWorldPoints(wpSet,worldPoints)` adds world points to the world points set `wpSet`.

`wpSet = addWorldPoints(wpSet,worldPoints,pointIds)` also adds identifiers for 3-D points in `worldPoints` to `wpSet`.

`[wpSet,newPointIndices] = addWorldPoints( ___ )` additionally returns the indices in the world point set of the added world points.

Examples

Add World Points to World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset
```

Add the world points to the world point set.

```
wpSet = addWorldPoints(wpSet,worldPoints)
```

Add World Points to World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate pointIds.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset
```

```
wpSet =
  worldpointset with properties:

                   WorldPoints: [0×3 single]
                       ViewIds: [1×0 uint32]
                      PointIds: [1×0 uint32]
              ViewingDirection: [0×3 single]
                DistanceLimits: [0×2 single]
           RepresentativeViewId: [0×1 double]
     RepresentativeFeatureIndex: [0×1 double]
                         Count: 0
               Correspondences: [0×3 table]
```

Add the world points to the world point set.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds)
```

```
wpSet =
  worldpointset with properties:

                   WorldPoints: [100×3 single]
                       ViewIds: [1×0 uint32]
                      PointIds: [33 96 61 3 43 1 25 4 19 6 36 90 76 58 20 10 69 99 35 60 21 57 10
              ViewingDirection: [100×3 single]
                DistanceLimits: [100×2 single]
           RepresentativeViewId: [100×1 double]
     RepresentativeFeatureIndex: [100×1 double]
                         Count: 100
               Correspondences: [100×3 table]
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`worldPoints` — World points coordinates

M-by-3 matrix

World points coordinates, specified as an *M*-by-3 matrix in the form [*x y z*].

`pointIds` — Identifiers for 3-D points

[ ] (default) | *M*-element row vector of integers

This property is read-only.

Identifiers for 3-D points in `WorldPoints`, specified as an *M*-element row vector of integers.

Output Arguments

`wpSet` — World point set

`worldpointset` object

World point set, returned as a `worldpointset` object.

`newPointIndices` — Added world point indices
M-element column vector of integers

Added world point indices, returned as an *M*-element column vector of integers. *M* is the number of world points added to the world point set.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`worldpointset`

Functions
`removeWorldPoints` | `updateWorldPoints`

findViewsOfWorldPoint

Find views that observe a world point

Syntax

```
viewIds = findViewsOfWorldPoint(wpSet,pointIndices)
[viewIds,featureIndices] = findViewsOfWorldPoint(wpSet,pointIndices)

viewIds = findViewsOfWorldPoint(wpSet,pointIds)
[viewIds,featureIndices] = findViewsOfWorldPoint(wpSet,pointIds)
```

Description

`viewIds = findViewsOfWorldPoint(wpSet,pointIndices)` finds the identifiers for views `viewIds` that observe the 3-D world points specified by `pointIndices` in a world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`[viewIds,featureIndices] = findViewsOfWorldPoint(wpSet,pointIndices)` additionally returns the indices of the 2-D feature points that correspond to each 3-D world point for each associated view. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`viewIds = findViewsOfWorldPoint(wpSet,pointIds)` finds the identifiers for views `viewIds` that observe the 3-D world points specified by `pointIds` in a world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

`[viewIds,featureIndices] = findViewsOfWorldPoint(wpSet,pointIds)` additionally returns the indices of the 2-D feature points that correspond to each 3-D world point for each associated view. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Find Views of World Points

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Add 3-D to 2-D correspondences for view 1.

```
viewId1 = 1;
pointIndices1   = 1:10;
featureIndices1 = 1:10;
wpSet  = addCorrespondences(wpSet,viewId1,pointIndices1,featureIndices1);
```

Add 3-D to 2-D correspondences for view 2.

```
viewId2 = 2;
pointIndices2   = 6:10;
featureIndices2 = 1:5;
wpSet  = addCorrespondences(wpSet,viewId2,pointIndices2,featureIndices2);
```

Find views of world points.

```
pointIndex = 6:10;
viewIds = findViewsOfWorldPoint(wpSet,pointIndex);
```

Find Views of World Points

Generate 3-D world points and point Ids.

```
worldPoints = rand(100,3);
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Add 3-D to 2-D correspondences for view 1.

```
viewId1 = 1;
featureIndices1 = 1:10;
wpSet  = addCorrespondences(wpSet,viewId1,pointIds(1:10),featureIndices1);
```

Add 3-D to 2-D correspondences for view 2.

```
viewId2 = 2;
featureIndices2 = 1:5;
wpSet  = addCorrespondences(wpSet,viewId2,pointIds(6:10),featureIndices2);
```

Find views of world points.

```
viewIds = findViewsOfWorldPoint(wpSet,pointIds(6:10))
```

```
viewIds=5×1 cell array
    {2×1 uint32}
    {2×1 uint32}
    {2×1 uint32}
    {2×1 uint32}
    {2×1 uint32}
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`pointIndices` — World point indices

scalar | *N*-element vector of integers

World point indices, specified as a scalar or an *N*-element vector of integers. This input is valid when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, specified as an *M*-element vector of integers. *M* is the number of world points. This input is valid when points were added using `addWorldPoints`.

Output Arguments

`viewIds` — View identifiers

M-element column vector | *N*-element cell array

View identifiers, returned as an *M*-element column vector or *N*-element cell array. The function returns an *M*-element vector column, where *M* is the number of views that observe the world point, when `pointIndex` is a scalar. The function returns an *N*-element cell array when `pointIndex` is an *N*-element vector of integers. Each cell contains a column vector of the view identifiers for the associated world point.

`featureIndices` — 2-D feature point indices

M-element column vector | *N*-element cell array

2-D feature point indices, returned as an *M*-element column vector or *N*-element cell array. The function returns an *M*-element vector column, where *M* is the number of views that observe the world point, when `pointIndex` is a scalar. The function returns an *N*-element cell array when `pointIndex` is an *N*-element vector of integers. Each cell contains a column vector of the corresponding feature indices for the associated world point.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

If the input argument `pointIndex` is a scalar, the output arguments `viewIds` and `featureIndices` returned by the generated code are *M*-element cell arrays. *M* is the number of views that observe the world point.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`findWorldPointsInView`

Objects
`imageviewset` | `worldpointset`

findWorldPointsInTracks

Find world points that correspond to point tracks

Syntax

```
pointIndices = findWorldPointsInTracks(wpSet,tracks)
[pointIndices,validIndex] = findWorldPointsInTracks(wpSet,tracks)

pointIds = findWorldPointsInTracks(wpSet,tracks)
[pointIds,validIndex] = findWorldPointsInTracks(wpSet,tracks)
```

Description

`pointIndices = findWorldPointsInTracks(wpSet,tracks)` finds the indices, `pointIndices`, of world points that correspond to the specified point tracks `tracks`.Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`[pointIndices,validIndex] = findWorldPointsInTracks(wpSet,tracks)` additionally returns a vector that indicates whether each point track has a corresponding world point. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`pointIds = findWorldPointsInTracks(wpSet,tracks)` finds the point identifiers, `pointIds`, of world points that correspond to the specified point tracks `tracks`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

`[pointIds,validIndex] = findWorldPointsInTracks(wpSet,tracks)` additionally returns a vector that indicates whether each point track has a corresponding world point. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Find World Points in Tracks

Load precomputed world point set and image view set.

```
data = load(fullfile(toolboxdir('vision'),'visiondata','worldpointsetAndTracks.mat'));
```

Find point tracks across views.

```
tracks = findTracks(data.vSet);
```

Find 3-D world points corresponding to point tracks.

```
pointIndices = findWorldPointsInTracks(data.wpSet,tracks);
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`tracks` — Point tracks

M-element row vector of `pointTrack` objects

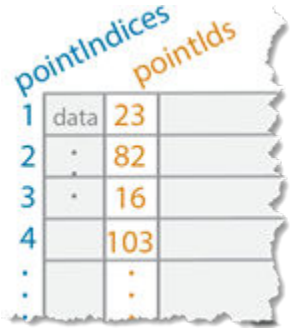
Point tracks, specified as an *M*-element row vector of `pointTrack` objects.

Output Arguments

`pointIndices` — World point indices

scalar | *N*-element vector of integers

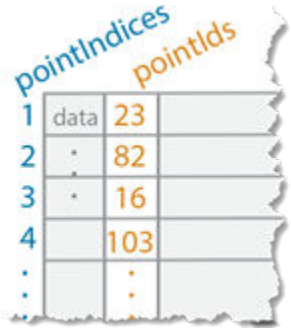
World point indices, specified as a scalar or an *N*-element vector of integers. This input is valid when sequential indices are used for the points in the `wpSet` object.



`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, specified as an *M*-element vector of integers. *M* is the number of world points. This input is valid when points were added using `addWorldPoints`.



`validIndex` — Logical index

M-element logical vector

Logical index of point track correspondences, returned as an *M*-element logical vector. *M* is the number of point tracks in the `tracks` argument. A value of `1` (`true`) indicates that the associated point track has a corresponding world point, and a value of `0` (`false`) indicates that it does not.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`findTracks` | `findWorldPointsInView` | `findViewsOfWorldPoint`

Objects

`imageviewset` | `worldpointset`

findWorldPointsInView

Find world points observed in view

Syntax

```
pointIndices = findWorldPointsInView(wpSet,viewId)
pointIds = findWorldPointsInView(wpSet,viewId)
[___,featureIndices] = findWorldPointsInView(wpSet,viewId)
```

Description

`pointIndices = findWorldPointsInView(wpSet,viewId)` finds the indices of the 3-D world points `pointIndices` observed in the specified view `viewId`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`pointIds = findWorldPointsInView(wpSet,viewId)` finds the point identifiers of the 3-D world points `pointIds` observed in the specified view `viewId`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

`[___,featureIndices] = findWorldPointsInView(wpSet,viewId)` additionally returns the indices of the corresponding 2-D feature points in the view.

Examples

Find World Points in World Points Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Add 3-D to 2-D correspondences for view 1.

```
viewId1 = 1;
pointIndices1   = 1:10;
featureIndices1 = 1:10;
wpSet  = addCorrespondences(wpSet,viewId1,pointIndices1,featureIndices1);
```

Add 3-D to 2-D correspondences for view 2.

```
viewId2 = 2;
pointIndices2   = 6:10;
featureIndices2 = 1:5;
wpSet  = addCorrespondences(wpSet,viewId2,pointIndices2,featureIndices2);
```

Find world points in view 2.

```
[pointIndices,featureIndices] = findWorldPointsInView(wpSet,viewId2)
```

```
pointIndices = 5×1

     6
     7
     8
     9
    10


featureIndices = 5×1

    1
    2
    3
    4
    5
```

Find World Points in World Points Set

Generate 3-D world points and point Ids.

```
worldPoints = rand(100,3);
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Add 3-D to 2-D correspondences for view 1.

```
viewId1 = 1;
featureIndices1 = 1:10;
wpSet  = addCorrespondences(wpSet,viewId1,pointIds(1:10),featureIndices1);
```

Add 3-D to 2-D correspondences for view 2.

```
viewId2 = 2;
featureIndices2 = 1:5;
wpSet  = addCorrespondences(wpSet,viewId2,pointIds(6:10),featureIndices2);
```

Find world points in view 2.

```
[pointIdsInView,featureIndices] = findWorldPointsInView(wpSet,viewId2);
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`viewId` — View identifier

scalar | *N*-element vector

View identifier, specified as a scalar or an *N*-element vector. *N* is the number of views specified. The dimensions of the `pointIndices` and `featureIndices` arguments depends on the value of the `viewId` argument.

`viewId`	`pointIndices`	`featureIndices`
scalar	*M*-element column vector	*M*-element column vector
N-element array	*N*-element cell array	*N*-element cell array

Output Arguments

`pointIndices` — 3-D world point indices

M-element column vector | *N*-element cell array

3-D world point indices, returned as an *M*-element column vector or an *N*-element cell array. *M* is the number of world points with corresponding feature points in the specified view. *N* is the number of views specified, and each element of the cell array contains a column vector of the world point indices with corresponding feature points in the associated view. This output is valid when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, returned as an *M*-element vector of integers. *M* is the number of world points found from the `wpSet` object. This output is valid when points were added using `addWorldPoints`.

`featureIndices` — 2-D feature point indices

M-element column vector | *N*-element cell array

2-D feature point indices, returned as an *M*-element column vector or an *N*-element cell array. *M* is the number of world points with corresponding feature points in the specified view. *N* is the number of views specified, and each element of the cell array contains a column vector of the feature point indices in the associated view with corresponding world points.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

If the input argument `viewId` is a scalar, the output arguments `pointIndices` and `featureIndices` returned by the generated code are *M*-element cell arrays. *M* is the number of world points with corresponding feature points in the specified view.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`findViewsOfWorldPoint` | `addCorrespondences`

Objects
`imageviewset`

removeCorrespondences

Remove 3-D to 2-D correspondences from world point set

Syntax

```
wpSet = removeCorrespondences(wpSet,viewId,pointIndices)
wpSet = removeCorrespondences(wpSet,viewId,pointIds)
```

Description

`wpSet = removeCorrespondences(wpSet,viewId,pointIndices)` removes the correspondences between the 3-D world points, specified by the point indices `pointIndices`, and 2-D feature points for the specified view `viewId` from a world point set. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = removeCorrespondences(wpSet,viewId,pointIds)` removes the correspondences between the 3-D world points, specified by point identifiers `pointIds` and 2-D feature points for the specified view `viewId` from a world point set. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Remove Correspondences in World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Add correspondences for a view.

```
viewId = 1;
pointIndices   = 1:10;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIndices,featureIndices);
```

Remove the first 5 correspondences.

```
pointIndices = 1:5;
wpSet  = removeCorrespondences(wpSet,viewId,pointIndices);
```

Remove Correspondences in World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate pointIds.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Add correspondences for a view.

```
viewId = 1;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIds(1:10),featureIndices);
```

Remove the first 5 correspondences.

```
wpSet  = removeCorrespondences(wpSet,viewId,pointIds(1:5));
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

World point set, specified as a `worldpointset` object.

`viewId` — View identifier

scalar

View identifier, specified as a scalar.

`pointIndices` — World point indices

M-element column vector of integers

World point indices, specified as an *M*-element column vector of integers. *M* is the number of world points from which to remove correspondences. Use this input when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, returned as an *M*-element vector of integers. *M* is the number of world points from which to remove correspondences. Use this input when points were added using `addWorldPoints`.

Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`addCorrespondences` | `findViewsOfWorldPoint`

Objects
`imageviewset` | `worldpointset`

removeWorldPoints

Remove world points from world point set

Syntax

```
wpSet = removeWorldPoints(wpSet,pointIndices)
wpSet = removeWorldPoints(wpSet,pointIds)
```

Description

`wpSet = removeWorldPoints(wpSet,pointIndices)` removes the world points at the specified indices `pointIndices` from the world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = removeWorldPoints(wpSet,pointIds)` removes the world points specified by `pointIDs` from the world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Remove Points from World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add the world points to the world point set. Display the world point set.

```
[wpSet,newPointIndices] = addWorldPoints(wpSet,worldPoints);
wpSet
```

```
wpSet =
  worldpointset with properties:

                   WorldPoints: [100×3 single]
                       ViewIds: [1×0 uint32]
                      PointIds: [1×0 uint32]
              ViewingDirection: [100×3 single]
                DistanceLimits: [100×2 single]
          RepresentativeViewId: [100×1 double]
    RepresentativeFeatureIndex: [100×1 double]
                         Count: 100
               Correspondences: [100×3 table]
```

Remove the first 50 world points from the world point set.

```
wpSet = removeWorldPoints(wpSet,newPointIndices(1:50))

wpSet =
  worldpointset with properties:

                   WorldPoints: [50×3 single]
                       ViewIds: [1×0 uint32]
                      PointIds: [1×0 uint32]
              ViewingDirection: [50×3 single]
                DistanceLimits: [50×2 single]
           RepresentativeViewId: [50×1 double]
    RepresentativeFeatureIndex: [50×1 double]
                         Count: 50
               Correspondences: [50×3 table]
```

Remove Points from World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate pointIds.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add the world points to the world point set. Display the world point set.

```
[wpSet,newPointIndices] = addWorldPoints(wpSet,worldPoints,pointIds);
```

```
wpSet
```

Remove the first 50 world points from the world point set.

```
wpSet = removeWorldPoints(wpSet,pointIds(1:50))

wpSet =
  worldpointset with properties:

                   WorldPoints: [50×3 single]
                       ViewIds: [1×0 uint32]
                      PointIds: [16 67 75 39 12 5 27 54 47 62 80 89 77 95 83 30 71 14 29 11 53 50
              ViewingDirection: [50×3 single]
                DistanceLimits: [50×2 single]
           RepresentativeViewId: [50×1 double]
    RepresentativeFeatureIndex: [50×1 double]
                         Count: 50
               Correspondences: [50×3 table]
```

Input Arguments

`wpSet` — World point set

`worldpointset` object

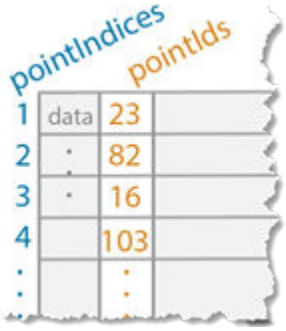
World point set, specified as a `worldpointset` object.

`pointIndices` — World point indices

M-element array vector of integers

World point indices, specified as an *M*-element vector of integers. *M* is the number of world points to remove from the `worldpointset` object. Use this input when sequential indices are used to identify points.

- `pointIds` — Unique identifiers assigned to points when using the `addWorldPoints` to add points.
- `pointIndices` — Sequential identifiers assigned to points.

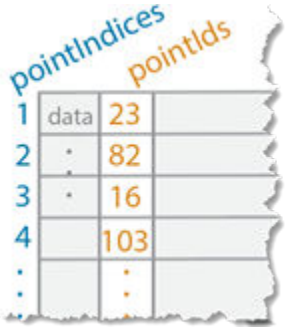


`pointIds` — World point identifiers

M-element array vector of integers

World point identifiers, specified as an *M*-element vector of integers. *M* is the number of world points to remove from the `worldpointset` object. Use this input if you used the `addWorldPoints` function when points were added.

- `pointIds` — Unique identifiers assigned to points when using the `addWorldPoints` to add points.
- `pointIndices` — Sequential identifiers assigned to points.



Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2020b

R2024a: Use unique point identifiers to specify world points

New for this release, you can use `pointIds` to specify unique point IDs to specify world points. In prior releases if you did not specify unique identifiers when points were added, then points were saved with sequential identifiers. World points can only be saved with one method, either using sequential `pointIndices` or with unique IDs `pointIDs`. The function will return an error if both types of IDs are passed into it.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`addWorldPoints` | `updateWorldPoints`

Objects
`imageviewset` | `worldpointset`

updateCorrespondences

Update 3-D to 2-D correspondences in world point set

Syntax

```
wpSet = updateCorrespondences(wpSet,viewId,pointIndices,featureIndices)
wpSet = updateCorrespondences(wpSet,viewId,pointIds,featureIndices)
```

Description

`wpSet = updateCorrespondences(wpSet,viewId,pointIndices,featureIndices)` updates the correspondences between the 3-D world points specified by `pointIndices` and 2-D feature points `featureIndices` for the specified view `viewId` in a world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = updateCorrespondences(wpSet,viewId,pointIds,featureIndices)` updates the correspondences between the 3-D world points specified by the point identifiers `pointIds` and 2-D feature points `featureIndices` for the specified view `viewId` in a world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Update Correspondences in World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Add correspondences for a view.

```
viewId = 1;
pointIndices   = 1:10;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIndices,featureIndices);
```

Update the feature indices.

```
newFeatureIndices = 11:20;
wpSet  = updateCorrespondences(wpSet,viewId,pointIndices,newFeatureIndices);
```

Update Correspondences in World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate identifiers for 3-D world points.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add world points.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Add correspondences for a view.

```
viewId = 1;
featureIndices = 1:10;
wpSet  = addCorrespondences(wpSet,viewId,pointIds(1:10),featureIndices);
```

Update the feature indices.

```
newFeatureIndices = 11:20;
wpSet  = updateCorrespondences(wpSet,viewId,pointIds(1:10),newFeatureIndices);
```

Input Arguments

`wpSet` — World point set
`worldpointset` object

World point set, specified as a `worldpointset` object.

`viewId` — View identifier
scalar

View identifier, specified as a scalar.

`pointIndices` — World point indices
M-element column vector of integers

World point indices, specified as an *M*-element column vector of integers. *M* is the number of world points for which to update correspondences in the world point set. Use this input when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers
M-element array vector of integers

World point identifiers, returned as an *M*-element vector of integers. *M* is the number of world points from which to update correspondences. Use this input when points were added using `addWorldPoints`.

`featureIndices` — 2-D feature point indices
M-element column vector of integers

Feature point indices, specified as an *M*-element column vector of integers. *M* is the number of world points for which to update correspondences in the world point set.

Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`removeCorrespondences` | `addCorrespondences`

Objects
`imageviewset` | `worldpointset`

updateWorldPoints

Update world points in world point set

Syntax

```
wpSet = updateWorldPoints(wpSet,pointIndices,worldPoints)
wpSet = updateWorldPoints(wpSet,pointIds,worldPoints)
```

Description

`wpSet = updateWorldPoints(wpSet,pointIndices,worldPoints)` updates the locations of the world points at the specified indices `pointIndices` of the world point set `wpSet`. Use this syntax if you added points using the `addWorldPoints` function without specifying unique point identifiers `pointIds`.

`wpSet = updateWorldPoints(wpSet,pointIds,worldPoints)` updates the locations of the world points specified by the point identifiers `pointIds`, of the world point set `wpSet`.Use this syntax if you added points using the `addWorldPoints` function by specifying unique point identifiers `pointIds`.

Examples

Update World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add the world points to the world point set.

```
wpSet = addWorldPoints(wpSet,worldPoints);
```

Update the first 50 world points with new locations.

```
pointIndices = 1:50;
newWorldPoints = worldPoints(pointIndices,:) + [0 0 5];
wpSet = updateWorldPoints(wpSet,pointIndices,newWorldPoints);
```

Update World Point Set

Generate 3-D world points.

```
worldPoints = rand(100,3);
```

Generate pointIds.

```
pointIds = randperm(100);
```

Create a `worldpointset` object.

```
wpSet = worldpointset;
```

Add the world points to the world point set.

```
wpSet = addWorldPoints(wpSet,worldPoints,pointIds);
```

Update the first 50 world points with new locations.

```
newWorldPoints = selectWorldPoints(wpSet,pointIds(1:50)) + [0 0 5];
wpSet = updateWorldPoints(wpSet,pointIds(1:50),newWorldPoints);
```

Input Arguments

`wpSet` — World point set
`worldpointset` object

World point set, specified as a `worldpointset` object.

`pointIndices` — World point indices
scalar | *N*-element vector of integers

World point indices, specified as a scalar or an *N*-element vector of integers. This input is valid when sequential indices are used for the points in the `wpSet` object.

`pointIds` — World point identifiers
M-element array vector of integers

World point identifiers, specified as an *M*-element vector of integers. *M* is the number of world points. This input is valid when points were added using `addWorldPoints`.

`worldPoints` — World points coordinates
M-by-3 matrix

World points coordinates, specified as an *M*-by-3 matrix in the form [*x y z*].

Output Arguments

`wpSet` — World point set
`worldpointset` object

World point set, returned as a `worldpointset` object.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`worldpointset`

Functions
`addWorldPoints` | `removeWorldPoints`

pcviewset

Manage data for point cloud based odometry and SLAM

Description

The `pcviewset` object stores point cloud odometry and simultaneous localization and mapping (SLAM) data as a set of views and pairwise connections between views.

Creation

Syntax

```
vSet = pcviewset
```

Description

`vSet = pcviewset` creates a pcviewset object with default property names. Use object functions to perform actions such as adding, modifying, or removing views or connections.

Properties

`Views` — View attributes

empty three-column table (default) | three-column table

This property is read-only.

View attributes, specified as a three-column table. The table contains columns as described in this table.

Column	Description
`ViewID`	View identifier, specified as an integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`PointCloud`	Point cloud, specified as a `pointCloud` object.

`Connections` — Pairwise connections between views

empty four-column table (default) | four-column table

This property is read-only.

Pairwise connections between views, specified as a four-column table. The table contains columns as described in this table. Each row corresponds to one connection.

Column	Description
`ViewID1`	View identifier for the first view, specified as a unique integer.
`ViewID2`	View identifier for the second view, specified as a unique integer.
`RelativePose`	Relative pose of the second view with respect to the first view, specified as a `rigidtform3d` object.
`InformationMatrix`	Information matrix, specified as a 6-by-6 matrix. The information matrix represents the uncertainty of the measurement error and is the inverse of the covariance matrix.

`NumViews` — Number of views
`0` (default) | nonnegative integer

This property is read-only.

Number of views, specified as a nonnegative integer.

`NumConnections` — Number of connections
`0` (default) | nonnegative integer

This property is read-only.

Number of connections, specified as a nonnegative integer.

Object Functions

addView	Add views to view set
updateView	Update view in view set
deleteView	Delete view from view set
hasView	Check if view is in view set
addConnection	Add connection between views in view set
updateConnection	Update connection between views in a view set
deleteConnection	Delete a connection between views in view set
hasConnection	Check if connection between two views is in view set
findView	Find views associated with view identifiers
findConnection	Find connections associated with view identifiers
connectedViews	Connected views in view set
poses	Absolute poses associated with views in view set
createPoseGraph	Create pose graph
optimizePoses	Optimize absolute poses using relative pose constraints
plot	Plot point cloud view set views and connections

Examples

Perform Lidar Odometry Using Point Cloud Registration

Create a view set to hold odometry.

```
 vSet = pcviewset;
```

Create a Velodyne reader to read point clouds.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
ptCloud = readFrame(veloReader);
```

Preprocess one frame to remove the ground plane and to downsample point cloud.

```
elevationDelta = 25;
gridStep = 0.2;
groundPtsIdx = segmentGroundFromLidarData(ptCloud, ...
    ElevationAngleDelta=elevationDelta);
ptCloud = select(ptCloud,~groundPtsIdx,Output='full');
ptCloud = pcdownsample(ptCloud,'gridAverage',gridStep);
```

Initialize attributes for the first view.

```
absPose = rigidtform3d;
relPose = rigidtform3d;
```

Add the first view to the point cloud view set.

```
vSet = addView(vSet,1,absPose,PointCloud=ptCloud);
```

Initialize a point cloud map using the first view.

```
ptCloudMap = copy(ptCloud);

skipFrames = 5;
prevViewId = 1;
prevPtCloud = ptCloud;
```

Loop over frames to update odometry and the point cloud map.

```
for viewId = 6:skipFrames:40
    % Read point cloud.
    ptCloud = readFrame(veloReader,viewId);

    % Preprocess the frame.
    groundPtsIdx = segmentGroundFromLidarData(ptCloud, ...
        ElevationAngleDelta=elevationDelta);
    ptCloud = select(ptCloud,~groundPtsIdx,Output='full');
    ptCloud = pcdownsample(ptCloud,'gridAverage',gridStep);

    % Register new point cloud against the previous one.
    regGridStep = 5;
    relPose = pcregisterndt(ptCloud,prevPtCloud,regGridStep, ...
        InitialTransform=relPose);

    % Update the absolute pose.
    absPose = rigidtform3d(absPose.A * relPose.A);

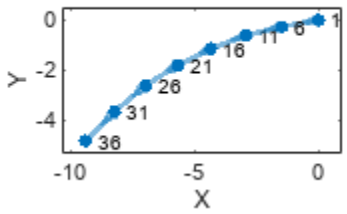
    % Add new view and connection to the previous view.
    vSet = addView(vSet,viewId,absPose,PointCloud=ptCloud);
    vSet = addConnection(vSet,prevViewId,viewId,relPose);

    prevPtCloud = ptCloud;
```

```
    prevViewId = viewId;
end
```

Display the view set in a default 2-D view.

```
plot(vSet,ShowViewIds='on')
view(2)
```



Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `pcviewset` object and its object functions are updated to use `rigidtform3d` objects, which support the premultiply convention. The `pcviewset` object now:

- Returns the `Views` property with the `AbsolutePose` column as a `rigidtform3d` object
- Returns the `Connections` property with the `RelativePose` column as a `rigidtform3d` object

Before, the properties returned `AbsolutePose` and `RelativePose` as `rigid3d` objects, which use the postmultiply convention.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`rigidtform3d` | `table` | `digraph`

Topics
“Build a Map from Lidar Data Using SLAM”

addView

Add views to view set

Syntax

```
vSet = addView(vSet,viewId)
vSet = addView(vSet,viewId,absPose)
vSet = addView( ___ ,"PointCloud",ptCloud)

vSet = addView(vSet,viewTable)
```

Description

`vSet = addView(vSet,viewId)` adds the view specified by `viewId` to the view set, `vSet`.

`vSet = addView(vSet,viewId,absPose)` specifies the absolute pose of the view.

`vSet = addView( ___ ,"PointCloud",ptCloud)` specifies the point cloud associated with the view in addition to any of the input argument combinations in previous syntaxes.

`vSet = addView(vSet,viewTable)` adds one or more views in the table specified by `viewTable`.

Examples

Update View in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add a view to the point cloud view set.

```
viewId  = 1;
ptCloud = pcread("teapot.ply");

vSet = addView(vSet,viewId,"PointCloud",ptCloud);
```

Update the absolute pose of the view.

```
rotationAngles = [0 0 0];
translation = [2 0 0];
absPose = rigidtform3d(rotationAngles,translation);
vSet = updateView(vSet,viewId,absPose);
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId` — View identifier
positive integer

View identifier, specified as a positive integer. View identifiers are unique to a specific view.

`absPose` — Absolute pose of the view
`rigidtform3d` object

Absolute pose of the view, specified as a `rigidtform3d` object.

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`viewTable` — One or more views
two-column table | three-column table

One or more views, specified as a two-column or three-column table. The table must contain the columns `ViewId` and `AbsolutePose`. The `Points` column is optional.

Column	Description
`ViewID`	View identifier, specified as an integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Points`	Point cloud for the view, specified as a `pointCloud` object.

Output Arguments

`vSet` — View set with added views
`pcviewset` object

View set with added views, returned as a `pcviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify poses (the `absPose` argument and the `AbsolutePose` column of the `viewTable` argument) as `rigidtform3d` objects, which use the premultiply convention.

Although you can still specify poses as `rigid3d` objects, these objects are not recommended because they use the postmultiply convention. The `addView` function automatically converts the poses to the premultiply convention and returns a `pcviewset` object that represents the poses as `rigidtform3d` objects.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When you specify table as an input argument, then code generation has limitations. For information about the limitations, see “Code Generation for Tables” (MATLAB Coder) and “Table Limitations for Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects

`pcviewset`

updateView

Update view in view set

Syntax

```
vSet = updateView(vSet,viewId,absPose)
vSet = updateView( ___ ,"PointCloud",ptCloud)

vSet = updateView(vSet,viewTable)
```

Description

`vSet = updateView(vSet,viewId,absPose)` updates the view specified by `viewId` with the absolute pose `absPose`.

`vSet = updateView( ___ ,"PointCloud",ptCloud)` specifies the point cloud associated with the view.

`vSet = updateView(vSet,viewTable)` updates the views specified in the table `viewTable`.

Examples

Update View in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add a view to the point cloud view set.

```
viewId  = 1;
ptCloud = pcread("teapot.ply");

vSet = addView(vSet,viewId,"PointCloud",ptCloud);
```

Update the absolute pose of the view.

```
rotationAngles = [0 0 0];
translation = [2 0 0];
absPose = rigidtform3d(rotationAngles,translation);
vSet = updateView(vSet,viewId,absPose);
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

View set, specified as an `pcviewset` object.

`viewId` — View identifier
integer

Point cloud view identifier, specified as a unique integer.

`absPose` — Absolute pose
`rigidtform3d` object

Absolute pose, specified as a `rigidtform3d` object.

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`viewTable` — New view or set of views
two-column table | three-column table

New view or set of views, specified as a two- or three- column table. The table must contain the columns `ViewId` and `AbsolutePose`, and an optional column, `Points`.

Column	Description
`ViewID`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Points`	A `pointCloud`.

Output Arguments

`vSet` — View set with updated view
`pcviewset` object

View set with updated view, returned as a `pcviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify poses (the `absPose` argument and the `AbsolutePose` column of the `viewTable` argument) as `rigidtform3d` objects, which use the premultiply convention.

Although you can still specify poses as `rigid3d` objects, these objects are not recommended because they use the postmultiply convention. The `updateView` function automatically converts the poses to the premultiply convention and returns a `pcviewset` object that represents the poses as `rigidtform3d` objects.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When you specify table as an input argument, then code generation has limitations. For information about the limitations, see “Code Generation for Tables” (MATLAB Coder) and “Table Limitations for Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects

`rigidtform3d` | `pcviewset`

deleteView

Delete view from view set

Syntax

```
vSet = deleteView(vSet,viewId)
```

Description

`vSet = deleteView(vSet,viewId)` deletes the view specified by `viewId` from the view set, `vSet`.

Examples

Delete View from Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add two views to the point cloud view set.

```
viewId1 = 1;
viewId2 = 2;
ptCloud1 = pcread('teapot.ply');
ptCloud2 = pctransform(ptCloud1,rigidtform3d([0 0 0],[2 0 0]));

vSet = addView(vSet,viewId1,'PointCloud',ptCloud1);
vSet = addView(vSet,viewId2,'PointCloud',ptCloud2);
```

Delete the first view.

```
vSet = deleteView(vSet,viewId1);
```

Input Arguments

`vSet` — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId` — View identifier

positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

vSet — View set with deleted views

`pcviewset` object

View set with deleted views, returned as a `pcviewset` object.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcviewset`

hasView

Check if view is in view set

Syntax

```
tf = hasView(vSet,viewId)
```

Description

`tf = hasView(vSet,viewId)` returns `1` (true), if the view specified by `viewId` is in the view set, `vSet`. The function returns `0` (false), if the view is not in `vSet`.

Examples

Check If View Is in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add a view with ID `1` to the point cloud view set.

```
ptCloud = pcread('teapot.ply');
vSet = addView(vSet,1,'PointCloud',ptCloud);
```

Check if a view with ID `1` is in the view set.

```
hasView(vSet,1)
```

```
ans = logical
   1
```

Check if a view with ID `2` is in the view set.

```
hasView(vSet,2)
```

```
ans = logical
   0
```

Input Arguments

`vSet` — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId` — View identifier

positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`tf` — True or false result

`1 | 0`

True or false result indicating if view is in view set `vSet`, returned as a `1` or `0` data type `logical`.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcviewset`

addConnection

Add connection between views in view set

Syntax

```
vSet = addConnection(vSet,viewId1,viewId2)
vSet = addConnection(vSet,viewId1,viewId2,relPose)
vSet = addConnection(vSet,viewId1,viewId2,relPose,infoMat)
```

Description

`vSet = addConnection(vSet,viewId1,viewId2)` adds a connection between views `viewId1` and `viewId2` to the view set, `vSet`.

`vSet = addConnection(vSet,viewId1,viewId2,relPose)` specifies the relative pose of `viewId2` with respect to `viewId1`.

`vSet = addConnection(vSet,viewId1,viewId2,relPose,infoMat)` specifies the information matrix associated with the connection.

Examples

Add Connection to Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Load point cloud data.

```
data = load('livingRoom.mat');
ptCloud1 = data.livingRoomData{1};
ptCloud2 = data.livingRoomData{2};
```

Add `ptCloud1` to the point cloud view set. Specify a `rigidtform3d` object as the origin.

```
absPose1 = rigidtform3d;
vSet = addView(vSet,1,absPose1,'PointCloud',ptCloud1);
```

Add `ptCloud2` to the point cloud view set. Specify a `rigidtform3d` object as the origin.

```
vSet = addView(vSet,2,absPose1,'PointCloud',ptCloud2);
```

Estimate the rigid transformation between the two point clouds.

```
ptCloud2Downsampled = pcdownsample(ptCloud2,'gridAverage',0.1);
gridStep = 0.5;
relPose = pcregisterndt(ptCloud2Downsampled,ptCloud1,gridStep);
```

Add a connection between the views.

```
vSet = addConnection(vSet,1,2,relPose);
```

Compute the accumulated absolute pose.

```
absPose2 = rigidtform3d(absPose1.A * relPose.A);
```

Update the absolute pose of second view.

```
vSet = updateView(vSet,2,absPose2);
```

Input Arguments

`vSet` — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId1` — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2

positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

`relPose` — Relative pose

`rigidtform3d` object

Relative pose of `viewId2` with respect to `viewId1`, specified as a `rigidtform3d` object.

`infoMat` — Information matrix

6-by-6 numeric matrix

Information matrix associated with the connection, specified as a 6-by-6 numeric matrix.

Output Arguments

`vSet` — View set with added connections

`pcviewset` object

View set with added connections between views, returned as a `pcviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `relPose` argument as a `rigidtform3d` object, which uses the premultiply convention.

Although you can still specify `relPose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. The `addConnection` function automatically converts the relative pose to the premultiply convention and returns a `pcviewset` object that represents the pose as a `rigidtform3d` object.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`rigidtform3d` | `pcviewset`

updateConnection

Update connection between views in a view set

Syntax

```
vSet = updateConnection(vSet,viewId1,viewId2,relPose)
vSet = updateConnection(vSet,viewId1,viewId2,relPose,infoMat)
```

Description

`vSet = updateConnection(vSet,viewId1,viewId2,relPose)` updates the connection between views `viewId1` and `viewId2` with the relative pose specified by `relPose`.

`vSet = updateConnection(vSet,viewId1,viewId2,relPose,infoMat)` specifies the information matrix associated with the connection.

Examples

Update Connection in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Load point cloud data.

```
data = load('livingRoom.mat');
ptCloud1 = data.livingRoomData{1};
ptCloud2 = data.livingRoomData{2};
```

Add two views to the point cloud view set.

```
vSet = addView(vSet,1,'PointCloud',ptCloud1);
vSet = addView(vSet,2,'PointCloud',ptCloud2);
```

Add a connection between the two views.

```
vSet = addConnection(vSet,1,2);
```

Estimate the rigid transformation between the two point clouds.

```
ptCloud2Downsampled = pcdownsample(ptCloud2,'gridAverage',0.1);

gridStep = 0.5;
relPose = pcregisterndt(ptCloud2Downsampled,ptCloud1,gridStep);
```

Compute the accumulated absolute pose for each view.

```
absPose1 = rigidtform3d;
absPose2 = rigidtform3d(absPose1.A * relPose.A);
```

Update the connection with relative pose between views.

```
vSet = updateConnection(vSet,1,2,relPose);
```

Update the views with the absolute poses.

```
vSet = updateView(vSet,1,absPose1);
vSet = updateView(vSet,2,absPose2);
```

Input Arguments

`vSet` — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId1` — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2

positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

`relPose` — Relative pose

`rigidtform3d` object

Relative pose of `viewId2` with respect to `viewId1`, specified as a `rigidtform3d` object.

`infoMat` — Information matrix

6-by-6 numeric matrix

Information matrix associated with the connection, specified as a 6-by-6 numeric matrix.

Output Arguments

`vSet` — View set with updated connections

`pcviewset` object

View set with updated connections between views, returned as a `pcviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `relPose` argument as a `rigidtform3d` object, which uses the premultiply convention.

Although you can still specify `relPose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. The `updateConnection` function automatically converts the

relative pose to the premultiply convention and returns a `pcviewset` object that represents the pose as a `rigidtform3d` object.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`rigidtform3d` | `pcviewset`

deleteConnection

Delete a connection between views in view set

Syntax

```
vSet = deleteConnection(vSet,viewId1,viewId2)
```

Description

`vSet = deleteConnection(vSet,viewId1,viewId2)` deletes the connection between views `viewId1` and `viewId2` in the view set, `vSet`.

Examples

Delete Connection Between Views in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add two views and one connection to the view set.

```
vSet = addView(vSet,1);
vSet = addView(vSet,2);
vSet = addConnection(vSet,1,2);
```

Delete the connection between the two views.

```
vSet = deleteConnection(vSet,1,2);
```

Input Arguments

`vSet` — Point cloud view set
`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId1` — View identifier 1
positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2
positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

vSet — View set with deleted connections
`pcviewset` object

View set with deleted connections between views, returned as a `pcviewset` object.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pcviewset`

hasConnection

Check if connection between two views is in view set

Syntax

```
tf = hasConnection(vSet,viewId1,viewId2)
```

Description

`tf = hasConnection(vSet,viewId1,viewId2)` returns `1` (true), if the connection between views `viewId1` and `viewId2` is in the view set, `vSet`. The function returns `0` (false), if the connection does not exist in `vSet`.

Examples

Check for Connection Between Views in Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add two views to the point cloud view set.

```
vSet = addView(vSet,1);
vSet = addView(vSet,2);
```

Add a connection to the point cloud view set.

```
vSet = addConnection(vSet,1,2);
```

Check if the connection is between views is in the point cloud view set.

```
tf = hasConnection(vSet,1,2)
```

```
tf = logical
   1
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

viewId1 — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2
positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`tf` — True or false result
`1` | `0`

True or false result indicating if the connection between views `viewId1` and `viewId2` exists in view set `vSet`, returned as a `1` or `0` data type `logical`.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pcviewset`

connectedViews

Connected views in view set

Syntax

```
viewTable = connectedViews(vSet,viewId)
```

Description

`viewTable = connectedViews(vSet,viewId)` returns a table of views from `vSet` that are connected to the view specified by `viewId`.

Examples

Get Connected Views in View Set

Create an empty point cloud view set object.

```
vSet = pcviewset;
```

Add views to the point cloud view set.

```
vSet = addView(vSet,1);
vSet = addView(vSet,2);
vSet = addView(vSet,3);
vSet = addView(vSet,4);
```

Connect views in the point cloud view set.

```
vSet = addConnection(vSet,1,2);
vSet = addConnection(vSet,2,3);
vSet = addConnection(vSet,2,4);
```

Get the connected views for the view with ID 2.

```
viewTable = connectedViews(vSet,2)
```

```
viewTable=3×3 table
    ViewId      AbsolutePose        PointCloud
    ______    _________________    _____________

      1       1×1 rigidtform3d    1×1 pointCloud
      3       1×1 rigidtform3d    1×1 pointCloud
      4       1×1 rigidtform3d    1×1 pointCloud
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewId` — View identifier
positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`viewTable` — One or more views
two-column table | three-column table

One or more views, returned as a three-column table. The table must contain the columns as described in this table.

Column	Description
`ViewId`	View identifier, specified as an integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Points`	Point cloud for the view, specified as a `pointCloud` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `connectedViews` function now returns the `AbsolutePose` value in the `viewTable` argument as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `AbsolutePose` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pcviewset` | `rigidtform3d`

poses

Absolute poses associated with views in view set

Syntax

```
sensorPoses = poses(vSet)
```

Description

`sensorPoses = poses(vSet)` returns a table of absolute poses associated with the views contained in the view set, `vSet`.

Examples

Get Absolute Poses from Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Load point cloud data.

```
data = load('livingRoom.mat');
ptCloud1 = data.livingRoomData{1};
ptCloud2 = data.livingRoomData{2};
```

Add `ptCloud1` to the view set.

```
absPose1 = rigidtform3d;
vSet = addView(vSet,1,absPose1,'PointCloud',ptCloud1);
```

Add `ptCloud2` to the view set.

```
vSet = addView(vSet,2,'PointCloud',ptCloud2);
```

Estimate the rigid transformation between the two point clouds.

```
ptCloud2Downsampled = pcdownsample(ptCloud2,'gridAverage',0.1);

gridStep = 0.5;
relPose = pcregisterndt(ptCloud2Downsampled,ptCloud1,gridStep);
```

Add a connection between the two views.

```
vSet = addConnection(vSet,1,2,relPose);
```

Compute the accumulated absolute pose.

```
absPose2 = rigidtform3d(absPose1.A * relPose.A);
```

Update the absolute pose of second view.

```
vSet = updateView(vSet,2,absPose2);
```

Get absolute poses.

```
sensorPoses = poses(vSet)
```

```
sensorPoses=2×2 table
    ViewId      AbsolutePose
    ______    _________________

      1       1×1 rigidtform3d
      2       1×1 rigidtform3d
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

Output Arguments

sensorPoses — Absolute poses

two-column table

Absolute poses, returned as a two-column table. The table contains columns as described in this table.

Column	Description
`ViewID`	View identifier, returned as a positive integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, returned as a `rigidtform3d` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `poses` function now returns the `AbsolutePose` value in the `sensorPoses` argument as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `AbsolutePose` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When generating code, the output argument `sensorPoses` is returned as a structure with fields *ViewID* and *AbsolutePose*.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcviewset` | `rigidtform3d`

optimizePoses

Optimize absolute poses using relative pose constraints

Syntax

```
vSetOptim = optimizePoses(vSet)
vSetOptim = optimizePoses(vSet,Name,Value)
```

Description

`vSetOptim = optimizePoses(vSet)` returns a point cloud view set whose absolute poses are optimized. `vSetOptim` and `vSet` are `pcviewset` objects.

The `optimizePoses` function performs pose graph optimization on the absolute poses for the `Views` in the view set using the relative pose constraints established by the `Connections` property. You can use `optimizePoses` to correct drift in odometry after detecting loop closures.

`vSetOptim = optimizePoses(vSet,Name,Value)` specifies options using one or more name-value pair arguments. For example, `'Tolerance',0.2` sets the tolerance of the optimization cost function to `0.2`.

Examples

Create and Optimize Poses Using Point Cloud View Set

Create a view set.

```
vSet = pcviewset;
```

Add four nodes and specify absolute poses.

```
absPoses = repelem(rigidtform3d,4,1);

absPoses(1).Translation = [0 0 0];
absPoses(2).Translation = [1 0 0];
absPoses(3).Translation = [2 0 0];
absPoses(4).Translation = [0.1 0 0];

vSet = addView(vSet,1,absPoses(1));
vSet = addView(vSet,2,absPoses(2));
vSet = addView(vSet,3,absPoses(3));
vSet = addView(vSet,4,absPoses(4));
```

Define 4 edges, 3 odometry and 1 loop closure.

```
relPoses = repelem(rigidtform3d,4,1);

relPoses(1).Translation = [1 0 0];
relPoses(2).Translation = [1 0 0];
relPoses(3).Translation = [-1.9 0 0];
relPoses(4).Translation = [0.2 0 0];
```

```
vSet = addConnection(vSet,1,2,relPoses(1)); % odometry
vSet = addConnection(vSet,2,3,relPoses(2)); % odometry
vSet = addConnection(vSet,3,4,relPoses(3)); % odometry
vSet = addConnection(vSet,4,1,relPoses(4)); % loop closure
```

Optimize view set.

```
vSetOptim = optimizePoses(vSet);
```

DIsplay original and optimized locations.

```
disp('Original absolute translations:')
```

```
Original absolute translations:
```

```
disp(vertcat(vSet.Views.AbsolutePose.Translation))
```

```
         0         0         0
    1.0000         0         0
    2.0000         0         0
    0.1000         0         0
```

```
disp('Optimized absolute translations:')
```

```
Optimized absolute translations:
```

```
disp(vertcat(vSetOptim.Views.AbsolutePose.Translation))
```

```
         0         0         0
    0.9250         0         0
    1.8500         0         0
   -0.1250         0         0
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MaxIterations',300` sets the maximum number of iterations to `300`.

MaxIterations — Maximum number of iterations

`300` (default) | positive integer

Maximum number of iterations before the function terminates optimization, specified as the comma-separated pair consisting of `'MaxIterations'` and a positive integer. Increase this value for greater accuracy in the results. Decrease this value for faster results.

`Tolerance` — Tolerance of optimization cost function
`1e-8` (default) | positive scalar

Tolerance of the optimization cost function between two consecutive iterations, specified as the comma-separated pair consisting of '`Tolerance`' and a positive scalar. If the cost function changes by less than the '`Tolerance`' value between two consecutive iterations, the function terminates optimization.

`Verbose` — Display progress information
`false` or `0` (default) | `true` or `1`

Display progress information, specified as the comma-separated pair consisting of `Verbose` and a numeric or logical `0` (`false`) or `1` (`true`). To display the progress information, set '`Verbose`' to `true`.

Output Arguments

`vSetOptim` — Optimized absolute poses
`pcviewset` object

Point cloud view set that contains optimized absolute poses, specified as a `pcviewset` object.

Tips

- To update a view set with optimized poses, use the `updateView` object function.
- The `optimizePoses` object function holds the first view fixed.

Algorithms

The `optimizePoses` function uses the Levenberg-Marquardt optimization algorithm with sparse Cholesky decomposition from the general (hyper) graph optimization (G2o) library, [1 on page 2-503].

Version History

Introduced in R2020a

R2024a: Added code generation support for non-host target platforms

Supports code generation for host and non-host target platforms.

References

[1] Kümmerle, Rainer, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. “G2o: A General Framework for Graph Optimization.” In *2011 IEEE International Conference on Robotics and Automation, 3607–13, 2011.* https://doi.org/10.1109/ICRA.2011.5979949.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Supports code generation for host and non-host target platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`createPoseGraph`

Objects

`imageviewset` | `pcviewset`

createPoseGraph

Create pose graph

Syntax

```
G = createPoseGraph(vSet)
```

Description

`G = createPoseGraph(vSet)` returns a pose graph derived from the views and connections in the view set, `vSet`.

Examples

Create Pose Graph from Point Cloud View Set

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Define three relative poses.

```
relPoses = repelem(rigidtform3d,3,1);
relPoses(1).Translation = [3 0 0];
relPoses(2).Translation = [5 0 0];
relPoses(3).Translation = [2 0 0];
```

Accumulate absolute poses.

```
absPoses = repelem(rigidtform3d,4,1);
absPoses(2).A = absPoses(1).A * relPoses(1).A;
absPoses(3).A = absPoses(2).A * relPoses(2).A;
absPoses(4).A = absPoses(3).A * relPoses(3).A;
```

Add four views to the point cloud view set.

```
vSet = addView(vSet,1,absPoses(1));
vSet = addView(vSet,2,absPoses(2));
vSet = addView(vSet,3,absPoses(3));
vSet = addView(vSet,4,absPoses(4));
```

Add three connections to the point cloud view set.

```
vSet = addConnection(vSet,1,2,relPoses(1));
vSet = addConnection(vSet,2,3,relPoses(2));
vSet = addConnection(vSet,3,4,relPoses(3));
```

Add a loop closure connection to the point cloud view set.

```
relPoses(4).Translation = [9 0 0];
vSet = addConnection(vSet,4,1,relPoses(4));
```

Create a pose graph.

```
G = createPoseGraph(vSet);
```

Input Arguments

vSet — View set

`pcviewset` object

View set, specified as a `pcviewset` object.

Output Arguments

G — Pose graph

`digraph` object

Pose graph, returned as a `digraph` object. The nodes in the object correspond to views, and the edges in the object correspond to connections.

The `EndNodes` of the `Edges` in the `G` digraph correspond to indices into the `Views` table, (not `ViewIds`).

Tips

- The `EndNodes` of the `Edges` in the `G` digraph correspond to indices into the `Views` table, (not `ViewIds`).
- Use “Directed and Undirected Graphs” to inspect, modify, or visualize the pose graph.
- Use the `optimizePoseGraph` function to optimize the pose graph. Use of this function requires the Navigation Toolbox.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`digraph` | `optimizePoses` | `optimizePoseGraph`

Objects

`pcviewset`

findView

Find views associated with view identifiers

Syntax

```
views = findView(vSet,viewIds)
```

Description

`views = findView(vSet,viewIds)` returns a table containing view attributes that correspond to the view identifiers in `viewIds`.

Examples

Find View Corresponding to View ID

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add two views to the view set.

```
viewId1 = 10;
viewId2 = 5;
ptCloud1 = pcread("teapot.ply");
ptCloud2 = pctransform(ptCloud1,rigidtform3d([0 0 0],[2 0 0]));
vSet = addView(vSet,viewId1,PointCloud=ptCloud1);
vSet = addView(vSet,viewId2,PointCloud=ptCloud2);
```

Find the view associated with the view identifier `5,` and then display the view table.

```
view = findView(vSet,5);
disp(view)
```

```
    ViewId      AbsolutePose         PointCloud
    ______    _________________    _______________

      5       1×1 rigidtform3d    1×1 pointCloud
```

Get the point cloud that corresponds to view identifier `5`.

```
ptCloud = view.PointCloud;
```

Get the absolute pose associated with view identifier `5`.

```
absPose = view.AbsolutePose;
```

Input Arguments

`vSet` — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewIds` — View identifier

M-element vector

View identifiers, specified as an *M*-element vector of integers. View identifiers are unique to a specific view.

Output Arguments

`views` — Table containing view attributes

three-column table

Table containing view attributes, specified as a three-column table. The table contains the columns `ViewId`, `AbsolutePose`, and `pointCloud` attribute descriptions.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcviewset`

findConnection

Find connections associated with view identifiers

Syntax

```
conn = findConnection(vSet,viewIds1,viewIds2)
```

Description

`conn = findConnection(vSet,viewIds1,viewIds2)` returns a table containing connection attributes for connections between `viewIds1` and `viewIds2`. `viewIds1` and `viewIds2` are *M*-element vectors of view identifiers.

Examples

Find Connection and Relative Pose Between Two Views

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Add two views to the view set.

```
viewId1 = 10;
viewId2 = 5;
vSet = addView(vSet,viewId1);
vSet = addView(vSet,viewId2);
```

Add a connection between `viewId1` and `viewId2`.

```
vSet = addConnection(vSet,viewId1,viewId2);
```

Find and display the connection between `viewId1` and `viewId2`.

```
conn = findConnection(vSet,viewId1,viewId2);
disp(conn)
```

```
    ViewId1    ViewId2      RelativePose      InformationMatrix
    _______    _______    _________________    _________________

      10           5       1×1 rigidtform3d      {6×6 double}
```

Get the relative pose between `viewId1` and `viewId2`.

```
relPose = conn.RelativePose;
```

Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

`viewIds1` — View identifiers
M-element vector

View identifiers, specified as an *M*-element vector of integers. View identifiers are unique to a specific view.

`viewIds2` — View identifiers
M-element vector

View identifiers, specified as an *M*-element vector of integers. View identifiers are unique to a specific view.

Output Arguments

`conn` — Table containing connection attributes
four-column table

Table containing view attributes, specified as a four-column table. The table contains the columns `ViewId1`, `ViewId2`, `RelativePose`, and `InformationMatrix` attribute descriptions.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pcviewset`

plot

Plot point cloud view set views and connections

Syntax

```
plot(vSet)
h = plot(vSet)
plot(vSet,Name=Value)
```

Description

`plot(vSet)` plots the views and connections in the point cloud view set, `vSet`.

`h = plot(vSet)` returns a graph plot, `h`

`plot(vSet,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `plot(vSet,ShowViewIds="on")` turns on the ID display.

Examples

Plot Point Cloud View with View Identifiers

Create an empty point cloud view set.

```
vSet = pcviewset;
```

Define 3 relative poses.

```
relPoses = repelem(rigidtform3d,3,1);
relPoses(1).Translation = [3 0 0];
relPoses(2).Translation = [5 0 0];
relPoses(3).Translation = [2 0 0];
```

Accumulate the absolute poses.

```
absPoses = repelem(rigidtform3d,4,1);
absPoses(2).A = absPoses(1).A * relPoses(1).A;
absPoses(3).A = absPoses(2).A * relPoses(2).A;
absPoses(4).A = absPoses(3).A * relPoses(3).A;
```

Add 4 views to the point cloud view set.

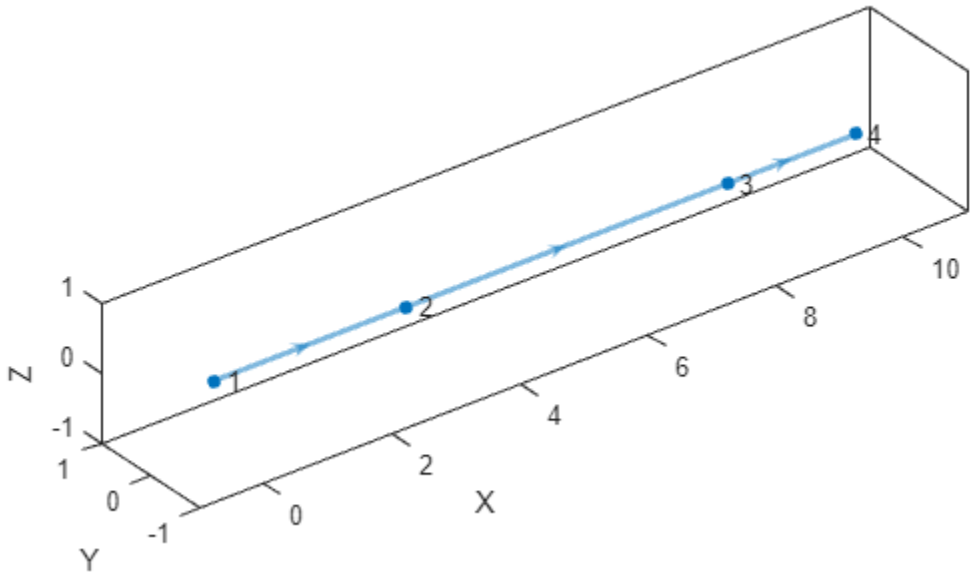
```
vSet = addView(vSet,1,absPoses(1));
vSet = addView(vSet,2,absPoses(2));
vSet = addView(vSet,3,absPoses(3));
vSet = addView(vSet,4,absPoses(4));
```

Add 3 connections to the view set.

```
vSet = addConnection(vSet,1,2,relPoses(1));
vSet = addConnection(vSet,2,3,relPoses(2));
vSet = addConnection(vSet,3,4,relPoses(3));
```

Display the point cloud view set with view IDs.

```
plot(vSet,ShowViewIds="on")
```



Input Arguments

vSet — Point cloud view set

`pcviewset` object

Point cloud view set, specified as a `pcviewset` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ShowViewId="on"` turns on the ID display.

Parent — Axes on which to plot view set

`Axes` object (default)

Axes on which to plot view set, specified as an `Axes` object. To create an `Axes` object, use the `axes` function.

`ShowViewIds` — Display of view identifiers
`"off"` (default) | `"on"`

Display of view identifiers, specified as `"on"` or `"off"`.

Output Arguments

`h` — Graph plot
`GraphPlot` object

Graph plot, returned as a `GraphPlot` object. You can use the `GraphPlot` object functions and properties to inspect and adjust the plotted graph.

Version History

Introduced in R2020a

See Also

Functions
`axes` | `GraphPlot`

Objects
`pcviewset`

imageviewset

Manage data for structure-from-motion, visual odometry, and visual SLAM

Description

The `imageviewset` object manages view attributes and pairwise connections between views of data used in structure-from-motion, visual odometry, and simultaneous localization and mapping (SLAM) data. View attributes can be feature descriptors, feature points, or absolute camera poses. Pairwise connections between views can be point matches, relative camera poses, or an information matrix. You can also use this object to find point tracks used by `triangulateMultiview` and `bundleAdjustment` functions.

Creation

Syntax

```
vSet = imageviewset()
```

Description

`vSet = imageviewset()` returns an `imageviewset` object. You can add views and connections using the `addView` and `addConnection` object functions.

Properties

`Views` — View attributes

empty three-column table (default) | three-column table

This property is read-only.

View attributes, specified as a three-column table. The table contains columns as described in this table.

Column	Description
`ViewId`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` or `se3` object.
`Features`	Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures`.

Column	Description
`Points`	Image points, specified as one of these options: • *M*-by-2 matrix of coordinates in the format [*x*,*y*] • *M*-element feature point array. For more details, see “Point Feature Types”.

`Connections` — Pairwise connections between views
empty four-column table (default) | four-column table

This property is read-only.

Pairwise connections between views, specified as a four-column table. The table contains columns as described in this table. Each row corresponds to one connection.

Column	Description
`ViewId1`	View identifier for the first view, specified as a unique integer.
`ViewId2`	View identifier for the second view, specified as a unique integer.
`RelativePose`	Relative pose of the second view with respect to the first view, specified as a `rigidtform3d` or `simtform3d` object.
`InformationMatrix`	Information matrix, specified as a 6-by-6 matrix (when you specify `RelativePose` as a `rigidtform3d` object) or 7-by-7 matrix (when you specify `RelativePose` as a `simtform3d` object). The information matrix represents the uncertainty of the measurement error and is the inverse of the covariance matrix.
`Matches`	Indices of matched feature points between two views, specified as *M*-by-2 matrix.

`NumViews` — Number of views
`0` (default) | nonnegative integer

This property is read-only.

Number of views, specified as a nonnegative integer.

`NumConnections` — Number of connections
`0` (default) | nonnegative integer

This property is read-only.

Number of connections, specified as a nonnegative integer.

`ViewsSE3` — View attributes
empty three-column table (default) | three-column table

View attributes, specified as a three-column table. This property applies only when adding views with `se3` objects. The table contains these columns:

Column	Description
`ViewID`	View identifier, specified as an integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, specified as an `se3` object.
`Features`	Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures` object.
`Points`	Image points, specified as an *M*-by-2 matrix of coordinates of the form [*x*,*y*] or an *M*-element feature point array.

Object Functions

addView	Add views to view set
updateView	Update view in view set
deleteView	Delete view from view set
hasView	Check if view is in view set
addConnection	Add connection between views in view set
updateConnection	Update connection between views in a view set
deleteConnection	Delete a connection between views in view set
hasConnection	Check if connection between two views is in view set
findView	Find views associated with view identifiers
findConnection	Find connections associated with view identifiers
connectedViews	Returns directly and indirectly connected views
poses	Absolute poses associated with views in view set
posesSe3	Absolute Se3 poses associated with views in view set
createPoseGraph	Create pose graph
findTracks	Find matched points across multiple views
optimizePoses	Optimize absolute poses using relative pose constraints
plot	Plot image view set views and connections

Examples

Find Point Tracks Across Sequence of Images

Load images in the workspace.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
images = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = im2gray(readimage(images,1));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create an image view set and add one view to the set.

```
vSet = imageviewset;
vSet = addView(vSet,1,'Features',featuresPrev,'Points',pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:numel(images.Files)
  I = im2gray(readimage(images, i));
  points = detectSURFFeatures(I);
  [features, points] = extractFeatures(I,points);
  vSet = addView(vSet,i,'Features',features,'Points',points);
  pairsIdx = matchFeatures(featuresPrev,features);
  vSet = addConnection(vSet,i-1,i,'Matches',pairsIdx);
  featuresPrev = features;
end
```

Find point tracks across views in the image view set.

```
tracks = findTracks(vSet);
```

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `imageviewset` object and its object functions are updated to use `rigidtform3d` and `simtform3d` objects, which support the premultiply convention. The `imageviewset` object now:

- Returns the `Views` property with the `AbsolutePose` column as a `rigidtform3d` object
- Returns the `Connections` property with the `RelativePose` column as a `rigidtform3d` or `simtform3d` object

Before, the properties returned `AbsolutePose` and `RelativePose` as `rigid3d` and `affine3d` objects, which use the postmultiply convention.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2022b: Visual SLAM: Improve accuracy of pose estimation

The `imageviewset` adds the `connectedViews` object function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The `plot` method of `imageviewset` object is not supported for code generation.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`digraph` | `table` | `matchFeatures` | `optimizePoses` | `triangulateMultiview` | `bundleAdjustment`

Objects
`rigidtform3d` | `worldpointset` | `pcviewset`

Topics
“Implement Visual SLAM in MATLAB”
“Monocular Visual-Inertial SLAM”
“Monocular Visual Simultaneous Localization and Mapping”

addView

Add views to view set

Syntax

```
vSet = addView(vSet,viewId)
vSet = addView(vSet,viewId,absPose)
vSet = addView( ___ ,Name=Value)

vSet = addView(vSet,viewTable)
```

Description

`vSet = addView(vSet,viewId)` adds the view specified by `viewId` to the view set, `vSet`.

`vSet = addView(vSet,viewId,absPose)` specifies the absolute pose of the view.

`vSet = addView( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any of the input argument combinations in previous syntaxes.

`vSet = addView(vSet,viewTable)` adds one or more views in the table specified by `viewTable`.

Examples

Add View to Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Detect interest points in the image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add the interest points as a view to the image view set.

```
vSet = addView(vSet,1,'Points',points);
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

viewId — View identifier

positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

absPose — Absolute pose of the view
`rigidtform3d` object | `se3` object

Absolute pose of the view, specified as a `rigidtform3d` or an `se3` object.

`viewTable` — One or more views
two-column table | three-column table

One or more views, specified as a two-column or three-column table. The table must contain the columns `ViewId` and `AbsolutePose`. The `Points` column is optional.

Column	Description
`ViewID`	View identifier, specified as an integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` or an `se3` object.
`Points`	Image points, specified as an *M*-by-2 matrix of coordinates of the form [*x*,*y*] or an *M*-element feature point array.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `"Points",detectSURFFeatures(im2gray(I))`

`Features` — Feature vectors
M-by-*N* matrix | `binaryFeatures` object

Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures` object.

`Points` — Image points
M-by-2 matrix | *M*-element feature point array

Image points, specified as an *M*-by-2 matrix of coordinates in the format [*x*,*y*] or an *M*-element feature point array. For more details, see “Point Feature Types”.

Output Arguments

`vSet` — View set with added views
`imageviewset` object

View set with added views, returned as an `imageviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify poses (the `absPose` argument and the `AbsolutePose` column of the `viewTable` argument) as `rigidtform3d` objects, which use the premultiply convention.

Although you can still specify poses as `rigid3d` objects, these objects are not recommended because they use the postmultiply convention. The `addView` function automatically converts the poses to the premultiply convention and returns an `imageviewset` object that represents the poses as `rigidtform3d` objects.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

When you specify table as an input argument, then code generation has limitations. For information about the limitations, see “Code Generation for Tables” (MATLAB Coder) and “Table Limitations for Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview`

Objects
`imageviewset` | `rigidtform3d` | `se3`

updateView

Update view in view set

Syntax

```
vSet = updateView(vSet,viewId,absPose)
vSet = updateView( ___ ,"PointCloud",ptCloud)

vSet = updateView(vSet,viewTable)
```

Description

`vSet = updateView(vSet,viewId,absPose)` updates the view specified by `viewId` with the absolute pose `absPose`.

`vSet = updateView( ___ ,"PointCloud",ptCloud)` specifies the point cloud associated with the view.

`vSet = updateView(vSet,viewTable)` updates the views specified in the table `viewTable`.

Examples

Update View in Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Detect interest points in an image.

```
imageDir= fullfile(toolboxdir("vision"),"visiondata","structureFromMotion");
I = imread(fullfile(imageDir,"image1.jpg"));
points = detectSURFFeatures(im2gray(I));
```

Add a view to the image view set.

```
vSet = addView(vSet,1,Points=points);
```

Update the absolute pose of the view.

```
absPose = rigidtform3d(eye(3),[0 0 1]);
vSet = updateView(vSet,1,absPose);
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId` — View identifier
integer

Point cloud view identifier, specified as a unique integer.

`absPose` — Absolute pose
`rigidtform3d` object | `se3` object

Absolute pose, specified as a `rigidtform3d` or an `se3` object.

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`viewTable` — New view or set of views
two-column table | three-column table

New view or set of views, specified as a two- or three- column table. The table must contain the columns `ViewId` and `AbsolutePose`, and an optional column, `Points`.

Column	Description
`ViewID`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` or an `se3` object.
`Points`	A `pointCloud`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Points",detectSURFFeatures(im2gray(I))`

`Features` — Feature vectors
M-by-*N* matrix | `binaryFeatures` object

Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures` object.

`Points` — Image points
M-by-2 matrix | *M*-element feature point array

Image points, specified as an *M*-by-2 matrix of coordinates in the format [*x*,*y*] or an *M*-element feature point array. For more details, see “Point Feature Types”.

Output Arguments

`vSet` — View set with updated view
`imageviewset` object

View set with updated view, returned as an `imageviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify poses (the `absPose` argument and the `AbsolutePose` column of the `viewTable` argument) as `rigidtform3d` objects, which use the premultiply convention.

Although you can still specify poses as `rigid3d` objects, these objects are not recommended because they use the postmultiply convention. The `updateView` function automatically converts the poses to the premultiply convention and returns an `imageviewset` object that represents the poses as `rigidtform3d` objects.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When you specify table as an input argument, then code generation has limitations. For information about the limitations, see “Code Generation for Tables” (MATLAB Coder) and “Table Limitations for Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`detectSURFFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview`

Objects

`rigidtform3d` | `imageviewset` | `se3`

deleteView

Delete view from view set

Syntax

```
vSet = deleteView(vSet,viewId)
```

Description

`vSet = deleteView(vSet,viewId)` deletes the view specified by `viewId` from the view set, `vSet`.

Examples

Delete View from Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Detect interest points in an image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add a view to the image view set.

```
vSet = addView(vSet,1,'Points',points);
```

Delete the view.

```
vSet = deleteView(vSet,1);
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId` — View identifier

positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

vSet — View set with deleted views
`imageviewset` object

View set with deleted views, returned as an `imageviewset` object.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset`

hasView

Check if view is in view set

Syntax

```
tf = hasView(vSet,viewId)
```

Description

`tf = hasView(vSet,viewId)` returns `1` (true), if the view specified by `viewId` is in the view set, `vSet`. The function returns `0` (false), if the view is not in `vSet`.

Examples

Check If View Is in Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Detect interest points in an image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir, 'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add a view with ID `1` to the image view set.

```
vSet = addView(vSet,1,'Points',points);
```

Check if a view with ID `1` is in the view set.

```
hasView(vSet,1)
```

```
ans = logical
   1
```

Check if a view with ID `2` is in the view set.

```
hasView(vSet,2)
```

```
ans = logical
   0
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId` — View identifier
positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`tf` — True or false result
`1` | `0`

True or false result indicating if view is in view set `vSet`, returned as a `1` or `0` data type `logical`.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset`

addConnection

Add connection between views in view set

Syntax

```
vSet = addConnection(vSet,viewId1,viewId2)
vSet = addConnection(vSet,viewId1,viewId2,relPose)
vSet = addConnection(vSet,viewId1,viewId2,relPose,infoMat)
vSet = addConnection(___,"Matches",featureMatches)
```

Description

`vSet = addConnection(vSet,viewId1,viewId2)` adds a connection between views `viewId1` and `viewId2` to the view set, `vSet`.

`vSet = addConnection(vSet,viewId1,viewId2,relPose)` specifies the relative pose of `viewId2` with respect to `viewId1`.

`vSet = addConnection(vSet,viewId1,viewId2,relPose,infoMat)` specifies the information matrix associated with the connection.

`vSet = addConnection(___,"Matches",featureMatches)` specifies the indices of matched points between two views in addition to any of the input argument combinations in previous syntaxes.

Examples

Add Connection to Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Read two images into the workspace.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata', ...
  'structureFromMotion');
I1 = im2gray(imread(fullfile(imageDir,'image1.jpg')));
I2 = im2gray(imread(fullfile(imageDir,'image2.jpg')));
```

Detect interest points in each image.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract feature descriptors from the interest points.

```
[features1,validPoints1] = extractFeatures(I1,points1);
[features2,validPoints2] = extractFeatures(I2,points2);
```

Add the features and points for the two images to the image view set.

```
vSet = addView(vSet,1,'Features',features1,'Points',validPoints1);
vSet = addView(vSet,2,'Features',features2,'Points',validPoints2);
```

Match the features between the two images.

```
indexPairs = matchFeatures(features1,features2);
```

Store the matched features as a connection in the image view set.

```
vSet = addConnection(vSet,1,2,'Matches',indexPairs);
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId1` — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2

positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

`relPose` — Relative pose

`rigidtform3d` object | `simtform3d` object | `se3` object

Relative pose of `viewId2` with respect to `viewId1`, specified as a `rigidtform3d`, `simtform3d`, or `se3` object.

`infoMat` — Information matrix

6-by-6 numeric matrix

Information matrix associated with the connection, specified as a 6-by-6 numeric matrix.

`featureMatches` — Indices of matched points between two views

M-b-2 matrix

Indices of matched points between two views, specified as an *M*-b-2 matrix.

Output Arguments

`vSet` — View set with added connections

`imageviewset` object

View set with added connections between views, returned as an `imageviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify `relPose` as a `rigidtform3d` or `simtform3d` object, which use the premultiply convention.

Although you can still specify `relPose` as a `rigid3d` or `affine3d` object, these objects are not recommended because they use the postmultiply convention. The `addConnection` function automatically converts the pose to the premultiply convention and returns an `imageviewset` object that represents the pose as a `rigidtform3d` or `simtform3d` object.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview`

Objects
`rigidtform3d` | `simtform3d` | `imageviewset` | `se3`

updateConnection

Update connection between views in a view set

Syntax

```
vSet = updateConnection(vSet,viewId1,viewId2,relPose)
vSet = updateConnection(vSet,viewId1,viewId2,relPose,infoMat)
vSet = updateConnection(___,"Matches",featureMatches)
```

Description

`vSet = updateConnection(vSet,viewId1,viewId2,relPose)` updates the connection between views `viewId1` and `viewId2` with the relative pose specified by `relPose`.

`vSet = updateConnection(vSet,viewId1,viewId2,relPose,infoMat)` specifies the information matrix associated with the connection.

`vSet = updateConnection(___,"Matches",featureMatches)` specifies the indices of matched points between two views in addition to any of the input argument combinations in previous syntaxes.

Examples

Update Connection in Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Read two images into the workspace.

```
imageDir= fullfile(toolboxdir("vision"),"visiondata","structureFromMotion");
I1 = im2gray(imread(fullfile(imageDir,"image1.jpg")));
I2 = im2gray(imread(fullfile(imageDir,"image2.jpg")));
```

Detect interest points in each image.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract feature descriptors from the interest points.

```
[features1,validPoints1] = extractFeatures(I1,points1);
[features2,validPoints2] = extractFeatures(I2,points2);
```

Add features and the points for the two images to the image view set.

```
vSet = addView(vSet,1,Features=features1,Points=validPoints1);
vSet = addView(vSet,2,Features=features2,Points=validPoints2);
```

Match the features between the two images. Then, store the matched features as a connection in the image view set.

```
indexPairs = matchFeatures(features1,features2);
vSet = addConnection(vSet,1,2,Matches=indexPairs);
```

Create a relative pose between the views. Then,update the connection in the image view set.

```
theta = 30; % degrees
rot = [ cosd(theta) -sind(theta) 0; ...
        sind(theta) cosd(theta) 0; ...
        0           0  1];

trans = [2 3 4];
tform = rigidtform3d(rot,trans);
vSet = updateConnection(vSet,1,2,tform);
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId1` — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2

positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

`relPose` — Relative pose

`rigidtform3d` object | `simtform3d` object | `se3` object

Relative pose of `viewId2` with respect to `viewId1`, specified as a `rigidtform3d`, `simtform3d`, or `se3` object.

`infoMat` — Information matrix

6-by-6 numeric matrix

Information matrix associated with the connection, specified as a 6-by-6 numeric matrix.

`featureMatches` — Indices of matched points between two views

M-b-2 matrix

Indices of matched points between two views, specified as an *M*-b-2 matrix.

Output Arguments

`vSet` — View set with updated connections

`imageviewset` object

View set with updated connections, returned as an `imageviewset` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `relPose` argument as a `rigidtform3d` or `simtform3d` object, which use the premultiply convention.

Although you can still specify `relPose` as a `rigid3d` or `affine3d` object, these objects are not recommended because they use the postmultiply convention. The `updateConnection` function automatically converts the relative pose to the premultiply convention and returns an `imageviewset` object that represents the pose as a `rigidtform3d` object.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`detectSURFFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview`

Objects

`rigidtform3d` | `simtform3d` | `imageviewset` | `se3`

deleteConnection

Delete a connection between views in view set

Syntax

```
vSet = deleteConnection(vSet,viewId1,viewId2)
```

Description

`vSet = deleteConnection(vSet,viewId1,viewId2)` deletes the connection between views `viewId1` and `viewId2` in the view set, `vSet`.

Examples

Delete Connection Between Views in Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Read two images into the workspace.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I1 = im2gray(imread(fullfile(imageDir,'image1.jpg')));
I2 = im2gray(imread(fullfile(imageDir,'image2.jpg')));
```

Detect interest points in each image.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract feature descriptors from the interest points.

```
[features1,validPoints1] = extractFeatures(I1,points1);
[features2,validPoints2] = extractFeatures(I2,points2);
```

Add the features and points for the two images to the image view set.

```
vSet = addView(vSet,1,'Features',features1,'Points',validPoints1);
vSet = addView(vSet,2,'Features',features2,'Points',validPoints2);
```

Match the features between the two images. Then, store the matches features as a connection in the image view set.

```
indexPairs = matchFeatures(features1,features2);
vSet = addConnection(vSet,1,2,'Matches',indexPairs);
```

Delete the connection between the two views.

```
vSet = deleteConnection(vSet,1,2);
```

Input Arguments

`vSet` — Image view set
`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId1` — View identifier 1
positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2
positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`vSet` — View set with deleted connections
`imageviewset` object

View set with deleted connections, returned as an `imageviewset` object.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset`

hasConnection

Check if connection between two views is in view set

Syntax

```
tf = hasConnection(vSet,viewId1,viewId2)
```

Description

`tf = hasConnection(vSet,viewId1,viewId2)` returns `1` (true), if the connection between views `viewId1` and `viewId2` is in the view set, `vSet`. The function returns `0` (false), if the connection does not exist in `vSet`.

Examples

Check for Connection Between Views in View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Add two views to the image view set.

```
vSet = addView(vSet,1);
vSet = addView(vSet,2);
```

Add a connection to the image view set.

```
vSet = addConnection(vSet,1,2);
```

Check if the connection between views is in the view set.

```
tf = hasConnection(vSet,1,2)
```

```
tf = logical
   1
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

viewId1 — View identifier 1

positive integer

View identifier 1, specified as an integer. View identifiers are unique to a specific view.

`viewId2` — View identifier 2
positive integer

View identifier 2, specified as an integer. View identifiers are unique to a specific view.

Output Arguments

`tf` — True or false result
`1` | `0`

True or false result indicating if view is in view set `vSet`, returned as a `1` or `0` data type `logical`.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset`

connectedViews

Returns directly and indirectly connected views

Syntax

```
viewTable = connectedViews(vSet,viewId)
[viewTable,dist] = connectedViews( ___ )
viewTable = connectedViews(vSet,viewId,Name=Value)
```

Description

`viewTable = connectedViews(vSet,viewId)` returns a table of views from `vSet` that are connected and indirectly connected to the view specified by `viewId`.

`[viewTable,dist] = connectedViews( ___ )` also returns the distances between the `viewId` view and each view in `viewTable`.

`viewTable = connectedViews(vSet,viewId,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes For example, `MaxDistance=1` additionally sets the distance between two connected views.

Examples

Get Connected Views in View Set

Create an empty `imageviewset` object.

```
vSet = imageviewset;
```

Add views to the image view set.

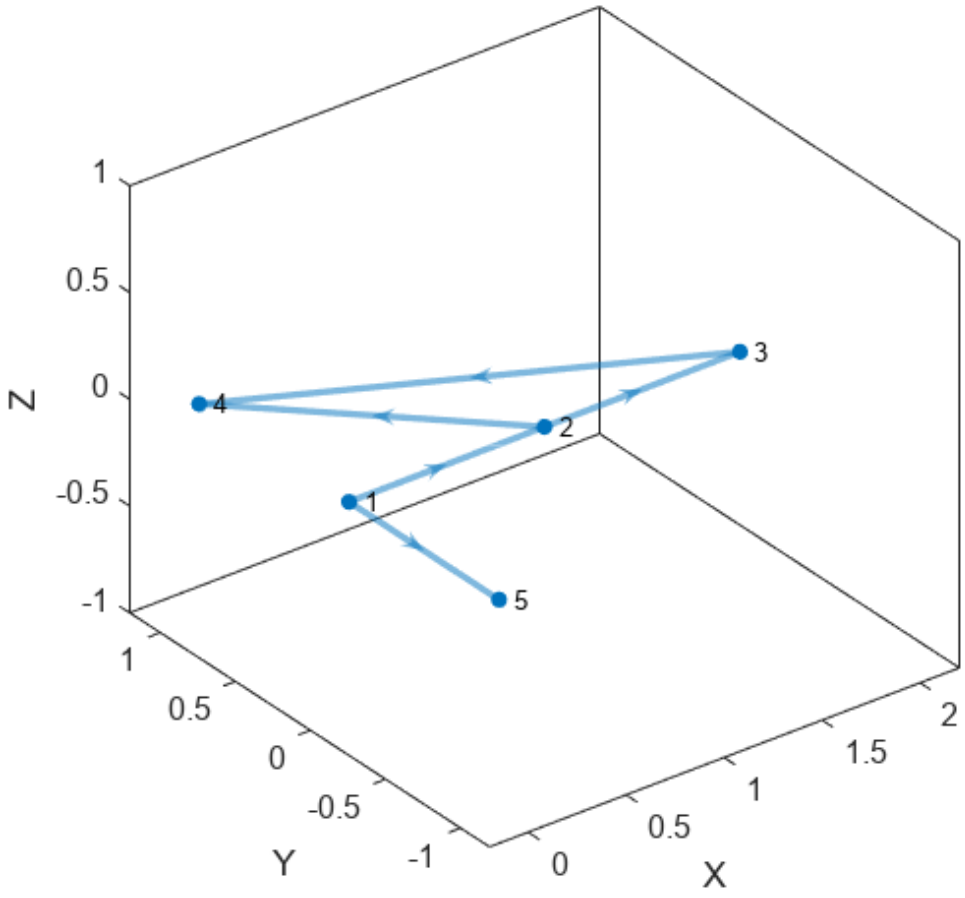
```
pose1 = rigidtform3d;
vSet = addView(vSet,1,pose1,Features=rand(10,3),Points=rand(10,2));
pose2 = rigidtform3d([0 0 0],[1 0 0]);
vSet = addView(vSet,2,pose2,Features=rand(10,3),Points=rand(10,2));
pose3 = rigidtform3d([0 0 0],[2 0 0]);
vSet = addView(vSet,3,pose3,Features=rand(10,3),Points=rand(10,2));
pose4 = rigidtform3d([0 0 0],[0 1 0]);
vSet = addView(vSet,4,pose4,Features=rand(10,3),Points=rand(10,2));
pose5 = rigidtform3d([0 0 0],[0 -1 0]);
vSet = addView(vSet,5,pose5,Features=rand(10,3),Points=rand(10 ,2));
```

Connect views in the image view set.

```
vSet = addConnection(vSet,1,2,Matches=[1 2; 2 3; 3 4; 4 5]);
vSet = addConnection(vSet,2,3,Matches=[1 2; 2 3; 3 4]);
vSet = addConnection(vSet,2,4,Matches=[4 1]);
vSet = addConnection(vSet,3,4,Matches=[4 1]);
vSet = addConnection(vSet,1,5,Matches=[1 2; 4 1]);
```

Display the view set with view IDs.

```
plot(vSet,ShowViewIds="on");
```



Get the strongly connected views of view 2 with the at least 3 matched feature points.

```
viewTableStrong = connectedViews(vSet,2,MinNumMatches=3)
```

```
viewTableStrong=2×4 table
    1    1×1 rigidtform3d    10×3 double    10×2 double
    3    1×1 rigidtform3d    10×3 double    10×2 double
```

Get views with maximum distance of 2, within view 3.

```
[viewTableNeaby,dist] = connectedViews(vSet,3,MaxDistance=2)
```

```
viewTableNeaby=3×4 table
    2    1×1 rigidtform3d    10×3 double    10×2 double
    4    1×1 rigidtform3d    10×3 double    10×2 double
    1    1×1 rigidtform3d    10×3 double    10×2 double
```

```
dist = 3×1

    1
    1
    2
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewId` — View identifier

positive integer

View identifier, specified as an integer. View identifiers are unique to a specific view.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `MaxDistance=2` sets the distance between two connected views to 2.

`MaxDistance` — Maximum distance between connected views and `viewId` view

`1` (default) | positive integer

Maximum distance between connected views and the view that corresponds to `viewId`, specified as a positive integer. The distance between any two connected views must be equal to `1`. To find indirectly connected views, set `MaxDistance` to a value greater than `1`.

`MinNumMatches` — Minimum number of matched feature points

`0` (default) | nonnegative integer

Minimum number of matched feature points in a connection for the view to be counted as a connected view, specified as a nonnegative integer.

Output Arguments

`viewTable` — One or more views

three-column table

View attributes, specified as a three-column table. The table contains columns as described in this table.

Column	Description
`ViewId`	View identifier for the view, specified as a unique integer
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.
`Features`	Feature vectors, specified as an *M*-by-*N* matrix of *M* feature vectors or as a `binaryFeatures`.
`Points`	Image points, specified as one of these options: • *M*-by-2 matrix of coordinates in the format [*x*,*y*] • *M*-element feature point array. For more details, see “Point Feature Types”.

`dist` — Distance between identified view and each view in the view table
positive integers

Distance between the view identified in `viewId` and each view in the view table `viewTable`, specified as an *M*-by-1 vector of positive integers.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `connectedViews` function now returns the `AbsolutePose` value in the `viewTable` argument as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `AbsolutePose` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Added a new Name-Value argument, `MaxDistance`. Use `MaxDistance` to find indirectly-connected views by setting it to a value greater than `1`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset` | `rigidtform3d`

poses

Absolute poses associated with views in view set

Syntax

```
sensorPoses = poses(vSet)
```

Description

`sensorPoses = poses(vSet)` returns a table of absolute poses associated with the views contained in the view set, `vSet`.

Examples

Retrieve Absolute Poses from Image View Set

Load images into the workspace.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata","structureFromMotion");
images = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = im2gray(readimage(images,1));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create an image view set and add the extracted feature points to the image view set.

```
vSet = imageviewset;
vSet = addView(vSet,1,Points=pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:numel(images.Files)
 I = im2gray(readimage(images,i));
 points = detectSURFFeatures(I);
 [features,points] = extractFeatures(I,points);
 vSet = addView(vSet,i,Features=features,Points=points);
 pairsIdx = matchFeatures(featuresPrev,features);
 vSet = addConnection(vSet,i-1,i,Matches=pairsIdx);
 featuresPrev = features;
end
```

Get the absolute poses.

```
sensorPoses = poses(vSet)
```

```
sensorPoses=5×2 table
    ViewId      AbsolutePose
    ______    _________________
```

```
1       1×1 rigidtform3d
2       1×1 rigidtform3d
3       1×1 rigidtform3d
4       1×1 rigidtform3d
5       1×1 rigidtform3d
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

Output Arguments

sensorPoses — Absolute poses

two-column table

Absolute poses, returned as a two-column table. The table contains columns as described in this table.

Column	Description
`ViewID`	View identifier, returned as a positive integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, returned as a `rigidtform3d` or `se3` object.

Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `poses` function now returns the `AbsolutePose` value in the `sensorPoses` argument as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `AbsolutePose` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When generating code, the output argument `sensorPoses` is returned as a structure with fields *ViewID* and *AbsolutePose*.

Use posesSe3 instead of poses for code generation if you're using se3 poses as input to `vset`

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset` | `rigidtform3d` | `se3`

posesSe3

Absolute Se3 poses associated with views in view set

Syntax

```
sensorPoses = posesSe3(vSet)
```

Description

`sensorPoses = posesSe3(vSet)` returns a table of absolute Se3 poses associated with the views contained in the view set, `vSet`. Use this function when you need to perform code generation using Se3 poses.

Examples

Retrieve Absolute Poses from Image View Set

Load images into the workspace.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata","structureFromMotion");
images = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = im2gray(readimage(images,1));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create an image view set and add the extracted feature points to the image view set.

```
vSet = imageviewset;
vSet = addView(vSet,1,Points=pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:numel(images.Files)
 I = im2gray(readimage(images,i));
 points = detectSURFFeatures(I);
 [features,points] = extractFeatures(I,points);
 vSet = addView(vSet,i,Features=features,Points=points);
 pairsIdx = matchFeatures(featuresPrev,features);
 vSet = addConnection(vSet,i-1,i,Matches=pairsIdx);
 featuresPrev = features;
end
```

Get the absolute poses.

```
sensorPoses = poses(vSet)
```

```
sensorPoses=5×2 table
    ViewId      AbsolutePose
    ______    _________________
```

```
1       1×1 rigidtform3d
2       1×1 rigidtform3d
3       1×1 rigidtform3d
4       1×1 rigidtform3d
5       1×1 rigidtform3d
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

Output Arguments

case when

sensorPoses — Absolute poses

two-column table

Absolute poses, returned as a two-column table. The table contains columns as described in this table.

Column	Description
`ViewID`	View identifier, returned as a positive integer. View identifiers are unique to a specific view.
`AbsolutePose`	Absolute pose of the view, returned as an `se3` object.

Version History

Introduced in R2026a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When generating code, the output argument `sensorPoses` is returned as a structure with fields *ViewID* and *AbsolutePose*.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset` | `rigidtform3d` | `se3`

optimizePoses

Optimize absolute poses using relative pose constraints

Syntax

```
vSetOptim = optimizePoses(vSet)
[vSetOptim,poseScale] = optimizePoses(vSet)
vSetOptim = optimizePoses(vSet, minNumMatches)
vSetOptim = optimizePoses( ___ ,Name,Value)
```

Description

`vSetOptim = optimizePoses(vSet)` returns an image view set whose absolute poses are optimized. `vSetOptim` and `vSet` are `imageviewset` objects.

The `optimizePoses` function performs pose graph optimization on the absolute poses for the `Views` in the view set using the relative pose constraints established by the `Connections` property. You can use `optimizePoses` to correct drift in odometry after detecting loop closures.

`[vSetOptim,poseScale] = optimizePoses(vSet)` also returns the scales associated with the optimized absolute poses. This output applies only when the `RelativePose` of at least one connection is represented as a `simtform3d` object.

`vSetOptim = optimizePoses(vSet, minNumMatches)` additionally specifies the minimum number of matched feature points in a connection in order for the connection to be included in optimization.

`vSetOptim = optimizePoses( ___ ,Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Create and Optimize Poses Using Image View Set

Create a view set.

```
vSet = imageviewset;
```

Add four nodes and specify absolute poses.

```
absPoses = repelem(rigidtform3d, 4, 1);

absPoses(1).Translation = [ 0   0 0];
absPoses(2).Translation = [ 1   0 0];
absPoses(3).Translation = [ 2   0 0];
absPoses(4).Translation = [ 0.1 0 0];

vSet = addView(vSet, 1, absPoses(1));
vSet = addView(vSet, 2, absPoses(2));
```

```
vSet = addView(vSet, 3, absPoses(3));
vSet = addView(vSet, 4, absPoses(4));
```

Define 4 edges, 3 odometry and 1 loop closure.

```
relPoses = repelem(rigidtform3d, 4, 1);

relPoses(1).Translation = [ 1   0 0];
relPoses(2).Translation = [ 1   0 0];
relPoses(3).Translation = [-1.9 0 0];
relPoses(4).Translation = [ 0.2 0 0];

vSet = addConnection(vSet, 1, 2, relPoses(1)); % odometry
vSet = addConnection(vSet, 2, 3, relPoses(2)); % odometry
vSet = addConnection(vSet, 3, 4, relPoses(3)); % odometry
vSet = addConnection(vSet, 4, 1, relPoses(4)); % loop closure
```

Optimize view set.

```
vSetOptim = optimizePoses(vSet);
```

DIsplay original and optimized locations.

```
disp('Original absolute translations:')
```

```
Original absolute translations:
```

```
disp(vertcat(vSet.Views.AbsolutePose.Translation))
```

```
         0         0         0
    1.0000         0         0
    2.0000         0         0
    0.1000         0         0
```

```
disp('Optimized absolute translations:')
```

```
Optimized absolute translations:
```

```
disp(vertcat(vSetOptim.Views.AbsolutePose.Translation))
```

```
         0         0         0
    0.9250         0         0
    1.8500         0         0
   -0.1250         0         0
```

Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

minNumMatches — Minimum number of connections

positive integer

Minimum number of connections, specified as a positive integer. Set `minNumMatches` to the minimum number of matched feature points in a connection for the connection to be included in optimization.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"MaxIterations",300`

`MaxIterations` — Maximum number of iterations
`300` (default) | positive integer

Maximum number of iterations before optimization is terminated, specified as a positive integer. Increase this value for more accurate results. Decrease this value for faster results.

`Tolerance` — Tolerance of the optimization cost function
`1e-8` (default) | positive scalar

Tolerance of the optimization cost function between two iterations, specified as a positive scalar. Optimization is terminated when the cost function changes by less than the `Tolerance` value between two iterations.

`Verbose` — Display progress information
`false` (default) | `true`

Display progress information, specified as a logical `true` or `false`. Set `Verbose` to `true` to display progress information.

Output Arguments

`vSetOptim` — Optimized absolute poses
`imageviewset` object

Image view set that contains optimized absolute poses, returned as an `imageviewset` object.

`poseScale` — Scales of optimized absolute poses
vector of positive values

Scales of optimized absolute poses, returned as a vector of positive values. This output applies only when the `RelativePose` of the pairwise “Connections” on page 2-0 property, of at least one connection is represented as a `simtform3d` object.

Tips

- To update a view set with optimized poses, use the `updateView` object function.
- The optimizePoses object function holds the first view fixed.

Algorithms

The `optimizePoses` function uses the Levenberg Marquardt optimization algorithm with sparse Cholesky factorization from the general (hyper) graph optimization (g2o) library, (1 on page 2-553).

Version History

Introduced in R2020a

R2024a: Added code generation support for non-host target platforms

Supports code generation for host and non-host target platforms.

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `optimizePoses` function now supports image view sets that represent poses as `rigidtform3d` or `simtform3d` objects, which use the premultiply convention.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Kuemmerle, R., G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "g2o: A General Framework for Graph Optimization IEEE International Conference on Robotics and Automation". *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA, 2011.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Supports code generation for host and non-host target platforms.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`createPoseGraph`

Objects
`imageviewset`

createPoseGraph

Create pose graph

Syntax

```
G = createPoseGraph(vSet)
```

Description

`G = createPoseGraph(vSet)` returns a pose graph derived from the views and connections in the view set, `vSet`.

You can use the pose graph for inspection, modification, visualization, and pose graph optimization.

Examples

Create Pose Graph from Image View Set

Create an empty image view set.

```
vSet = imageviewset;
```

Define three relative poses.

```
relPoses = repelem(rigidtform3d,3,1);
relPoses(1).Translation = [3 0 0];
relPoses(2).Translation = [5 0 0];
relPoses(3).Translation = [2 0 0];
```

Accumulate absolute poses.

```
absPoses = repelem(rigidtform3d,4,1);
absPoses(2).A = absPoses(1).A*relPoses(1).A;
absPoses(3).A = absPoses(2).A*relPoses(2).A;
absPoses(4).A = absPoses(3).A*relPoses(3).A;
```

Add four views to the image view set.

```
vSet = addView(vSet,1,absPoses(1));
vSet = addView(vSet,2,absPoses(2));
vSet = addView(vSet,3,absPoses(3));
vSet = addView(vSet,4,absPoses(4));
```

Add three connections to the image view set.

```
vSet = addConnection(vSet,1,2,relPoses(1));
vSet = addConnection(vSet,2,3,relPoses(2));
vSet = addConnection(vSet,3,4,relPoses(3));
```

Add a loop closure connection to the image view set.

```
relPoses(4).Translation = [9 0 0];
vSet = addConnection(vSet,4,1,relPoses(4));
```

Create a pose graph.

```
G = createPoseGraph(vSet)
```

```
G =
  digraph with properties:

    Edges: [4×4 table]
    Nodes: [4×2 table]
```

Input Arguments

vSet — View set

`imageviewset` object

View set, specified as an `imageviewset` object.

Output Arguments

G — Pose graph

`digraph` object

Pose graph, returned as a `digraph` object. The nodes in the object correspond to views, and the edges in the object correspond to connections.

The `EndNodes` of the `Edges` in the `G` digraph correspond to indices into the `Views` table, (not `ViewIds`).

Tips

- The `EndNodes` of the `Edges` in the `G` digraph correspond to indices into the `Views` table, (not `ViewIds`).
- Use “Directed and Undirected Graphs” to inspect, modify, or visualize the pose graph.
- Use the `optimizePoseGraph` function to optimize the pose graph. Use of this function requires the Navigation Toolbox.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`digraph` | `optimizePoses` | `optimizePoseGraph`

Objects
`imageviewset`

findView

Find views associated with view identifiers

Syntax

```
views = findView(vSet,viewIds)
```

Description

`views = findView(vSet,viewIds)` finds the views associated with the specified view `viewIds` and returns the view attributes.

Examples

Find View Corresponding to View ID

Create an empty image view set.

```
vSet = imageviewset;
```

Add two images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata","structureFromMotion");
I1 = im2gray(imread(fullfile(imageDir,"image1.jpg")));
I2 = im2gray(imread(fullfile(imageDir,"image2.jpg")));
```

Detect interest points in the two images.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract features descriptors from the detected points.

```
[features1,validPoints1] = extractFeatures(I1,points1);
[features2,validPoints2] = extractFeatures(I2,points2);
```

Add the points to the imageviewset object.

```
vSet = addView(vSet,1,Features=features1,Points=validPoints1);
vSet = addView(vSet,2,Features=features2,Points=validPoints2);
```

Find the view that corresponds to `viewID` equal to 1.

```
view = findView(vSet,1)
```

```
view=1×4 table
    ViewId      AbsolutePose         Features              Points
    ______    _________________    _______________    ____________________

      1       1×1 rigidtform3d    {574×64 single}    {574×1 SURFPoints}
```

Check the feature points that correspond to `viewID` equal to 1.

```
view.Points{:}
```

```
ans =
  574×1 SURFPoints array with properties:

              Scale: [574×1 single]
    SignOfLaplacian: [574×1 int8]
        Orientation: [574×1 single]
           Location: [574×2 single]
             Metric: [574×1 single]
              Count: 574
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewIds` — View identifier

M-element vector

View identifiers, specified as an *M*-element vector of integers. Each view identifier is unique to a specific view.

Output Arguments

`views` — View attributes

four-column table

View attributes, returned as a four-column table. The table contains the columns `ViewId`, `AbsolutePose`, `Features`, and `Points` attribute descriptions.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`imageviewset`

findConnection

Find connections associated with view identifiers

Syntax

```
conn = findConnection(vSet,viewIds1,viewIds2)
```

Description

`conn = findConnection(vSet,viewIds1,viewIds2)` finds the connections between the specified view identifiers `viewIds1` and `viewIds2` and returns the connection attributes `conn`.

Examples

Find Connection Between Two Image View Sets

Create an empty imageviewset object.

```
vSet = imageviewset;
```

Add three views to the view set.

```
viewId1 = 10;
viewId2 = 5;
viewId3 = 2;
vSet = addView(vSet,viewId1);
vSet = addView(vSet,viewId2);
vSet = addView(vSet,viewId3);
```

Add a connection between viewId1 and viewId2.

```
relPose1 = rigidtform3d(eye(3),[0 0 1]);
vSet = addConnection(vSet,viewId1,viewId2,relPose1);
```

Add a connection between viewId2 and viewId3.

```
relPose2 = rigidtform3d(eye(3),[0 0 2]);
vSet = addConnection(vSet,viewId2,viewId3,relPose2);
```

Find the connection between viewId1 and viewId2.

```
conn = findConnection(vSet,viewId1,viewId2)
```

```
conn=1×5 table
    ViewId1    ViewId2       RelativePose       InformationMatrix      Matches
    _______    _______    __________________    _________________    ___________

      10          5       {1×1 rigidtform3d}      {6×6 double}       {0×2 uint32}
```

Check the relative pose between viewId1 and viewId2.

```
conn.RelativePose{:}

ans =
  rigidtform3d with properties:

    Dimensionality: 3
       Translation: [0 0 1]
                 R: [3×3 double]

                 A: [1     0     0     0
                     0     1     0     0
                     0     0     1     1
                     0     0     0     1]
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewIds1` — Initial view identifiers

M-element vector

Initial view identifiers, specified as an *M*-element vector of integers. Each view identifier is unique to a specific view.

`viewIds2` — Connected view identifiers

M-element vector

Connected view identifiers, specified as an *M*-element vector of integers. Each view identifier is unique to a specific view.

Output Arguments

`conn` — Connection attributes

four-column table

Pairwise connections between views, returned as a four-column table. The table contains columns as described in this table. Each row corresponds to one connection.

Column	Description
`ViewId1`	View identifier for the first view, specified as a unique integer.
`ViewId2`	View identifier for the second view, specified as a unique integer.
`RelativePose`	Relative pose of the second view with respect to the first view, specified as a `rigidtform3d` or `simtform3d` object.

Column	Description
`InformationMatrix`	Information matrix, specified as a 6-by-6 matrix (when you specify `RelativePose` as a `rigidtform3d` object) or 7-by-7 matrix (when you specify `RelativePose` as a `simtform3d` object). The information matrix represents the uncertainty of the measurement error and is the inverse of the covariance matrix.
`Matches`	Indices of matched feature points between two views, specified as *M*-by-2 matrix.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`imageviewset`

findTracks

Find matched points across multiple views

Syntax

```
tracks = findTracks(vSet)
tracks = findTracks(vSet,viewIds)
tracks = findTracks( ___ ,'MinTrackLength',trackLength)
```

Description

`tracks = findTracks(vSet)` finds and returns point tracks across multiple views in the view set, `vSet`. Each track contains 2-D projections of the same 3-D world point.

`tracks = findTracks(vSet,viewIds)` finds point tracks across a subset of views specified by `viewIds`.

`tracks = findTracks( ___ ,'MinTrackLength',trackLength)` specifies the minimum length of the tracks.

Examples

Find Point Tracks Across Sequence of Images

Load images in the workspace.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
images = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = im2gray(readimage(images,1));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create an image view set and add one view to the set.

```
vSet = imageviewset;
vSet = addView(vSet,1,'Features',featuresPrev,'Points',pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:numel(images.Files)
  I = im2gray(readimage(images, i));
  points = detectSURFFeatures(I);
  [features, points] = extractFeatures(I,points);
  vSet = addView(vSet,i,'Features',features,'Points',points);
  pairsIdx = matchFeatures(featuresPrev,features);
  vSet = addConnection(vSet,i-1,i,'Matches',pairsIdx);
  featuresPrev = features;
end
```

Find point tracks across views in the image view set.

```
tracks = findTracks(vSet);
```

Input Arguments

`vSet` — Image view set

`imageviewset` object

Image view set, specified as an `imageviewset` object.

`viewIds` — View identifiers

vector of integers

View identifiers, specified as a vector of positive integers. View identifiers are unique to a specific view.

`trackLength` — Minimum length of the tracks

positive integer equal to or greater than 2

Minimum length of the tracks, specified as a positive integer equal to or greater than 2.

Output Arguments

`tracks` — Point tracks

`pointTrack` object

Point tracks across multiple views, returned as a `pointTrack` object.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`detectSURFFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview`

Objects

`imageviewset` | `pointTrack`

plot

Plot image view set views and connections

Syntax

```
plot(vSet)
h = plot(vSet)
plot(vSet,Name=Value)
```

Description

`plot(vSet)` plots the views and connections in the view set, `vSet`.

`h = plot(vSet)` returns a graph plot, `h`

`plot(vSet,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `plot(vSet,ShowViewIds="off")` turns off the ID display.

Examples

Plot View Set View Identifiers

Create an empty image view set.

```
vSet = imageviewset;
```

Define three relative poses.

```
relPoses = repelem(rigidtform3d,3,1);
relPoses(1).Translation = [3 0 0];
relPoses(2).Translation = [5 0 0];
relPoses(3).Translation = [2 0 0];
```

Calculate and add absolute poses.

```
absPoses = repelem(rigidtform3d,4,1);
absPoses(2).A = absPoses(1).A*relPoses(1).A;
absPoses(3).A = absPoses(2).A*relPoses(2).A;
absPoses(4).A = absPoses(3).A*relPoses(3).A;
```

Add four views to the image view set.

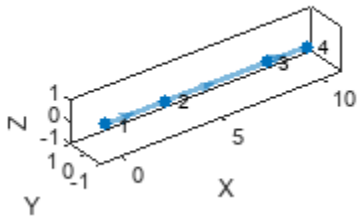
```
vSet = addView(vSet,1,absPoses(1));
vSet = addView(vSet,2,absPoses(2));
vSet = addView(vSet,3,absPoses(3));
vSet = addView(vSet,4,absPoses(4));
```

Add connections between the added views.

```
vSet = addConnection(vSet,1,2,relPoses(1));
vSet = addConnection(vSet,2,3,relPoses(2));
vSet = addConnection(vSet,3,4,relPoses(3));
```

Plot the view set with view identifers.

```
plot(vSet,ShowViewIds="on")
```



Input Arguments

vSet — Image view set

`imageviewset` object

Image view set, specified as a `imageviewset` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ShowViewId="off"` turns off the ID display.

Parent — Axes on which to plot view set

`Axes` object (default)

Axes on which to plot view set, specified as an `Axes` object. To create an `Axes` object, use the `axes` function.

ShowViewIds — Display of view identifiers

`"off"` (default) | `"on"`

Display of view identifiers, specified as `"on"` or `"off"`.

Output Arguments

h — Graph plot

`GraphPlot` object

Graph plot, returned as a `GraphPlot` object. You can use the `GraphPlot` object functions and properties to inspect and adjust the plotted graph.

Version History

Introduced in R2020a

See Also

Functions

`axes` | `GraphPlot`

Objects

`pcviewset`

ssdObjectDetector

Detect objects using SSD deep learning detector

Description

The `ssdObjectDetector` detects objects from an image, using a single shot detector (SSD) object detector. To detect objects in an image, pass the trained detector to the `detect` function. You can also use the trained detector for multiclass object detection. For information about SSD object detection, see “Getting Started with SSD Multibox Detection”.

Creation

Syntax

```
detector = ssdObjectDetector(net,classes,aboxes)
detector = ssdObjectDetector(baseNet,classes,aboxes,DetectionNetworkSource=
layer)
detector = ssdObjectDetector( ___ ,Name=Value)
```

Description

`detector = ssdObjectDetector(net,classes,aboxes)` creates an object detector using the SSD deep learning network `net`.

If `net` is a pretrained SSD deep learning network, the function creates a pretrained SSD object detector. The `classes` and `aboxes` are the classes and anchor boxes used when the network was trained.

If `net` is an untrained SSD deep learning network, the function creates a SSD object detector to use for training and inference. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the SSD network. Use the `trainSSDObjectDetector` function to train the network before performing object detection.

`detector = ssdObjectDetector(baseNet,classes,aboxes,DetectionNetworkSource=layer)` creates a SSD object detector by adding detection heads to specified feature extraction layers within a base network, `baseNet`.

If `baseNet` is a pretrained deep learning network, then the function creates a SSD object detector and configures it to perform transfer learning with the specified object classes and anchor boxes.

If `baseNet` is an untrained deep learning network, then the function creates a SSD object detector and configures it for object detection. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the SSD network.

You must train the detector on a training dataset before performing object detection. Use the `trainSSDObjectDetector` function to train the detector.

Before R2021a, use the equivalent syntax `detector = ssdObjectDetector(baseNet,classes,aboxes,"DetectionNetworkSource",layer)`.

`detector = ssdObjectDetector( ___ ,Name=Value)` specifies properties using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `InputSize=[224 224 3]` sets the *height*, *width*, and number of *channels* for the image size used for training.

Input Arguments

`net` — SSD deep learning network

`dlnetwork` object

SSD deep learning network, specified as a `dlnetwork` object. The input network can be either an untrained or a pretrained deep learning network. The input network must not have loss layers.

`baseNet` — Base network

`dlnetwork` object

Base network for creating the SSD deep learning network, specified as a `dlnetwork` object. The network can be either an untrained or a pretrained deep learning network. When you specify this argument, you must also specify the feature extraction layers by using the `layer` input argument.

`classes` — Names of object classes

string vector | cell array of character vectors | categorical vector

Names of object classes for training the detector, specified as a string vector, cell array of character vectors, or categorical vector. This argument sets the `ClassNames on page 2-0` property of the `ssdObjectDetector` object.

Data Types: `char` | `string` | `categorical`

`aboxes` — Anchor boxes

N-by-1 cell array

Anchor boxes for training the detector, specified as an *N*-by-1 cell array. *N* is the number of output layers in the SSD deep learning network. Each cell contains an *M*-by-2 matrix, where *M* is the number of anchor boxes in that layer. Each row in the *M*-by-2 matrix denotes the size of an anchor box in the form [*height width*].

The first element in the cell array specifies the anchor boxes to associate with the first output layer, the second element in the cell array specifies the anchor boxes to associate with the second output layer, and so on. For accurate detection results, specify large anchor boxes for the first output layer and small anchor boxes for the last output layer. That is, the anchor box sizes must decrease for each output layer in the order in which the layers appear in the SSD deep learning network.

This argument sets the `AnchorBoxes on page 2-0` property of the `ssdObjectDetector` object.

Data Types: `cell`

`layer` — Names of feature extraction layers

cell array of character vectors | string array

Names of the feature extraction layers in the base network, specified as a cell array of character vectors or a string array.

Example: `layer = {'conv10','fire9-concat'}`

Example: `layer = ["conv10","fire9-concat"]`

Data Types: `char` | `string` | `cell`

Properties

`InputSize` — Image size used for training

vector

This property is read-only.

Image size used for training, stored as a vector of form [*height width*] or [*height width channels*]. To set this property, specify it at object creation. The size of the training images must be a multiple of 32.

`ModelName` — Name of classification model

character vector | string scalar

This property is read-only.

Name of the classification model, specified as a character vector or string scalar. By default, the name is set to the heading of the second column of the `trainingData` table specified in the `trainSSDObjectDetector` function. You can modify this name after creating your `ssdObjectDetector` object.

`Network` — Trained SSD multibox object detection network

`dlnetwork` object

This property is read-only.

Trained SSD multibox object detection network, specified as a `dlnetwork` object. The `dlnetwork` object stores the layers that define the convolutional neural network used within the SSD detector.

`AnchorBoxes` — Size of anchor boxes

P-by-1 cell array

This property is read-only.

Size of anchor boxes, specified as a *P*-by-1 cell array for *P* number of feature extraction layers used for object detection in the SSD network. Each element of the array contains an *M*-by-2 matrix of anchor box sizes, in the format [*height width*]. Each cell can contain a different number of anchor boxes.

You can set this property by using the input argument `aboxes`.

`ClassNames` — Names of object classes

cell array

This property is read-only.

Names of the object classes that the SSD detector is trained to find, specified as a cell array. You can set this property by using the input argument `classes`.

Object Functions

detect Detect objects using SSD multibox object detector

Examples

Create Custom SSD Object Detector

This example shows how to create a SSD object detection network by using a pretrained ResNet-50 convolutional neural network as the base network.

Load a pretrained deep learning network to use as the base network. This example uses a ResNet-50 pretrained network as the base network. For information about other available pretrained networks, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

```
baseNet = imagePretrainedNetwork("resnet50");
```

Use `analyzeNetwork` to display the architecture of the base network.

```
analyzeNetwork(baseNet)
```

Specify the class names and anchor boxes to use for training the SSD deep learning network created using ResNet-50 as the base network.

```
classNames = ["person" "car" "dog"];
anchorBoxes = {[30 60;60 30;50 50;100 100], ...
               [40 70;70 40;60 60;120 120], ...
               [50 80;80 60;70 70;140 140]};
```

Specify the names of the feature extraction layers in the base network to use as the detection heads.

```
featureExtractionLayers = ["activation_11_relu" "activation_22_relu" "activation_40_relu"];
```

Create an SSD object detector by using the specified base network and the detection heads.

```
detector = ssdObjectDetector(baseNet,classNames,anchorBoxes,DetectionNetworkSource=featureExtrac
```

Display and inspect the properties of the SSD object detector.

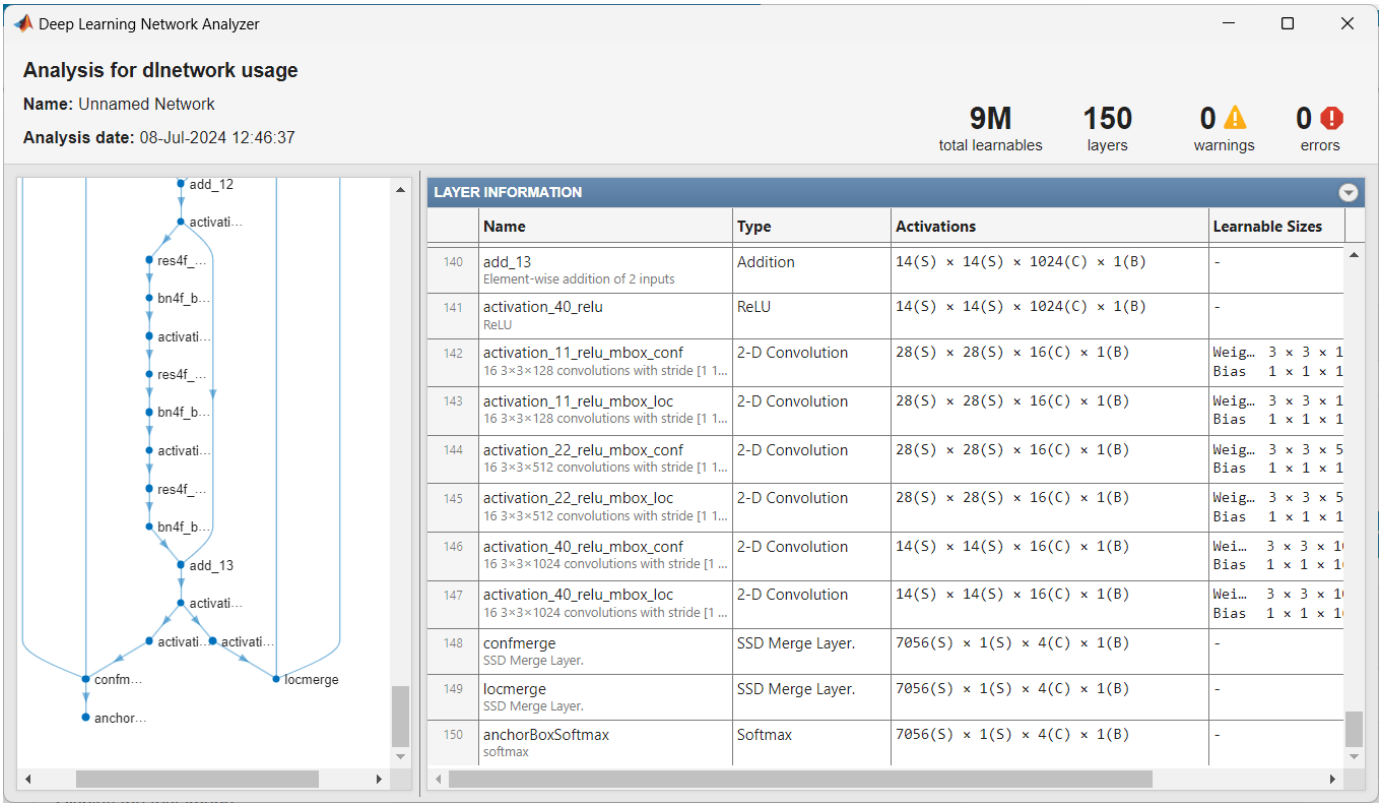
```
disp(detector)
```

```
  ssdObjectDetector with properties:

       Network: [1×1 dlnetwork]
    AnchorBoxes: {3×1 cell}
     ClassNames: [3×1 string]
      InputSize: [224 224 3]
      ModelName: ''
```

Use `analyzeNetwork` to display the SSD network architecture and get information about the network layers.

```
analyzeNetwork(detector.Network)
```



Detect Vehicles Using SSD Object Detector

Load a pretrained single shot detector (SSD) object to detect vehicles in an image. The detector is trained with images of cars on a highway scene.

```
vehicleDetector = load("ssdVehicleDetector.mat","detector");
detector = vehicleDetector.detector;
```

Read a test image into the workspace.

```
I = imread("highway.png");
```

Display the test image.

```
imshow(I)
```



Run the pretrained SSD object detector by using the `detect` function. The output contains the bounding boxes, scores, and the labels for vehicles detected in the image. The labels are derived from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 2×4

   139    78    96    81
    99    67   165   146


scores = 2×1 single column vector

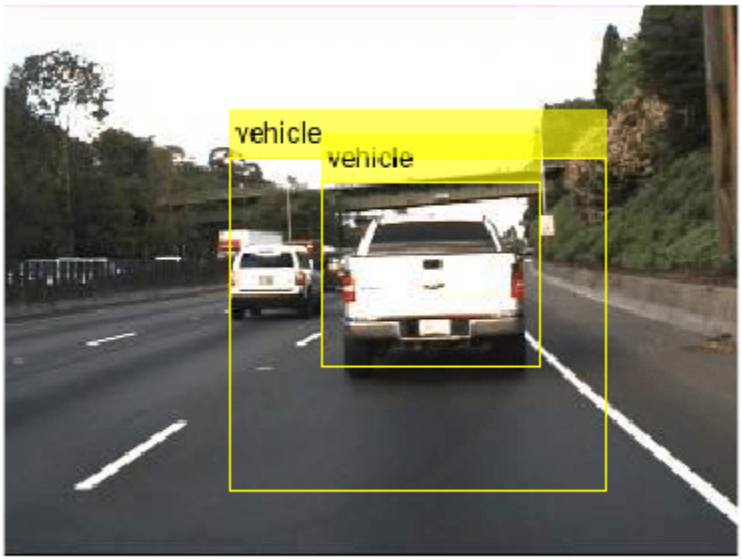
    0.8349
    0.6302


labels = 2×1 categorical
     vehicle
     vehicle
```

Annotate the image with the detection results.

```
if ~isempty(bboxes)
    detectedI = insertObjectAnnotation(I,"rectangle",bboxes,cellstr(labels));
else
   detectedI = insertText(I,[10 10],"No Detections");
end

imshow(detectedI)
```



Version History

Introduced in R2020a

R2024b: Returns `Network` property as `dlnetwork` object
Behavior changed in R2024b

The `Network` property now returns a `dlnetwork` object instead of a `DAGNetwork` object.

R2024b: `LayerGraph` objects are not recommended
Not recommended starting in R2024b

Starting in R2024b, `LayerGraph` objects are not recommended. If you want to use a pretrained SSD network or a base feature extraction network, specify `net` or `baseNet` as a `dlnetwork` object instead.

There are no plans to remove support for `LayerGraph` objects. However, `dlnetwork` objects have these advantages and are recommended instead:

- `dlnetwork` objects are a unified data type that supports network building, prediction, built-in training, visualization, compression, verification, and custom training loops.
- `dlnetwork` objects support a wider range of network architectures that you can create or import from external platforms.
- Training and prediction with `dlnetwork` objects is typically faster than with `LayerGraph` objects.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Only the `detect` method of the `ssdObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported. All name-value arguments must be compile-time constants.
- The channel and batch size of the input image must be fixed size.
- The `labels` output is returned as a categorical array.
- In the generated code, the input is rescaled to the size of the input layer of the network. But the bounding box that the `detect` method returns is in reference to the original input size.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation,

- Only the `detect` method of the `ssdObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported.
- The channel and batch size of the input image must be fixed size.
- The `labels` output is returned as a categorical array.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainSSDObjectDetector` | `yolov4ObjectDetector` | `acfObjectDetector` | `selectStrongestBboxMulticlass`

Topics
“Object Detection Using SSD Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using Single Shot Multibox Detector”
“Getting Started with SSD Multibox Detection”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Datastores for Deep Learning” (Deep Learning Toolbox)

detect

Detect objects using SSD multibox object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[___,labels] = detect(detector,I)
[___] = detect(___,roi)

detectionResults = detect(detector,ds)

[___] = detect(___,Name,Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using an single shot multibox detector (SSD). The locations of objects detected are returned as a set of bounding boxes.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[bboxes,scores] = detect(detector,I)` also returns the detection scores for each bounding box.

`[___,labels] = detect(detector,I)` also returns a categorical array of labels assigned to the bounding boxes, using either of the preceding syntaxes. The labels used for object classes are defined during training using the `trainSSDObjectDetector` function.

`[___] = detect(___,roi)` detects objects within the rectangular search region specified by `roi`.

`detectionResults = detect(detector,ds)` detects objects within the series of images returned by the `read` function of the input datastore.

`[___] = detect(___,Name,Value)` specifies options using one or more `Name,Value` pair arguments. For example, `detect(detector,I,"Threshold",0.75)` sets the detection score threshold to `0.75`. Any detections with a lower score are removed.

Examples

Detect Vehicles Using SSD Object Detector

Load a pretrained single shot detector (SSD) object to detect vehicles in an image. The detector is trained with images of cars on a highway scene.

```
vehicleDetector = load("ssdVehicleDetector.mat","detector");
detector = vehicleDetector.detector;
```

Read a test image into the workspace.

```
I = imread("highway.png");
```

Display the test image.

```
imshow(I)
```



Run the pretrained SSD object detector by using the `detect` function. The output contains the bounding boxes, scores, and the labels for vehicles detected in the image. The labels are derived from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 2×4

   139    78    96    81
    99    67   165   146


scores = 2×1 single column vector

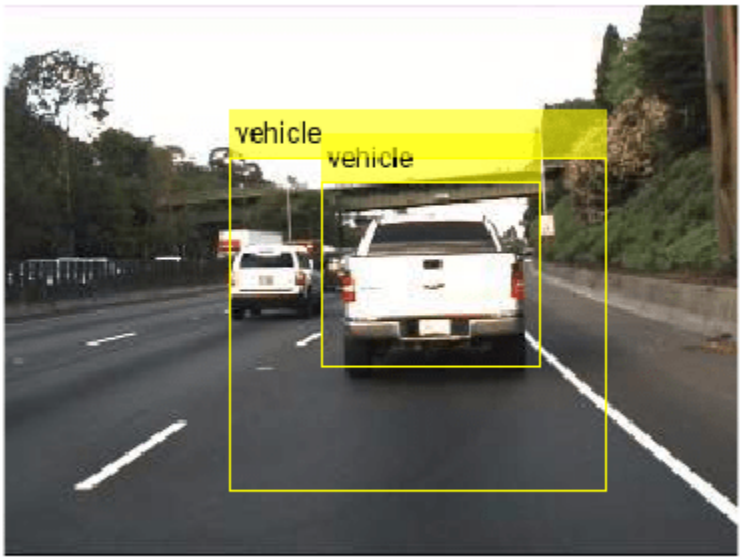
    0.8349
    0.6302


labels = 2×1 categorical
     vehicle
     vehicle
```

Annotate the image with the detection results.

```
if ~isempty(bboxes)
    detectedI = insertObjectAnnotation(I,"rectangle",bboxes,cellstr(labels));
else
   detectedI = insertText(I,[10 10],"No Detections");
end

imshow(detectedI)
```



Input Arguments

`detector` — SSD object detector

`ssdObjectDetector` object

SSD object detector, specified as an `ssdObjectDetector` object. To create this object, call the `trainSSDObjectDetector` function with training data as input.

`I` — Input image

H-by-*W*-by-*C*-by-*B* numeric array of images

Input image, specified as an *H*-by-*W*-by-*C*-by-*B* numeric array of images. Images must be real, nonsparse, grayscale or RGB image.

- *H* — Height in pixels.
- *W* — Width in pixels
- *C* — The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to `1`. For RGB color images, it must be equal to `3`.
- *B* — Number of images in the array.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was

trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`ds` — Datastore
`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be a grayscale, RGB, or multichannel image. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as a four-element vector of the form [*x y width height*]. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"SelectStrongest",true`

`Threshold` — Detection threshold
`0.5` (default) | scalar

Detection threshold, specified as a scalar in the range [0, 1]. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Return the strongest bounding box per object. To select these boxes, `detect` calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  For example:

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.5);
  ```

- `false` — Return all detected bounding boxes. You can then create your own custom operation to eliminate overlapping bounding boxes.

`MaxSize` — Maximum region size
`size(I)` (default) | [*height width*] vector

Maximum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `MaxSize` is set to the height and width of the input image, `I`.

`MinSize` — Minimum region size
`[1 1]` (default) | [*height width*] vector

Minimum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

To reduce computation time, set this value to the known minimum region size for the objects being detected in the image. By default, `MinSize` is set to `[1 1]`.

`MiniBatchSize` — Minimum batch size
`128` (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of images. Images are grouped into minibatches and processed as a batch, which can improve computation efficiency at the cost of increased memory demand. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of the following:

- `"auto"` — Automatically apply a number of optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available when using a GPU only. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Disable all acceleration.

The default option is `"auto"`. If `"auto"` is specified, MATLAB applies a number of compatible optimizations. If you use the `"auto"` option, MATLAB does not ever generate a MEX function.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is only available for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is only available when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

`bboxes` — Location of objects detected

M-by-4 matrix | *B*-by-1 cell array

Location of objects detected within the input image or images, returned as an *M*-by-4 matrix or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-4 matrices when the input contains an array of images.

Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

`scores` — Detection scores

M-by-1 vector | *B*-by-1 cell array

Detection confidence scores, returned as an *M*-by-1 vector or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-1 vectors when the input contains an array of images. A higher score indicates higher confidence in the detection.

`labels` — Labels for bounding boxes

M-by-1 categorical array | *B*-by-1 cell array

Labels for bounding boxes, returned as an *M*-by-1 categorical array or a *B*-by-1 cell array. *M* is the number of labels in an image, and *B* is the number of *M*-by-1 categorical arrays when the input contains an array of images. You define the class names used to label the objects when you train the input `detector`.

`detectionResults` — Detection results

3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column contains *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported. All argument values must be compile-time constants.
- The `MiniBatchSize` name-value argument is not supported for code generation when the image input `I` is an *H*-by-*W*-by-*C*-by-*B* numeric array of images. All images will processed in one batch.
- The channel and batch size of the input image must be fixed size.
- The `labels` output is returned as a categorical array.
- In the generated code, the input is rescaled to the size of the input layer of the network. But the bounding box that the `detect` method returns is in reference to the original input size.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation,

- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported.
- The channel and batch size of the input image must be fixed size.
- The `labels` output is returned as a categorical array.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

Apps

Image Labeler | Video Labeler

Objects

`boxLabelDatastore` | `yolov4ObjectDetector`

Functions

`trainSSDObjectDetector` | `selectStrongestBboxMulticlass` | `evaluateObjectDetection`

Topics
“Object Detection Using SSD Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using Single Shot Multibox Detector”
“Getting Started with SSD Multibox Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

boxLabelDatastore

Datastore for bounding box label data

Description

The `boxLabelDatastore` object creates a datastore for bounding box label data. Use this object to read labeled bounding box data for object detection.

To read bounding box label data from a `boxLabelDatastore` object, use the `read` function. This object function returns a cell array with either two or three columns. You can create a datastore that combines the `boxLabelDatastore` object with an `ImageDatastore` object using the `combine` object function. Use the combined datastore to train object detectors using the training functions such as `trainYOLOv4ObjectDetector` and `trainSSDObjectDetector`. To modify the `ReadSize` property, you can use dot notation.

Creation

Syntax

```
blds = boxLabelDatastore(tbl1,...,tbln)
blds = boxLabelDatastore(tbl1,...,tbln,bSet)
```

Description

`blds = boxLabelDatastore(tbl1,...,tbln)` creates a `boxLabelDatastore` object from one or more tables containing labeled bounding box data.

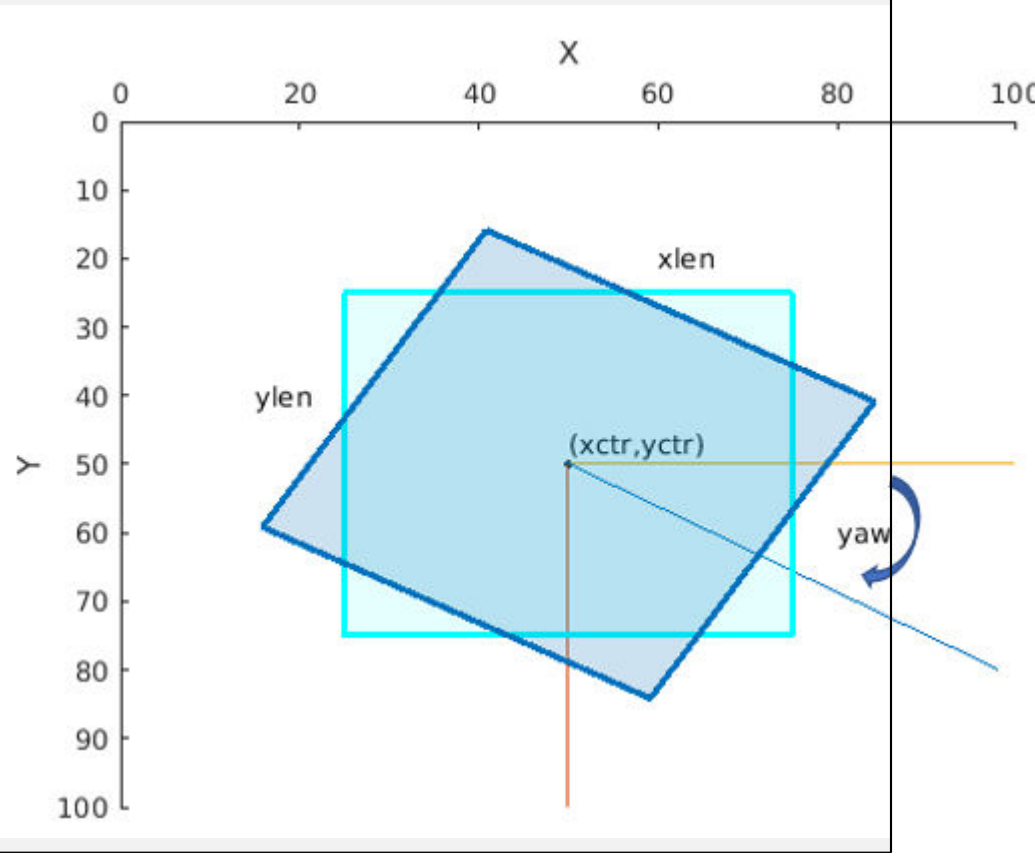
`blds = boxLabelDatastore(tbl1,...,tbln,bSet)` creates a `boxLabelDatastore` object for block-based labeled bounding box data. The blocks have resolution level, block size, and block positions specified by the block locations in `bSet`.

Input Arguments

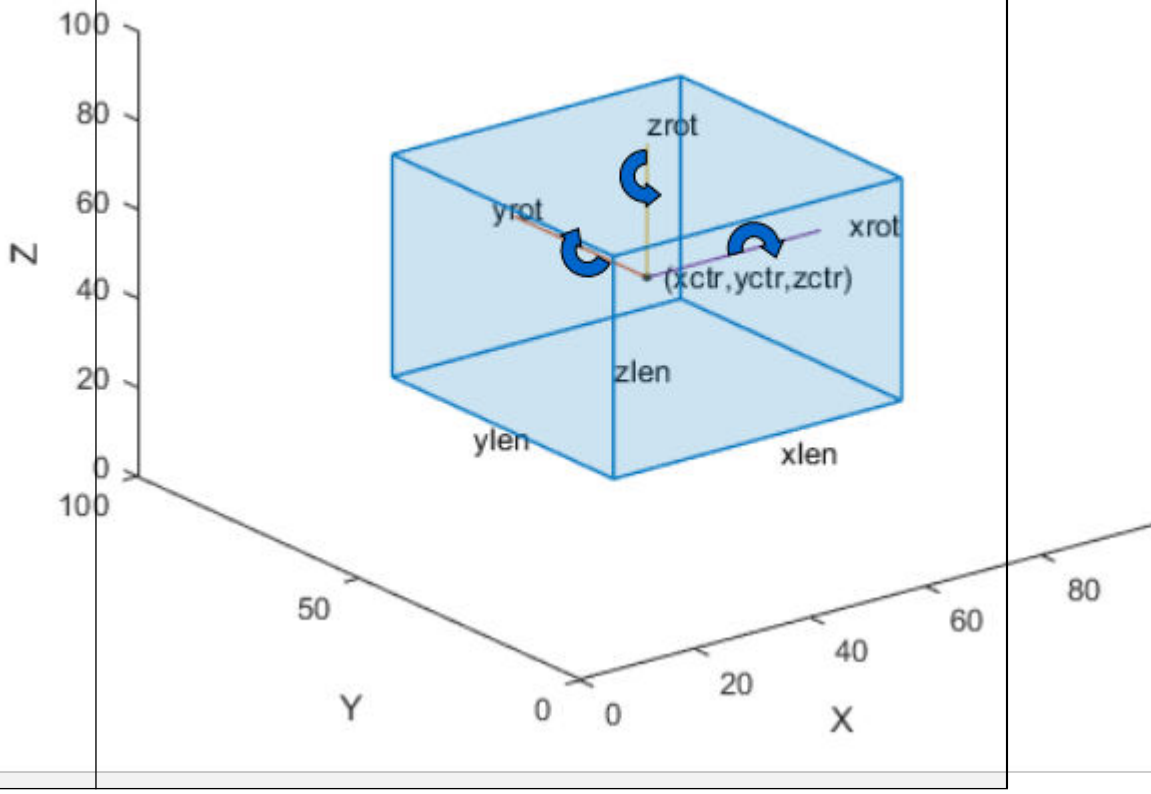
`tbl1,...,tbln` — Labeled bounding box data (as separate arguments)

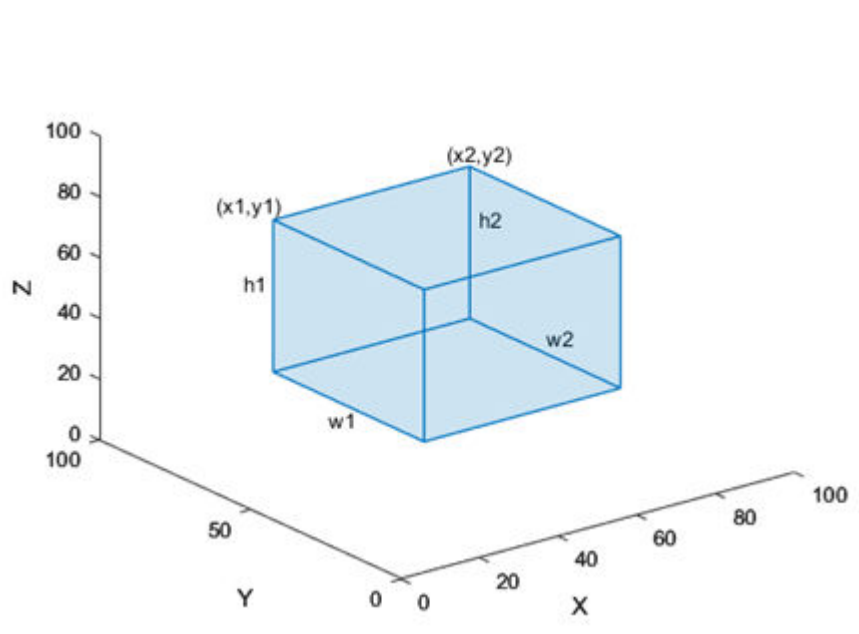
table with one or more columns

Labeled bounding box data, specified as a table with one or more columns. Each table corresponds to a set of labels. The bounding boxes can be axis-aligned rectangles, rotated rectangles, or cuboids. The table below describes the format of the bounding boxes.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

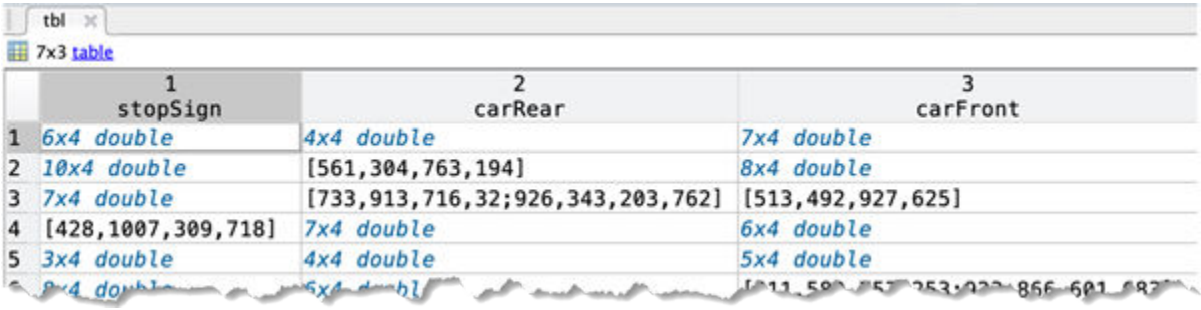
Bounding Box	Description
Cuboid	Defined in spatial coordinates as an *M*-by-9 numeric matrix with rows of the form [*xctr yctr zctr xlen ylen zlen xrot yrot zrot*], where: • *M* is the number of cuboids. • *xctr*, *yctr*, and *zctr* specify the center of the cuboid. • *xlen*, *ylen*, and *zlen* specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation. • *xrot*, *yrot*, and *zrot* specify the rotation angles of the cuboid around the *x*-axis, *y*-axis, and *z*-axis, respectively. The *xrot*, *yrot*, and *zrot* rotation angles are in degrees about the cuboid center. Each rotation is clockwise-positive with respect to the positive direction of the associated spatial axis. The function computes rotation matrices assuming ZYX order Euler angles [*xrot yrot zrot*]. The figure shows how these values determine the position of a cuboid.



Bounding Box	Description
Projected Cuboid	M-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid. 

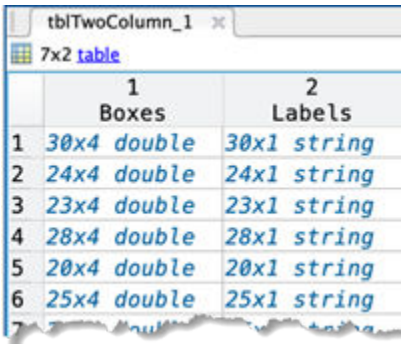
- A table with one or more columns:

  All columns contain bounding boxes. Each column must be a cell vector containing *M*-by-*N* matrices. *M* is the number of images and *N* represents a single object class, such as *stopSign*, *carRear*, or *carFront*.



- A table with two columns.

  The first column contains bounding boxes. The second column must be a cell vector that contains the label names corresponding to each bounding box. Each element in the cell vector must be an *M*-by-1 categorical or string vector, where *M* represents the number of labels.



To create a ground truth table, use the Image Labeler or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.

Data Types: `table`

bSet — Block locations

`blockLocationSet` object

Block locations, specified as a `blockLocationSet` object. You can create this object by using the `balanceBoxLabels` function or the `blockLocationsWithROI` function.

Properties

LabelData — Labeled bounding box data

N-by-2 cell array

This property is read-only.

Labeled bounding box data, specified as an *N*-by-2 cell matrix of *N* images. The first column must be a cell vector that contains bounding boxes. Each element in the cell contains a vector representing either an axis-aligned rectangle, rotated rectangle, or a cuboid. The second column must be a cell vector that contains the label names corresponding to each bounding box. An *M*-by-1 categorical vector represents each label name.

Bounding Box Descriptions

Bounding Box	Cell Vector	Format
Axis-aligned rectangle	*M*-by-4 for *M* bounding boxes	[*x*,*y*,*width*,*height*]
Rotated rectangle	*M*-by-5 for *M* bounding boxes	[*xcenter*,*ycenter*,*width*,*height*,*yaw*]
Cuboid	*M*-by-9 for *M* bounding boxes	[*xcenter*,*ycenter*,*zcenter*,*width*,*height*,*depth*,*rx*,*ry*,*rz*]
Projected cuboid	*M*-by-8 vector for *M* bounding boxes	[*x1*,*y1*,*w1*,*h1*,*x2*,*y2*,*w2*,*h2*]

`ReadSize` — Maximum number of rows of label data

1 (default) | positive integer

Maximum number of rows of label data to read in each call to the `read` function, specified as a positive integer.

Object Functions

combine	Combine data from multiple datastores
countEachLabel	Count occurrence of pixel or box labels
hasdata	Determine if data is available to read from label datastore
numpartitions	Number of partitions for label datastore
partition	Partition label datastore
preview	Read first row of data in datastore
progress	Percentage of data read from a datastore
read	Read data from label datastore
readall	Read all data in label datastore
reset	Reset label datastore to initial state
shuffle	Return shuffled version of label datastore
subset	Create subset of datastore or FileSet
transform	Transform datastore
isPartitionable	Determine whether datastore is partitionable
isShuffleable	Determine whether datastore is shuffleable

Examples

Estimate Anchor Boxes for YOLO v2 Object Detection Network

This example shows how to estimate anchor boxes using a table containing the training data. The first column contains the training images and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Create a `boxLabelDatastore` object using the labeled bounding boxes from the training data.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Specify the class names using the labels from the training data.

```
classes = trainingData.Properties.VariableNames(2:end);
```

Estimate the anchor boxes using the `boxLabelDatastore` object.

```
numAnchors = 5;
anchorBoxes = estimateAnchorBoxes(blds,numAnchors);
```

Specify the image size.

```
inputImageSize = [128 228 3];
```

Use a pretrained ResNet-50 network as a base network for the YOLO v2 network.

```
baseNet = imagePretrainedNetwork("resnet50");
```

Specify the network layer to use for feature extraction. You can use the `analyzeNetwork` function to see all the layer names in a network.

```
featureLayer = "activation_49_relu";
```

Create the YOLO v2 object detection network.

```
detector = yolov2ObjectDetector(baseNet,classes,anchorBoxes, ...
    DetectionNetworkSource=featureLayer)
```

```
detector =
  yolov2ObjectDetector with properties:

                 Network: [1×1 dlnetwork]
               InputSize: [224 224 3]
       TrainingImageSize: [224 224]
             AnchorBoxes: [5×2 double]
              ClassNames: vehicle
    ReorganizeLayerSource: ''
              LossFactors: [5 1 1 1]
               ModelName: ''
```

Visualize the network using the network analyzer.

```
analyzeNetwork(detector.Network)
```

Combine Box Label Datastore and Image Label Datastore

Load a table of vehicle class training data that contains bounding boxes with labels.

```
data = load('vehicleTrainingData.mat');
trainingData = data.vehicleTrainingData;
```

Add the fullpath to the local vehicle data folder.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` object using the file names in the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` object using the table with label data.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the `imageDatastore` and `boxLabelDatastore` objects.

```
cds = combine(imds,blds);
```

Read the data for training. Use the `read` object function to return images, bounding boxes, and labels.

```
read(cds)
```

```
ans=1×3 cell array
    {128×228×3 uint8}    {[126 78 20 16]}    {[vehicle]}
```

Combine Multiple Class Ground Truth Data

Load a table of vehicle class training data that contains bounding boxes with labels.

```
load('vehicleTrainingData.mat');
```

Load a table of stop signs and cars class training data that contains bounding boxes with labels.

```
load('stopSignsAndCars.mat');
```

Create ground truth tables from the training data.

```
vehiclesTbl  = vehicleTrainingData(:,2:end);
stopSignsTbl = stopSignsAndCars(:,2:end);
```

Create a `boxLabelDatastore` object using two tables: one with vehicle label data and the other with the stop signs and cars label data.

```
blds = boxLabelDatastore(vehiclesTbl,stopSignsTbl);
```

Create an `imageDatastore` object using the file names in the training data tables.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
vehicleFiles = fullfile(dataDir,vehicleTrainingData.imageFilename);
stopSignFiles = fullfile(dataDir,stopSignsAndCars.imageFilename);
imds = imageDatastore([vehicleFiles;stopSignFiles]);
```

Combine the `imageDatastore` and `boxLabelDatastore` objects.

```
cds = combine(imds,blds);
```

Read the data for training. Use the `read` object function to return images, bounding boxes, and labels.

```
read(cds)
```

```
ans=1×3 cell array
    {128×228×3 uint8}    {[126 78 20 16]}    {[vehicle]}
```

Sample Block Sets to Use in Blocked Image Object Detection

Load box labels data that contains boxes and labels for one image. The height and width of each box is 20-by-20 pixels.

```
d = load("balanceBoxLabelsData.mat");
boxLabels = d.BoxLabels;
```

Create a blocked image of size 500-by-500 pixels.

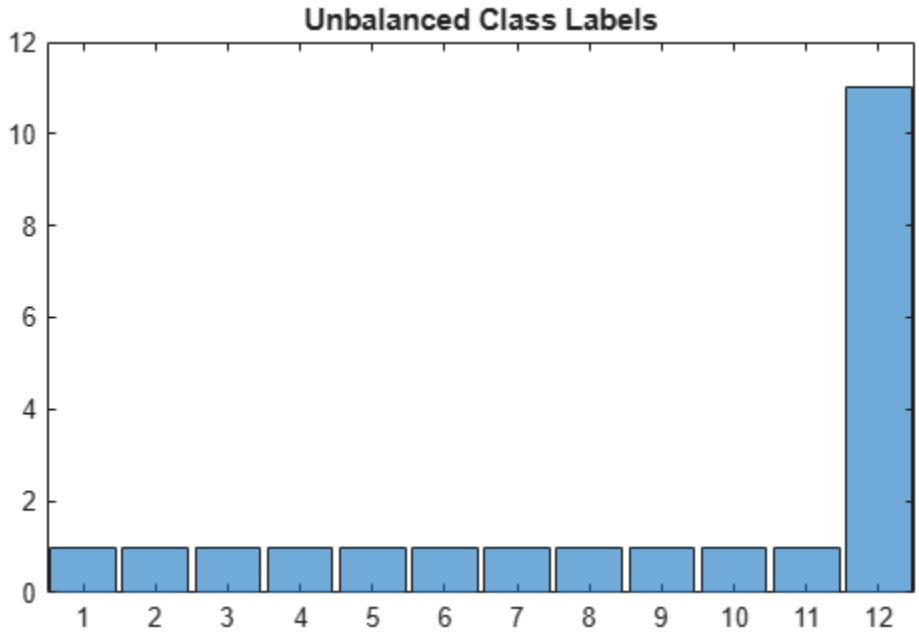
```
blockedImages = blockedImage(zeros([500 500]));
```

Choose the images size of each observation.

```
blockSize = [50 50];
```

Visualize using a histogram to identify any class imbalance in the box labels.

```
blds = boxLabelDatastore(boxLabels);
datasetCount = countEachLabel(blds);
figure
unbalancedLabels = datasetCount.Label;
unbalancedCount  = datasetCount.Count;
h1 = histogram(Categories=unbalancedLabels,BinCounts=unbalancedCount);
title("Unbalanced Class Labels")
```



Measure the distribution of box labels. If the coefficient of variation is more than 1, then there is class imbalance.

```
cvBefore = std(datasetCount.Count)/mean(datasetCount.Count)
```

```
cvBefore =
1.5746
```

Choose a heuristic value for number of observations by finding the mean of the counts of each class, multiplied by the number of classes.

```
numClasses = height(datasetCount);
numObservations = mean(datasetCount.Count) * numClasses;
```

Control the amount a box can be cut using `OverlapThreshold`. Using a lower threshold value will cut objects more at the border of a block. Increase this value to reduce the amount an object can be clipped at the border, at the expense of a less balanced box labels.

```
ThresholdValue = 0.5;
```

Balance `boxLabels` using the `balanceBoxLabels` function.

```
locationSet = balanceBoxLabels(boxLabels,blockedImages,blockSize, ...
        numObservations,OverlapThreshold=ThresholdValue);
```

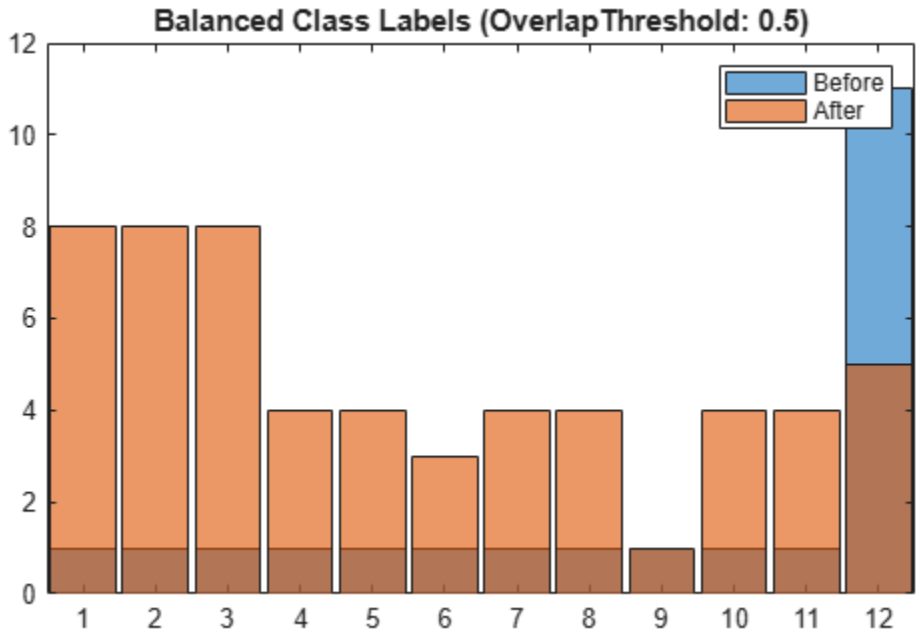
```
[==================================================] 100%
Elaps[==================================================] 100%
Elapsed time: 00:00:00
Estimated time remaining: 00:00:00
Balancing box labels complete.
```

Count the labels that are contained within the image blocks.

```
bldsBalanced = boxLabelDatastore(boxLabels,locationSet);
balancedDatasetCount = countEachLabel(bldsBalanced);
```

Overlay another histogram against the original label count to see if the box labels are balanced. If the labels appear to be not balanced by looking at the histograms, increase the value for `numObservations`.

```
hold on
balancedLabels = balancedDatasetCount.Label;
balancedCount  = balancedDatasetCount.Count;
h2 = histogram(Categories=balancedLabels,BinCounts=balancedCount);
title(h2.Parent,"Balanced Class Labels (OverlapThreshold: " + ThresholdValue + ")" )
legend(h2.Parent,["Before" "After"])
```



Measure the distribution of the new balanced box labels.

```
cvAfter = std(balancedCount)/mean(balancedCount)
```

```
cvAfter =
0.4588
```

Version History

Introduced in R2019b

R2022b: Project cuboids from 3-D world coordinates to 2-D image coordinates

Updated to support 3-D projected cuboid labels.

See Also

Apps
Image Labeler | Video Labeler

Functions
`balanceBoxLabels` | `blockLocationsWithROI` | `analyzeNetwork` | `estimateAnchorBoxes`

Objects
`imageDatastore` | `blockLocationSet` | `blockedImage`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)

“Training Data for Object Detection and Semantic Segmentation”

pixelLabelImageDatastore

(To be removed) Datastore for semantic segmentation networks

Note `pixelLabelImageDatastore` will be removed in a future release. Use the `imageDatastore` and `pixelLabelDatastore` objects and the `combine` function instead.

Description

Use `pixelLabelImageDatastore` to create a datastore for training a semantic segmentation network using deep learning.

Creation

Syntax

```
pximds = pixelLabelImageDatastore(gTruth)
pximds = pixelLabelImageDatastore(imds,pxds)
pximds = pixelLabelImageDatastore( ___ ,Name,Value)
```

Description

`pximds = pixelLabelImageDatastore(gTruth)` returns a datastore for training a semantic segmentation network based on the input `groundTruth` object or array of `groundTruth` objects. Use the output `pixelLabelImageDatastore` object with the `trainnet` function to train convolutional neural networks for semantic segmentation.

`pximds = pixelLabelImageDatastore(imds,pxds)` returns a datastore based on the input image datastore and the pixel label datastore objects. `imds` is an `ImageDatastore` object that represents the training input to the network. `pxds` is a `PixelLabelDatastore` object that represents the required network output.

`pximds = pixelLabelImageDatastore( ___ ,Name,Value)` additionally uses name-value arguments to set the `DispatchInBackground` and `OutputSizeMode` properties. For 2-D data, you can also use name-value pairs to specify the `ColorPreprocessing`, `DataAugmentation`, and `OutputSize` augmentation properties. You can specify multiple name-value arguments.

For example, `pixelLabelImageDatastore(gTruth,'PatchesPerImage',40)` creates a pixel label image datastore that randomly generates 40 patches from each ground truth object in `gTruth`.

Input Arguments

gTruth — Ground truth data

`groundTruth` object | array of `groundTruth` objects

Ground truth data, specified as a `groundTruth` object or as an array of `groundTruth` objects. Each `groundTruth` object contains information about the data source, the list of label definitions, and all marked labels for a set of ground truth labels.

`imds` — Collection of images
`ImageDatastore` object

Collection of images, specified as an `ImageDatastore` object.

`pxds` — Collection of pixel labeled images
`PixelLabelDatastore` object

Collection of pixel labeled images, specified as a `PixelLabelDatastore` object. The object contains the pixel labeled images for each image contained in the `imds` input object.

Properties

`Images` — Image file names
character vector | cell array of character vectors

This property is read-only.

Image file names used as the source for ground truth images, specified as a character vector or a cell array of character vectors.

`PixelLabelData` — Pixel label file names
character | cell array of characters

This property is read-only.

Pixel label data file names used as the source for ground truth label images, specified as a character or a cell array of characters.

`ClassNames` — Class names
cell array of character vectors

This property is read-only.

Class names, specified as a cell array of character vectors.

`ColorPreprocessing` — Color channel preprocessing
`'none'` (default) | `'gray2rgb'` | `'rgb2gray'`

Color channel preprocessing for 2-D data, specified as `'none'`, `'gray2rgb'`, or `'rgb2gray'`. Use this property when you need the image data created by the data source must be only color or grayscale, but the training set includes both. Suppose you need to train a network that expects color images but some of your training images are grayscale. Set `ColorPreprocessing` to `'gray2rgb'` to replicate the color channels of the grayscale images in the input image set. Using the `'gray2rgb'` option creates *M*-by-*N*-by-3 output images.

The `ColorPreprocessing` property is not supported for 3-D data. To perform color channel preprocessing of 3-D data, use the `transform` function.

`DataAugmentation` — Preprocessing applied to input images
`'none'` (default) | `imageDataAugmenter` object

Preprocessing applied to input images, specified as an `imageDataAugmenter` object or `'none'`. When `DataAugmentation` is `'none'`, no preprocessing is applied to input images. Training data can be augmented in real-time during training.

The `DataAugmentation` property is not supported for 3-D data. To preprocess 3-D data, use the `transform` function.

`DispatchInBackground` — Dispatch observations in background
`false` (default) | `true`

Dispatch observations in the background during training, prediction, and classification, specified as `false` or `true`. To use background dispatching, you must have Parallel Computing Toolbox. If `DispatchInBackground` is `true` and you have Parallel Computing Toolbox, then `pixelLabelImageDatastore` asynchronously reads patches, adds noise, and queues patch pairs.

`MiniBatchSize` — Number of observations in each batch
positive integer

Number of observations that are returned in each batch. The default value is equal to the `ReadSize` of image datastore `imds`. You can change the value of `MiniBatchSize` only after you create the datastore. For training, prediction, or classification, the `MiniBatchSize` property is set to the mini-batch size defined in `trainingOptions`.

`NumObservations` — Total number of observations in the datastore
positive integer

This property is read-only.

Total number of observations in the denoising image datastore. The number of observations is the length of one training epoch.

`OutputSize` — Size of output images
`[]` (default) | vector of two positive integers

This property is read-only.

Size of output images, specified as a vector of two positive integers. The first element specifies the number of rows in the output images, and the second element specifies the number of columns. When you specify `OutputSize`, image sizes are adjusted as necessary. By default, this property is empty, which means that the images are not adjusted.

The `OutputSize` property is not supported for 3-D data. To set the output size of 3-D data, use the `transform` function.

`OutputSizeMode` — Method used to resize output images
`'resize'` (default) | `'centercrop'` | `'randcrop'`

Method used to resize output images, specified as one of the following. This property applies only when you set `OutputSize` to a value other than `[]`.

- `'resize'` — Scale the image to fit the output size. For more information, see `imresize`.
- `'centercrop'` — Take a crop from the center of the training image. The crop has the same size as the output size.
- `'randcrop'` — Take a random crop from the training image. The random crop has the same size as the output size.

Data Types: `char` | `string`

Object Functions

combine	Combine data from multiple datastores
countEachLabel	Count occurrence of pixel or box labels
hasdata	Determine if data is available to read
partitionByIndex	(To be removed) Partition pixelLabelImageDatastore according to indices
preview	Preview subset of data in datastore
read	Read data from label datastore
readall	Read all data in datastore
readByIndex	(To be removed) Read data specified by index from pixelLabelImageDatastore
reset	Reset datastore to initial state
shuffle	Return shuffled version of label datastore
transform	Transform datastore

Examples

Create Datastore for Training Semantic Segmentation Network

Load the training data.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
imageDir = fullfile(dataSetDir,"trainingImages");
labelDir = fullfile(dataSetDir,"trainingLabels");
```

Create an image datastore for the images.

```
imds = imageDatastore(imageDir);
```

Create a `pixelLabelDatastore` for the ground truth pixel labels.

```
classNames = ["triangle" "background"];
labelIDs   = [255 0];
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Combine the image and pixel label datastore for training.

```
trainingData = pixelLabelImageDatastore(imds,pxds);
```

Tips

- The `pixelLabelDatastore` `pxds` and the `imageDatastore` `imds` store files that are located in a folder in lexicographical order. For example, if you have twelve files named `'file1.jpg'`, `'file2.jpg'`, ... , `'file11.jpg'`, and `'file12.jpg'`, then the files are stored in this order:

  ```
  'file1.jpg'
  'file10.jpg'
  'file11.jpg'
  'file12.jpg'
  'file2.jpg'
  'file3.jpg'
  ...
  'file9.jpg'
  ```

  Files that are stored in a cell array are read in the same order as they are stored.

If the order of files in `pxds` and `imds` are not the same, then you may encounter a mismatch when you read a ground truth image and corresponding label data using a `pixelLabelImageDatastore`. If this occurs, then rename the pixel label files so that they have the correct order. For example, rename `'file1.jpg'`, ... , `'file9.jpg'` to `'file01.jpg'`, ..., `'file09.jpg'`.

- To extract semantic segmentation data from a `groundTruth` object generated by the Video Labeler, use the `pixelLabelTrainingData` function.

Version History

Introduced in R2018a

R2025a: Warns
Warns starting in R2025a

The `pixelLabelImageDatastore` issues a warning that it will be removed in a future release. Use the `imageDatastore` and `pixelLabelDatastore` objects and the `combine` function instead.

R2022b: To be removed
Not recommended starting in R2022b

The `pixelLabelImageDatastore` will be removed in a future release.

See Also

Functions
`trainnet` | `combine`

Objects
`groundTruth` | `pixelLabelDatastore` | `pixelLabelTrainingData` | `imageDatastore`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)

partitionByIndex

(To be removed) Partition `pixelLabelImageDatastore` according to indices

Note `pixelLabelImageDatastore` object and the `partitionByIndex` function will be removed in a future release. For more information, see “Version History”.

Syntax

```
pximds2 = partitionByIndex(pximds,ind)
```

Description

`pximds2 = partitionByIndex(pximds,ind)` partitions a subset of observations in a pixel label image datastore, `pximds`, into a new datastore, `pximds2`. The desired observations are specified by indices, `ind`.

Input Arguments

`pximds` — Pixel label image datastore

`pixelLabelImageDatastore`

Pixel label image datastore, specified as a `pixelLabelImageDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`pximds2` — Output datastore

`pixelLabelImageDatastore` object

Output datastore, returned as a `pixelLabelImageDatastore` object containing a subset of files from `pximds`.

Version History

Introduced in R2018a

R2025a: Warns

Warns starting in R2025a

The `pixelLabelImageDatastore` object and the `partitionByIndex` function issue a warning that it will be removed in a future release. Use the `imageDatastore` and `pixelLabelDatastore` objects and the `combine` function instead of the `pixelLabelImageDatastore` object. There is no replacement for the `partitionByIndex` function.

R2022b: To be removed
Not recommended starting in R2022b

The `pixelLabelImageDatastore` object and the `partitionByIndex` function will be removed in a future release.

See Also

`readall` | `read`

readByIndex

(To be removed) Read data specified by index from `pixelLabelImageDatastore`

Note `pixelLabelImageDatastore` object and the `readByIndex` function will be removed in a future release. For more information, see “Version History”.

Syntax

```
data = readByIndex(pximds,ind)
[data,info] = readByIndex(pximds,ind)
```

Description

`data = readByIndex(pximds,ind)` returns a subset of observations from a pixel label image datastore, `pximds`. The desired observations are specified by indices, `ind`.

`[data,info] = readByIndex(pximds,ind)` also returns information about the observations, including metadata, in `info`.

Input Arguments

`pximds` — Pixel label image datastore

`pixelLabelImageDatastore`

Pixel label image datastore, specified as a `pixelLabelImageDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`data` — Observations from datastore

table

Observations from the datastore, returned as a table with `length(ind)` number of rows.

`info` — Information about read data

structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`Filename`	`Filename` is a fully resolved path containing the path string, name of the file, and file extension.

Field Name	Description
`FileSize`	Total file size, in bytes. For MAT-files, `FileSize` is the total number of key-value pairs in the file.

Version History

Introduced in R2018a

R2025a: Warns
Warns starting in R2025a

The `pixelLabelImageDatastore` object and the `readByIndex` function issue a warning that it will be removed in a future release. Use the `imageDatastore` and `pixelLabelDatastore` objects and the `combine` function instead of the `pixelLabelImageDatastore` object. There is no replacement for the `readByIndex` function.

R2022b: To be removed
Not recommended starting in R2022b

The `pixelLabelImageDatastore` object and the `readByIndex` function will be removed in a future release.

See Also

`readall` | `read`

dicePixelClassificationLayer

(To be removed) Create pixel classification layer using generalized Dice loss for semantic segmentation

Note The `DicePixelClassificationLayer` object will be removed in a future release. Use the `trainnet` function and specify the loss using the `generalizedDice` function. For more information, see “Version History”.

Description

A Dice pixel classification layer provides a categorical label for each image pixel or voxel using generalized Dice loss.

The layer uses generalized Dice loss to alleviate the problem of class imbalance in semantic segmentation problems. Generalized Dice loss controls the contribution that each class makes to the loss by weighting classes by the inverse size of the expected region.

Creation

Syntax

```
layer = dicePixelClassificationLayer
layer = dicePixelClassificationLayer(Name,Value)
```

Description

`layer = dicePixelClassificationLayer` creates a Dice pixel classification output layer for semantic image segmentation networks. The layer outputs the categorical label for each image pixel or voxel processed by a CNN. The layer automatically ignores undefined pixel labels during training.

`layer = dicePixelClassificationLayer(Name,Value)` returns a Dice pixel classification output layer using name-value arguments to set the optional `Classes` and `Name` properties. You can specify multiple name-value arguments.

For example, `dicePixelClassificationLayer("Name","pixclass")` creates a Dice pixel classification layer with the name `"pixclass"`.

Properties

`Classes` — Classes of the output layer

`"auto"` (default) | categorical vector | string array | cell array of character vectors

Classes of the output layer, specified as a categorical vector, string array, cell array of character vectors, or `"auto"`. If `Classes` is `"auto"`, then the software automatically sets the classes at training time. If you specify the string array or cell array of character vectors `str`, then the software sets the classes of the output layer to `categorical(str,str)`.

Data Types: `char` | `categorical` | `string` | `cell`

`OutputSize` — Output size
`'auto'` (default)

This property is read-only.

The output size of the layer. The value is `'auto'` prior to training, and is specified as a numeric value at training time.

`LossFunction` — Loss function
`'generalizedDiceLoss'` (default)

This property is read-only.

Loss function used for training, specified as `'generalizedDiceLoss'`.

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
1 (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

Examples

Create 2-D Semantic Segmentation Network with Dice Pixel Classification Layer

Predict the categorical label of every pixel in an input image using a generalized Dice loss function.

```
layers = [
      imageInputLayer([480 640 3])
      convolution2dLayer(3,16,'Stride',2,'Padding',1)
      reluLayer
      transposedConv2dLayer(2,4,'Stride',2)
      softmaxLayer
```

```
      dicePixelClassificationLayer
      ]

layers =
  6×1 Layer array with layers:

     1   ''   Image Input                     480×640×3 images with 'zerocenter' normalization
     2   ''   2-D Convolution                16 3×3 convolutions with stride [2  2] and paddi
     3   ''   ReLU                           ReLU
     4   ''   2-D Transposed Convolution     4 2×2 transposed convolutions with stride [2  2]
     5   ''   Softmax                        Softmax
     6   ''   Dice Pixel Classification Layer   Generalized Dice loss
```

More About

Dice Loss

The Dice loss function is based on the Sørensen-Dice similarity coefficient for measuring overlap between two segmented images.

The generalized Dice loss function L used by `dicePixelClassificationLayer` for the loss between one image Y and the corresponding ground truth T is given by:

$$L = 1 - \frac{2\sum_{k=1}^{K} w_k \sum_{m=1}^{M} Y_{km} T_{km}}{\sum_{k=1}^{K} w_k \sum_{m=1}^{M} Y_{km}^2 + T_{km}^2}$$

K is the number of classes, M is the number of elements along the first two dimensions of Y, and w_k is a class specific weighting factor that controls the contribution each class makes to the loss. This weighting helps counter the influence of larger regions on the Dice score, making it easier for the network to learn how to segment smaller regions. w_k is typically the inverse area of the expected region:

$$w_k = \frac{1}{\left(\sum_{m=1}^{M} T_{km}\right)^2}$$

There are several variations of generalized Dice Loss functions [1], [2]. The function used in `dicePixelClassificationLayer` has squared terms to ensure that the derivative is 0 when the prediction matches the ground truth [3].

Version History

Introduced in R2019b

R2024a: `dicePixelClassificationLayer` will be removed

Not recommended starting in R2024a

The `dicePixelClassificationLayer` object will be removed in a future release. Follow these steps to update your code:

- Define your network as a `dlnetwork` object. You can use functions such as `addLayers` and `connectLayers` to build the network. Do not include output layers in the network.

- Define a custom loss function that uses the `generalizedDice` function. Here is a sample loss function appropriate for Dice pixel classification:

```
function loss = modelLoss(Y,T)
  z = generalizedDice(Y,T);
  loss = 1 - mean(z,"all");
end
```

- Train the network using the `trainnet` function with the custom loss function. For example, this code trains a `dlnetwork` network called `net` using the training data `images` and the custom loss function `modelLoss`.

```
netTrained = trainnet(images,net,@modelLoss,options);
```

References

[1] Crum, William R., Oscar Camara, and Derek LG Hill. "Generalized overlap measures for evaluation and validation in medical image analysis." *IEEE Transactions on Medical Imaging*. 25.11, 2006, pp. 1451–1461.

[2] Sudre, Carole H., et al. "Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations." *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, Cham, 2017, pp. 240–248.

[3] Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". *Fourth International Conference on 3D Vision (3DV)*. Stanford, CA, 2016: pp. 565–571.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM® GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).
- For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.
- The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer

description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

See Also

`dlnetwork` | `generalizedDice` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“3-D Brain Tumor Segmentation Using Deep Learning”
“Get Started with Semantic Segmentation Using Deep Learning”
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)

focalLossLayer

(To be removed) Create focal loss layer using focal loss function

Note The `FocalLossLayer` object will be removed in a future release. Use the `trainnet` function and specify the loss as the `focalCrossEntropy` function. For more information, see "Version History".

Description

A focal loss layer predicts object classes using focal loss. Add the focal loss layer to train an object detection, semantic segmentation, or a classification network when imbalance exists between foreground and background classes. To compensate for class imbalance, the focal loss function multiplies the cross entropy function with a modulating factor that increases the sensitivity of the network to misclassified observations.

Creation

Syntax

```
layer = focalLossLayer
layer = focalLossLayer(Name,Value)
```

Description

`layer = focalLossLayer` creates a focal loss layer for deep learning networks.

`layer = focalLossLayer(Name,Value)` sets properties of the focal loss layer by using one or more name-value pair arguments. Enclose each property name in quotes.

For example, `focalLossLayer('Name','focalloss')` creates a focal loss layer with the name `'focalloss'` and the specified balancing and focusing parameters.

Properties

`Alpha` — Balancing parameter
0.25 (default) | positive real number

Balancing parameter of the focal loss function, specified as a positive real number. The `Alpha` value scales the loss function linearly and is typically set to `0.25`. If you decrease `Alpha`, increase `Gamma`.

`Gamma` — Focusing parameter
2.0 (default) | positive real number

Focusing parameter of the focal loss function, specified as a positive real number. Increasing the value of `Gamma` increases the sensitivity of the network to misclassified observations.

`Classes` — Object classes to detect

`'auto'` (default) | string vector | categorical vector | cell array of character vectors

Object classes that the network is trained to detect, specified as a string vector, categorical vector, cell array of character vectors, or `'auto'`. When you set `Classes` to `'auto'`, the classes are automatically set at training time. When you specify a string vector or cell array of character vectors, then the elements of `Classes` are sorted according to the output of the `categories` function.

Data Types: `string` | `categorical` | `cell` | `char`

`Name` — Layer name

`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

Examples

Create Focal Loss Layer

Specify class names.

```
classes = ["Vehicle","Background"];
```

Specify the balancing parameter and focusing parameter of the focal loss function. Create a focal loss layer named `"focallosslayer"` for the two classes, displaying results.

```
layer = focalLossLayer(Classes=classes,Name="focallosslayer")
```

```
layer =
  FocalLossLayer with properties:

           Name: 'focallosslayer'

   Hyperparameters
          Gamma: 2
          Alpha: 0.2500
        Classes: [2x1 categorical]
    LossFunction: 'focalLoss'
```

Create 2-D Semantic Segmentation Network with Focal Loss Layer

Create a DeepLab v3+ network based on ResNet-18.

```
imageSize = [480 640 3];
numClasses = 5;
network = "resnet18";
lgraph = deeplabv3plusLayers(imageSize,numClasses,network,DownsamplingFactor=16)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'data'}
```

```
    OutputNames: {'classification'}
         Layers: [100x1 nnet.cnn.layer.Layer]
    Connections: [113x2 table]
```

Display the output layer of the network. The output layer of the DeepLab v3+ network is a `PixelClassificationLayer` that uses cross-entropy loss to predict the categorical label for every pixel in an input 2-D image.

```
lgraph.Layers(end)
```

```
ans =
  PixelClassificationLayer with properties:

            Name: 'classification'
         Classes: 'auto'
    ClassWeights: 'none'
      OutputSize: 'auto'

   Hyperparameters
    LossFunction: 'crossentropyex'
```

Replace the output `PixelClassificationLayer` with the `FocalLossLayer` to handle imbalanced classes in data.

```
layer = focalLossLayer("Name","focalloss");
lgraph = replaceLayer(lgraph,"classification",layer);
```

Display the network.

```
analyzeNetwork(lgraph);
```

Create 3-D Semantic Segmentation Network with Focal Loss Layer

Create a 3-D U-Net network for semantic segmentation by using the `unet3dLayers` function. Set the encoder-decoder depth to 2 and specify the number of output channels for the first convolution layer as 16.

```
imageSize = [128 128 128 3];
numClasses = 5;
lgraph = unet3dLayers(imageSize,numClasses, ...
    EncoderDepth=2,NumFirstEncoderFilters=16);
figure
plot(lgraph)
```

Create a focal loss layer and replace the `Segmentation-Layer` in the network with the focal loss layer. The layer predicts the categorical label for every voxel in an input 3-D volume.

```
layer = focalLossLayer(Name="focalloss");
lgraph = replaceLayer(lgraph,"Segmentation-Layer",layer)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'ImageInputLayer'}
```

```
    OutputNames: {'focalloss'}
         Layers: [40x1 nnet.cnn.layer.Layer]
    Connections: [41x2 table]
```

Display the network.

```
analyzeNetwork(lgraph);
```

More About

Focal Loss

The focal loss function is based on cross-entropy loss. Focal loss compensates for class imbalance by using a modulating factor that emphasizes hard negatives during training.

The focal loss function, L, used by the `focalLossLayer` object for the loss between one image Y and the corresponding ground truth T is given by:

$$L = -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} T_{mk}\alpha(1-Y_{mk})^{\gamma}\ln(Y_{mk})$$

K is the number of classes. M is the number of observations, which is typically the number of elements along the first two dimensions of Y multiplied by the number of anchor boxes. The function used by the `focalLossLayer` object ensures that the input to the natural logarithm is greater than 0.

Version History

Introduced in R2020a

R2024a: `focalLossLayer` will be removed

Not recommended starting in R2024a

The `focalLossLayer` object will be removed in a future release. Follow these steps to update your code:

- Define your network as a `dlnetwork` object. You can use functions such as `addLayers` and `connectLayers` to build the network. Do not include output layers in the network.
- Train the network using the `trainnet` function and specify the loss using the `focalCrossEntropy` function. For example, this code trains a `dlnetwork` network called `net` using the training data `images`.

  ```
  netTrained = trainnet(images,net,@focalCrossEntropy,options);
  ```

References

[1] Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. "Focal Loss for Dense Object Detection." In 2017 *IEEE® International Conference on Computer Vision (ICCV),* 2999–3007. Venice: IEEE, 2017. https://doi.org/10.1109/ICCV.2017.324.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).
- For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.
- The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

See Also

`dlnetwork` | `focalCrossEntropy` | `trainnet` | `trainSSDObjectDetector` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Get Started with Object Detection Using Deep Learning”
“Getting Started with SSD Multibox Detection”
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)

patchEmbeddingLayer

Patch embedding layer

Description

A patch embedding layer maps patches of pixels to vectors. Use this layer in vision transformer neural networks to encode information about patches in images.

Creation

Syntax

```
layer = patchEmbeddingLayer(patchSize,outputSize)
layer = patchEmbeddingLayer(patchSize,outputSize,Name=Value)
```

Description

`layer = patchEmbeddingLayer(patchSize,outputSize)` creates a patch embedding layer and sets the `PatchSize` and `OutputSize` properties.

This feature requires a Deep Learning Toolbox license.

`layer = patchEmbeddingLayer(patchSize,outputSize,Name=Value)` sets additional properties using one or more name-value arguments.

Properties

Patch Embedding

`PatchSize` — Size of patches to split input images into
positive integer | row vector of positive integers

This property is read-only.

Size of patches to split input images into, specified as a positive integer or row vector of positive integers.

If `PatchSize` is a vector, then each element of `PatchSize` is the size of the patch in the corresponding spatial dimension of the input. If `PatchSize` is a scalar, then the layer uses the same value for all spatial dimensions of the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`SpatialFlattenMode` — Mode for flattening output of convolution operation
`"column-major"` (default) | `"row-major"`

Mode for flattening the output of the convolution operation, specified as `"column-major"` or `"row-major"`.

If `SpatialFlattenMode` is `"column-major"`, then the flatten operation outputs the data in its column-major representation. For example, consider the input:

```
A = [
   1 2 3
   4 5 6
   7 8 9];
```

The output in column-major representation is:

```
AFlat = [1 4 7 2 5 8 3 6 9];
```

If `SpatialFlattenMode` is `"row-major"`, then the flatten operation outputs the data in its row-major representation. For example, consider the input:

```
A = [
   1 2 3
   4 5 6
   7 8 9];
```

The output in row-major representation is:

```
AFlat = [1 2 3 4 5 6 7 8 9];
```

Set this option when creating or importing models that require this representation.

`OutputSize` — Size of output vectors
positive integer

This property is read-only.

Size of output vectors, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`InputSize` — Number of input channels
`"auto"` (default) | positive integer

This property is read-only.

Number of input channels, specified as one of these values:

- `"auto"` — Automatically determine the number of input channels at training time.
- Positive integer — Configure the layer for the specified number of input channels. `InputSize` and the number of channels in the layer input data must match. For example, if the input is an RGB image, then `InputSize` must be 3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Parameters and Initialization

`WeightsInitializer` — Function to initialize weights
`"glorot"` (default) | `"he"` | `"narrow-normal"` | `"zeros"` | `"ones"` | function handle

Function to initialize the weights, specified as one of these values:

- `"glorot"` — Initialize the weights with the Glorot initializer “[1] [2]” on page 2-623 (also known as Xavier initializer). The Glorot initializer independently samples from a normal distribution with zero mean and a variance of `2/(numIn + numOut)`, where `numIn` and `numOut` are the values of the `InputSize` and `OutputSize` properties, respectively.
- `"he"` – Initialize the weights with the He initializer [3]. The He initializer samples from a normal distribution with zero mean and a variance of `2/numIn`, where `numIn` is the value of the `InputSize` property.
- `"narrow-normal"` — Initialize the weights by independently sampling from a normal distribution with zero mean and a standard deviation of 0.01.
- `"zeros"` — Initialize the weights with zeros.
- `"ones"` — Initialize the weights with ones.
- Function handle — Initialize the weights with a custom function. If you specify a function handle, then the function must have the form `weights = func(sz)`, where `sz` is the size of the weights.

The layer initializes the weights only when the `Weights` property is empty.

Data Types: `char` | `string` | `function_handle`

`BiasInitializer` — Function to initialize biases
`"zeros"` (default) | `"narrow-normal"` | `"ones"` | function handle

Function to initialize the biases, specified as one of these values:

- `"zeros"` — Initialize the biases with zeros.
- `"ones"` — Initialize the biases with ones.
- `"narrow-normal"` — Initialize the biases by independently sampling from a normal distribution with a mean of zero and a standard deviation of 0.01.
- Function handle — Initialize the biases with a custom function. If you specify a function handle, then the function must have the form `bias = func(sz)`, where `sz` is the size of the biases.

The layer initializes the biases only when the `Bias` property is empty.

The `PatchEmbeddingLayer` object stores this property as a character vector or a function handle.

Data Types: `char` | `string` | `function_handle`

`Weights` — Learnable weights
`[]` (default) | numeric array

Learnable weights.

If `PatchSize` is a positive integer, then `Weights` is an `PatchSize`-by-...-by-`PatchSize`-by-`InputSize`-by-`OutputSize` numeric array or `[]`, where the number of dimensions of size `PatchSize` is the number of spatial dimensions of the input.

If `PatchSize` is a vector, then `Weights` is an `PatchSize(1)`-by-...-by-`PatchSize(K)`-by-`InputSize`-by-`OutputSize` numeric array or `[]`, where `K` is the number of spatial dimensions of the input.

The layer weights are learnable parameters. You can specify the initial value of the weights directly using the `Weights` property of the layer. When you train a network, if the `Weights` property of the layer is nonempty, then the `trainnet` function uses the `Weights` property as the initial value. If the

`Weights` property is empty, then the software uses the initializer specified by the `WeightsInitializer` property of the layer.

Data Types: `single` | `double`

`Bias` — Layer biases
`[]` (default) | column vector

Layer biases, specified as a numeric column vector of length `OutputSize` or `[]`.

The layer biases are learnable parameters. When you train a neural network, if `Bias` is nonempty, then the `trainnet` function uses the `Bias` property as the initial value. If `Bias` is empty, then software uses the initializer specified by the `BiasInitializer` property.

Data Types: `single` | `double`

Learning Rate and Regularization

`WeightLearnRateFactor` — Learning rate factor for weights
`1` (default) | nonnegative scalar

Learning rate factor for the weights, specified as a nonnegative scalar.

The software multiplies this factor by the global learning rate to determine the learning rate for the weights in this layer. For example, if `WeightLearnRateFactor` is `2`, then the learning rate for the weights in this layer is twice the current global learning rate. The software determines the global learning rate based on the settings you specify using the `trainingOptions` function.

Data Types: `double`

`BiasLearnRateFactor` — Learning rate factor for biases
`1` (default) | nonnegative scalar

Learning rate factor for the biases, specified as a nonnegative scalar.

The software multiplies this factor by the global learning rate to determine the learning rate for the biases in this layer. For example, if `BiasLearnRateFactor` is `2`, then the learning rate for the biases in the layer is twice the current global learning rate. The software determines the global learning rate based on the settings you specify using the `trainingOptions` function.

The `PatchEmbeddingLayer` object stores this property as `double` type.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`WeightL2Factor` — L_2 regularization factor for weights
`1` (default) | nonnegative scalar

L_2 regularization factor for the weights, specified as a nonnegative scalar.

The software multiplies this factor by the global L_2 regularization factor to determine the L_2 regularization for the weights in this layer. For example, if `WeightL2Factor` is `2`, then the L_2 regularization for the weights in this layer is twice the global L_2 regularization factor. You can specify the global L_2 regularization factor using the `trainingOptions` function.

Data Types: `double`

`BiasL2Factor` — L_2 regularization factor for biases
`0` (default) | nonnegative scalar

L_2 regularization factor for the biases, specified as a nonnegative scalar.

The software multiplies this factor by the global L_2 regularization factor to determine the L_2 regularization for the biases in this layer. For example, if `BiasL2Factor` is `2`, then the L_2 regularization for the biases in this layer is twice the global L_2 regularization factor. The software determines the global L_2 regularization factor based on the settings you specify using the `trainingOptions` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Layer

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
`1` (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Create Patch Embedding Layer

Create a patch embedding layer that embeds patches of size 16 with an output size of 768.

```
patchSize = 16;
embeddingOutputSize = 768;
layer = patchEmbeddingLayer(patchSize,embeddingOutputSize)
```

```
layer =
  PatchEmbeddingLayer with properties:

                     Name: ''
                PatchSize: 16
                InputSize: 'auto'
               OutputSize: 768
        SpatialFlattenMode: 'column-major'
        WeightsInitializer: 'glorot'
           BiasInitializer: 'zeros'
    WeightLearnRateFactor: 1
      BiasLearnRateFactor: 1
           WeightL2Factor: 1
             BiasL2Factor: 1

   Learnable Parameters
                  Weights: []
                     Bias: []

   State Parameters
    No properties.

  Show all properties
```

Create a `dlnetwork` object.

```
net = dlnetwork;
```

Specify layers of the network, including a patch embedding layer.

```
inputSize = [384 384 3];

maxPosition = (inputSize(1)/patchSize)^2 + 1;

numHeads = 4;
numKeyChannels = 4*embeddingOutputSize;

numClasses = 1000;

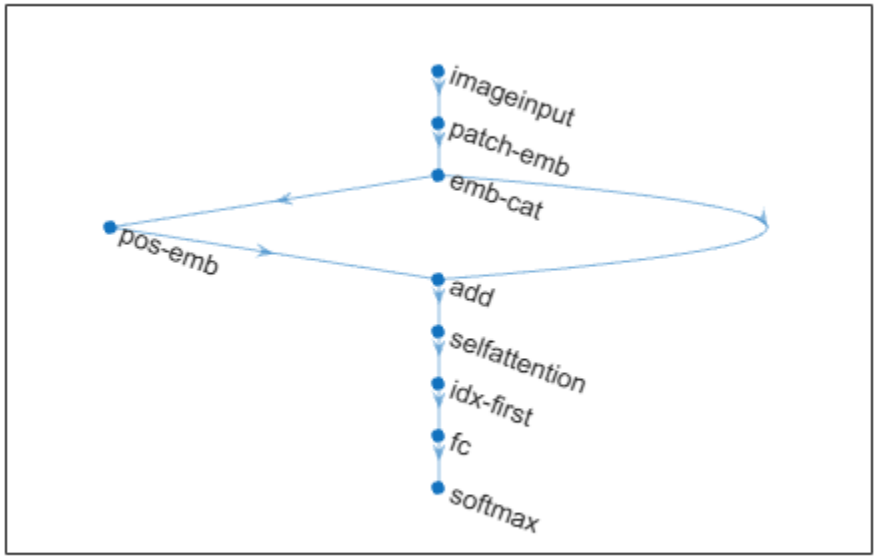
layers = [
    imageInputLayer(inputSize)
    patchEmbeddingLayer(patchSize,embeddingOutputSize,Name="patch-emb")
    embeddingConcatenationLayer(Name="emb-cat")
    positionEmbeddingLayer(embeddingOutputSize,maxPosition,Name="pos-emb");
    additionLayer(2,Name="add")
    selfAttentionLayer(numHeads,numKeyChannels,AttentionMask="causal")
    indexing1dLayer(Name="idx-first")
    fullyConnectedLayer(numClasses)
    softmaxLayer];
net = addLayers(net,layers);
```

Connect the embedding concatenation layer with the "in2" input of the addition layer.

```
net = connectLayers(net,"emb-cat","add/in2");
```

View the neural network architecture.

```
plot(net)
```



Algorithms

Patch Embedding Layer

A patch embedding layer maps patches of pixels to vectors. You can use this layer in vision transformer neural networks to encode information about patches in images.

The layer uses a convolution operation with the layer weights and biases to extract and project patches from the input. In particular, the layer:

1. Splits the input into non-overlapping patches.
2. Flattens the patches.
3. Projects the flattened patches to the output size.
4. Flattens the spatial dimensions of the projected output.

Layer Input and Output Formats

Most layers in a layer array or layer graph pass data to subsequent layers as formatted `dlarray` objects. The format of a `dlarray` object is a string of characters in which each character describes the corresponding dimension of the data. The format consists of one or more of these characters:

- `"S"` — Spatial
- `"C"` — Channel
- `"B"` — Batch
- `"T"` — Time
- `"U"` — Unspecified

For example, you can describe 2-D image data that is represented as a 4-D array, where the first two dimensions correspond to the spatial dimensions of the images, the third dimension corresponds to the channels of the images, and the fourth dimension corresponds to the batch dimension, as having the format `"SSCB"` (spatial, spatial, channel, batch).

You can interact with these `dlarray` objects in automatic differentiation workflows, such as those for:

- developing a custom layer
- using a `functionLayer` object
- using the `forward` and `predict` functions with `dlnetwork` objects

This table shows the supported input formats of `PatchEmbeddingLayer` objects and the corresponding output format. If the software passes the output of the layer to a custom layer that does not inherit from the `nnet.layer.Formattable` class, or to a `FunctionLayer` object with the `Formattable` property set to `0` (`false`), then the layer receives an unformatted `dlarray` object with dimensions ordered according to the formats in this table. The formats listed here are only a subset of the formats that the layer supports. The layer might support additional formats, such as formats with additional `"S"` (spatial) or `"U"` (unspecified) dimensions.

Input Format	Output Format
`"SCB"` (spatial, channel, batch)	`"SCB"` (spatial, channel, batch)
`"SSCB"` (spatial, spatial, channel, batch)	`"SCB"` (spatial, channel, batch)
`"SSSCB"` (spatial, spatial, spatial, channel, batch)	`"SCB"` (spatial, channel, batch)
`"SCBT"` (spatial, channel, batch, time)	`"SCBT"` (spatial, channel, batch, time)
`"SSCBT"` (spatial, spatial, channel, batch, time)	`"SCBT"` (spatial, channel, batch, time)
`"SSSCBT"` (spatial, spatial, spatial, channel, batch, time)	`"SCBT"` (spatial, channel, batch, time)
`"SC"` (spatial, channel)	`"SC"` (spatial, channel)
`"SSC"` (spatial, spatial, channel)	`"SC"` (spatial, channel)
`"SSSC"` (spatial, spatial, spatial, channel)	`"SC"` (spatial, channel)

In `dlnetwork` objects, `PatchEmbeddingLayer` objects also support these input and output format combinations.

Input Format	Output Format
`"SCT"` (spatial, channel, time)	`"SCT"` (spatial, channel, time)
`"SSCT"` (spatial, spatial, channel, time)	`"SCT"` (spatial, channel, time)
`"SSSCT"` (spatial, spatial, spatial, channel, time)	`"SCT"` (spatial, channel, time)

Version History

Introduced in R2023b

R2024a: Specify spatial flattening mode of patch embedding layers

Specify the mode for flattening output of the convolution operation using the `SpatialFlattenMode` option. Set this option when creating or importing models that require this representation.

References

[1] Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An Image is Worth 16x16 words: Transformers for Image Recognition at Scale." Preprint, submitted June 3, 2021. https://doi.org/10.48550/arXiv.2010.11929.

[2] Glorot, Xavier, and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–356. Sardinia, Italy: AISTATS, 2010. https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

[3] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026–34. Santiago, Chile: IEEE, 2015. https://doi.org/10.1109/ICCV.2015.123

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation supports only 1-D and 2-D spatial data. 3-D spatial or more than 3-D spatial data format such as "SSS" or "SSSS" is not supported.
- You can generate generic C/C++ code that does not depend on third-party libraries and deploy the generated code to hardware platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Code generation supports only 1-D and 2-D spatial data. 3-D spatial or more than 3-D spatial data format such as "SSS" or "SSSS" is not supported.
- You can generate CUDA code that is independent of deep learning libraries and deploy the generated code to platforms that use NVIDIA GPU processors.

See Also

`visionTransformer` | `selfAttentionLayer` | `embeddingConcatenationLayer` | `indexing1dLayer` | `trainnet` | `trainingOptions` | `dlnetwork`

Topics
“Train Vision Transformer Network for Image Classification”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning Tips and Tricks” (Deep Learning Toolbox)
“Data Sets for Deep Learning” (Deep Learning Toolbox)

pixelClassificationLayer

(To be removed) Create pixel classification layer for semantic segmentation

Note The `pixelClassificationLayer` object will be removed in a future release. Use the `trainnet` function and specify the loss using the `crossentropy` function. For more information, see "Version History".

Description

A pixel classification layer provides a categorical label for each image pixel or voxel.

Creation

Syntax

```
layer = pixelClassificationLayer
layer = pixelClassificationLayer(Name=Value)
```

Description

`layer = pixelClassificationLayer` creates a pixel classification output layer for semantic image segmentation networks. The layer outputs the categorical label for each image pixel or voxel processed by a CNN. The layer automatically ignores undefined pixel labels during training.

`layer = pixelClassificationLayer(Name=Value)` returns a pixel classification output layer using one or more name-value arguments to set the optional `Classes`, `ClassWeights`, and `Name`. For example, `pixelClassificationLayer(Name="pixclass")` creates a pixel classification layer with the name `pixclass`.

Properties

`Classes` — Classes of the output layer

`"auto"` (default) | categorical vector | string array | cell array of character vectors

Classes of the output layer, specified as a categorical vector, string array, cell array of character vectors, or `"auto"`. If `Classes` is `"auto"`, then the software automatically sets the classes at training time. If you specify the string array or cell array of character vectors `str`, then the software sets the classes of the output layer to `categorical(str,str)`.

Data Types: `char` | `categorical` | `string` | `cell`

`ClassWeights` — Class weights

`"none"` (default) | vector of real scalar

Class weights, specified as `"none"` or as a vector of real scalar. The elements of the vector correspond to the classes in `Classes`. If you specify `ClassWeights`, then you must specify `Classes`.

Use class weighting to balance classes when there are underrepresented classes in the training data.

`OutputSize` — Output size
`"auto"` (default)

This property is read-only.

The output size of the layer. The value is `"auto"` prior to training, and is specified as a numeric value at training time.

`LossFunction` — Loss function
`"crossentropyex"` (default)

This property is read-only.

Loss function used for training, specified as `"crossentropyex"`.

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

Examples

Use Pixel Classification Layer to Create Semantic Segmentation Network

Predict the categorical label of every pixel in an input image.

```
layers = [
         imageInputLayer([32 32 3])
         convolution2dLayer(3,16,Stride=2,Padding=1)
         reluLayer
         transposedConv2dLayer(3,1,Stride=2,Cropping=1)
         softmaxLayer
```

```
        pixelClassificationLayer
      ]

layers =
  6×1 Layer array with layers:

     1   ''   Image Input                 32×32×3 images with 'zerocenter' normalization
     2   ''   2-D Convolution             16 3×3 convolutions with stride [2  2] and padding [1
     3   ''   ReLU                        ReLU
     4   ''   2-D Transposed Convolution  1 3×3 transposed convolutions with stride [2  2] and
     5   ''   Softmax                     Softmax
     6   ''   Pixel Classification Layer  Cross-entropy loss
```

Use Weighting to Balance Classes in Training Data

Balance classes using inverse class frequency weighting when some classes are underrepresented in the training data. First, count class frequencies over the training data using pixelLabelDatastore. Then, set the 'ClassWeights' in pixelClassificationLayer to the computed inverse class frequencies.

Set the location of image and pixel label data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
imDir = fullfile(dataDir,"building");
pxDir = fullfile(dataDir,"buildingPixelLabels");
```

Create a pixel label image datastore using the ground truth images in `imds` and the pixel labeled images in `pxds`.

```
imds = imageDatastore(imDir);
classNames = ["sky" "grass" "building" "sidewalk"];
pixelLabelID = [1 2 3 4];
pxds = pixelLabelDatastore(pxDir,classNames,pixelLabelID);
```

Tabulate class distribution in dataset.

```
tbl = countEachLabel(pxds)
```

```
tbl=4×3 table
        Name        PixelCount    ImagePixelCount
    ____________    __________    _______________

    {'sky'     }    3.1485e+05       1.536e+06
    {'grass'   }    1.5979e+05       1.536e+06
    {'building'}    1.0312e+06       1.536e+06
    {'sidewalk'}         25313       9.216e+05
```

Calculate inverse frequency class weights.

```
totalNumberOfPixels = sum(tbl.PixelCount);
frequency = tbl.PixelCount / totalNumberOfPixels;
inverseFrequency = 1./frequency
```

```
inverseFrequency = 4×1

    4.8632
```

```
    9.5827
    1.4848
   60.4900
```

Set 'ClassWeights' to the inverse class frequencies.

```
  layer = pixelClassificationLayer(...
      Classes=tbl.Name,ClassWeights=inverseFrequency)
```

```
layer =
  PixelClassificationLayer with properties:

           Name: ''
        Classes: [sky    grass    building    sidewalk]
   ClassWeights: [4×1 double]
     OutputSize: 'auto'

   Hyperparameters
    LossFunction: 'crossentropyex'
```

Version History

Introduced in R2017b

R2024a: `pixelClassificationLayer` will be removed

Not recommended starting in R2024a

The `pixelClassificationLayer` object will be removed in a future release. Follow these steps to update your code:

- Define your network as a `dlnetwork` object. You can use functions such as `addLayers` and `connectLayers` to build the network. Do not include output layers in the network.
- Define a custom loss function that uses the `crossentropy` function. Here is a sample loss function appropriate for pixel classification:

  ```
  function loss = modelLoss(Y,T)
    mask = ~isnan(T);
    targets(isnan(T)) = 0;
    loss = crossentropy(Y,T,Mask=mask,NormalizationFactor="mask-included");
  end
  ```

- Train the network using the `trainnet` function with the custom loss function. For example, this code trains a `dlnetwork` network called `net` using the training data `images` and the custom loss function `modelLoss`.

  ```
  netTrained = trainnet(images,net,@modelLoss,options);
  ```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer

description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).
- For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.
- The code generator represents characters in an 8-bit ASCII codeset that the locale setting determines. Therefore, the use of non-ASCII characters in class names, layer names, layer description, or network names might result in errors. For more information, see “Encoding of Characters in Code Generation” (MATLAB Coder).

See Also

`dlnetwork` | `crossentropy` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)

rcnnBoxRegressionLayer

(Not recommended) Box regression layer for Fast and Faster R-CNN

Note `RCNNBoxRegressionLayer` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Description

A box regression layer refines bounding box locations by using a smooth L1 loss function. Use this layer to create a Fast or Faster R-CNN object detection network.

Creation

Syntax

```
layer = rcnnBoxRegressionLayer
layer = rcnnBoxRegressionLayer('Name',Name)
```

Description

`layer = rcnnBoxRegressionLayer` creates a box regression layer for a Fast or Faster R-CNN object detection network.

`layer = rcnnBoxRegressionLayer('Name',Name)` creates a box regression layer and sets the optional `Name` property.

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

Examples

Create R-CNN Box Regression Layer

Create an R-CNN box regression layer with the name `'rcnn_box_reg'`.

```
rcnnBoxRegression = rcnnBoxRegressionLayer('Name','rcnn_box_reg');
```

Version History

Introduced in R2018b

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.

See Also

`trainFasterRCNNObjectDetector` | `trainFastRCNNObjectDetector` | `regressionLayer`

Topics

“Create Fast R-CNN Object Detection Network”
“Create Faster R-CNN Object Detection Network”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

regionProposalLayer

(Not recommended) Region proposal layer for Faster R-CNN

Note `RegionProposalLayer` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see "Version History".

Description

A region proposal layer outputs bounding boxes around potential objects in an image as part of the region proposal network (RPN) within Faster R-CNN. These outputs are further refined by additional layers within Faster R-CNN to produce the final object detection results.

There are two inputs to this layer:

- `'scores'` — The classification scores produced by the RPN classification branch
- `'boxDeltas'` — The bounding box deltas produced by the RPN regression branch

Use the input names when connecting or disconnecting the region proposal layer to other layers using `connectLayers` or `disconnectLayers` (requires Deep Learning Toolbox).

Creation

Syntax

```
layer = regionProposalLayer(anchorBoxes)
layer = regionProposalLayer(anchorBoxes,'Name',Name)
```

Description

`layer = regionProposalLayer(anchorBoxes)` creates a region proposal layer for building Faster R-CNN object detection networks, and sets the `AnchorBoxes` property.

`layer = regionProposalLayer(anchorBoxes,'Name',Name)` creates a region proposal layer and sets the optional `Name` property.

Properties

`AnchorBoxes` — Anchor boxes
M-by-2 matrix

Anchor boxes, specified as an *M*-by-2 matrix defining the [*height width*] of *M* anchor boxes.

Anchor boxes are predefined bounding box templates of fixed size. The size of each anchor box is typically determined based on a priori knowledge of the scale and aspect ratio of objects in the training dataset. An RPN network is trained to predict the translation and rescaling needed to align the anchor boxes with the ground truth bounding boxes. [1]

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
2 (default)

Number of inputs of the layer. This layer has two inputs.

Data Types: `double`

`InputNames` — Input names
`{'scores' 'boxDeltas'}` (default)

Input names of the layer. This layer has two inputs, named `'scores'` and `'boxDeltas'`.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Create Region Proposal Layer

Define three square anchor boxes for the region proposal layer.

```
anchorBoxes = [
    16 16
    64 64
    128 128
    ];
```

Create a region proposal layer with the name `'region_proposal'`.

```
regionProposal = regionProposalLayer(anchorBoxes,'Name','region_proposal');
```

Version History

Introduced in R2018b

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

References

[1] Ren, S., K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *Advances in Neural Information Processing Systems*. Vol. 28, 2015.

See Also

`trainFasterRCNNObjectDetector` | `layerGraph` | `connectLayers` | `removeLayers`

Topics

“Create Faster R-CNN Object Detection Network”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

roiAlignLayer

Non-quantized ROI pooling layer for Mask-CNN

Description

An ROI align layer outputs fixed size feature maps for every rectangular ROI within an input feature map. Use this layer to create a Mask R-CNN network.

Given an input feature map of size [*H W C N*], where *C* is the number of channels and *N* is the number of observations, the output feature map size is [*h w C* `sum`(*M*)], where *h* and *w* are the specified output size. *M* is a vector of length *N* and *M*(*i*) is the number of ROIs associated with the *i*-th input feature map.

There are two inputs to this layer:

- `'in'` — The input feature map
- `'roi'` — A list of ROIs to pool

Use the input names when connecting or disconnecting the ROI align layer to other layers using `connectLayers` or `disconnectLayers` (requires Deep Learning Toolbox).

Creation

Syntax

```
layer = roiAlignLayer(outputSize)
layer = roiAlignLayer(outputSize,Name,Value)
```

Description

`layer = roiAlignLayer(outputSize)` creates an ROI align layer with pooled output size `outputSize`. The `outputSize` input sets the OutputSize property.

`layer = roiAlignLayer(outputSize,Name,Value)` set properties of the ROI align layer by using one or more name-value pair arguments. Enclose each property name in quotes.

For example, `roiAlignLayer([7 7],'Name','roialignlayer')` creates an ROI align layer with a pooled output size of 7-by-7 pixels and name `'roialignlayer'`.

Properties

`OutputSize` — Pooled output size
vector of two positive integers

Pooled output size, specified as a vector of two positive integers `[h w]`, where `h` is the height and `w` is the width.

Data Types: `double`

`ROIScale` — Scale of input feature map to input image
1 (default) | positive number

Scale of the input feature map to the input image, specified as a positive number.

Data Types: `double`

`SamplingRatio` — Number of samples in each pooled bin
`'auto'` (default) | row vector of two positive integers

Number of samples in each pooled bin, specified as `'auto'` or a row vector of two positive integers. The two elements are the number of vertical and horizontal samples, respectively.

If you do not specify the sampling ratio, then the number of vertical samples has the default value `ceil(`*`roiHeight/outputHeight`*`)`. Likewise, the number of horizontal samples has the default value `ceil(`*`roiWidth/outputWidth`*`)`.

Data Types: `double` | `char`

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
2 (default)

Number of inputs of the layer. This layer accepts two inputs.

Data Types: `double`

`InputNames` — Input names
`{'in' 'roi'}` (default)

Input names of the layer.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Create ROI Align Layer

Specify the pooled output size.

```
outputSize = [7 7];
```

Create an ROI align layer named 'roialign'.

```
layer = roiAlignLayer(outputSize,'Name','roialign')
```

```
layer =
  ROIAlignLayer with properties:

            Name: 'roialign'
       NumInputs: 2
      InputNames: {'in'  'roi'}
      OutputSize: [7 7]

   Hyperparameters
         ROIScale: 1
    SamplingRatio: 'auto'
```

More About

ROI Align Layer

An ROI align layer outputs fixed size feature maps for every rectangular ROI within an input feature map. The layer first partitions an ROI into fixed sized bins of size `OutputSize` without quantizing the grid points. Each bin is further sampled at `SamplingRatio` locations. The value at each sampled point is inferred using bilinear interpolation. The average of the sampled values is returned as the output value of each pooled bin.

Version History

Introduced in R2020b

See Also

`roiMaxPooling2dLayer`

Topics

“Get Started with Object Detection Using Deep Learning”
“Get Started with Instance Segmentation Using Deep Learning”
“Getting Started with Mask R-CNN for Instance Segmentation”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

roiInputLayer

(Not recommended) ROI input layer for Fast R-CNN

Note `ROIInputLayer` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see "Version History".

Description

An ROI input layer inputs images to a Fast R-CNN object detection network.

Creation

Syntax

```
layer = roiInputLayer
layer = roiInputLayer('Name',Name)
```

Description

`layer = roiInputLayer` creates an ROI input layer.

`layer = roiInputLayer('Name',Name)` creates an ROI input layer and sets the optional `Name` property.

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumOutputs` — Number of outputs
`1` (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Connect ROI Input Layer to ROI Max Pooling Layer

Create a `dlnetwork` object.

```
net = dlnetwork;
```

Create an array of layers that includes an ROI input layer and an ROI max pooling layer with output size [4 4]. Add the layers to the network.

```
outputSize = [4 4];
layers = [
    roiInputLayer(Name="roi_input")
    roiMaxPooling2dLayer(outputSize,Name="roi_pool")]
```

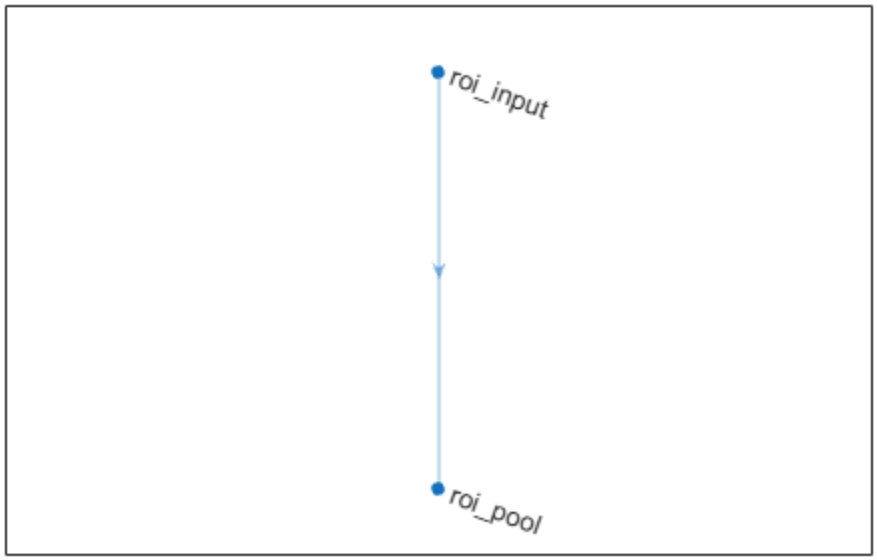
```
layers =
  2×1 Layer array with layers:

     1   'roi_input'   ROI Input         M-by-4 rectangular ROIs
     2   'roi_pool'    ROI Max Pooling   ROI Max Pooling with pooled output size [4 4]
```

```
net = addLayers(net,layers);
```

Connect the output of the ROI input layer with the `"roi"` input of the ROI max pooling layer. Remove the connection between the ROI input layer and the `"in"` input of the ROI max pooling layer.

```
net = connectLayers(net,"roi_input","roi_pool/roi");
net = disconnectLayers(net,"roi_input","roi_pool/in");
plot(net)
```



Version History

Introduced in R2018b

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

`trainFastRCNNObjectDetector` | `roiMaxPooling2dLayer` | `imageInputLayer`

Topics

“Create Fast R-CNN Object Detection Network”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

roiMaxPooling2dLayer

Neural network layer used to output fixed-size feature maps for rectangular ROIs

Description

An ROI max pooling layer outputs fixed size feature maps for every rectangular ROI within the input feature map. Use this layer to create a Fast or Faster R-CNN object detection network.

Given an input feature map of size [*H W C N*], where *C* is the number of channels and *N* is the number of observations, the output feature map size is [*height width C* `sum`(*M*)], where *height* and *width* are the output size. *M* is a vector of length *N* and *M*(*i*) is the number of ROIs associated with the *i*-th input feature map.

There are two inputs to this layer:

- `'in'` — The input feature map that will be cropped
- `'roi'` — A list of ROIs to pool

Use the input names when connecting or disconnecting the ROI max pooling layer to other layers using `connectLayers` or `disconnectLayers` (requires Deep Learning Toolbox).

Creation

Syntax

```
layer = roiMaxPooling2dLayer(outputSize)
layer = roiMaxPooling2dLayer(outputSize,'Name',Name)
```

Description

`layer = roiMaxPooling2dLayer(outputSize)` creates a max pooling layer for ROIs and sets the `OutputSize` property.

`layer = roiMaxPooling2dLayer(outputSize,'Name',Name)` creates a max pooling layer for ROIs and sets the optional `Name` property. To create a network containing an ROI max pooling layer, you must specify a layer name.

Properties

`OutputSize` — Pooled output size
two-element vector of positive integers

Pooled output size, specified as a two-element vector of positive integers of the form `[`*`height width`*`]`.

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
2 (default)

Number of inputs of the layer. This layer has two inputs.

Data Types: `double`

`InputNames` — Input names
`{'in' 'roi'}` (default)

Input names of the layer. This layer has two inputs, named `'in'` and `'roi'`.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Connect ROI Input Layer to ROI Max Pooling Layer

Create a `dlnetwork` object.

```
net = dlnetwork;
```

Create an array of layers that includes an ROI input layer and an ROI max pooling layer with output size [4 4]. Add the layers to the network.

```
outputSize = [4 4];
layers = [
    roiInputLayer(Name="roi_input")
    roiMaxPooling2dLayer(outputSize,Name="roi_pool")]
```

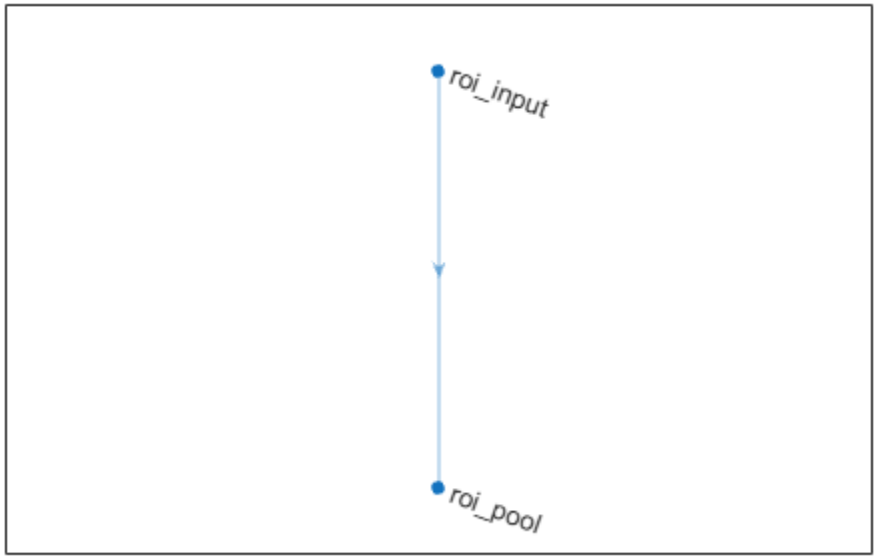
```
layers =
  2×1 Layer array with layers:
```

```
     1   'roi_input'   ROI Input         M-by-4 rectangular ROIs
     2   'roi_pool'    ROI Max Pooling   ROI Max Pooling with pooled output size [4 4]
```

```
net = addLayers(net,layers);
```

Connect the output of the ROI input layer with the `"roi"` input of the ROI max pooling layer. Remove the connection between the ROI input layer and the `"in"` input of the ROI max pooling layer.

```
net = connectLayers(net,"roi_input","roi_pool/roi");
net = disconnectLayers(net,"roi_input","roi_pool/in");
plot(net)
```



Version History

Introduced in R2018b

See Also

`maxPooling2dLayer` | `dlnetwork` | `connectLayers` | `removeLayers`

Topics

“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

rpnClassificationLayer

(Not recommended) Classification layer for region proposal networks (RPNs)

Note `RPNClassificationLayer` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Description

A region proposal network (RPN) classification layer classifies image regions as either *object* or *background* by using a cross entropy loss function. Use this layer to create a Faster R-CNN object detection network.

Creation

Syntax

```
layer = rpnClassificationLayer
layer = rpnClassificationLayer('Name',Name)
```

Description

`layer = rpnClassificationLayer` creates a two-class classification layer for a Faster R-CNN object detection network.

`layer = rpnClassificationLayer('Name',Name)` creates a two-class classification layer and sets the optional `Name` property.

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

Examples

Create Classification Branch of RPN

Create an RPN softmax layer with the name `'rpn_softmax'`.

```
rpnSoftmax = rpnSoftmaxLayer('Name','rpn_softmax')
```

```
rpnSoftmax =
  RPNSoftmaxLayer with properties:

    Name: 'rpn_softmax'
```

Create an RPN classification layer with the name `'rpn_cls'`.

```
rpnClassification = rpnClassificationLayer('Name','rpn_cls')
```

```
rpnClassification =
  RPNClassificationLayer with properties:

    Name: 'rpn_cls'
```

Add the RPN softmax and RPN classification layers to a `Layer` array, to form the classification branch of an RPN.

```
numAnchors = 3;
rpnClassLayers = [
    convolution2dLayer(1,numAnchors*2,'Name','conv1x1_box_cls')
    rpnSoftmax
    rpnClassification
    ]
```

```
rpnClassLayers =
  3×1 Layer array with layers:

     1   'conv1x1_box_cls'   2-D Convolution             6 1×1 convolutions with stride [1  1] ar
     2   'rpn_softmax'       RPN Softmax                 rpn softmax
     3   'rpn_cls'           RPN Classification Output   cross-entropy loss with 'object' and 'ba
```

Version History

Introduced in R2018b

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`trainFasterRCNNObjectDetector` | `classificationLayer` | `rpnSoftmaxLayer`

Topics
“Create Faster R-CNN Object Detection Network”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

rpnSoftmaxLayer

(Not recommended) Softmax layer for region proposal network (RPN)

Note `RPNSoftmaxLayer` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see "Version History".

Description

A region proposal network (RPN) softmax layer applies a softmax activation function to the input. Use this layer to create a Faster R-CNN object detection network.

Creation

Syntax

```
layer = rpnSoftmaxLayer
layer = rpnSoftmaxLayer('Name',Name)
```

Description

`layer = rpnSoftmaxLayer` creates a softmax layer for a Faster R-CNN object detection network.

`layer = rpnSoftmaxLayer('Name',Name)` creates a softmax layer and sets the optional `Name` property.

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
`1` (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Create Classification Branch of RPN

Create an RPN softmax layer with the name `'rpn_softmax'`.

```
rpnSoftmax = rpnSoftmaxLayer('Name','rpn_softmax')
```

```
rpnSoftmax =
  RPNSoftmaxLayer with properties:

    Name: 'rpn_softmax'
```

Create an RPN classification layer with the name `'rpn_cls'`.

```
rpnClassification = rpnClassificationLayer('Name','rpn_cls')
```

```
rpnClassification =
  RPNClassificationLayer with properties:

    Name: 'rpn_cls'
```

Add the RPN softmax and RPN classification layers to a `Layer` array, to form the classification branch of an RPN.

```
numAnchors = 3;
rpnClassLayers = [
    convolution2dLayer(1,numAnchors*2,'Name','conv1x1_box_cls')
    rpnSoftmax
    rpnClassification
    ]
```

```
rpnClassLayers =
  3×1 Layer array with layers:

     1   'conv1x1_box_cls'   2-D Convolution             6 1×1 convolutions with stride [1  1] ar
     2   'rpn_softmax'       RPN Softmax                 rpn softmax
     3   'rpn_cls'           RPN Classification Output   cross-entropy loss with 'object' and 'ba
```

Version History

Introduced in R2018b

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

`trainFasterRCNNObjectDetector` | `rpnClassificationLayer` | `softmaxLayer`

Topics

“Create Faster R-CNN Object Detection Network”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

ssdMergeLayer

Create SSD merge layer for object detection

Description

An SSD merge layer merges the outputs of feature maps for subsequent regression and classification loss computation. Use the merged feature maps to compute focal loss for classification and smooth L1 loss for regression.

Creation

Syntax

```
layer = ssdMergeLayer(NumChannels,NumInputs)
layer = ssdMergeLayer(NumChannels,NumInputs,'Name',Name)
```

Description

`layer = ssdMergeLayer(NumChannels,NumInputs)` creates an SSD merge layer, specifying the `NumChannels` and `NumInputs` properties as the number of feature map channels and number of inputs to merge, respectively.

`layer = ssdMergeLayer(NumChannels,NumInputs,'Name',Name)` creates an SSD merge layer and sets the optional `Name` property using a name-value pair. Enclose the property name in single quotes.

For example, `ssdMergeLayer(4,6,'Name','sm1')` creates an SSD merge layer with the name `'sm1'` that merges four dimensions and six feature maps.

Properties

`NumChannels` — Number of feature map channels
positive integer

Number of feature map channels per anchor box, specified as a positive integer. For regression problems, `NumChannels` is 4 (for x, y, w, and h). For classification problems, `NumChannels` equals the number of classes that are being detected plus one for the background class.

`NumInputs` — Number of inputs to merge
positive integer

Number of inputs to merge, specified as a positive integer.

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

Examples

Create SSD Merge Layer for Regression

Specify the number of feature map channels per anchor box. Regression merges four channels.

```
numChannels = 4;
```

Specify the number of inputs to merge.

```
numInputs = 6;
```

Create an SSD merge layer named `regressionMergeLayer`.

```
layer = ssdMergeLayer(numChannels,numInputs,'Name','regressionMergeLayer')
```

```
layer =
  SSDMergeLayer with properties:

           Name: 'regressionMergeLayer'
    NumChannels: 4
      NumInputs: 6
```

Setup SSD Merge Layer For Classification

Specify the number of classes, e.g. person and vehicle.

```
numClasses = 2
```

```
numClasses =
2
```

Specify the number of inputs.

```
numInputs = 6
```

```
numInputs =
6
```

Create an SSD merge layer for classification. The number of channels equals the number of classes + 1 for the background class.

```
numChannels = numClasses + 1;
clsMergeLayer = ssdMergeLayer(numChannels, numInputs,'Name','clsMergeLayer')
```

```
clsMergeLayer =
  SSDMergeLayer with properties:

           Name: 'clsMergeLayer'
    NumChannels: 3
      NumInputs: 6
```

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), or the NVIDIA TensorRT high performance inference library.

See Also

`trainNetwork` | `trainSSDObjectDetector`

Topics

“Get Started with Object Detection Using Deep Learning”
“Getting Started with SSD Multibox Detection”
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)

yolov2OutputLayer

(To be removed) Create output layer for YOLO v2 object detection network

Note `yolov2OutputLayer` will be removed in a future release. Create a YOLO v2 object detection network using the `yolov2ObjectDetector` object instead. For more information, see "Version History".

Description

The `yolov2OutputLayer` function creates a `YOLOv2OutputLayer` object, which represents the output layer for you only look once version 2 (YOLO v2) object detection network. The output layer provides the refined bounding box locations of the target objects.

Creation

Syntax

```
layer = yolov2OutputLayer(anchorBoxes)
layer = yolov2OutputLayer(anchorBoxes,Name,Value)
```

Description

`layer = yolov2OutputLayer(anchorBoxes)` creates a `YOLOv2OutputLayer` object, `layer`, which represents the output layer for YOLO v2 object detection network. The layer outputs the refined bounding box locations that are predicted using a predefined set of anchor boxes specified at the input.

`layer = yolov2OutputLayer(anchorBoxes,Name,Value)` sets the additional properties using name-value pairs and the input from the preceding syntax. Enclose each property name in single quotes. For example, `yolov2OutputLayer('Name','yolo_Out')` creates an output layer with the name 'yolo_Out'.

Input Arguments

`anchorBoxes` — Set of anchor boxes
M-by-2 matrix

Set of anchor boxes, specified as an *M*-by-2 matrix, where each row is of the form [*height width*]. The matrix defines the height and the width of *M* number of anchor boxes. This input sets the `AnchorBoxes` property of the output layer. You can use the clustering approach for estimating anchor boxes from the training data. For more information, see "Estimate Anchor Boxes from Training Data".

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`LossFunction` — Loss function
`'mean-squared-error'` (default)

This property is read-only.

Loss function, set as `'mean-squared-error'`. For more information about the loss function, see “Loss Function for Bounding Box Refinement” on page 2-657.

`AnchorBoxes` — Set of anchor boxes
M-by-2 matrix

This property is read-only.

Set of anchor boxes used for training, specified as a *M*-by-2 matrix defining the width and the height of *M* number of anchor boxes. This property is set by the input `anchorBoxes`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`LossFactors` — Weights in the loss function
`[5 1 1 1]` (default) | 1-by-4 vector

This property is read-only.

Weights in the loss function, specified as a 1-by-4 vector of form [K_1 K_2 K_3 K_4]. Weights increase the stability of the network model by penalizing incorrect bounding box predictions and false classifications. For more information about the weights in loss the function, see “Loss Function for Bounding Box Refinement” on page 2-657.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Classes` — Classes of the output layer
'auto' (default) | categorical vector | string array | cell array of character vectors

Classes of the output layer, specified as a categorical vector, string array, cell array of character vectors, or `'auto'`. Use this name-value pair to specify the names of the object classes in the input training data.

If the value is set to `'auto'`, then the software automatically sets the classes at training time. If you specify the string array or cell array of character vectors `str`, then the software sets the classes of the output layer to `categorical(str)`. The default value is `'auto'`.

Data Types: `char` | `string` | `cell` | `categorical`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

Examples

Create YOLO v2 Output Layer

Create a YOLO v2 output layer with two anchor boxes.

Define the height and the width of the anchor boxes.

```
anchorBoxes = [16 16;32 32];
```

Specify the names of the object classes in the training data.

```
classNames = {'Vehicle','Person'};
```

Generate a YOLO v2 output layer with the name `"yolo_Out"`.

```
layer = yolov2OutputLayer(anchorBoxes,'Name','yolo_Out','Classes',classNames);
```

Inspect the properties of the YOLO v2 output layer.

```
layer
```

```
layer =
  YOLOv2OutputLayer with properties:

            Name: 'yolo_Out'

   Hyperparameters
         Classes: [2x1 categorical]
    LossFunction: 'mean-squared-error'
     AnchorBoxes: [2x2 double]
     LossFactors: [5 1 1 1]
```

You can read the values for `Classes` property by using dot notation `layer.Classes`. The function stores the class names as a categorical array.

```
layer.Classes
```

```
ans = 2x1 categorical
     Vehicle
```

```
Person
```

More About

Loss Function for Bounding Box Refinement

During training, the output layer of YOLO v2 network predicts refined bounding box locations by optimizing the mean squared error loss between predicted bounding boxes and the ground truth. The loss function is defined as

$$
\begin{aligned}
&K_1 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ K_1 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \\
&+ K_2 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \\
&+ K_3 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \\
&+ K_4 \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2
\end{aligned}
$$

where:

- S is the number of grid cells
- B is the number of bounding boxes in each grid cell.
- 1_{ij}^{obj} is 1 if the *jth* bounding box in grid cell i is responsible for detecting the object. Otherwise it is set to 0. A grid cell i is responsible for detecting the object, if the overlap between the ground truth and a bounding box in that grid cell is greater than or equal to 0.6.
- 1_{ij}^{noobj} is 1 if the *jth* bounding box in grid cell i does not contain any object. Otherwise it is set to 0.
- 1_{i}^{obj} is 1 if an object is detected in grid cell i. Otherwise it is set to 0.
- K_1, K_2, K_3, and K_4 are the weights. To adjust the weights, modify the `LossFactors` property.

The loss function can be split into three parts:

- Localization loss

  The first and second terms in the loss function comprise the localization loss. It measures error between the predicted bounding box and the ground truth. The parameters for computing the localization loss include the position, size of the predicted bounding box, and the ground truth. The parameters are defined as follows.

  - (x_i, y_i), is the center of the *jth* bounding box relative to grid cell i.
  - $(\hat{x}_i, \hat{y}_i)$, is the center of the ground truth relative to grid cell i.

- w_i and h_i is the width and the height of the *jth* bounding box in grid cell i, respectively. The size of the predicted bounding box is specified relative to the input image size.
- $\widehat{w}_i$ and $\widehat{h}_i$ is the width and the height of the ground truth in grid cell i, respectively.
- K_1 is the weight for localization loss. Increase this value to increase the weightage for bounding box prediction errors.

- Confidence loss

  The third and fourth terms in the loss function comprise the confidence loss. The third term measures the objectness (confidence score) error when an object is detected in the *jth* bounding box of grid cell i. The fourth term measures the objectness error when no object is detected in the *jth* bounding box of grid cell i. The parameters for computing the confidence loss are defined as follows.

  - C_i is the confidence score of the *jth* bounding box in grid cell i.
  - $\hat{C}_i$ is the confidence score of the ground truth in grid cell i.
  - K_2 is the weight for objectness error, when an object is detected in the predicted bounding box. You can adjust the value of K_2 to weigh confidence scores from grid cells that contain objects.
  - K_3 is the weight for objectness error, when an object is not detected in the predicted bounding box. You can adjust the value of K_3 to weigh confidence scores from grid cells that do not contain objects.

  The confidence loss can cause the training to diverge when the number of grid cells that do not contain objects is more than the number of grid cells that contain objects. To remedy this, increase the value for K_2 and decrease the value for K_3.

- Classification loss

  The fifth term in the loss function comprises the classification loss. For example, suppose that an object is detected in the predicted bounding box contained in grid cell i. Then, the classification loss measures the squared error between the class conditional probabilities for each class in grid cell i. The parameters for computing the classification loss are defined as follows.

  - $p_i(c)$ is the estimated conditional class probability for object class c in grid cell i.
  - $\widehat{p}_i(c)$ is the actual conditional class probability for object class c in grid cell i.
  - K_4 is the weight for classification error when an object is detected in the grid cell. Increase this value to increase the weightage for classification loss.

Tips

To improve prediction accuracy, you can:

- Train the network with more number of images. You can expand the training dataset through data augmentation. For information on how to apply data augmentation for training dataset, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).
- Perform multiscale training by using the `trainYOLOv2ObjectDetector` function. To do so, specify the `TrainingImageSize` argument of `trainYOLOv2ObjectDetector` function for training the network.
- Choose anchor boxes appropriate to the dataset for training the network. You can use the `estimateAnchorBoxes` function to compute anchor boxes directly from the training data.

Version History

Introduced in R2019a

R2025a: To be removed

Warns starting in R2025a

The `yolov2OutputLayer` object will be removed in a future release. When you call the `yolov2OutputLayer` object, it issues a warning that it will be removed. Create a YOLO v2 object detection network by using the `yolov2ObjectDetector` object instead, using these steps:

- Define your network as a `dlnetwork` object. You can load a pretrained feature extraction network by using the `imagePretrainedNetwork` function. Alternatively, you can use functions such as `addLayers` and `connectLayers` to build the network. Do not include output layers in the network.
- Create a `yolov2ObjectDetector` object using the `dlnetwork` as the custom network. You can specify the anchor boxes, class names, and loss factors using the `AnchorBoxes`, `ClassNames`, and `LossFactors` name-value arguments, respectively.

R2024b: To be removed

Not recommended starting in R2024b

The `yolov2OutputLayer` object will be removed in a future release. Create a YOLO v2 object detection network using the `yolov2ObjectDetector` object instead, using these steps:

- Define your network as a `dlnetwork` object. You can use functions such as `addLayers` and `connectLayers` to build the network. Do not include output layers in the network.
- Create a `yolov2ObjectDetector` using the `dlnetwork` as the custom network. You can specify the anchor boxes, class names, and loss factors using the `AnchorBoxes`, `ClassNames`, and `LossFactors` name-value arguments, respectively.

References

[1] Joseph. R, S. K. Divvala, R. B. Girshick, and F. Ali. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788. Las Vegas, NV: CVPR, 2016.

[2] Joseph. R and F. Ali. "YOLO 9000: Better, Faster, Stronger." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525. Honolulu, HI: CVPR, 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.

See Also

`yolov2ObjectDetector` | `trainYOLOv2ObjectDetector` | `spaceToDepthLayer`

Topics

“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using YOLO v2”
“Getting Started with YOLO v2”
“Anchor Boxes for Object Detection”

yolov2TransformLayer

Create transform layer for YOLO v2 object detection network

Description

The `yolov2TransformLayer` function creates a `YOLOv2TransformLayer` object, which represents the transform layer for you only look once version 2 (YOLO v2) object detection network. The transform layer in YOLO v2 object detection network improves the stability of the network by constraining the location predictions. The transform layer extracts activations of the last convolutional layer and transforms the bounding box predictions to fall within the bounds of the ground truth.

Creation

Syntax

```
layer = yolov2TransformLayer(numAnchorBoxes)
layer = yolov2TransformLayer(numAnchorBoxes,Name,Value)
```

Description

`layer = yolov2TransformLayer(numAnchorBoxes)` creates the transform layer for YOLO v2 object detection network.

`layer = yolov2TransformLayer(numAnchorBoxes,Name,Value)` sets the `Name` property using a name-value pair. Enclose the property name in single quotes. For example, `yolov2TransformLayer('Name','yolo_Transform')` creates a transform layer with the name 'yolo_Transform'.

Input Arguments

`numAnchorBoxes` — Number of anchor boxes
positive integer

Number of anchor boxes used for training, specified as a positive integer. This input sets the `NumAnchorBoxes` property of the transform layer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Properties

`Name` — Layer name
`''` (default) | character vector

Layer name, specified as a character vector. For `Layer` array input, the `trainnet` and `dlnetwork` functions automatically assign names to unnamed layers.

Data Types: `char`

`NumAnchorBoxes` — Number of anchor boxes
positive integer

This property is read-only.

Number of anchor boxes used for training, specified as a positive integer. This property is set by the input `numAnchorBoxes`.

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs to the layer, stored as `1`. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names, stored as `{'in'}`. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
`1` (default)

This property is read-only.

Number of outputs from the layer, stored as `1`. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names, stored as `{'out'}`. This layer has a single output only.

Data Types: `cell`

Examples

Create YOLO v2 Transform Layer

Specify the number of anchor boxes.

```
numAnchorBoxes = 5;
```

Create a YOLO v2 transform layer with the name "yolo_Transform".

```
layer = yolov2TransformLayer(numAnchorBoxes,'Name','yolo_Transform');
```

Inspect the properties of the YOLO v2 transform layer.

```
layer
```

```
layer =
  YOLOv2TransformLayer with properties:

             Name: 'yolo_Transform'
    NumAnchorBoxes: 5

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

Algorithms

Layer Input and Output Formats

Most layers in a layer array or layer graph pass data to subsequent layers as formatted `dlarray` objects. The format of a `dlarray` object is a string of characters in which each character describes the corresponding dimension of the data. The format consists of one or more of these characters:

- `"S"` — Spatial
- `"C"` — Channel
- `"B"` — Batch
- `"T"` — Time
- `"U"` — Unspecified

For example, you can describe 2-D image data that is represented as a 4-D array, where the first two dimensions correspond to the spatial dimensions of the images, the third dimension corresponds to the channels of the images, and the fourth dimension corresponds to the batch dimension, as having the format `"SSCB"` (spatial, spatial, channel, batch).

You can interact with these `dlarray` objects in automatic differentiation workflows, such as those for:

- developing a custom layer
- using a `functionLayer` object
- using the `forward` and `predict` functions with `dlnetwork` objects

This table shows the supported input formats of `yolov2TransformLayer` objects and the corresponding output format. If the software passes the output of the layer to a custom layer that does not inherit from the `nnet.layer.Formattable` class, or to a `FunctionLayer` object with the `Formattable` property set to `0` (`false`), then the layer receives an unformatted `dlarray` object with dimensions ordered according to the formats in this table. The formats listed here are only a subset of the formats that the layer supports. The layer might support additional formats, such as formats with additional `"S"` (spatial) or `"U"` (unspecified) dimensions.

Input Format	Output Format
`"SSCB"` (spatial, spatial, channel, batch)	`"SSCB"` (spatial, spatial, channel, batch)

Version History

Introduced in R2019a

R2024b: Support for formatted `dlarray` data

`yolov2TransformLayer` can accept formatted `dlarray` data with the format `"SSCB"` (spatial, spatial, channel, batch). When you pass formatted `dlarray` data to the layer, the layer returns data of the same format. For more information, see “Layer Input and Output Formats” on page 2-663.

References

[1] Joseph. R, S. K. Divvala, R. B. Girshick, and F. Ali. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788. Las Vegas, NV: CVPR, 2016.

[2] Joseph. R and F. Ali. "YOLO 9000: Better, Faster, Stronger." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525. Honolulu, HI: CVPR, 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`yolov2ObjectDetector` | `trainYOLOv2ObjectDetector` | `spaceToDepthLayer`

Topics

“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using YOLO v2”

“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with YOLO v2”
“Anchor Boxes for Object Detection”

pixelLabelDatastore

Datastore for pixel label data

Description

You can use a `PixelLabelDatastore` object to read pixel label data for semantic segmentation.

To read pixel label data from a `PixelLabelDatastore`, use the `read` function. This function returns a categorical array that contains a categorical label assigned to every (*i*,*j*) pixel location or (*i*,*j*,*k*) voxel location.

Creation

Create a `PixelLabelDatastore` object using the `pixelLabelDatastore` function described here. Once the object is created, you can use functions that access and manage the data. You can use dot notation to modify the `ReadSize` and `ReadFcn` properties.

Syntax

```
pxds = pixelLabelDatastore(gTruth)
pxds = pixelLabelDatastore(location,classNames,pixelLabelIDs)
pxds = pixelLabelDatastore( ___ ,Name=Value)
```

Description

`pxds = pixelLabelDatastore(gTruth)` creates a `PixelLabelDatastore` object from a `groundTruth` object or an array of `groundTruth` objects.

`pxds = pixelLabelDatastore(location,classNames,pixelLabelIDs)` creates a `PixelLabelDatastore` object from image files that store pixel label data, in the folder or files specified by `location`. The function creates the object using pixel IDs that map image pixel label values to class names.

`pxds = pixelLabelDatastore( ___ ,Name=Value)` creates a `PixelLabelDatastore` object using name-value arguments to set one or both of the `ReadSize` or `AlternateFileSystemRoots` properties. For image file input, you can also use name-value pair arguments to set the `ReadFcn` property or to specify options on page 2-668 about files to include in the datastore. You can specify multiple name-value pairs. Enclose each property name in quotes.

For example, `pixelLabelDatastore(loc,names,ids,ReadSize=8,FileExtensions="png")` creates a pixel label datastore that includes only PNG images and that reads eight images during each call to the `read` function.

Input Arguments

`gTruth` — Ground truth data
`groundTruth` object | array of `groundTruth` objects

Ground truth data, specified as a `groundTruth` object or as an array of `groundTruth` objects. Each `groundTruth` object contains information about the data source, the list of label definitions, and all marked labels for a set of ground truth labels.

`location` — Folder or image file names

character vector | cell array of character vectors | string array

Folder or image file names, specified as a character vector, string array, or cell array of character vectors. Images must contain `uint8` data.

`location`	Description
character vector, string scalar	Name of a folder. Files within subfolders of the specified folder are not automatically included in the datastore. The datastore includes only images with supported file formats and ignores any other format. See a list of supported file formats by using the `imformats` function.
cell array of character vectors, array of strings	File names of multiple images.

`pixelLabelDatastore` expands the file names and stores the full file paths in the `Files` property.

You can use the wildcard character (*) when specifying `location`. This character indicates that all matching files or all files in the matching folders are included in the datastore.

If the files are not in the current folder, then you must include the full or relative path.

If the files are not available locally, then the full path of the files or folders must be an internationalized resource identifier (IRI), such as `hdfs://`*`hostname`*`:`*`portnumber`*`/`*`path_to_file`*. For information on using a `datastore` with Amazon S3™ and HDFS™, see “Work with Remote Data”.

Example: `"file1.jpg"`

Example: `"../dir/data/file1.png"`

Example: `["C:\dir\data\file1.tif","C:\dir\data\file2.tif"]`

Example: `'C:\dir\data\*.jpg'`

Data Types: `char` | `string`

`classNames` — Class names

cell array of character vectors | cell array of strings

Class names, specified as a cell array of strings or character vectors. `pixelLabelDatastore` converts the names to a cell array and stores the names in the `ClassNames` property.

Example: `["sky" "grass" "building" "sidewalk"]`

`pixelLabelIDs` — IDs to relate pixel labels to class names

vector | *m*-by-3 matrix | cell array of column vectors | cell array of *m*-by-3 matrices

IDs to map pixel labels to `ClassNames`, specified as a vector, an *m*-by-3 matrix, a cell array of column vectors, or a cell array of *m*-by-3 matrices. Values must be integers in the range [0, 255]. *m*-by-3 matrices are only supported for RGB images.

Format	Description
vector	The length of the vector must equal the number of class names. Values must be unique.
m-by-3 matrix	*m* corresponds to the number of class names. Each row contains a 3-element vector representing the RGB pixel value to associate with each class name. Vectors must be unique. Use this format for pixel label data stored as RGB images.
cell array of column vectors cell array of *m*-by-3 matrices	Use a cell array to map multiple pixel label IDs to one class name.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `pixelLabelDatastore(IncludeSubfolders=true)` specifies to include all files and subfolders within each folder.

`IncludeSubfolders` — Include subfolders

`false` (default) | `true`

Include subfolders, specified as `false` (`0`), or `true` (`1`). Specify `true` to include all files and subfolders within each folder or `false` to include only the files within each folder.

The `IncludeSubfolders` argument is not supported when you specify ground truth data `gTruth`.

`FileExtensions` — File extensions

character vector | cell array of character vectors | string scalar | string array

File extensions of files to include in the datastore, specified as a character vector, cell array of character vectors, string scalar, or string array. By default, the datastore includes files with all extensions supported by the `imformats` function.

The `FileExtensions` argument is not supported when you specify ground truth data `gTruth`.

Example: `["png" "gif" "bmp"]`

Properties

`Files` — Files included in datastore

character vector | cell array of character vectors

This property is read-only.

Files included in the datastore, specified as a character vector or cell array of character vectors. Each character vector is a full path to a file. When you create a `PixelLabelDatastore` object, use the `location` argument to set this property.

`ClassNames` — Class names

cell array of character vectors

This property is read-only.

Class names, specified as a cell array of character vectors.

`ReadSize` — Maximum number of image files

1 (default) | positive integer

Maximum number of image files to read in each call to the `read` function, specified as a positive integer.

`AlternateFileSystemRoots` — Alternate file system root paths

string vector | cell array

Alternate file system root paths, specified as the name-value argument consisting of `"AlternateFileSystemRoots"` and a string vector or a cell array. Use `"AlternateFileSystemRoots"` when you create a datastore on a local machine, but need to access and process the data on another machine (possibly of a different operating system). Also, when processing data using the Parallel Computing Toolbox and the MATLAB Parallel Server™, and the data is stored on your local machines with a copy of the data available on different platform cloud or cluster machines, you must use `"AlternateFileSystemRoots"` to associate the root paths.

- To associate a set of root paths that are equivalent to one another, specify `"AlternateFileSystemRoots"` as a string vector. For example,

  ```
  ["Z:\datasets","/mynetwork/datasets"]
  ```

- To associate multiple sets of root paths that are equivalent for the datastore, specify `"AlternateFileSystemRoots"` as a cell array containing multiple rows where each row represents a set of equivalent root paths. Specify each row in the cell array as either a string vector or a cell array of character vectors. For example:

  - Specify `"AlternateFileSystemRoots"` as a cell array of string vectors.

    ```
    {["Z:\datasets", "/mynetwork/datasets"];...
     ["Y:\datasets", "/mynetwork2/datasets","S:\datasets"]}
    ```

  - Alternatively, specify `"AlternateFileSystemRoots"` as a cell array of cell array of character vectors.

    ```
    {{'Z:\datasets','/mynetwork/datasets'};...
     {'Y:\datasets', '/mynetwork2/datasets','S:\datasets'}}
    ```

The value of `"AlternateFileSystemRoots"` must satisfy these conditions:

- Contains one or more rows, where each row specifies a set of equivalent root paths.
- Each row specifies multiple root paths and each root path must contain at least two characters.
- Root paths are unique and are not subfolders of one another.
- Contains at least one root path entry that points to the location of the files.

For more information, see “Set Up Datastore for Processing on Different Machines or Clusters”.

Example: `["Z:\datasets","/mynetwork/datasets"]`

Data Types: `string` | `cell`

ReadFcn — Function that reads pixel labeled image data

`@readDatastoreImage` (default) | function handle

Function that reads pixel labeled image data, specified as a function handle. The custom read function must take an image file name as input, and then output the corresponding pixel labeled data as a categorical array. For example, if `customreader` is the specified function to read the image data, then it must have a signature similar to the following:

```
function C = customreader(filename)
...
end
```

If the read function has more than one output argument, then only the first one is used. The rest are ignored.

The `ReadFcn` property is not supported when you specify ground truth data `gTruth`.

Example: `@customreader`

Object Functions

combine	Combine data from multiple datastores
countEachLabel	Count occurrence of pixel or box labels
hasdata	Determine if data is available to read from label datastore
numpartitions	Number of partitions for label datastore
partition	Partition label datastore
preview	Read first row of data in datastore
read	Read data from label datastore
readall	Read all data in label datastore
readimage	Read specified pixel label data file
reset	Reset label datastore to initial state
shuffle	Return shuffled version of label datastore
transform	Transform datastore
subset	Create subset of datastore or FileSet
isPartitionable	Determine whether datastore is partitionable
isShuffleable	Determine whether datastore is shuffleable

Examples

Read and Display Pixel Label Data

Overlay pixel label data on an image.

Set the location of the image and pixel label data.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
imDir = fullfile(dataDir,'building');
pxDir = fullfile(dataDir,'buildingPixelLabels');
```

Create an image datastore.

```
imds = imageDatastore(imDir);
```

Create a pixel label datastore.

```
classNames = ["sky" "grass" "building" "sidewalk"];
pixelLabelID = [1 2 3 4];
pxds = pixelLabelDatastore(pxDir,classNames,pixelLabelID);
```

Read the image and pixel label data. `read(pxds)` returns a categorical matrix, `C`. The element C(i,j) in the matrix is the categorical label assigned to the pixel at the location I(i,j).

```
I = read(imds);
C = read(pxds);
```

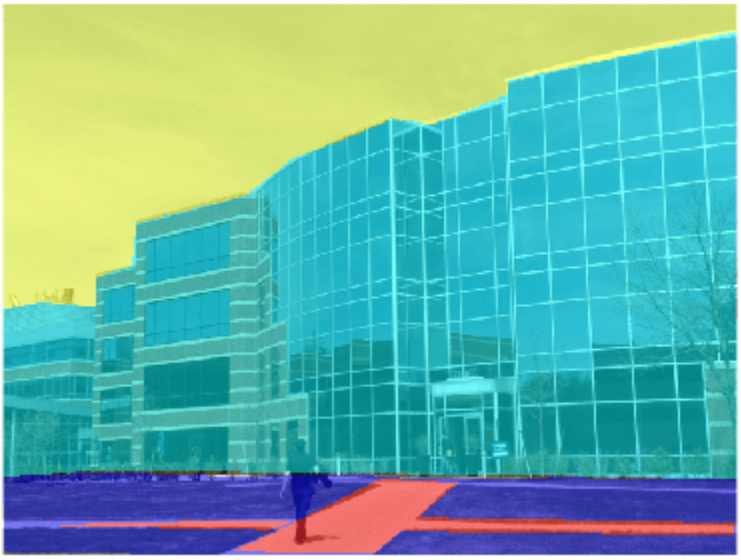
Display the label categories in `C`.

```
categories(C{1})
```

```
ans = 4×1 cell
    {'sky'     }
    {'grass'   }
    {'building'}
    {'sidewalk'}
```

Overlay and display the pixel label data onto the image.

```
B = labeloverlay(I,C{1});
figure
imshow(B)
```



Read and Display 3-D Pixel Label Data

Specify the location of 3-D volume and pixel label data. This data is a labeled 3-D MRI scan of a brain.

```
dataDir = fullfile(toolboxdir("images"),"imdata");
imDir = fullfile(dataDir,"BrainMRILabeled","images");
pxDir = fullfile(dataDir,"BrainMRILabeled","labels");
```

Specify a custom read function. This example specifies a function called `samplePXDSMatReader` (defined at the end of the example) that read 3-D image data from MAT image files.

```
matReader = @samplePXDSMatReader;
```

Create an image datastore.

```
imds = imageDatastore(imDir,"FileExtensions",".mat","ReadFcn",matReader);
```

Create a pixel label datastore.

```
classNames = ["edema","nonEnhancingTumor","enhancingTumour"];
pixelLabelID = [1 2 3];
pxds = pixelLabelDatastore(pxDir,classNames,pixelLabelID,"FileExtensions",".mat","ReadFcn",matRea
```

Read volume and pixel label data. `L` is a categorical matrix, where `L(i,j,k)` is the categorical label assigned to `V(i,j,k)`.

```
V = read(imds);
L = read(pxds);
```

Display the label categories.

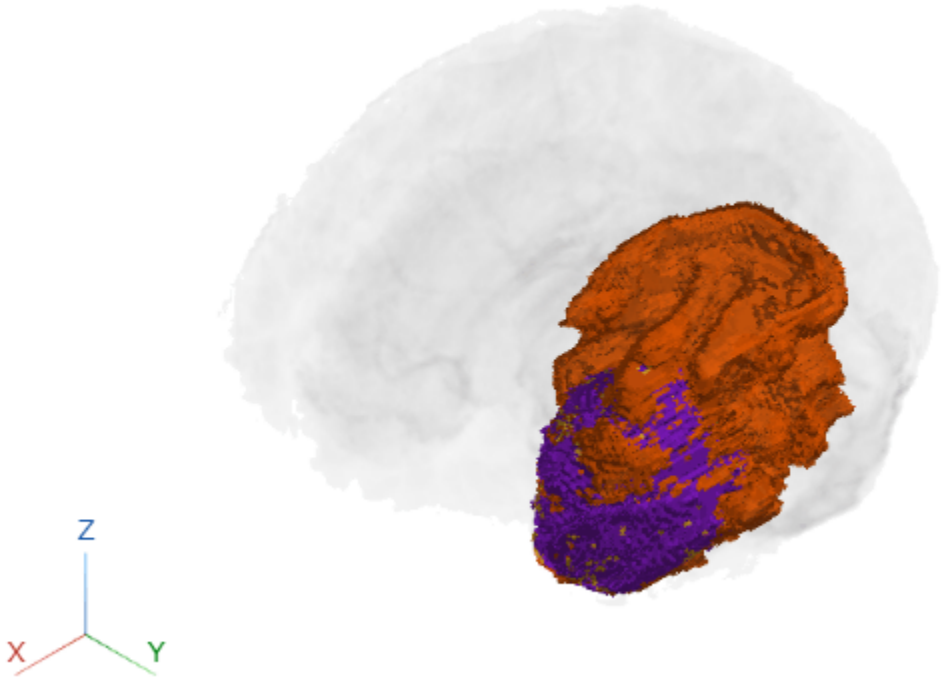
```
C = categories(L{1})
```

```
C = 3×1 cell array
    "'edema'"
    "'nonEnhancingTumor'"
    "'enhancingTumour'"
```

Visualize the result using the `viewer3d` and `volshow` functions. Display the labels by specifying the `OverlayData` and `OverlayAlpha` name-value arguments.

```
viewer = viewer3d(BackgroundColor="white",BackgroundGradient="off",CameraZoom=2);

h = volshow(V,Parent=viewer, ...
    RenderingStyle="GradientOpacity", ...
    GradientOpacityValue=0.8, ...
    Alphamap=linspace(0,0.2,256), ...
    OverlayData=L{1}, ...
    OverlayAlpha=0.8);
```



This example defines a helper function, `samplePXDSMatReader`, to read the 3-D image data from the image files. This function loads a MAT file and returns the first variable saved in that file.

```
function data = samplePXDSMatReader(filename)
    inp = load(filename);
    f = fields(inp);
    data = inp.(f{1});
end
```

Tips

- A `pixelLabelDatastore` stores files in lexicographical order. For example, if you have twelve files named `"file1.jpg"`, `"file2.jpg"`, ... , `"file11.jpg"`, and `"file12.jpg"`, then the files are stored in this order:

  ```
  "file1.jpg"
  "file10.jpg"
  "file11.jpg"
  "file12.jpg"
  "file2.jpg"
  "file3.jpg"
  ...
  "file9.jpg"
  ```

  In contrast, an `imageDatastore` stores files in the order they are added to the datastore. If you simultaneously read a ground truth image and pixel label data, then you may encounter a

mismatch between the images and the labels. If this occurs, then rename the pixel label files so that they have the correct order. For example, rename `"file1.jpg"`, ... , `"file9.jpg"` to `"file01.jpg"`, ..., `"file09.jpg"`.

- To extract semantic segmentation data from a `groundTruth` object generated by the Video Labeler app, use the `pixelLabelTrainingData` function.

Version History

Introduced in R2017b

See Also

Apps

Image Labeler | Video Labeler | Medical Image Labeler

Functions

`evaluateSemanticSegmentation` | `semanticseg`

Objects

`ImageDatastore` | `groundTruth` | `semanticSegmentationMetrics` | `randomPatchExtractionDatastore`

Topics

“3-D Brain Tumor Segmentation Using Deep Learning”
“Get Started with Semantic Segmentation Using Deep Learning”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Training Data for Object Detection and Semantic Segmentation”

KAZEPoints

Object for storing KAZE interest points

Description

This object provides the ability to pass data between the `detectKAZEFeatures` and `extractFeatures` functions. You can also use this object to manipulate and plot the data returned by these functions. Use the object to fill points interactively, where you might want to mix a non-KAZE interest point detector with a KAZE descriptor.

Creation

Syntax

```
points = KAZEPoints(location)
points = KAZEPoints(location,Name=Value)
```

Description

`points = KAZEPoints(location)` constructs a `KAZEPoints` object from an *M*-by-2 array [*x y*] of location coordinates.

The scalar KAZEPoints object contains many points. Therefore `numel(KAZEPoints)` always returns `1`. This value can be different than the result of `length(KAZEPoints)`, which returns the true number of points contained in the object.

`points = KAZEPoints(location,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `points = KAZEPoints(Metric=10)` sets the strength of detected KAZE features to `10`.

Input Arguments

`location` — Location coordinates

M-by-2 array

Location of points, specified as an *M*-by-2 array of [*x y*] coordinates.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `points = KAZEPoints(Metric=10)` sets the strength of detected KAZE features to `10`.

`Scale` — Scale

`1.6` (default) | scalar | *M*-element vector

Scale, specified as a scalar or an or *M*-element vector. The scale sets the size at which the interest points are detected and must be a value greater than or equal to `1.6`.

`Metric` — Strength of detected feature
0.0 (default) | numeric scalar | vector

Strength of detected features, specified as a numeric scalar. The KAZE algorithm uses a determinant of an approximated Hessian.

`Orientation` — Orientation
0.0 (default) | radians | scalar | *M*-element vector

Orientation of the detected feature, specified as an angle in radians. The angle is measured from the *x*-axis with the origin set by the `location` input. `Orientation` property can be useful for visualizations. The `extractFeatures` function sets this property. Do not set it manually.

Properties

`Location` — Location of points
`[]` (default) | *M*-by-2 matrix

This property is read-only.

Location of points, specified as an *M*-by-2 matrix. Each row is of the form [*x* *y*] and represents the location of a point. *M* is the number of points. You cannot set this property, use the `location` input argument instead.

`Count` — Number of points
`0` (default) | nonnegative integer

This property is read-only.

Number of points held by the object, specified as a nonnegative integer.

Object Functions

isempty	Determine if points object is empty
length	Number of stored points
plot	Plot detected feature points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation

Examples

Detect KAZE Features

Detect KAZE features and display 10 strongest points.

Read an image.

```
I = imread('cameraman.tif');
```

Detect KAZE features in the image.

```
points = detectKAZEFeatures(I);
```

Select the 10 strongest points.

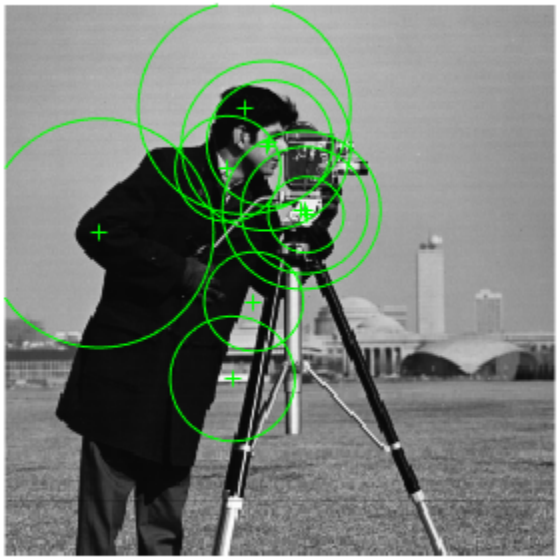
```
strongest = selectStrongest(points,10);
```

Display the selected points.

```
imshow(I);
hold on;
```

Display the location and scale. The size of the circles displayed relate to the scale.

```
plot(strongest);
hold on;
```



Display the [x y] coordinates for the strongest points in the MATLAB Command Window.

```
strongest.Location
```

```
ans = 10×2 single matrix

  138.5041   95.8063
  139.9253   95.8802
  111.8975   48.2950
  106.4036  174.1800
   44.3964  106.4899
  122.0368   65.9064
  116.2702  138.2877
  123.6542   64.7193
  104.2719   76.5821
  140.6228   97.9271
```

Detect KAZE Features and Display Specific Points

Read an image into the workspace.

```
I = imread("cameraman.tif");
```

Detect KAZE features in the image.

```
points = detectKAZEFeatures(I);
```

Select and display the last 5 points detected.

```
imshow(I);
hold on;
plot(points(end-4:end));
hold off;
```



Tips

- Although `KAZEPoints` may hold many points, it is a scalar object. Therefore, `numel(points)` always returns `1`. This value may differ from the resulting value of `length(points)`, which returns the true number of points held by the object.

Version History

Introduced in R2017b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To index locations with this object, use the syntax: `points.Location(idx,:)`, for `points` object. See visionRecovertformCodeGeneration_kernel.m, which is used in the “Introduction to Code Generation with Feature Matching and Registration” example.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`detectKAZEFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `matchFeatures` | `extractFeatures` | `detectSURFFeatures`

Objects

`MSERRegions` | `SURFPoints` | `cornerPoints` | `BRISKPoints`

binaryFeatures

Object for storing binary feature vectors

Description

This object provides the ability to pass data between the `extractFeatures` and `matchFeatures` functions. It can also be used to manipulate and plot the data returned by `extractFeatures`.

Creation

Syntax

```
features = binaryFeatures(featureVectors)
```

Description

`features = binaryFeatures(featureVectors)` constructs a `binaryFeatures` object from the *M*-by-*N* input matrix, `featureVectors`. This matrix contains *M* feature vectors stored in *N* uint8 containers.

Input Arguments

`featureVectors` — Input feature vectors
M-by-*N* matrix

Input feature vectors, specified as an *M*-by-*N* input matrix. This matrix contains *M* binary feature vectors stored in *N* uint8 containers.

Read-only Properties

`Features` — Feature vectors
M-by-*N* matrix

Feature vectors, saved as an *M*-by-*N* input matrix. This matrix contains *M* binary feature vectors stored in *N* uint8 containers.

`NumBits` — Number of bits per feature vector
integer

Number of bits per feature vector, saved as an integer. `NumBits` equals the number of uint8 feature vector containers times 8.

`NumFeatures` — Number of feature vectors
integer

Number of feature vectors contained in the `binaryFeatures` object, saved as a positive integer.

Examples

Match Two Sets of Binary Feature Vectors

Input feature vectors.

```
features1 = binaryFeatures(uint8([1 8 7 2; 8 1 7 2]));
features2 = binaryFeatures(uint8([8 1 7 2; 1 8 7 2]));
```

Match the vectors using the Hamming distance.

```
[indexPairs matchMetric] = matchFeatures(features1, features2)
```

```
indexPairs = 2×2 uint32 matrix

   1   2
   2   1


matchMetric = 2×1 single column vector

     0
     0
```

Version History

Introduced in R2013a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`extractFeatures` | `extractHOGFeatures` | `matchFeatures`

cameraIntrinsics

Object for storing intrinsic camera parameters

Description

Store information about a camera’s intrinsic calibration parameters, including lens distortion.

Creation

Syntax

```
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize)
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize,
PropertyName=Value)
```

Description

`intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize)` returns a camera intrinsics object that contains the focal length and the camera's principal point. The three input arguments set the FocalLength, PrincipalPoint, and ImageSize properties, respectively.

`intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize, PropertyName=Value)` specifies options using one or more properties on page 2-682 in addition to the previous syntax. For example, `RadialDistortion=[0 10]` sets the radial lens distortion property, to the vector `[0 10]`.

Properties

`FocalLength` — Focal length

2-element vector

Focal length in x and y, stored as a 2-element vector $[f_x\ f_y]$ in pixels.

$f_x = F * s_x$
$f_y = F * s_y$

F is the focal length in world units, typically in millimeters. s_x and s_y are the number of pixels per world unit in the x and y direction respectively. Thus, f_x and f_y are in pixels.

The focal length F influences the angle of view and thus affects the area of the scene that appears focused in an image. For a fixed subject distance:

- A short focal length offers a wide angle of view allowing to capture large area of the scene under focus. It emphasizes both the subject and the scene background.
- A long focal length offers a narrow angle of view, thus reducing the area of the scene under focus. It emphasizes more on the subject and restricts the amount of background from being captured.

You cannot set the value of `FocalLength` after you create the `cameraIntrinsics` object.

`PrincipalPoint` — Optical center of camera

two-element vector

Optical center of camera, stored as a two-element vector [*cx cy*] in pixels. The vector contains the coordinates of the optical center of the camera.

You cannot set the value of `PrincipalPoint` after you create the `cameraIntrinsics` object.

`ImageSize` — Image size produced by the camera

two-element vector

Image size produced by the camera, stored as a two-element vector, [*mrows ncols*].

You cannot set the value of `ImageSize` after you create the `cameraIntrinsics` object.

`RadialDistortion` — Radial lens distortion coefficients

`[0 0]` (default) | 2-element vector | 3-element vector | 6-element vector

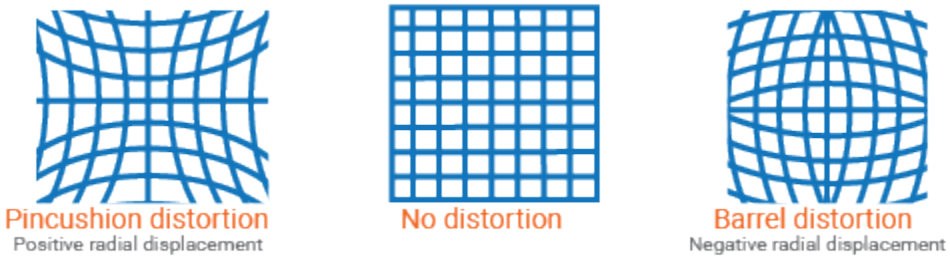
Radial lens distortion coefficients, specified as a 2-, 3-, or 6-element vector.

- 2-element vector — [*k1 k2*].
- 3-element vector — [*k1 k2 k3*].
- 6-element vector — [*k1 k2 k3 k4 k5 k6*].

In the Robot Operating System (ROS), the 3-element vector is referred to as the plumb_bob model, while the 6-element vector is known as the rational_polynomial model.

Radial distortion is the displacement of image points along radial lines extending from the principal point.

- As image points move away from the principal point (positive radial displacement), image magnification decreases and a pincushion-shaped distortion occurs on the image.
- As image points move toward the principal point (negative radial displacement), image magnification increases and a barrel-shaped distortion occurs on the image.



The camera parameters object calculates the radial-distorted location of a point, denoted as ($x_{\text{distorted}}$, $y_{\text{distorted}}$):

$$x_{distorted} = x\left(\frac{1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6}{1 + k_4 * r^2 + k_5 * r^4 + k_6 * r^6}\right)$$

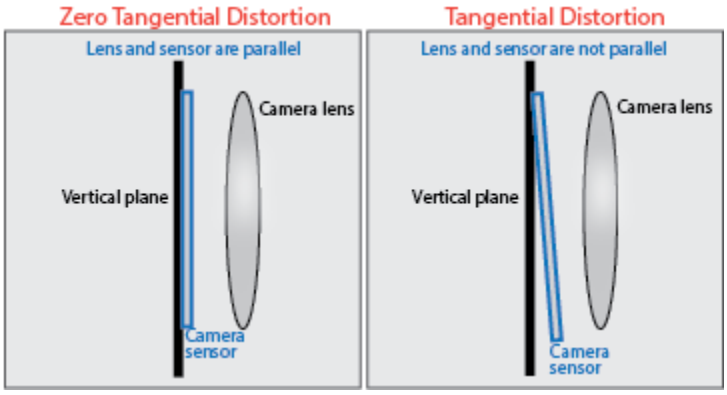
$$y_{distorted} = y\left(\frac{1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6}{1 + k_4 * r^2 + k_5 * r^4 + k_6 * r^6}\right)$$

x, y is a undistorted image point in normalized image coordinates in world units, with the origin at the optical center.
$r^2 = x^2 + y^2$
k_1, k_2, ..., k_6 are radial distortion coefficients of the lens. Typically, two coefficients are sufficient and k_3, ..., k_6 are only needed for wide-angle lenses.

`TangentialDistortion` — Tangential distortion coefficients

`[0 0]` (default) | two-element vector

Tangential distortion coefficients, specified as a 2-element vector, $[p_1\ p_2]$. Tangential distortion occurs when the lens and the image plane are not parallel.



The camera parameters object calculates the tangential distorted location of a point. You can denote the distorted points as $(x_{distorted}, y_{distorted})$, as follows:

$$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$$
$$y_{distorted} = y + [p_1 * (r^2 + 2*y^2) + 2 * p_2 * x * y]$$

x, y are undistorted pixel locations
p_1 and p_2 are tangential distortion coefficients of the lens
$r^2 = x^2 + y^2$
The undistorted pixel locations appear in normalized image coordinates, with the origin at the optical center. The coordinates are expressed in world units.

`Skew` — Camera axes skew

`0` (default) | numeric scalar

Camera axes skew, specified as a numeric scalar. If the x and the y axes are exactly perpendicular, then the skew must be `0`.

`K` — Camera intrinsic matrix

3-by-3 identity matrix (default) | 3-by-3 matrix

This property is read-only.

Camera intrinsic matrix, specified as a 3-by-3 matrix. The matrix has the format

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The coordinates $[c_x\ c_y]$ represent the optical center (the principal point), in pixels. When the x and y axis are exactly perpendicular, the skew parameter, s, equals `0`.

$f_x = F*s_x$
$f_y = F*s_y$

F is the focal length in world units, typically expressed in millimeters.
s_x and s_y are the number of pixels per world unit in the x and y direction respectively.
f_x and f_y are expressed in pixels.

Examples

Create an Object Containing Fundamental Camera Parameters

Define camera parameters without lens distortion or skew.

Specify the focal length and principal point in pixels.

```
focalLength    = [800 800];
principalPoint = [320 240];
imageSize      = [480 640];
```

Create a camera intrinsics object.

```
intrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize)
```

```
intrinsics =
  cameraIntrinsics with properties:

             FocalLength: [800 800]
          PrincipalPoint: [320 240]
               ImageSize: [480 640]
        RadialDistortion: [0 0]
    TangentialDistortion: [0 0]
                    Skew: 0
                       K: [3×3 double]
```

Version History

Introduced in R2017a

R2022b: Supports premultiply matrix convention
Behavior changed in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, a property of the `cameraIntrinsics` object has changed to support the premultiply convention.

- The new `K` property replaces the old `IntrinsicMatrix` property. The value of `K` is the transpose of `IntrinsicMatrix`.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2024b: Import OpenCV pinhole camera model with six radial distortion coefficients

The `cameraIntrinsics` has been updated to support OpenCV pinhole camera model with 6 radial distortion coefficients.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Objects

`cameraParameters`

Functions

`undistortImage`

Topics

“Using the Single Camera Calibrator App”
“Import Camera Intrinsic Parameters from ROS”
“3-D Motion Reconstruction Using Multiple Cameras”

cameraIntrinsicsKB

Intrinsic camera parameters based on Kannala-Brandt model

Description

The `cameraIntrinsicsKB` object stores information about the intrinsic calibration parameters about the intrinsic calibration parameters, using the Kannala-Brandt [1] model, for a fisheye lens.

Creation

Create a `cameraIntrinsicsKB` object by using the `cameraIntrinsicsFromOpenCV` function.

Properties

`FocalLength` — Focal length
two-element vector

This property is read-only.

Focal length, stored as a two-element vector of the form [f_x f_y], with values in pixels.
$f_x = F \cdot s_x$
$f_y = F \cdot s_y$

F is the focal length in world units, typically millimeters. s_x and s_y are the number of pixels per world unit in the x- and y-directions, respectively.

The focal length F influences the angle of view, and thus affects the area of the scene that appears focused in an image. For a fixed subject distance:

- A short focal length offers a wide angle of view enabling the camera to capture a large area of the scene under focus. It emphasizes both the subject and the scene background.
- A long focal length offers a narrow angle of view, thus reducing the area of the scene under focus. It emphasizes the subject more, and restricts the amount of background from captured by the camera.

For more information on how to set this property at object creation, see the `intrinsicMatrix` argument of the `cameraIntrinsicsFromOpenCV` function.

`PrincipalPoint` — Optical center of camera
two-element vector

This property is read-only.

Optical center of the camera, stored as a two-element vector of the form [cx cy], with values in pixels. The vector contains the coordinates of the optical center of the camera.

For more information on how to set this property at object creation, see the `intrinsicMatrix` argument of the `cameraIntrinsicsFromOpenCV` function.

`ImageSize` — Image size produced by camera

two-element vector

This property is read-only.

Image size produced by the camera, stored as a two-element vector of the form [*mrows ncols*]. Specify this value at object creation by using the `imageSize` argument of the `cameraIntrinsicsFromOpenCV` function.

`DistortionCoefficients` — Polynomial coefficients

four-element vector

This property is read-only.

Polynomial coefficients, stored as a four-element vector of the form [*k1 k2 k3 k4*]. These values specify the polynomial coefficients for the fisheye distortion function:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8)$$

$$\theta = \text{atan}(\gamma)$$

$$k_1, k_2, k_3, k_4$$

where *k1*, *k2*, *k3*, and *k4* are the distortion coefficients of the lens.

For more information on how to set this property at object creation, see the `distortionCoefficients` argument of the `cameraIntrinsicsFromOpenCV` function.

K — Camera intrinsic matrix

3-by-3 identity matrix (default) | 3-by-3 matrix

This property is read-only.

Camera intrinsic matrix, stored as a 3-by-3 matrix. The matrix has this format:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *cx* and *cy* are the corresponding values of the `PrincipalPoint`, and *fx* and *fy* are the corresponding values of `FocalLength`.

For more information on how to set this property at object creation, see the `intrinsicMatrix` argument of the `cameraIntrinsicsFromOpenCV` function.

Examples

Undistort Image Using Fisheye Camera Parameters from OpenCV

Define OpenCV fisheye parameters in the workspace.

```
intrinsicMatrix = [875.88  0.00   1005.62; ...
                   0.00    874.76 741.52; ...
```

```
                    0.00,   0.00,  1.00];
distortionCoeffs = [0.08, -0.16, 0.35, -0.26];
```

Define the image size returned by the camera.

```
imageSize =[1500 2000];
```

Import the intrinsic camera parameters for a fisheye lens from OpenCV.

```
intrinsicsKB = cameraIntrinsicsFromOpenCV(intrinsicMatrix,distortionCoeffs,imageSize)
```

```
intrinsicsKB =
  cameraIntrinsicsKB with properties:

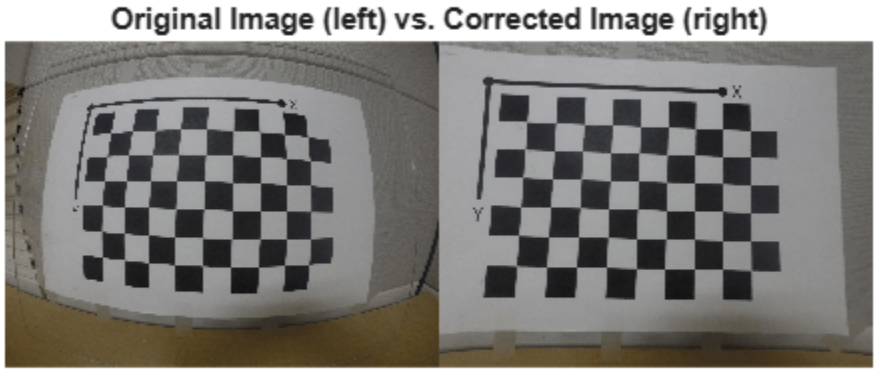
              FocalLength: [875.8800 874.7600]
            PrincipalPoint: [1.0066e+03 742.5200]
                 ImageSize: [1500 2000]
    DistortionCoefficients: [0.0800 -0.1600 0.3500 -0.2600]
                         K: [3×3 double]
```

Load an image to undistort.

```
filename = fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","gopro","gopro01.jpg");
I = imread(filename);
```

Undistort the image and display the results.

```
J = undistortImage(I,intrinsicsKB);
imshowpair(I,J,"montage");
title("Original Image (left) vs. Corrected Image (right)");
```



Algorithms

The Kannale-Brandt model extends the ideal pinhole model by accounting for lens distortion to represent a real camera.

The distorted points are ($x_{\text{distorted}}$, $y_{\text{distorted}}$), where:

$$x_{distorted} = \frac{\theta_d}{\gamma} \times x$$

$$y_{distorted} = \frac{\theta_d}{\gamma} \times y$$

The undistorted pixel locations in normalized image coordinates are (x,y), where:

$$\gamma^2 = x^2 + y^2$$

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8)$$

$$\theta = \text{atan}(\gamma)$$

$$k_1, k_2, k_3, k_4$$

where *k1*, *k2*, *k3*, and *k4* are the distortion coefficients of the lens.

Version History

Introduced in R2024a

R2025a: Generate C/C++ code using MATLAB Coder

The `cameraIntrinsicsKB` function now supports the generation of C/C++ code (requires MATLAB Coder).

References

[1] Kannala, J., and S.S. Brandt. *A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses.* IEEE Transactions on Pattern Analysis and Machine Intelligence 28, no. 8 (August 2006): 1335–40. https://doi.org/10.1109/TPAMI.2006.153.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`cameraIntrinsicsFromOpenCV` | `undistortImage` | `undistortPoints` | `world2img` | `img2world2d`

Topics
“What Is Camera Calibration?”
“Fisheye Calibration Basics”

pcplayer

Visualize streaming 3-D point cloud data

Description

Visualize 3-D point cloud data streams from devices such as Microsoft Kinect®.

To improve performance, `pcplayer` automatically downsamples the rendered point cloud during interaction with the figure. The downsampling occurs only for rendering the point cloud and does not affect the saved points.

You can set the default center of rotation for the point cloud viewer to rotate around the axes center or around a point. Set the default behavior from the “Computer Vision Toolbox Preferences”.

Creation

Syntax

```
player = pcplayer(xlimits,ylimits,zlimits)
player = pcplayer(xlimits,ylimits,zlimits,Name=Value)
```

Description

`player = pcplayer(xlimits,ylimits,zlimits)` returns a player with `xlimits`,`ylimits`, and `zlimits` set for the axes limits.

`player = pcplayer(xlimits,ylimits,zlimits,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ViewPlane="XY"` sets the plane to the x-y axes from which to visualize the point cloud.

Input Arguments

`xlimits` — Range of *x*-axis coordinates
1-by-2 vector

Range of *x*-axis coordinates, specified as a 1-by-2 vector in the format [*min max*]. `pcplayer` does not display data outside these limits.

`ylimits` — Range of *y*-axis coordinates
1-by-2 vector

Range of *y*-axis coordinates, specified as a 1-by-2 vector in the format [*min max*]. `pcplayer` does not display data outside these limits.

`zlimits` — Range of *z*-axis coordinates
1-by-2 vector

Range of *z*-axis coordinates, specified as a 1-by-2 vector in the format [*min max*].`pcplayer` does not display data outside these limits.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ViewPlane="XY"` sets the plane to the x-y axes from which to visualize the point cloud.

`MarkerSize` — Diameter of marker

`6` (default) | positive scalar

Diameter of marker, specified as a positive scalar. The value specifies the approximate diameter of the point marker. MATLAB graphics defines the unit as points. A marker size larger than six can reduce the rendering performance.

`BackgroundColor` — Background color

`[0 0 0]` (black) (default) | RGB triplet | hexadecimal color code | character vector

Background color, specified as an RGB triplet, hexadecimal color code, or a character vector that specifies a long or short color name.

`VerticalAxis` — Vertical axis

`"Z"` (default) | `"X"` | `"Y"`

Vertical axis, specified as `"X"`, `"Y"`, or `"Z"`. When you reload a saved figure, any action on the figure resets the vertical axis to the *z*-axis.

`VerticalAxisDir` — Vertical axis direction

`"Up"` (default) | `"Down"`

Vertical axis direction, specified as `"Up"` or `"Down"`. When you reload a saved figure, any action on the figure resets the direction to the up direction.

`Projection` — Camera projection for 3-D views

`"perspective"` (default) | `"orthographic"`

Camera projection for 3-D views, specified as one of these values:

- `"perspective"` — Projects the viewing volume as the frustum of a pyramid (a pyramid whose apex has been cut off parallel to the base). Objects further from the camera appear smaller. Distance causes foreshortening, which enables you to perceive the depth of 3-D objects. This projection type is useful when you want to display realistic views of real objects. Perspective projection does not preserve the relative dimensions of objects. Instead, it displays a distant line segment smaller than a nearer line segment of the same length. Lines that are parallel in the data might not appear parallel in the scene.
- `"orthographic"` — This projection type maintains the correct relative dimensions of graphics objects regarding the distance of a given point from the viewer. Relative distance from the camera does not affect the size of objects. Lines that are parallel in the data parallel appear parallel on the screen. This projection type is useful when it is important to maintain the actual size of objects and the angles between objects.

`ViewPlane` — Plane to visualize point cloud

`"XY"` (default) | `"YX"` | `"XZ"` | `"ZX"` | `"YZ"` | `"ZY"`

Plane to visualize point cloud, specified as `"XY"`, `"YX"`, `"XZ"`, `"ZX"`, `"YZ"`, `"ZY"`. The `ViewPlane` sets the line of sight from the camera, centered in the selected plane, to the center of the plot.

`ColorSource` — Point cloud property for color map
`"auto"` (default) | `"X"` | `"Y"` | `"Z"` | `"Intensity"` | `"Color"` | `"Row"` | `"Column"`

Point cloud property for color map, specified as `"auto"`,`"X"`, `"Y"`, `"Z"`, `"Intensity"`, `"Color"`, `"Row"`, `"Column"`. The `pcshow` function uses the value of `ColorSource` to assign a color map for rendering the point cloud. The values use these properties:

- `"X"` — *x* coordinates in the `Location` property of the input point cloud.
- `"Y"` — *y* coordinates in the `Location` property of the input point cloud.
- `"Z"` — *z* coordinates in the `Location` property of the input point cloud.
- `"Intensity"` — `Intensity` property of the input point cloud.
- `"Color"` — `Color` property of the input point cloud.
- `"Row"` — Using the row values of an organized point cloud.
- `"Column"` — Using the column values of an organized point cloud.

When you set `ColorSource` to `"auto"`, the function sets the color map depending on which property values are available in the input point cloud object:

Available Point Cloud Property	Color Source
`Location` only	Maps the Z-value to a color value in the current color map.
`Location` and `Intensity`	Maps the z-value to a color value in the current color map.
`Location` and `Color`	Maps the color values to the current color map.
`Location`, `Intensity`, and `Color`	Maps the color values to the current color map.

`AxesVisibility` — State of axes visibility
`"off"` or logical `0` (`false`) (default) | `"on"` or logical `1` (`true`)

State of axes visibility, specified as `"off"` or `"on"`, or as numeric or logical `0` (`false`) or `1` (`true`). A value of `"off"` is equivalent to `false`, and `"on"` is equivalent to `true`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `"off"` — Hide the axes without deleting it. You still can access the properties of an invisible axes object. (Child objects such as lines remain visible.)
- `"on"` — Display the axes and its children.

`Parent` — Axes on which to display the visualization
`axes` graphics object

Axes on which to display the visualization, specified as an `Axes` object. To create an `Axes` object, use the `axes` function. To display the visualization in a new figure, leave `"Parent"` unspecified.

Properties

Axes — Player axes handle

`axes` graphics object

Player axes handle, specified as an `axes` graphics object.

Object Functions

hide	Hide player figure
isOpen	Visible or hidden status for player
show	Show player
view	Display point cloud

Examples

Terminate a Point Cloud Processing Loop

Create the player and add data.

```
player = pcplayer([0 1],[0 1],[0 1]);
```

Display continuous player figure. Use the `isOpen` function to check if player figure window is open.

```
while isOpen(player)
     ptCloud = pointCloud(rand(1000,3,"single"));
     view(player,ptCloud);
end
```

Terminate while-loop by closing pcplayer figure window.

More About

Axes Toolbar Operations

Keyboard shortcuts are enabled only for the rotate (default) mode. To view the axes toolbar operations, point to the figure and select one of these options. Click the Restore View icon to reset the display.



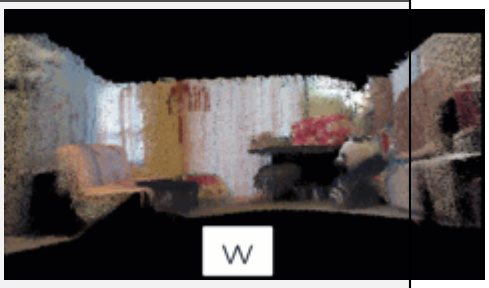
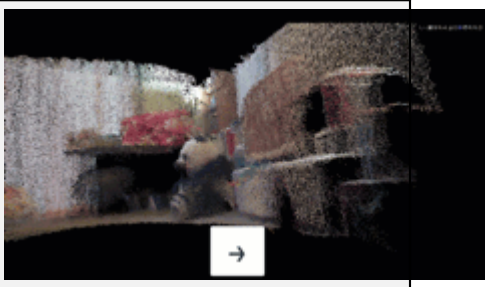
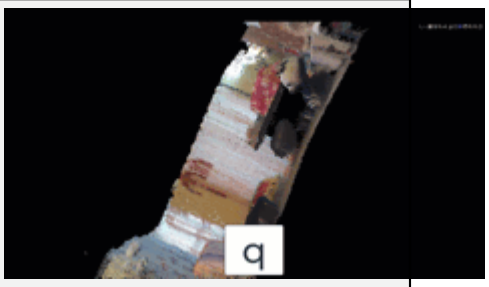
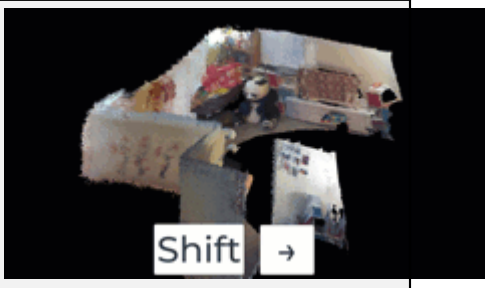
Feature	Description
Vertical Axis	Select an *X*-, *Y*-, or *Z*-direction to set the direction of that axis of the point cloud as the positive vertical direction. Use the keyboard arrow keys to move the point cloud with respect to the direction selected.
View Plane	Select the viewing angle of the point cloud figure as the *XY*, *YX*, *XZ*, *ZX*, *YZ*, or *ZY* plane.
Background Color	Display a color chart from which you can select a background color.
Axes On/Off	Turn the display axes on or off.

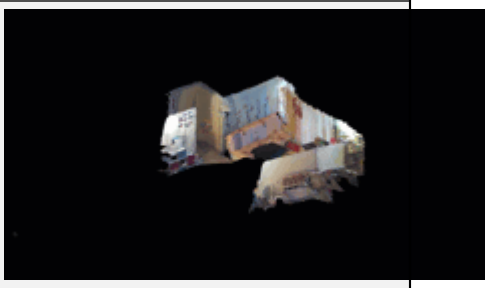
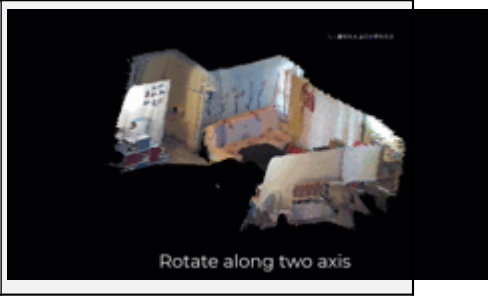
<table>
<tr><th>Feature</th><th>Description</th></tr>
<tr><td>Projection</td><td>Select perspective or orthogonal view.</td></tr>
<tr><td>Export</td><td>•  — Save the content as a tightly cropped image or PDF.
•  — Copy the content as an image.
•  — Copy the content as a vector graphic.</td></tr>
<tr><td>Brush</td><td>Enable data brushing mode.</td></tr>
<tr><td>Data Tips</td><td>View the point data values of a point by selecting it in the point cloud figure. The Data Tips display shows the x, y, and z values of the points. Point clouds for depth images and lidar contain these additional data properties:<table><tr><th>Point Cloud Data Source</th><th>Data Value Properties</th></tr><tr><td>Depth image (RGB-D sensor)</td><td>Color, Row, and Column</td></tr><tr><td>Lidar</td><td>Intensity, Range, Azimuth angle, Elevation angle, Row, and Column</td></tr></table></td></tr>
<tr><td>Rotate 3D</td><td>Drag to rotate the point cloud figure. You can also right-click any point on the point cloud figure and select Rotate Around a Point to specify that point as the center of rotation for the figure. To return to axes rotation, right-click again and select Rotate Around Axes Center.

While in Rotate 3D mode, you can also select the color source for the point cloud by right-clicking anywhere in the point cloud figure and selecting Color Source, and then selecting an option from the list. The figure applies the active colormap to the point cloud based on the values in the selected source. For a point cloud of only position data, the Color Source options are X, Y, Z, and User Specified. Point clouds for depth image and lidar data include these additional color source options:<table><tr><th>Point Cloud Data Source</th><th>Color Source Options</th></tr><tr><td>Depth image (RGB-D sensor)</td><td>Color, Row, and Column</td></tr><tr><td>Lidar</td><td>Intensity, Range, Azimuth angle, Elevation angle, Row, and Column</td></tr></table></td></tr>
<tr><td>Pan, Zoom in, and Zoom out</td><td>Maneuver the point cloud in the figure, or zoom the point cloud figure in or out.</td></tr>
<tr><td>Restore View</td><td>Restore the original view of the axes.</td></tr>
</table>

Move Camera Through a Scene

To navigate into a point cloud and observe the scene, use these keyboard shortcuts:

Action	Keyboard Shortcut	Example
Move forward or backward through the scene. The camera and the target can move.	**w** — Forward **s** — Backward	
Move left or right through the scene. The camera and the target can move.	**a** — Left **d** — Right	
Look around the scene (pitch/yaw rotation of the camera). Camera position is fixed, target moves.	**Up arrow** — Camera pitches up **Down arrow** — Camera pitches down **Left arrow** — Camera turns left in yaw **Right arrow** — Camera turns right in yaw	
Roll the scene (roll rotation of the camera).	**q** — Camera moves clockwise. (Scene appears to move counterclockwise.) **e** — Camera moves counterclockwise. (Scene appears to move clockwise.)	
Rotate the scene (rotate the camera around the point cloud). Camera position changes, target is fixed.	**Shift** + **Left** or **Right** — Camera orbits horizontally around the point cloud. (Scene appears to stand still as camera moves externally around it horizontally.) **Shift** + **Up** or **Down** — Camera orbits vertically around the point cloud. (Scene appears to stand still as camera moves externally around it vertically.)	
Zoom in or out.	**z** — Zoom in **x** — Zoom out	

Action	Keyboard Shortcut	Example
Look at a view plane.	1 — **XY** 2 — **YX** 3 — **XZ** 4 — **ZX** 5 — **YZ** 6 — **ZY** Camera positioned in the selected plane with a line of sight to the center of the plot.	
Switch between rotate and pan.	**Space bar** — Press and hold the space bar to switch the interaction to pan. Release the space bar to switch back to rotate.	
Rotate in X-, Y-, and Z-dimensions.	**T** — Adds rotate interaction in the third dimension.	

Color Formats

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
"white"	"w"	[1 1 1]	

Performance and Tips

- To improve performance, the function automatically downsamples the rendered point cloud when you rotate, pan, or zoom in or out by clicking the mouse. Downsampling is not applied for zoom in or out by using the mouse scroll mechanism. When using data tips, the figure does not apply downsampling. The downsampling occurs only for rendering the point cloud and does not affect the saved points.
- The keyboard shortcuts described in “Move Camera Through Scene” on page 3-1210 are not supported when loading a saved figure or for popped-out windows using Live Editor.
- When you reload a saved figure, any action on the figure resets the vertical axis to the *z*-axis and in the up direction.
- To visualize a large point cloud (greater than one million points), use the `pcviewer` object.

Version History

Introduced in R2015b

R2022b: Projection display default behavior changed to perspective

The default projection for the camera 3-D view display is changed from orthographic to perspective. You can use the new `Projection` name-value argument to specify the projection as `"perspective"` or `"orthographic"`. Prior to R2022b, the display rendered the point cloud in an orthographic view only.

See Also

Functions

`showShape` | `pcshow` | `pcshowpair` | `pcwrite` | `pcread` | `pcmerge` | `plot3` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `scatter3` | `pcregistericp`

Objects

`planeModel` | `pointCloud`

Topics

“Choose Function to Visualize Detected Objects”

view

Display point cloud

Syntax

```
view(player,ptCloud)
view(player,xyzPoints)
view(player,xyzPoints,color)
view(player,xyzPoints,colorMap)
```

Description

`view(player,ptCloud)` displays a point cloud in the `pcplayer` figure window, `player`. The points, locations, and colors are stored in the `ptCloud` object.

`view(player,xyzPoints)` displays the points of a point cloud at the locations specified by the `xyzPoints` matrix. The color of each point is determined by the *z* value.

`view(player,xyzPoints,color)` displays a point cloud with colors specified by `color`.

`view(player,xyzPoints,colorMap)` displays a point cloud with colors specified by `colorMap`.

Examples

View Rotating 3-D Point Cloud

Load point cloud.

```
ptCloud = pcread("teapot.ply");
```

Define a rotation matrix and 3-D transform.

```
x = pi/180;
R = [ cos(x) sin(x) 0 0
     -sin(x) cos(x) 0 0
      0         0   1 0
      0         0   0 1];

tform = affine3d(R);
```

Compute *x*, *y*, and *z* limits that ensure that the rotated teapot is not clipped.

```
lower = min([ptCloud.XLimits ptCloud.YLimits]);
upper = max([ptCloud.XLimits ptCloud.YLimits]);

xlimits = [lower upper];
ylimits = [lower upper];
zlimits = ptCloud.ZLimits;
```

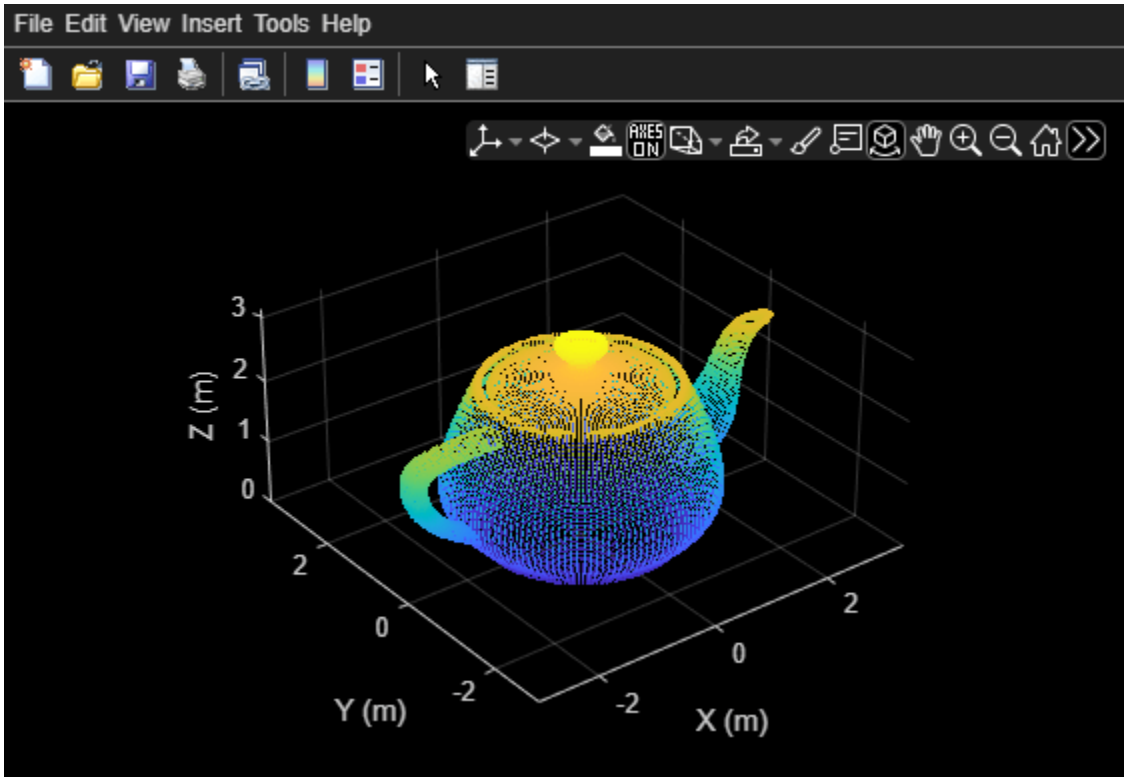
Create the player and customize player axis labels.

```
player = pcplayer(xlimits,ylimits,zlimits);

xlabel(player.Axes,"X (m)");
ylabel(player.Axes,"Y (m)");
zlabel(player.Axes,"Z (m)");
```

Rotate the teapot around the *z*-axis.

```
for i = 1:360
    ptCloud = pctransform(ptCloud,tform);
    view(player,ptCloud);
end
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object. The object contains the locations, intensities, and RGB colors to render the point cloud.

Point Cloud Property	Color Rendering Result
`Location` only	Maps the z-value to a color value in the current color map.
`Location` and `Intensity`	Maps the intensity to a color value in the current color map.

Point Cloud Property	Color Rendering Result
`Location` and `Color`	Use provided color.
`Location`, `Intensity`, and `Color`	Use provided color.

`player` — Player

`pcplayer` object

Player for visualizing 3-D point cloud data streams, specified as a `pcplayer` object.

`xyzPoints` — Point cloud *x*, *y*, and *z* locations

M-by-3 numeric matrix | *M*-by-*N*-by-3 numeric matrix

Point cloud *x*, *y*, and *z* locations, specified as either an *M*-by-3 or an *M*-by-*N*-by-3 numeric matrix. The *M*-by-*N*-by-3 numeric matrix is commonly referred to as an organized point cloud. The `xyzPoints` numeric matrix contains *M* or *M*-by-*N* [*x*,*y*,*z*] points. The *z* values in the numeric matrix, which generally correspond to depth or elevation, determine the color of each point.

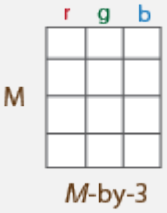
`color` — Color for points in the point cloud

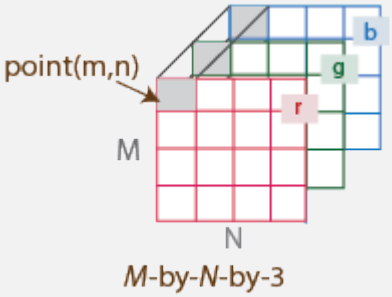
1-by-3 RGB vector | short name of color | long name of color | *M*-by-3 matrix | *M*-by-*N*-by-3 matrix

Color for points in the point cloud, specified as a 1-by-3-RGB vector, an *M*-by-3 matrix, an *M*-by-*N*-by-3 matrix, a short color name, or a long color name. For details on color values, see the Color Value on page 3-1212 table.

- For `single` or `double` datatype, the `Color` RGB value must be specified in the range [0, 1].
- For `uint8` datatype, the `Color` RGB value must be specified in the range [0, 255].
- The function stores the `Color` value as either an *M*-by-3 or *M*-by-*N*-by-3. If you specify the color as a string or a 1-by-3 vector, the function converts it to one of these dimensions.

Color	Format	Example
Specify one color for all points	short or long color name	`"r"` `"red"`
	1-by-3 vector (RGB triplet)	 `[255 0 0]`
Specify a color for each point	cell array of character vectors	`["red","yellow","blue"]`

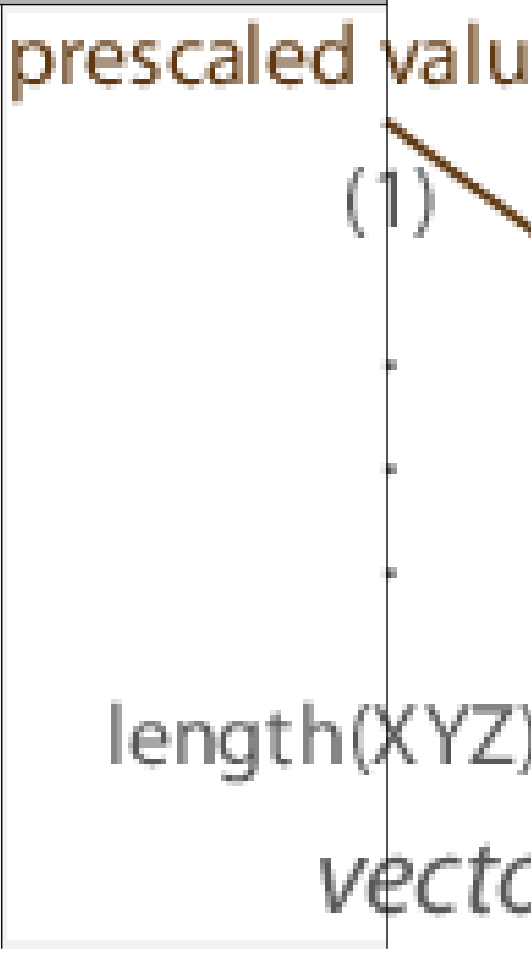
	M-by-3 matrix, as a list of RGB values	`255 0 0`
		`255 0 0`
		`0 255 255`
		

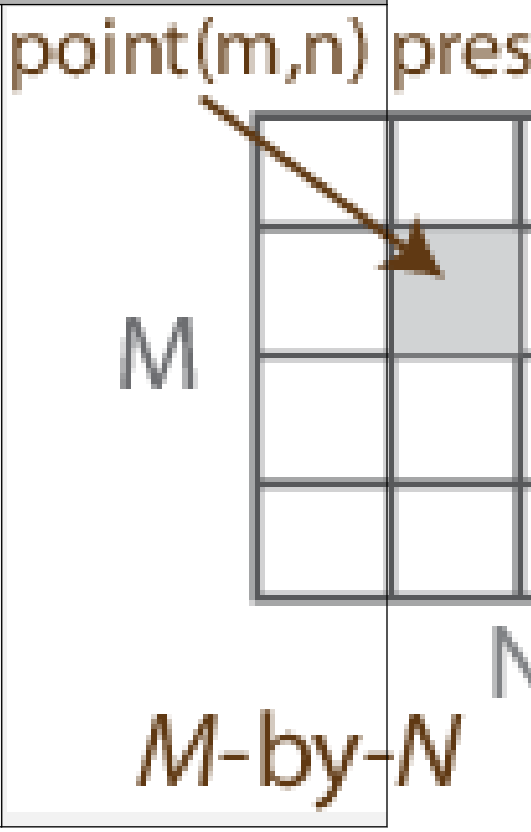
Color	Format	Example
	M-by-*N*-by-3 matrix for an organized point cloud containing RGB values for each point, where *M*-by-*N* is the dimension of the point cloud.	

`colorMap` — Point cloud color map for points
M-by-1 vector | *M*-by-*N* matrix

Point cloud color map for points, specified as one of:

- *M*-by-1 vector
- *M*-by-*N* matrix

Points Input	Color Selection	Valid Values of `C`	
`xyzPoints`	Different color for each point	Vector or *M*-by-*N* matrix. The matrix must contain values that are linearly mapped to a color listed in the current `colormap`.	

Points Input	Color Selection	Valid Values of C	
			

More About

Color Value

Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	
"green"	"g"	[0 1 0]	
"blue"	"b"	[0 0 1]	
"cyan"	"c"	[0 1 1]	
"magenta"	"m"	[1 0 1]	
"yellow"	"y"	[1 1 0]	
"black"	"k"	[0 0 0]	
"white"	"w"	[1 1 1]	

Version History

Introduced in R2015b

pointTrack

Object for storing matching points from multiple views

Description

Use this object to store 2-D points in an image that match across multiple views of the same image.

Creation

```
track = pointTrack(viewIDs,points)
```

Description

`track = pointTrack(viewIDs,points)` returns an object that stores matching 2-D points from multiple views. You can also create this point track object using the `findTracks` object function of the `imageviewset` object.

Input Arguments

`viewIDs` — View IDs of image poses

M-element vector

View IDs of image poses, specified as an *M*-element vector of scalar integers.

`points` — 2-D points that match across multiple image views

M-by-2 matrix

2-D points that match across multiple image views, specified as an *M*-by-2 matrix of (*x*,*y*) point coordinates. You can use the `matchFeatures` function to find these points, and then save them using this object.

Output Arguments

`track` — Point track object

`pointTrack` object

Point track object, returned as a `pointTrack` object. You can use this object to store matching 2-D points from multiple views. You can also create this point track object using the `findTracks` object function of the `imageviewset` object.

Examples

Create a Point Track Object

Save (*x* , *y*) points and view IDs.

```
points = [10,20;11,21;12,22];
viewIDs = [1 2 3];
```

Create a `pointTrack` object to save points and IDs. The matched 2-D image points correspond to the same 3-D world point. Separate `pointTrack` objects are needed to track different 3-D world points across views.

```
track = pointTrack(viewIDs,points);
```

Version History

Introduced in R2016a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`triangulateMultiview` | `matchFeatures` | `bundleAdjustment` | `findTracks`

Objects

`imageviewset` | `vision.PointTracker`

Topics

“3-D Point Cloud Registration and Stitching”
“Coordinate Systems”

vision.PointTracker

Namespace: `vision`

Track points in video using Kanade-Lucas-Tomasi (KLT) algorithm

Description

The point tracker object tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm. You can use the point tracker for video stabilization, camera motion estimation, and object tracking. It works particularly well for tracking objects that do not change shape and for those that exhibit visual texture. The point tracker is often used for short-term tracking as part of a larger tracking framework.

As the point tracker algorithm progresses over time, points can be lost due to lighting variation, out of plane rotation, or articulated motion. To track an object over a long period of time, you may need to reacquire points periodically.

To track a set of points:

1. Create the `vision.PointTracker` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
pointTracker = vision.PointTracker
pointTracker = vision.PointTracker(Name,Value)
```

Description

`pointTracker = vision.PointTracker` returns a point tracker object that tracks a set of points in a video.

`pointTracker = vision.PointTracker(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `pointTracker = vision.PointTracker('NumPyramidLevels',3)`

Initialize Tracking Process:

To initialize the tracking process, you must use `initialize` to specify the initial locations of the points and the initial video frame.

`initialize(pointTracker,points,I)` initializes points to track and sets the initial video frame. The initial locations `points`, must be an *M*-by-2 array of [x y] coordinates. The initial video frame, `I`, must be a 2-D grayscale or RGB image and must be the same size and data type as the video frames passed to the object.

The `detectFASTFeatures`, `detectSURFFeatures`, `detectHarrisFeatures`, and `detectMinEigenFeatures` functions are few of the many ways to obtain the initial points for tracking.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

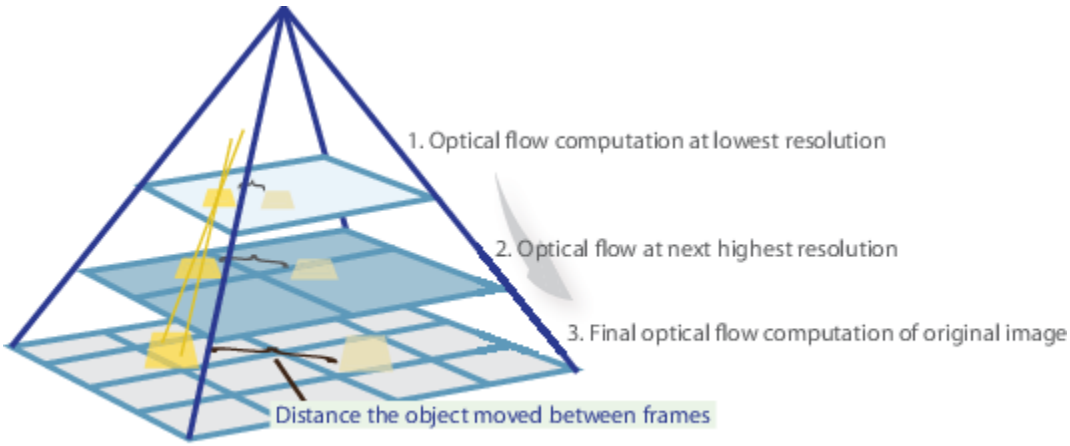
If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`NumPyramidLevels` — Number of pyramid levels

integer (default) | 3

Number of pyramid levels, specified as integer. The point tracker implementation of the KLT algorithm uses image pyramids. The tracker generates an image pyramid, where each level is reduced in resolution by a factor of two compared to the previous level. Selecting a pyramid level greater than 1, enables the algorithm to track the points at multiple levels of resolution, starting at the lowest level. Increasing the number of pyramid levels allows the algorithm to handle larger displacements of points between frames. However, computation cost also increases. Recommended values are between 1 and 4.



Each pyramid level is formed by down-sampling the previous level by a factor of two in width and height. The point tracker begins tracking each point in the lowest resolution level, and continues tracking until convergence. The object propagates the result of that level to the next level as the initial guess of the point locations. In this way, the tracking is refined with each level, up to the original image. Using the pyramid levels allows the point tracker to handle large pixel motions, which can comprise distances greater than the neighborhood size.

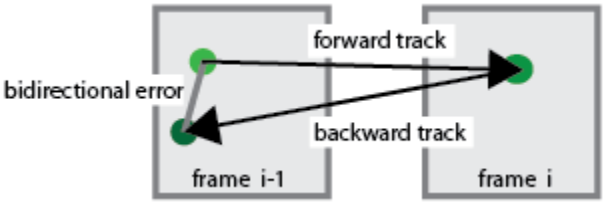
`MaxBidirectionalError` — Forward-backward error threshold

`inf` (default) | scalar

Forward-backward error threshold, specified as a scalar. If you set the value to less than `inf`, the tracker tracks each point from the previous to the current frame. It then tracks the same points back

to the previous frame. The object calculates the bidirectional error. This value is the distance in pixels from the original location of the points to the final location after the backward tracking. The corresponding points are considered invalid when the error is greater than the value set for this property. Recommended values are between `0` and `3` pixels.

Using the bidirectional error is an effective way to eliminate points that could not be reliably tracked. However, the bidirectional error requires additional computation. When you set the `MaxBidirectionalError` property to `inf`, the object does not compute the bidirectional error.



`BlockSize` — Size of neighborhood
`[31 31]` (default) | two-element vector

Size of neighborhood around each point being tracked, specified as a two-element vector, [*height*, *width*]. The *height* and *width* must be odd integers. This neighborhood defines the area for the spatial gradient matrix computation. The minimum value for `BlockSize` is `[5 5]`. Increasing the size of the neighborhood, increases the computation time.

`MaxIterations` — Maximum number of search iterations
`30` (default) | integer

Maximum number of search iterations for each point, specified as an integer. The KLT algorithm performs an iterative search for the new location of each point until convergence. Typically, the algorithm converges within 10 iterations. This property sets the limit on the number of search iterations. Recommended values are between `10` and `50`.

Usage

Syntax

```
[points,point_validity] = pointTracker(I)
[points,point_validity,scores] = pointTracker(I)
setPoints(pointTracker,points)
setPoints(pointTracker,points,point_validity)
```

Description

`[points,point_validity] = pointTracker(I)` tracks the points in the input frame, `I`.

`[points,point_validity,scores] = pointTracker(I)` additionally returns the confidence score for each point.

`setPoints(pointTracker,points)` sets the points for tracking. The function sets the *M*-by-2 `points` array of [*x* *y*] coordinates with the points to track. You can use this function if the points need to be redetected because too many of them have been lost during tracking.

`setPoints(pointTracker,points,point_validity)` additionally lets you mark points as either valid or invalid. The input logical vector `point_validity` of length *M*, contains the true or false value corresponding to the validity of the point to be tracked. The length *M* corresponds to the number of points. A false value indicates an invalid point that should not be tracked. For example, you can use this function with the `estgeotform2d` function to determine the transformation between the point locations in the previous and current frames. You can mark the outliers as invalid.

Input Arguments

`I` — Video frame
grayscale | truecolor (RGB)

Video frame, specified as grayscale or truecolor (RGB).

Output Arguments

`points` — Tracked points
M-by-2 array

Tracked points, returned as an *M*-by-2 array of [*x*, *y*] coordinates that correspond to the new locations of the points in the input frame, `I`.

`point_validity` — Reliability of track
M-by-1 logical array

Reliability of track for each point, returned as an *M*-by-1 logical array. A point can be invalid for several reasons. The point can become invalid if it falls outside of the image. Also, it can become invalid if the spatial gradient matrix computed in its neighborhood is singular. If the bidirectional error is greater than the `MaxBidirectionalError` threshold, this condition can also make the point invalid.

`scores` — Confidence score
M-by-1 array

Confidence score between `0` and `1`, returned as an *M*-by-1 array. The values correspond to the degree of similarity between the neighborhood around the previous location and new location of each point. These values are computed as a function of the sum of squared differences between the previous and new neighborhoods. The greatest tracking confidence corresponds to a perfect match score of `1`.

Object Functions

To use an object function, specify the System object™ as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.PointTracker

initialize Initialize video frame and points to track

Common to All System Objects

step Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Track a Face in Scene

Create System objects for reading and displaying video and for drawing a bounding box of the object.

```
videoReader = VideoReader('visionface.avi');
videoPlayer = vision.VideoPlayer('Position',[100,100,680,520]);
```

Read the first video frame, which contains the object, define the region.

```
objectFrame = readFrame(videoReader);
objectRegion = [264,122,93,93];
```

As an alternative, you can use the following commands to select the object region using a mouse. The object must occupy the majority of the region:

figure; imshow(objectFrame);

objectRegion=round(getPosition(imrect))

Show initial frame with a red bounding box.

```
objectImage = insertShape(objectFrame,'rectangle',objectRegion,'ShapeColor','red');
figure;
imshow(objectImage);
title('Red box shows object region');
```

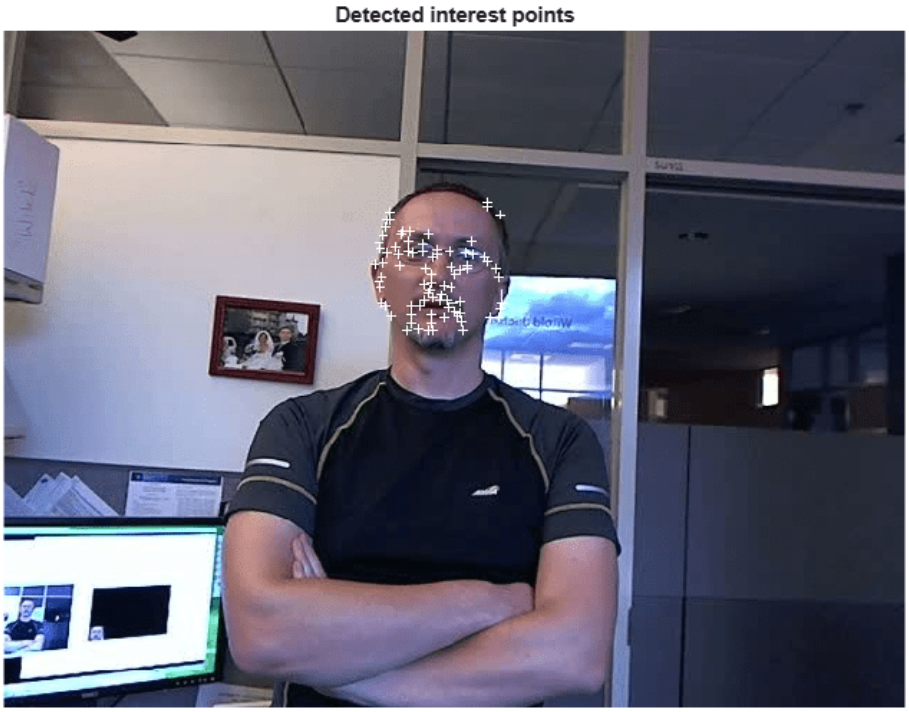


Detect interest points in the object region.

```
points = detectMinEigenFeatures(im2gray(objectFrame),'ROI',objectRegion);
```

Display the detected points.

```
pointImage = insertMarker(objectFrame,points.Location,'+','MarkerColor','white');
figure;
imshow(pointImage);
title('Detected interest points');
```



Create a tracker object.

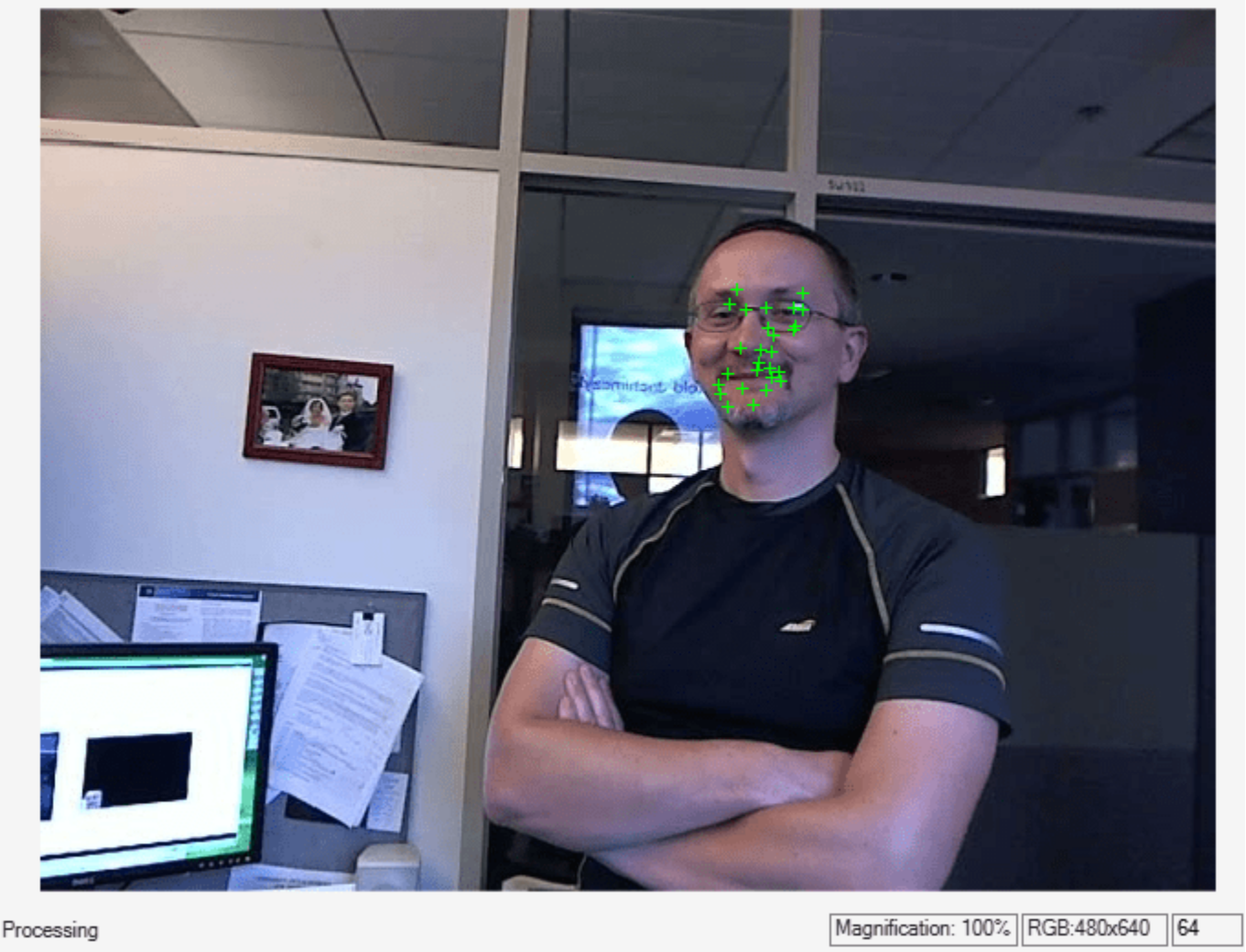
```
tracker = vision.PointTracker('MaxBidirectionalError',1);
```

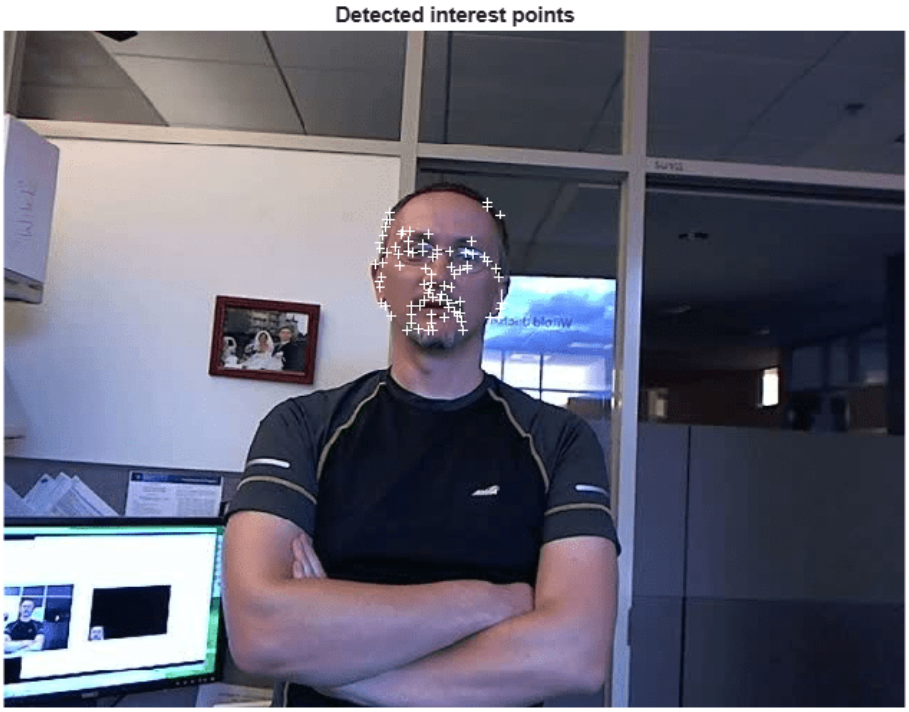
Initialize the tracker.

```
initialize(tracker,points.Location,objectFrame);
```

Read, track, display points, and results in each video frame.

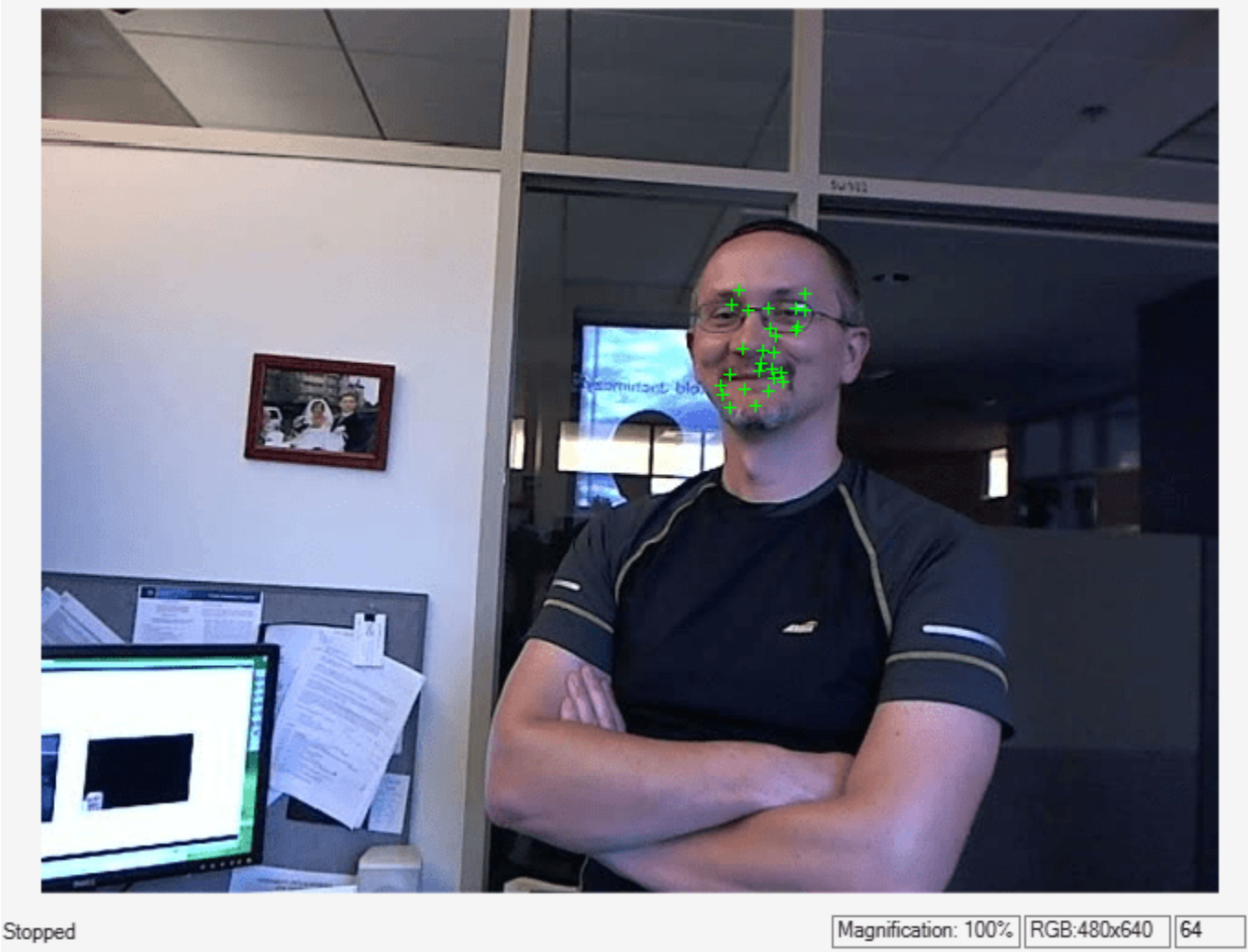
```
while hasFrame(videoReader)
      frame = readFrame(videoReader);
      [points,validity] = tracker(frame);
      out = insertMarker(frame,points(validity, :),'+');
      videoPlayer(out);
end
```


Processing
Magnification: 100%
RGB:480x640
64



Release the video player.

```
release(videoPlayer);
```



Version History

Introduced in R2012b

References

[1] Lucas, Bruce D. and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision,”*Proceedings of the 7th International Joint Conference on Artificial Intelligence*, April, 1981, pp. 674–679.

[2] Tomasi, Carlo and Takeo Kanade. *Detection and Tracking of Point Features*, Computer Science Department, Carnegie Mellon University, April, 1991.

[3] Shi, Jianbo and Carlo Tomasi. “Good Features to Track,” *IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[4] Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. “Forward-Backward Error: Automatic Detection of Tracking Failures,” *Proceedings of the 20th International Conference on Pattern Recognition*, 2010, pages 2756–2759, 2010.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`insertMarker` | `detectHarrisFeatures` | `detectMinEigenFeatures` | `detectSURFFeatures` | `estgeotform2d` | `imrect`

Objects

`vision.HistogramBasedTracker`

Topics

“Face Detection and Tracking Using CAMShift”
“Face Detection and Tracking Using the KLT Algorithm”
“Face Detection and Tracking Using Live Video Acquisition”
“Detect and Track Face”

External Websites

Detect and Track Multiple Faces in a Live Video Stream

viewSet

(To be removed) Object for managing data for structure-from-motion and visual odometry

Note The `viewSet` object will be removed in a future release. Use the `imageviewset` object instead. For more information, see “Version History”.

Description

A `viewSet` object stores views and connections between views. A view includes feature points and an absolute camera pose. A connection between two views includes point correspondences and the relative camera pose between them. Once you populate a `viewSet` object, you can use it to find point tracks across multiple views and retrieve the camera poses to be used by the `triangulateMultiview` and `bundleAdjustment` functions.

Creation

Syntax

```
vSet = viewSet
```

Description

`vSet = viewSet` creates an empty `viewSet` object. You can add views and connections using the `addView` and `addConnection` object functions.

Properties

`NumViews` — Number of views

integer

This property is read-only.

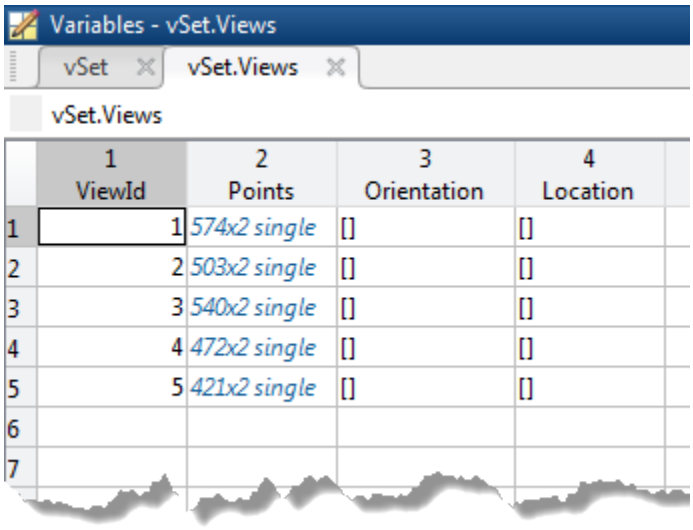
Number of views, stored as an integer.

`Views` — View attributes

four-column table

This property is read-only.

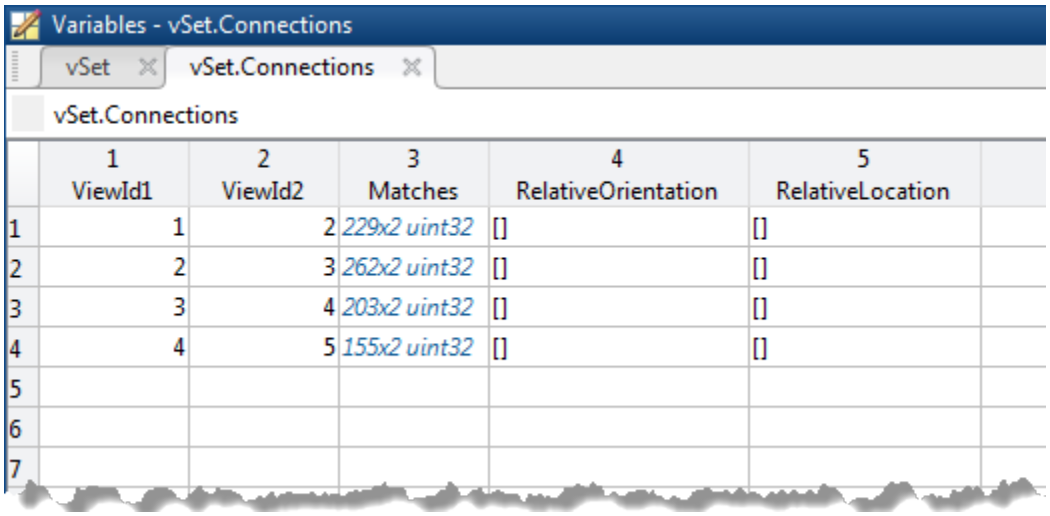
View attributes, stored as a four-column table. The table contains columns for `ViewID`, `Points`, `Orientation`, and `Location`. Use the `poses` method to obtain the IDs, orientation, and location for the points.



`Connections` — Pairwise connections between views
five-column table

This property is read-only.

Pairwise connections between views, stored as a five-column table. The columns are `ViewID1`, `ViewID2`, `Matches`, `RelativeOrientation`, and `RelativeLocation`. The number of entries in the table represent the number of connections. Each index in the `Matches` column represents a connection between the two views indicated by the view IDs.



Object Functions

addConnection	(To be removed) Add connection between two views
addView	(To be removed) Add new view to view set object
deleteConnection	(To be removed) Delete connection between two views from view set object
deleteView	(To be removed) Delete existing view from view set object
findTracks	(To be removed) Find matched points across multiple views
hasConnection	(To be removed) Check if connection exists between two views

hasView	(To be removed) Check if view exists
poses	(To be removed) Returns camera poses associated to views
updateConnection	(To be removed) Modify connection between two views in view set object
updateView	(To be removed) Modify existing view in view set object

Examples

Find Point Tracks Across Sequence of Images

Load images.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
imds = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = rgb2gray(read(imds));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create a `viewSet` object.

```
vSet = viewSet;
vSet = addView(vSet,1,'Points',pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:length(imds.Files)
 I = rgb2gray(read(imds));
 points = detectSURFFeatures(I);
 [features, points] = extractFeatures(I,points);
 vSet = addView(vSet,i,'Points',points);
 pairsIdx = matchFeatures(featuresPrev,features);
 vSet = addConnection(vSet,i-1,i,'Matches',pairsIdx);
 featuresPrev = features;
end
```

Find point tracks.

```
tracks = findTracks(vSet);
```

Version History

Introduced in R2016a

R2020a: `viewSet` object will be removed

Not recommended starting in R2020a

The `viewSet` object will be removed in a future release. Manage views and pairwise connections between views of data using the `imageviewset` object instead. The `imageviewset` object adds additional support for simultaneous localization and mapping (SLAM) data,

data used in structure-from-motion, visual odometry, and simultaneous localization and mapping (SLAM) data. View attributes can be feature descriptors, feature points, or absolute camera poses.

Pairwise connections between views can be point matches, relative camera poses, or an information matrix.

See Also

`detectSURFFeatures` | `detectHarrisFeatures` | `detectMinEigenFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `detectMSERFeatures` | `matchFeatures` | `bundleAdjustment` | `triangulateMultiview` | `table` | `pointTrack`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Code Generation for Depth Estimation from Stereo Video”
“Using the Single Camera Calibrator App”
“What Is Structure from Motion?”

addView

(To be removed) Add new view to view set object

Note The `viewSet` object and its `addView` function will be removed in a future release. Use the `imageviewset` object and its `addView` function instead. For more information, see "Version History".

Syntax

```
vSet = addView(vSet,viewId)
vSet = addView(vSet,viewId,Name,Value)
```

Description

`vSet = addView(vSet,viewId)` adds the view specified by `viewID` to the specified `viewSet` object.

`vSet = addView(vSet,viewId,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Add View to View Set Object

Create an empty viewSet object.

```
vSet = viewSet;
```

Detect interest points in the image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add the points to the object.

```
vSet = addView(vSet,1,'Points',points,'Orientation',eye(3),'Location',[0,0,0]);
```

Input Arguments

vSet — View set object
`viewSet` object

`viewSet` object.

viewId — Camera pose view ID
integer

Camera pose view ID in the `viewSet` object, specified as an integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Location','[0,0,0]'`

`Points` — Image points
M-by-2 matrix | points object

Image points, specified as the comma-separated pair consisting of '`Points`' and an *M*-by-2 matrix of [*x*,*y*] coordinates or any points object.

`Orientation` — Absolute orientation of the camera
3-by-3 matrix

Absolute orientation of the camera, specified as the comma-separated pair consisting of '`Orientation`' and a 3-by-3 matrix.

`Location` — Absolute location of the camera
three-element vector

Absolute location of the camera, specified as the comma-separated pair consisting of '`Location`' and a three-element vector.

Output Arguments

`vSet` — View set object
`viewSet` object

`viewSet` object containing the added view specified by `viewId`.

Version History

Introduced in R2016a

R2020a: `addView` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `addView` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Add views to the `imageviewset` object using its `addView` function.

See Also

`addView` | `imageviewset`

updateView

(To be removed) Modify existing view in view set object

Note The `viewSet` object and its `updateView` function will be removed in a future release. Use the `imageviewset` object and its `updateView` function instead. For more information, see "Version History".

Syntax

```
vSet = updateView(vSet,viewId)
vSet = updateView(vSet,viewId,Name,Value)
vSet = updateView(vSet,views)
```

Description

`vSet = updateView(vSet,viewId)` modifies the view specified by `viewId` in the specified `viewSet` object, `vSet`.

`vSet = updateView(vSet,viewId,Name,Value)` specifies additional options using one or more name-value pair arguments.

`vSet = updateView(vSet,views)` modifies a view or a set of views specified by the `view` table.

Examples

Update View in View Set Object

Create an empty `viewSet` object.

```
vSet = viewSet;
```

Detect interest points in the image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(rgb2gray(I));
```

Add the points to the object.

```
vSet = addView(vSet,1,'Points',points);
```

Update the view to specify the camera pose.

```
vSet = updateView(vSet, 1,'Orientation',eye(3),'Location',[0,0,0]);
```

Input Arguments

`vSet` — View set object

`viewSet` object

`viewSet` object.

`viewId` — Camera pose view ID

integer

Camera pose view ID in the `viewSet` object, specified as an integer.

`views` — Camera views

table

Camera views, specified as a table. The table must contain a column named `ViewID`, and one or more columns named `Points`, `Orientation`, or `Location`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Location`, `'[0,0,0]'`

`Points` — Image points

M-by-2 matrix | points object

Image points, specified as the comma-separated pair consisting of '`Points`' and an *M*-by-2 matrix of [*x*,*y*] coordinates or any points object.

`Orientation` — Orientation of the second camera relative to the first camera

3-by-3 matrix

Orientation of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Orientation`' and a 3-by-3 matrix that represents the [*x*,*y*,*z*] orientation of the second camera.

`Location` — Location of the second camera relative to the first camera

three-element vector

Location of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Location`' and a three-element vector that represents the [*x*,*y*,*z*] location of the second camera in the first camera’s coordinate system.

Output Arguments

`vSet` — View set object

`viewSet` object

`viewSet` object containing the modified view specified by `viewId`.

Version History

Introduced in R2016a

R2020a: `updateView` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `updateView` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Update connections between views of the `imageviewset` object using its `updateView` function.

See Also

`imageviewset` | `updateView`

deleteView

(To be removed) Delete existing view from view set object

Note The `viewSet` object and its `deleteView` function will be removed in a future release. Use the `imageviewset` object and its `deleteView` function instead. For more information, see "Version History".

Syntax

```
vSet = deleteView(vSet,viewId)
```

Description

`vSet = deleteView(vSet,viewId)` deletes an existing view or a set of views from the specified `viewSet` object,`vSet`.

Examples

Delete a View from View Set Object

Create an empty viewSet object.

```
vSet = viewSet;
```

Detect interest points in the image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add a view.

```
vSet = addView(vSet,1,'Points',points);
```

Delete the view.

```
vSet = deleteView(vSet,1);
```

Input Arguments

vSet — View set object
`viewSet` object

A `viewSet` object.

viewId — View IDs
integer scalar | vector

View IDs, specified as an integer scalar for a single view, or as a vector of integers for a set of views.

Output Arguments

vSet — View set object
`viewSet` object

`viewSet` object.

Version History

Introduced in R2016a

R2020a: `deleteView` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `deleteView` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Delete views from the `imageviewset` object using its `deleteView` function.

See Also

`imageviewset` | `deleteView`

hasView

(To be removed) Check if view exists

Note The `viewSet` object and its `hasView` function will be removed in a future release. Use the `imageviewset` object and its `hasView` function instead. For more information, see "Version History".

Syntax

```
tf = hasView(vSet,viewId)
```

Description

`tf = hasView(vSet,viewId)` returns `1` if the view specified by `viewID` exists and `0` if it does not exist.

Examples

Check If View Exists

Create an empty viewSet object.

```
vSet = viewSet;
```

Detect interest points in the image.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I = imread(fullfile(imageDir,'image1.jpg'));
points = detectSURFFeatures(im2gray(I));
```

Add a new view.

```
vSet = addView(vSet, 1,'Points',points);
```

Confirm that the view with ID 1 exists.

```
tf = hasView(vSet,1);
```

Input Arguments

vSet — View set object
`viewSet` object

`viewSet` object.

viewId — View ID
integer

View ID in the `viewSet` object, specified as an integer.

Output Arguments

`tf` — Validity of view connection

logical

Validity of view connection, returned as a logical `1` or `0`.

Version History

Introduced in R2016a

R2020a: `hasView` function of `viewSet` object will be removed

Not recommended starting in R2020a

The `viewSet` object and its `hasView` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Check if a view exists in the `imageviewset` object using its `hasView` function.

See Also

`imageviewset` | `hasView`

addConnection

(To be removed) Add connection between two views

Note The `viewSet` object and its `addConnection` function will be removed in a future release. Use the `imageviewset` object and its `addConnection` function instead. For more information, see "Version History".

Syntax

```
vSet = addConnection(vSet,viewId1,viewId2)
vSet = addConnection(vSet,viewId1,viewId2,Name,Value)
```

Description

`vSet = addConnection(vSet,viewId1,viewId2)` adds a connection between two views in the specified `viewSet` object, `vSet`.

`vSet = addConnection(vSet,viewId1,viewId2,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Add Connection Between Two Views

Create an empty `viewSet` object.

```
vSet = viewSet;
```

Read a pair of images.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I1 = im2gray(imread(fullfile(imageDir,'image1.jpg')));
I2 = im2gray(imread(fullfile(imageDir,'image2.jpg')));
```

Detect interest points in the two images.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Add the points to the `viewSet` object.

```
vSet = addView(vSet,1,'Points',points1);
vSet = addView(vSet,2,'Points',points2);
```

Extract feature descriptors from both images.

```
features1 = extractFeatures(I1,points1);
features2 = extractFeatures(I2,points2);
```

Match features and store the matches.

```
indexPairs = matchFeatures(features1,features2);
```

Add the connection between the two views.

```
vSet = addConnection(vSet,1,2,'Matches',indexPairs);
```

Input Arguments

`vSet` — View set object

`viewSet` object

`viewSet` object.

`viewId1` — View ID 1

integer

View ID 1 in the `viewSet` object, specified as an integer.

`viewId2` — View ID 2

integer

View ID 2 in the `viewSet` object, specified as an integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Location','[0,0,0]'`

`Matches` — Indices of matched points between two views

M-by-2 matrix

Indices of matched points between two views, specified as the comma-separated pair consisting of '`Matches`' and an *M*-by-2 matrix.

`Orientation` — Orientation of the second camera relative to the first camera

3-by-3 matrix

Orientation of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Orientation`' and a 3-by-3 matrix that represents the [*x*,*y*,*z*] orientation of the second camera.

`Location` — Location of the second camera relative to the first camera

three-element vector

Location of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Location`' and a three-element vector that represents the [*x*,*y*,*z*] location of the second camera in the first camera's coordinate system.

Output Arguments

vSet — View set object
`viewSet` object

`viewSet` object.

Version History

Introduced in R2016a

R2020a: `addConnection` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `addConnection` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Add connections to the `imageviewset` object using its `addConnection` function.

See Also

`addConnection` | `imageviewset`

updateConnection

(To be removed) Modify connection between two views in view set object

Note The `viewSet` object and its `updateConnection` function will be removed in a future release. Use the `imageviewset` object and its `updateConnection` function instead. For more information, see “Version History”.

Syntax

```
vSet = updateConnection(vSet,viewId1,viewId2)
vSet = updateConnection(vSet,viewId1,viewId2,Name,Value)
```

Description

`vSet = updateConnection(vSet,viewId1,viewId2)` modifies a connection between two views in the specified view set object, `vSet`.

`vSet = updateConnection(vSet,viewId1,viewId2,Name,Value)` specifies additional options using one or more name-value pair arguments. Unspecified arguments have default values.

Examples

Update Connection Between Two Views in View Set Object

Create an empty `viewSet` object.

```
vSet = viewSet;
```

Read a pair of images.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I1 = rgb2gray(imread(fullfile(imageDir,'image1.jpg')));
I2 = rgb2gray(imread(fullfile(imageDir,'image2.jpg')));
```

Detect interest points in the two images.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Add the points to the `viewSet` object.

```
vSet = addView(vSet, 1,'Points',points1);
vSet = addView(vSet, 2,'Points',points2);
```

Extract feature descriptors.

```
features1 = extractFeatures(I1,points1);
features2 = extractFeatures(I2,points2);
```

Match features and store the matches.

```
indexPairs = matchFeatures(features1, features2);
vSet = addConnection(vSet,1,2,'Matches',indexPairs);
```

Update the connection to store a relative pose between the views.

```
vSet = updateConnection(vSet,1,2,'Orientation', eye(3),'Location',[1 0 0]);
```

Input Arguments

`vSet` — View set object

`viewSet` object

`viewSet` object.

`viewId1` — View ID 1

integer

View ID 1 in the `viewSet` object, specified as an integer.

`viewId2` — View ID 2

integer

View ID 2 in the `viewSet` object, specified as an integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Location', '[0,0,0]'`

`Matches` — Indices of matched points between two views

M-by-2 matrix

Indices of matched points between two views, specified as the comma-separated pair consisting of '`Matches`' and an *M*-by-2 matrix.

`Orientation` — Orientation of the second camera relative to the first camera

3-by-3 matrix

Orientation of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Orientation`' and a 3-by-3 matrix that represents the [*x*,*y*,*z*] orientation of the second camera.

`Location` — Location of the second camera relative to the first camera

three-element vector

Location of the second camera relative to the first camera, specified as the comma-separated pair consisting of '`Location`' and a three-element vector that represents the [*x*,*y*,*z*] location of the second camera in the first camera's coordinate system.

Output Arguments

vSet — View set object
`viewSet` object

`viewSet` object containing the modified connection.

Version History

Introduced in R2016a

R2020a: `updateConnection` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `updateConnection` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Update connections between views of the `imageviewset` object using its `updateConnection` function.

See Also

`imageviewset` | `updateConnection`

deleteConnection

(To be removed) Delete connection between two views from view set object

Note The `viewSet` object and its `deleteConnection` function will be removed in a future release. Use the `imageviewset` object and its `deleteConnection` function instead. For more information, see “Version History”.

Syntax

```
vSet = deleteConnection(vSet,viewId1,viewId2)
```

Description

`vSet = deleteConnection(vSet,viewId1,viewId2)` deletes a connection between two views in the specified `viewSet` object, `vSet`.

Examples

Delete a Connection Between Two Views in View Set Object

Create an empty viewSet object.

```
vSet = viewSet;
```

Read a pair of images.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
I1 = im2gray(imread(fullfile(imageDir,'image1.jpg')));
I2 = im2gray(imread(fullfile(imageDir,'image2.jpg')));
```

Detect interest points in the two images.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Add the points to the `viewSet` object.

```
vSet = addView(vSet,1,'Points',points1);
vSet = addView(vSet,2,'Points',points2);
```

Extract feature descriptors.

```
features1 = extractFeatures(I1,points1);
features2 = extractFeatures(I2,points2);
```

Match features and store the matches.

```
indexPairs = matchFeatures(features1, features2);
vSet = addConnection(vSet,1,2,'Matches',indexPairs);
```

Delete the connection between the views.

```
vSet = deleteConnection(vSet,1,2);
```

Input Arguments

`vSet` — View set object
`viewSet` object

`viewSet` object.

`viewId1` — View ID 1
integer

View ID 1 in the `viewSet` object, specified as an integer.

`viewId2` — View ID 2
integer

View ID 2 in the `viewSet` object, specified as an integer.

Output Arguments

`vSet` — View set object
`viewSet` object

`viewSet` object.

Version History

Introduced in R2016a

R2020a: `deleteConnection` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `deleteConnection` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Delete connections from the `imageviewset` object using its `deleteConnection` function.

See Also

`imageviewset` | `deleteConnection`

hasConnection

(To be removed) Check if connection exists between two views

Note The `viewSet` object and its `hasConnection` function will be removed in a future release. Use the `imageviewset` object and its `hasConnection` function instead. For more information, see "Version History".

Syntax

```
tf = hasConnection(vSet,viewId1,viewId2)
```

Description

`tf = hasConnection(vSet,viewId1,viewId2)` returns true if both views exist and have a connection.

Examples

Check Whether a Connection Exists Between Two Views

Create an empty `viewSet` object.

```
vSet = viewSet;
```

Add a pair of views.

```
vSet = addView(vSet,1);
vSet = addView(vSet,2);
```

Add a connection.

```
vSet = addConnection(vSet,1,2);
```

Confirm that the connection exists.

```
tf = hasConnection(vSet,1,2);
```

Input Arguments

vSet — View set object

`viewSet` object

`viewSet` object.

viewId1 — View ID 1

integer

View ID 1 in the `viewSet` object, specified as an integer.

`viewId2` — View ID 2
integer

View ID 2 in the `viewSet` object, specified as an integer.

Output Arguments

`tf` — Validity of view connection
logical

Validity of view connection, returned as a logical `1` or `0`.

Version History

Introduced in R2016a

R2020a: `hasConnection` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `hasConnection` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Check if a connection between views exists in the `imageviewset` object using its `hasConnection` function.

See Also

`imageviewset` | `hasConnection`

findTracks

(To be removed) Find matched points across multiple views

Note The `viewSet` object and its `findTracks` function will be removed in a future release. Use the `imageviewset` object and its `findTracks` function instead. For more information, see "Version History".

Syntax

```
tracks = findTracks(vSet)
tracks = findTracks(vSet,viewIds)
```

Description

`tracks = findTracks(vSet)` finds point tracks across multiple views.

`tracks = findTracks(vSet,viewIds)` finds point tracks across a subset of views.

Examples

Find Point Tracks Across Sequence of Images

Load images.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','structureFromMotion');
imds = imageDatastore(imageDir);
```

Compute features for the first image.

```
I = rgb2gray(read(imds));
pointsPrev = detectSURFFeatures(I);
[featuresPrev,pointsPrev] = extractFeatures(I,pointsPrev);
```

Create a `viewSet` object.

```
vSet = viewSet;
vSet = addView(vSet,1,'Points',pointsPrev);
```

Compute features and matches for the rest of the images.

```
for i = 2:length(imds.Files)
 I = rgb2gray(read(imds));
 points = detectSURFFeatures(I);
 [features, points] = extractFeatures(I,points);
 vSet = addView(vSet,i,'Points',points);
 pairsIdx = matchFeatures(featuresPrev,features);
 vSet = addConnection(vSet,i-1,i,'Matches',pairsIdx);
 featuresPrev = features;
end
```

Find point tracks.

```
tracks = findTracks(vSet);
```

Input Arguments

`vSet` — View set object
`viewSet` object

`viewSet` object.

`viewIds` — Subset of views
vector of integers

Subset of views in the `viewSet` object, specified as a vector of integers.

Output Arguments

`tracks` — Point track objects
array of `pointTrack` objects

Point track objects, returned as an array of `pointTrack` objects. Each track contains 2-D projections of the same 3-D world point.

Version History

Introduced in R2016a

R2020a: `findTracks` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `findTracks` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Finds point track across views of the `imageviewset` object using its `findTracks` function.

See Also

`imageviewset` | `findTracks`

poses

(To be removed) Returns camera poses associated to views

Note The `viewSet` object and its `poses` function will be removed in a future release. Use the `imageviewset` object and its `poses` function instead. For more information, see “Version History”.

Syntax

```
cameraPoses = poses(vSet)
cameraPoses = poses(vSet,viewIds)
```

Description

`cameraPoses = poses(vSet)` returns the camera poses that correspond to the views contained in the input `viewSet` object, `vSet`.

`cameraPoses = poses(vSet,viewIds)` returns the camera poses that correspond to a subset of views specified by the vector `viewIds`.

Examples

Retrieve Camera Poses from View Set Object

Create an empty `viewSet` object.

```
vSet = viewSet;
```

Add views to the object.

```
vSet = addView(vSet,1,'Orientation',eye(3),'Location',[0,0,0]);
vSet = addView(vSet,2,'Orientation',eye(3),'Location',[1,0,0]);
```

Retrieve the absolute camera poses.

```
camPoses = poses(vSet);
```

Input Arguments

`vSet` — View set object
`viewSet` object

`viewSet` object.

`viewIds` — View IDs
integer scalar | vector

View IDs, specified as an integer scalar for a single view, or as a vector of integers for a set of views.

Output Arguments

`cameraPoses` — Camera pose information
three-column table

Camera pose information, returned as a three-column table. The table contains columns for `ViewId`, `Orientation`, and `Location`. The view IDs correspond to the IDs in the `viewSet` object. The orientations are specified as 3-by-3 rotation matrices and locations are specified as three-element vectors. You can pass the `cameraPoses` table to the `triangulateMultiview` and the `bundleAdjustment` functions.

Version History

Introduced in R2016a

R2020a: `poses` function of `viewSet` object will be removed
Not recommended starting in R2020a

The `viewSet` object and its `poses` function will be removed in a future release. Instead, manage views and pairwise connections between views of data using use the `imageviewset` object. Return camera poses associated with views of the `imageviewset` object using its `poses` function.

See Also

`triangulateMultiview` | `bundleAdjustment` | `imageviewset` | `poses`

acfObjectDetector

Detect objects using aggregate channel features

Description

The `acfObjectDetector` object detects objects from an image using the aggregate channel features (ACF) object detector. To use the ACF detector on an image, pass the trained detector to the `detect` function.

The ACF object detector recognizes specific objects in images, based on the training images and the object ground truth locations used with the `trainACFObjectDetector` function.

Creation

Create an `acfObjectDetector` object by calling the `trainACFObjectDetector` function with training data.

```
detector = trainACFObjectDetector(trainingData,...)
```

Syntax

```
detector = acfObjectDetector(classifier,trainingOptions)
```

Description

`detector = acfObjectDetector(classifier,trainingOptions)` creates an ACF object detector based on the specified pretrained `classifier` and `trainingOptions`. You can use this syntax to recreate an ACF object detector for code generation.

Input Arguments

`classifier` — Pretrained `acfObjectDetector` object classifier
structure field

Pretrained `acfObjectDetector` object classifier, specified as a structure of an Adaboost classifier. You can obtain the classifier structure field by converting a pretrained ACF classifier object to a structure by using the `toStruct` function.

`trainingOptions` — Pretrained `acfObjectDetector` object training options
structure field

Pretrained `acfObjectDetector` object training options, specified as a structure of feature pyramid parameters. You can obtain the training option structure field by converting a pretrained ACF classifier object to a structure by using the `toStruct` function.

Properties

`ModelName` — Name of classification model

character vector | string scalar

Name of the classification model, specified as a character vector or string scalar. By default, the name is set to the heading of the second column of the `trainingData` table specified in the `trainACFObjectDetector` function. You can modify this name after creating your `acfObjectDetector` object.

Example: `'stopSign'`

`ObjectTrainingSize` — Size of training images

[*height width*] vector

This property is read-only.

Size of training images, specified as a [*height width*] vector.

Example: `[100 100]`

`NumWeakLearners` — Number of weak learners

integer

This property is read-only.

Number of weak learners used in the detector, specified as an integer. `NumWeakLearners` is less than or equal to the maximum number of weak learners for the last training stage. To restrict this maximum, you can use the `'MaxWeakLearners'` name-value pair in the `trainACFObjectDetector` function.

Object Functions

detect	Detect objects using ACF object detector
toStruct	Convert a trained aggregate channel features (ACF) object detector into structure

Examples

Train Stop Sign Detector Using ACF Object Detector

Use the `trainACFObjectDetector` with training images to create an ACF object detector that can detect stop signs. Test the detector with a separate image.

Load the training data.

```
load('stopSignsAndCars.mat')
```

Prefix the full path to the stop sign images.

```
stopSigns = fullfile(toolboxdir('vision'),'visiondata',stopSignsAndCars{:,1});
```

Create datastores to load the ground truth data for stop signs.

```
imds = imageDatastore(stopSigns);
blds = boxLabelDatastore(stopSignsAndCars(:,2));
```

Combine the image and box label datastores.

```
ds = combine(imds,blds);
```

Train the ACF detector. Set the number of negative samples to use at each stage to 2. You can turn off the training progress output by specifying `Verbose=false`, as a `Name-Value` argument.

```
acfDetector = trainACFObjectDetector(ds,NegativeSamplesFactor=2);
```

```
ACF Object Detector Training
The training will take 4 stages. The model size is 34x31.
Sample positive examples(~100% Completed)
Compute approximation coefficients...Completed.
Compute aggregated channel features...Completed.
--------------------------------------------
Stage 1:
Sample negative examples(~100% Completed)
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 19 weak learners.
--------------------------------------------
Stage 2:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 20 weak learners.
--------------------------------------------
Stage 3:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 54 weak learners.
--------------------------------------------
Stage 4:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 61 weak learners.
--------------------------------------------
ACF object detector training is completed. Elapsed time is 17.9982 seconds.
```

Test the ACF detector on a test image.

```
img = imread('stopSignTest.jpg');
[bboxes,scores] = detect(acfDetector,img);
```

Display the detection results and insert the bounding boxes for objects into the image.

```
for i = 1:length(scores)
   annotation = sprintf('Confidence = %.1f',scores(i));
   img = insertObjectAnnotation(img,'rectangle',bboxes(i,:),annotation);
end

figure
imshow(img)
```



Recreate ACF Object Detector for Code Generation

Load the ACF stop sign detector from the `stopSignDetector.mat` file, which is present in the current working folder as a supporting file.

```
stopSignDetector = load('stopSignDetectorACF.mat');
detector = stopSignDetector.detector
```

```
detector =

  acfObjectDetector with properties:

            ModelName: 'stopSign'
    ObjectTrainingSize: [34 31]
       NumWeakLearners: 61
```

Convert the detector into a structure by using the `toStruct` function.

```
detectorStruct = toStruct(detector);
```

To generate code, pass the structure to a MATLAB® function. Then inside the MATLAB function, create an identical ACF stop sign detector using the existing detector properties.

```
detector1 = acfObjectDetector(detectorStruct.Classifier,detectorStruct.TrainingOptions)
```

```
detector1 =

  acfObjectDetector with properties:

            ModelName: 'stopSign'
    ObjectTrainingSize: [34 31]
       NumWeakLearners: 61
```

You can pass `detector1` to the `detect` function as an input to detect stop signs from images.

Version History

Introduced in R2017a

References

[1] Dollar, P., R. Appel, S. Belongie, and P. Perona. "Fast Feature Pyramids for Object Detection." *Pattern Analysis and Machine Intelligence, IEEE Transactions*. Vol. 36, Issue 8, 2014, pp. 1532–1545.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use the `toStruct` function to pass the `acfObjectDetector` object into generated code. For more information, see “Generate Code for Detecting Objects in Images by Using ACF Object Detector”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Image Labeler | Video Labeler

Functions

`trainACFObjectDetector` | `detectPeopleACF` | `trainCascadeObjectDetector` | `groundTruth`

rcnnObjectDetector

(Not recommended) Detect objects using R-CNN deep learning detector

Note `rcnnObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see "Version History".

Description

The `rcnnObjectDetector` object detects objects from an image, using a R-CNN (region-based convolutional neural networks) object detector. To detect objects in an image, pass the trained detector to the `detect` function. To classify image regions, pass the detector to the `classifyRegions` function.

Use of the `rcnnObjectDetector` requires Statistics and Machine Learning Toolbox™ and Deep Learning Toolbox.

When using the `detect` or `classifyRegions` functions with `rcnnObjectDetector`, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see "GPU Computing Requirements" (Parallel Computing Toolbox).

Creation

Create a `rcnnObjectDetector` object by calling the `trainRCNNObjectDetector` function with training data (requires Deep Learning Toolbox).

```
detector = trainRCNNObjectDetector(trainingData,...)
```

Properties

`Network` — Convolutional neural network
`SeriesNetwork` | `DAGNetwork`

Convolutional neural network (CNN) used within the R-CNN detector, specified as a `SeriesNetwork` or `DAGNetwork` object.

`RegionProposalFcn` — Custom region proposal method
function handle

Custom region proposal method, specified as a function name. A custom function `proposalFcn` must have the following functional form:

```
[bboxes,scores] = proposalFcn(I)
```

The input argument `I` is an image. The function must return rectangular bounding boxes in an *M*-by-4 array. Each row of `bboxes` contains a four-element vector, [*x*,*y*,*width*,*height*], that specifies the

upper–left corner and size of a bounding box in pixels. The function must also return a score for each bounding box in an *M*-by-1 vector. Higher scores indicate that the bounding box is more likely to contain an object.

`ClassNames` — Object class names
cell array

Object class names, specified as a cell array. The array contains the names of the object classes the R-CNN detector was trained to find.

`BoxRegressionLayer` — Bounding box regression layer
character vector

This property is read-only.

Bounding box regression layer name, specified as a character vector. This property is set during training using the `BoxRegressionLayer` argument of `trainRCNNObjectDetector`.

Object Functions

detect	(Not recommended) Detect objects using R-CNN deep learning detector
classifyRegions	(Not recommended) Classify objects in image regions using R-CNN object detector

Examples

Train R-CNN Stop Sign Detector

Load training data and network layers.

```
load('rcnnStopSigns.mat', 'stopSigns', 'layers')
```

Add the image directory to the MATLAB path.

```
imDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondata',...
  'stopSignImages');
addpath(imDir);
```

Set network training options to use mini-batch size of 32 to reduce GPU memory usage. Lower the InitialLearnRate to reduce the rate at which network parameters are changed. This is beneficial when fine-tuning a pre-trained network and prevents the network from changing too rapidly.

```
options = trainingOptions('sgdm', ...
  'MiniBatchSize', 32, ...
  'InitialLearnRate', 1e-6, ...
  'MaxEpochs', 10);
```

Train the R-CNN detector. Training can take a few minutes to complete.

```
rcnn = trainRCNNObjectDetector(stopSigns, layers, options, 'NegativeOverlapRange', [0 0.3]);
```

```
*********************************************************************
Training an R-CNN Object Detector for the following object classes:

* stopSign
```

```
--> Extracting region proposals from 27 training images...done.

--> Training a neural network to classify objects in training data...

Training on single CPU.
Initializing input data normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:01 |       96.88% |       0.1651 |      1.0000e-06 |
|       2 |          50 |       00:00:15 |       96.88% |       0.0807 |      1.0000e-06 |
|       3 |         100 |       00:00:27 |       96.88% |       0.1340 |      1.0000e-06 |
|       5 |         150 |       00:00:38 |       96.88% |       0.0225 |      1.0000e-06 |
|       6 |         200 |       00:00:49 |       93.75% |       0.6584 |      1.0000e-06 |
|       8 |         250 |       00:01:00 |       93.75% |       0.5233 |      1.0000e-06 |
|       9 |         300 |       00:01:11 |      100.00% |   2.9456e-05 |      1.0000e-06 |
|      10 |         350 |       00:01:23 |      100.00% |       0.0009 |      1.0000e-06 |
|========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*********************************************************************
```

Test the R-CNN detector on a test image.

```
img = imread('stopSignTest.jpg');

[bbox, score, label] = detect(rcnn, img, MiniBatchSize=32);
```

Display strongest detection result.

```
[score, idx] = max(score);

bbox = bbox(idx, :);
annotation = sprintf('%s: (Confidence = %f)', label(idx), score);

detectedImg = insertObjectAnnotation(img, 'rectangle', bbox, annotation);

figure
imshow(detectedImg)
```



Remove the image directory from the path.

```
rmpath(imDir);
```

Resume Training an R-CNN Object Detector

Resume training an R-CNN object detector using additional data. To illustrate this procedure, half the ground truth data will be used to initially train the detector. Then, training is resumed using all the data.

Load training data and initialize training options.

```
load('rcnnStopSigns.mat', 'stopSigns', 'layers')

stopSigns.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
    stopSigns.imageFilename);

options = trainingOptions('sgdm', ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 1e-6, ...
    'MaxEpochs', 10, ...
    'Verbose', false);
```

Train the R-CNN detector with a portion of the ground truth.

```
rcnn = trainRCNNObjectDetector(stopSigns(1:10,:), layers, options, 'NegativeOverlapRange', [0 0.
```

Get the trained network layers from the detector. When you pass in an array of network layers to `trainRCNNObjectDetector`, they are used as-is to continue training.

```
network = rcnn.Network;
layers = network.Layers;
```

Resume training using all the training data.

```
rcnnFinal = trainRCNNObjectDetector(stopSigns, layers, options);
```

Create a Network for Multiclass R-CNN Object Detection

Create an R-CNN object detector for two object classes: dogs and cats.

```
objectClasses = {'dogs','cats'};
```

The network must be able to classify both dogs, cats, and a "background" class in order to be trained using `trainRCNNObjectDetector`. In this example, a one is added to include the background.

```
numClassesPlusBackground = numel(objectClasses) + 1;
```

The final fully connected layer of a network defines the number of classes that the network can classify. Set the final fully connected layer to have an output size equal to the number of classes plus a background class.

```
layers = [ ...
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    fullyConnectedLayer(numClassesPlusBackground);
    softmaxLayer()
    classificationLayer()];
```

These network layers can now be used to train an R-CNN two-class object detector.

Use A Saved Network In R-CNN Object Detector

Create an R-CNN object detector and set it up to use a saved network checkpoint. A network checkpoint is saved every epoch during network training when the `trainingOptions` 'CheckpointPath' parameter is set. Network checkpoints are useful in case your training session terminates unexpectedly.

Load the stop sign training data.

```
load('rcnnStopSigns.mat','stopSigns','layers')
```

Add full path to image files.

```
stopSigns.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
      stopSigns.imageFilename);
```

Set the 'CheckpointPath' using the `trainingOptions` function.

```
checkpointLocation = tempdir;
options = trainingOptions('sgdm','Verbose',false, ...
    'CheckpointPath',checkpointLocation);
```

Train the R-CNN object detector with a few images.

```
rcnn = trainRCNNObjectDetector(stopSigns(1:3,:),layers,options);
```

Load a saved network checkpoint.

```
wildcardFilePath = fullfile(checkpointLocation,'convnet_checkpoint__*.mat');
contents = dir(wildcardFilePath);
```

Load one of the checkpoint networks.

```
filepath = fullfile(contents(1).folder,contents(1).name);
checkpoint = load(filepath);

checkpoint.net
```

```
ans =

  SeriesNetwork with properties:

    Layers: [15×1 nnet.cnn.layer.Layer]
```

Create a new R-CNN object detector and set it up to use the saved network.

```
rcnnCheckPoint = rcnnObjectDetector();
rcnnCheckPoint.RegionProposalFcn = @rcnnObjectDetector.proposeRegions;
```

Set the Network to the saved network checkpoint.

```
rcnnCheckPoint.Network = checkpoint.net
```

```
rcnnCheckPoint =

  rcnnObjectDetector with properties:

            Network: [1×1 SeriesNetwork]
         ClassNames: {'stopSign'  'Background'}
    RegionProposalFcn: @rcnnObjectDetector.proposeRegions
```

Version History

Introduced in R2016b

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

References

[1] Girshick, Ross, et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” *2014 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2014, pp. 580–87. *DOI.org (Crossref)*, https://doi.org/10.1109/CVPR.2014.81.

[2] Girshick, Ross. “Fast R-CNN.” *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015, pp. 1440–48. *DOI.org (Crossref)*, https://doi.org/10.1109/ICCV.2015.169.

See Also

Apps
Image Labeler | Video Labeler

Functions
`fasterRCNNObjectDetector` | `vision.CascadeObjectDetector` | `selectStrongestBboxMulticlass`

Topics
“Image Category Classification Using Deep Learning”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”

bagOfFeatures

Bag of visual words object

Description

You can construct a bag of visual words for use in image category classification, and image retrieval.

Creation

Syntax

```
bag = bagOfFeatures(imds)
bag = bagOfFeatures(imds,'CustomExtractor',extractorFcn)
bag = bagOfFeatures(imds,Name=Value)
```

Description

`bag = bagOfFeatures(imds)` returns a bag of features object. The `bag` output object is generated using samples from the `imds` input. By default, the visual vocabulary is created from SURF features extracted from images in `imds`.

`bag = bagOfFeatures(imds,'CustomExtractor',extractorFcn)` returns a bag of features that uses a custom feature extractor function to extract features from images in `imds`. `extractorFcn` is a function handle to a custom feature extraction function.

`bag = bagOfFeatures(imds,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `bag = bagOfFeatures(imds,Verbose=true)` additionally sets `Verbose` to `true`.

This object supports parallel computing using multiple MATLAB workers. Enable parallel computing from the “Computer Vision Toolbox Preferences” dialog box. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Then select Computer Vision Toolbox.

Input Arguments

`imds` — Images
`ImageDatastore` object

Images, specified as an `ImageDatastore` object. The `bagOfFeatures` extracts an equal number of strongest features from the images contained in the `imds` object. The number of strongest features is defined as:

number of strongest features = `min`(*number of features found in each set*) x `StrongestFraction`

The object obtains the `StrongestFraction` value from the `StrongestFeatures` property.

`extractorFcn` — Custom feature extractor function
function handle

Custom feature extractor function, specified as a function handle. This custom function extracts features to learn the visual vocabulary of the object.

The function, `extractorFcn`, must be specified as a function handle for a file:

```
extractorFcn = @exampleBagOfFeaturesExtractor;
bag = bagOfFeatures(imds,CustomExtractor=extractorFcn)
```

where `exampleBagOfFeaturesExtractor` is a MATLAB function. For example:

```
function [features,featureMetrics,location] = exampleBagOfFeaturesExtractor(img)
...
```

The function must be on the path or in the current working directory. The arguments are defined as:

Argument	Input/Output	Description
`img`	Input	• Binary, grayscale, or truecolor image. • The input image is from the image set that was originally passed into `bagOfFeatures`.
`features`	Output	• A `binaryFeatures` object. • An *M*-by-*N* numeric matrix of image features, where *M* is the number of features and *N* is the length of each feature vector. • The feature length, *N*, must be greater than zero and be the same for all images processed during the `bagOfFeatures` creation process. • If you cannot extract features from an image, supply an empty feature matrix and an empty feature metrics vector. Use the empty matrix and vector if, for example, you did not find any keypoints for feature extraction. • Numeric, real, and nonsparse.
`featureMetrics`	Output	• An *M*-by-1 vector of feature metrics indicating the strength of each feature vector. • Used to apply the `'SelectStrongest'` criteria in `bagOfFeatures` framework. • Numeric, real, and nonsparse.
`location`	Output	• An *M*-by-2 matrix of 1-based [*x* *y*] values. • The [*x* *y*] values can be fractional. • Numeric, real, and nonsparse.

For more details on the custom extractor function and its input and output requirements, see “Create a Custom Feature Extractor”.

You can open an example function file, and use it as a template by typing the following command at the MATLAB command-line:

```
edit('exampleBagOfFeaturesExtractor.m')
```

Properties

`CustomExtractor` — Custom extraction function

function handle

Custom feature extractor function, specified as a handle to a function. The custom feature extractor function extracts features used to learn the visual vocabulary for `bagOfFeatures`. You must specify '`CustomExtractor`' and the function handle, `extractorFcn`, to a custom feature extraction function.

The function, `extractorFcn`, must be specified as a function handle for a file:

```
extractorFcn = @exampleBagOfFeaturesExtractor;
bag = bagOfFeatures(imds,CustomExtractor=extractorFcn)
```

where `exampleBagOfFeaturesExtractor` is a MATLAB function such as:

```
function [features,featureMetrics] = exampleBagOfFeaturesExtractor(img)
...
```

The function must be on the path or in the current working directory.

For more details on the custom extractor function and its input and output requirements, see “Create a Custom Feature Extractor”. You can open an example function file, and use it as a template by typing the following command at the MATLAB command-line:

```
edit('exampleBagOfFeaturesExtractor.m')
```

`TreeProperties` — Vocabulary tree properties

`[1 500]` (default) | two-element vector

Vocabulary tree properties, specified as a two-element vector in the form [*numLevels branchingFactor*]. *numLevels* is an integer that specifies the number of levels in the vocabulary tree. *branchingFactor* is an integer that specifies a factor to control the amount the vocabulary can grow at successive levels in the tree. The maximum number of visual words represented by the vocabulary tree is *branchingFactor*^*numLevels*. Typical values for *numLevels* is between `1` and `6`. Typical values for *branchingFactor* is between `10` and `500`. Use an empirical analysis to select optimal values.

Increase the branching factor to generate a larger vocabulary. Increasing the vocabulary improves classification and image retrieval accuracy, but will also increase the time to encode images. You can use a vocabulary tree with multiple levels to create vocabularies on the order of 10,000 visual words or more. A multilevel tree reduces the time required to encode images with large vocabularies, but will take longer to create. You can use a tree with one level for vocabularies that contain only 100 - 1000 visual words.

`StrongestFeatures` — Fraction of strongest features

`0.8` (default) | `[0,1]`

Fraction of strongest features, specified as the comma-separated pair consisting of '`StrongestFeatures`' and a value in the range [0,1]. The value represents the fraction of strongest features to use from each label in the `imds` input.

`Verbose` — Enable progress display to screen

`true` (default) | false

Enable progress display to screen, specified as the comma-separated pair consisting of 'Verbose' and the logical `true` or `false`.

`PointSelection` — Selection method for picking point locations
`"Grid"` (default) | `"Detector"`

Selection method for picking point locations for SURF feature extraction, specified as the comma-separated pair consisting of 'PointSelection' and either `"Grid"` or `"Detector"`. There are two stages for feature extraction. First, you select a method for picking the point locations, (SURF `"Detector"` or `"Grid"`), with the `PointSelection` property. The second stage extracts the features. The feature extraction uses a SURF extractor for both point selection methods.

When you set `PointSelection` to `"Detector"`, the feature points are selected using a speeded up robust feature (SURF) detector. Otherwise, the points are picked on a predefined grid with spacing defined by 'GridStep'. This property applies only when you are not specifying a custom extractor with the `CustomExtractor` property.

`GridStep` — Grid step size
`[8 8]` (default) | 1-by-2 [*x y*] vector

Grid step size in pixels, specified as the comma-separated pair consisting of `"Grid"` and an 1-by-2 [*x y*] vector. This property applies only when you set `PointSelection` to `"Grid"` and you are not specifying a custom extractor with the `CustomExtractor` property. The steps in the *x* and *y* directions define the spacing of a uniform grid. Intersections of the grid lines define locations for feature extraction.

`BlockWidth` — Patch size to extract upright SURF descriptor
`[32 64 96 128]` (default) | 1-by-*N* vector

Patch size to extract upright SURF descriptor, specified as the comma-separated pair consisting of 'BlockWidth' and a 1-by-*N* vector of *N* block widths. This property applies only when you are not specifying a custom extractor with the `CustomExtractor` property. Each element of the vector corresponds to the size of a square block from which the function extracts upright SURF descriptors. Use multiple square sizes to extract multiscale features. All the square specified are used for each extraction points on the grid. This property only applies when you set `PointSelection` to `"Grid"`. The block width corresponds to the scale of the feature. The minimum `BlockWidth` is 32 pixels.



`Upright` — Orientation of SURF feature vector
`true` (default) | `false`

Orientation of SURF feature vector, specified as a logical scalar. This property applies only when you are not specifying a custom extractor with the `CustomExtractor` property. Set this property to `true`

when you do not need to estimate the orientation of the SURF feature vectors. Set it to `false` when you need the image descriptors to capture rotation information.

Object Functions

encode Create histogram of visual word occurrences

Examples

Create a Bag of Visual Words

Create an image datastore.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Create the bag of features. This process can take a few minutes.

```
bag = bagOfFeatures(imds);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 12 images...done. Extracted 230400 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 184320
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 54/100 iterations (~2.44 seconds/iteration)...converged in 54 iterations

* Finished creating Bag-Of-Features
```

Compute histogram of visual word occurrences for one of the images. Store the histogram as feature vector.

```
img = readimage(imds, 1);
featureVector = encode(bag,img);
```

```
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
```

Create a Bag of Features with a Custom Feature Extractor

Create an image datastore.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Specify a custom feature extractor.

```
extractor = @exampleBagOfFeaturesExtractor;
bag = bagOfFeatures(imds,'CustomExtractor',extractor)
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Extracting features using a custom feature extraction function: exampleBagOfFeaturesExtractor.

* Extracting features from 12 images...done. Extracted 230400 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 184320
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 50/100 iterations (~2.40 seconds/iteration)...converged in 50 iterations

* Finished creating Bag-Of-Features

bag =
  bagOfFeatures with properties:

      CustomExtractor: @exampleBagOfFeaturesExtractor
       NumVisualWords: 500
       TreeProperties: [1 500]
    StrongestFeatures: 0.8000
```

Version History

Introduced in R2014b

References

[1] Csurka, G., D. Christopher, F. Lixin, W. Jutta, and B. Cédric. "Visual categorization with bags of keypoints." *Workshop on statistical learning in computer vision, ECCV,* 2004, pp. 1-2.

[2] Nister, D., and H. Stewenius. "Scalable Recognition with a Vocabulary Tree." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006, vol. 2, pp. 2161–2168.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

For code generation, the `bagOfFeatures` function does not support the `ImageDatastore` object as input. Instead, specify the input as a structure with fields `Images` and `Labels`. The `Images` field must contain a cell array of grayscale or RGB color images. The `Labels` field must contain image labels stored as categorical arrays.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

Objects

`imageCategoryClassifier` | `imageDatastore` | `bagOfFeaturesDBoW` | `dbowLoopDetector`

Functions

`trainImageCategoryClassifier`

Topics

“Monocular Visual Simultaneous Localization and Mapping”
“Image Category Classification Using Bag of Features”
“Image Classification with Bag of Visual Words”
“Create a Custom Feature Extractor”

encode

Create histogram of visual word occurrences

Syntax

```
featureVector = encode(bag,I)
[featureVector,words] = encode(bag,I)

featureVector= encode(bag,imds)
[featureVector,words] = encode(bag,imds)

[___] = encode(___,Name,Value)
```

Description

`featureVector = encode(bag,I)` returns a feature vector that represents a histogram of visual word occurrences contained in the input image, `I`. The input bag contains the `bagOfFeatures` object.

`[featureVector,words] = encode(bag,I)` optionally returns the visual words as a `visualWords` object. The `visualWords` object stores the visual words that occur in `I` and stores the locations of those words.

`featureVector= encode(bag,imds)` returns a feature vector that represents a histogram of visual word occurrences contained in `imds`. The input `bag` contains the `bagOfFeatures` object.

`[featureVector,words] = encode(bag,imds)` optionally returns an array of `visualWords` occurrences in `imds`. The `visualWords` object stores the visual words that occur in `I` and stores the locations of those words.

`[___] = encode(___,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `encode('SparseOutput',false)`

This method supports parallel computing using multiple MATLAB workers. Enable parallel computing from the “Computer Vision Toolbox Preferences” dialog box. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Then select **Computer Vision Toolbox**.

Examples

Encode an Image into a Feature Vector

Load a set of image.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Pick the first two images from each label.

```
trainingSet = splitEachLabel(imds,2);
```

Create bag of features.

```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 4 images...done. Extracted 76800 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 61440
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 49/100 iterations (~0.91 seconds/iteration)...converged in 49 iterations

* Finished creating Bag-Of-Features
```

Encode one of the images into a feature vector.

```
img = readimage(trainingSet,1);
featureVector = encode(bag,img);
```

```
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
```

Input Arguments

bag — Bag of features

`bagOfFeatures` object

Bag of features, specified as a `bagOfFeatures` object.

I — Input image

grayscale image | truecolor image

Input image, `I`, specified as a grayscale or a truecolor image.

imds — Images

`ImageDatastore` object

Images, specified as an `ImageDatastore` object.

Name-Value Pair Arguments

Example: 'SparseOutput',`false`

`Normalization` — Type of normalization
`'L2'` (default) | `'none'`

Type of normalization applied to the feature vector, specified as the comma-separated pair consisting of '`Normalization`' and either `'L2'` or `'none'`.

`SparseOutput` — Output sparsity
`false` (default) | `true`

Output sparsity, specified as the comma-separated pair consisting of '`SparseOutput`' and as `true` or `false`. Set this property to `true` to return the visual word histograms in a sparse matrix. Setting this property to `true` reduces memory consumption for large visual vocabularies where the visual word histograms contain many zero elements.

`Verbose` — Enable progress display to screen
`true` (default) | false

Enable progress display to screen, specified as the comma-separated pair consisting of '`Verbose`' and the logical `true` or `false`.

Output Arguments

`featureVector` — Histogram of visual word occurrences
1-by-`bag.VocabularySize` | *M*-by-`bag.VocabularySize`

Histogram of visual word occurrences, specified as *M*-by-`bag.VocabularySize` vector, where *M* is the total number of images in `imds`, `numel(imds.Files)`.

`words` — Visual words object
`visualWords` object

Visual words object, returned as a visual words object or an array of visual words objects. The `visualWords` object stores the visual words that occur in the images and stores the locations of those words.

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For code generation, the function does not support the `ImageDatastore` object as input.
- To input more than one image, specify them as a cell array.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

`bagOfFeatures`

imageCategoryClassifier

Predict image category

Description

The `imageCategoryClassifier` object contains a linear support vector machine (SVM) classifier trained to recognize an image category.

You must have a Statistics and Machine Learning Toolbox license to use this classifier. This classifier supports parallel computing using multiple MATLAB workers. Enable parallel computing using the "Computer Vision Toolbox Preferences" dialog. To open the Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Select **Computer Vision System Toolbox**.

Creation

Use the `trainImageCategoryClassifier` function to create the `imageCategoryClassifier` object.

```
bag = bagOfFeatures(trainingSet);
categoryClassifier = trainImageCategoryClassifier(trainingSet,bag);
```

Properties

`Labels` — Category labels
cell array

Category labels, specified as a cell array.

`NumCategories` — Number of trained categories
integer

Number of trained categories, stored as an integer value.

Object Functions

evaluate	Evaluate image classifier on collection of image sets
predict	Predict image category

Examples

Train, Evaluate, and Apply Image Category Classifier

Load two image categories.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Split the data set into a training and test data. Pick 30% of images from each set for the training data and the remainder 70% for the test data.

```
[trainingSet,testSet] = splitEachLabel(imds,0.3,'randomize');
```

Create bag of visual words.

```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 4 images...done. Extracted 76800 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 61440
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 55/100 iterations (~0.92 seconds/iteration)...converged in 55 iterations

* Finished creating Bag-Of-Features
```

Train a classifier with the training sets.

```
categoryClassifier = trainImageCategoryClassifier(trainingSet,bag);
```

```
Training an image category classifier for 2 categories.
-------------------------------------------------------
* Category 1: books
* Category 2: cups

* Encoding features for 4 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Evaluate the classifier using test images. Display the confusion matrix.

```
confMatrix = evaluate(categoryClassifier,testSet)
```

```
Evaluating image category classifier for 2 categories.
-------------------------------------------------------

* Category 1: books
* Category 2: cups

* Evaluating 8 images...done.
```

```
* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


             PREDICTED
KNOWN    | books   cups
--------------------------
books    | 0.75    0.25
cups     | 0.25    0.75

* Average Accuracy is 0.75.

confMatrix = 2×2

    0.7500    0.2500
    0.2500    0.7500
```

Find the average accuracy of the classification.

```
mean(diag(confMatrix))
```

```
ans =
0.7500
```

Apply the newly trained classifier to categorize new images.

```
img = imread(fullfile(setDir,'cups','bigMug.jpg'));
[labelIdx, score] = predict(categoryClassifier,img);
```

```
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
```

Display the classification label.

```
categoryClassifier.Labels(labelIdx)
```

```
ans = 1×1 cell array
    {'cups'}
```

Version History

Introduced in R2014b

References

[1] Csurka, G., C. R. Dance, L. Fan, J. Willamowski, and C. Bray *Visual Categorization with Bag of Keypoints*, Workshop on Statistical Learning in Computer Vision, ECCV 1 (1-22), 1-2.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

`imageDatastore` | `bagOfFeatures` | `trainImageCategoryClassifier` | `fitcecoc`

Topics

“Image Category Classification Using Bag of Features”
“Image Classification with Bag of Visual Words”

predict

Predict image category

Syntax

```
[labelIdx,score] = predict(categoryClassifier,I)
[labelIdx,score] = predict(categoryClassifier,imds)
[labelIdx,score] = predict( ___ ,'Verbose',true)
```

Description

`[labelIdx,score] = predict(categoryClassifier,I)` returns the predicted label index and score for the input image.

`predict` supports parallel computing using multiple MATLAB workers. Enable parallel computing using the “Computer Vision Toolbox Preferences” dialog. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Select Computer Vision Toolbox.

`[labelIdx,score] = predict(categoryClassifier,imds)` returns the predicted label index and score for the images specified in `imds`.

`[labelIdx,score] = predict( ___ ,'Verbose',true)` also enables progress display to the screen. Set to `false` to turn it off.

Examples

Predict Category for Image

Load two image category sets.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Separate the two sets into training and test data. Pick 30% of images from each set for the training data and the remainder 70% for the test data.

```
[trainingSet,testSet] = splitEachLabel(imds,0.3,'randomize');
```

Create a bag of visual words.

```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
```

```
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 4 images...done. Extracted 76800 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 61440
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 55/100 iterations (~0.94 seconds/iteration)...converged in 55 iterations

* Finished creating Bag-Of-Features
```

Train a classifier.

```
categoryClassifier = trainImageCategoryClassifier(trainingSet,bag);

Training an image category classifier for 2 categories.
--------------------------------------------------------
* Category 1: books
* Category 2: cups

* Encoding features for 4 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Predict category label for one of the images in test set.

```
img = readimage(testSet,1);
[labelIdx, score] = predict(categoryClassifier,img);

Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.

categoryClassifier.Labels(labelIdx)

ans = 1×1 cell array
    {'books'}
```

Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image, specified as either an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-D grayscale image.

`categoryClassifier` — Image category classifier

`imageCategoryClassifier` object

Image category classifier, specified as an `imageCategoryClassifier` object.

`imds` — Images
`ImageDatastore` object

Images, specified as an `ImageDatastore` object.

Output Arguments

`labelIdx` — Predicted label index
M-by-1 vector | scalar

Predicted label index, returned as either an *M*-by-1 vector for *M* images or a scalar value for a single image. The `labelIdx` output value corresponds to the index of an image set used to train the bag of features. The prediction index corresponds to the class with the lowest average binary loss of the ECOC SVM classifier.

`score` — Prediction score
1-by-*N* vector | *M*-by-*N* matrix

Prediction score, specified as a 1-by-*N* vector or an *M*-by-*N* matrix. *N* represents the number of classes. *M* represents the number of images in the `imageSet` input object, `imgSet`. The score provides a negated average binary loss per class. Each class is a support vector machine (SVM) multiclass classifier that uses the error-correcting output codes (ECOC) approach.

Version History

Introduced in R2014b

Extended Capabilities

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

`imageCategoryClassifier` | `evaluate`

evaluate

Evaluate image classifier on collection of image sets

Syntax

```
confMat = evaluate(categoryClassifier,imds)
[confMat,knownLabelIdx,predictedLabelIdx,score] = evaluate(
categoryClassifier,imds)
```

Description

`confMat = evaluate(categoryClassifier,imds)` returns a normalized confusion matrix, `confMat`.

`[confMat,knownLabelIdx,predictedLabelIdx,score] = evaluate(categoryClassifier,imds)` additionally returns the corresponding label indexes and score.

Examples

Train, Evaluate, and Apply Image Category Classifier

Load two image categories.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Split the data set into a training and test data. Pick 30% of images from each set for the training data and the remainder 70% for the test data.

```
[trainingSet,testSet] = splitEachLabel(imds,0.3,'randomize');
```

Create bag of visual words.

```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 4 images...done. Extracted 76800 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
```

```
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 61440
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 55/100 iterations (~0.92 seconds/iteration)...converged in 55 iterations

* Finished creating Bag-Of-Features
```

Train a classifier with the training sets.

```
categoryClassifier = trainImageCategoryClassifier(trainingSet,bag);
```

```
Training an image category classifier for 2 categories.
--------------------------------------------------------
* Category 1: books
* Category 2: cups

* Encoding features for 4 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Evaluate the classifier using test images. Display the confusion matrix.

```
confMatrix = evaluate(categoryClassifier,testSet)
```

```
Evaluating image category classifier for 2 categories.
-------------------------------------------------------

* Category 1: books
* Category 2: cups

* Evaluating 8 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


             PREDICTED
KNOWN    | books   cups
-------------------------
books    | 0.75    0.25
cups     | 0.25    0.75

* Average Accuracy is 0.75.

confMatrix = 2×2

    0.7500    0.2500
    0.2500    0.7500
```

Find the average accuracy of the classification.

```
mean(diag(confMatrix))
```

```
ans =
0.7500
```

Apply the newly trained classifier to categorize new images.

```
img = imread(fullfile(setDir,'cups','bigMug.jpg'));
[labelIdx, score] = predict(categoryClassifier,img);
```

```
Encoding images using Bag-Of-Features.
----------------------------------------
* Encoding an image...done.
```

Display the classification label.

```
categoryClassifier.Labels(labelIdx)
```

```
ans = 1×1 cell array
    {'cups'}
```

Input Arguments

`imds` — Data store object of images

`ImageDatastore` object

Images, specified in an `ImageDatastore` object.

`categoryClassifier` — Image category classifier

`imageCategoryClassifier` object

Image category classifier, specified as an `imageCategoryClassifier` object.

Output Arguments

`confMat` — Confusion matrix

matrix

Confusion matrix, returned as a matrix. The row indices correspond to known labels and the columns correspond to the predicted labels.

`knownLabelIdx` — Label index for image set

M-by-1 vector | scalar

Label index for image set, returned as an *M*-by-1 vector for *M* images. The `knownLabelIdx` output value corresponds to the index of an image set used to train the bag of features.

`predictedLabelIdx` — Predicted label index

M-by-1 vector

Predicted label index, returned as an *M*-by-1 vector for *M* images. The `predictedLabelIdx` output value corresponds to the index of an image set used to train the bag of features. The predicted index corresponds to the class with the largest value in the `score` output.

`score` — Prediction score

M-by-*N* matrix

Prediction score, specified as an *M*-by-*N* matrix. *N* represents the number of classes. *M* represents the number of images in the `imageSet` input object, `imgSet`. The score provides a negated average

binary loss per class. Each class is a support vector machine (SVM) multiclass classifier that uses the error-correcting output codes (ECOC) approach.

Version History

Introduced in R2014b

See Also

`imageCategoryClassifier` | `predict`

intrinsicsEstimationErrors

Object for storing standard errors of estimated camera intrinsics and distortion coefficients

Description

The `intrinsicsEstimationErrors` object contains the standard errors of estimated camera intrinsics and distortion coefficients. You can access the intrinsics and distortion standard errors using the object properties. You can display the standard errors using the `displayErrors` object function.

Creation

Syntax

```
[params,imgsUsed,Errors] = estimateCameraParameters(imagePoints,worldPoints)
```

Description

`[params,imgsUsed,Errors] = estimateCameraParameters(imagePoints,worldPoints)` creates an `intrinsicsEstimationErrors` object by calling the `estimateCameraParameters` function with image points and world points. The function returns a `stereoCalibrationErrors` or `cameraCalibrationErrors` object as the third output argument, which contains the `intrinsicsEstimationErrors` as a read-only property.

Properties

`SkewError` — Standard error of camera axes skew estimate
scalar

This property is read-only.

Standard error of the camera axes skew estimate, specified as a scalar.

`FocalLengthError` — Standard error of focal length estimate
scalar

Standard error of the focal length estimate, specified as a scalar.

`PrincipalPointError` — Standard error of principal point estimate
scalar

This property is read-only.

Standard error of the principal point estimate, specified as a scalar.

`RadialDistortionError` — Standard error of radial distortion estimate
scalar

This property is read-only.

Standard error of the radial distortion estimate, specified as a scalar.

`TangentialDistortionError` — Standard error of tangential distortion estimate
scalar

This property is read-only.

Standard error of the tangential distortion estimate, specified as a scalar.

Examples

Display Intrinsics Errors

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
 "calibration","mono"));
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the corners of the squares. Square size of the checkerboard is 29 millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1) size(I,2)];
[params,~,errors] = estimateCameraParameters(imagePoints,worldPoints, ...
                                    ImageSize=imageSize);
```

Display the standard intrinsics errors.

```
displayErrors(errors.IntrinsicsErrors,params);
```

```
Focal length (pixels):   [  714.1884 +/- 3.3219      710.3783 +/- 4.0579  ]
Principal point (pixels):[  563.6478 +/- 5.3967      355.7255 +/- 3.3036  ]
Radial distortion:       [   -0.3536 +/- 0.0091        0.1730 +/- 0.0488  ]
```

Version History

Introduced in R2013b

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Objects
`cameraCalibrationErrors` | `stereoCalibrationErrors` | `extrinsicsEstimationErrors`

Topics
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“Evaluating the Accuracy of Single Camera Calibration”

extrinsicsEstimationErrors

Object for storing standard errors of estimated camera extrinsics and distortion coefficients

Description

The `extrinsicsEstimationErrors` object contains the standard errors of estimated camera extrinsics. You can access the extrinsics standard errors using the object properties. You can display the standard errors using the `displayErrors` object function.

Creation

Syntax

```
[params,imgsUsed,Errors] = estimateCameraParameters(imagePoints,worldPoints)
```

Description

`[params,imgsUsed,Errors] = estimateCameraParameters(imagePoints,worldPoints)` creates an `extrinsicsEstimationErrors` object by calling the `estimateCameraParameters` function with image points and world points. The function returns a `stereoCalibrationErrors` or `cameraCalibrationErrors` object as the third argument, which contains the `extrinsicsEstimationErrors` as a read-only property.

Properties

`RotationVectorsError` — Standard error of the camera rotations estimate
scalar

This property is read-only.

Standard error of the camera rotations estimate, specified as a scalar.

`TranslationVectorsError` — Standard error of the camera translations estimate
scalar

Standard error of the camera translations estimate, specified as a scalar.

Examples

Display Extrinsics Errors

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
 "calibration","mono"));
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the corners of the squares. Square size of the checkerboard is 29 millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1),size(I,2)];
[params,~,errors] = estimateCameraParameters(imagePoints,worldPoints, ...
                                    ImageSize=imageSize);
```

Display the standard extrinsics errors.

```
displayErrors(errors.ExtrinsicsErrors,params);
```

```
Rotation vectors:
                          [   -0.6096 +/- 0.0054       -0.1789 +/- 0.0073       -0.3835 +/- 0.0024
                          [   -0.7283 +/- 0.0050       -0.0996 +/- 0.0072        0.1964 +/- 0.0027
                          [   -0.6722 +/- 0.0051       -0.1444 +/- 0.0074       -0.1329 +/- 0.0026
                          [   -0.5836 +/- 0.0056       -0.2901 +/- 0.0074       -0.5622 +/- 0.0025
                          [   -0.3157 +/- 0.0065       -0.1441 +/- 0.0075       -0.1067 +/- 0.0011
                          [   -0.7581 +/- 0.0052        0.1947 +/- 0.0072        0.4324 +/- 0.0030
                          [   -0.7515 +/- 0.0051        0.0767 +/- 0.0072        0.2070 +/- 0.0029
                          [   -0.6223 +/- 0.0053        0.0231 +/- 0.0073        0.3663 +/- 0.0024
                          [    0.3443 +/- 0.0063       -0.2226 +/- 0.0073       -0.0437 +/- 0.0014

Translation vectors (mm):
                          [ -146.0514 +/- 6.0391      -26.8688 +/- 3.7318      797.9024 +/- 3.9002
                          [ -209.4354 +/- 6.9637      -59.4568 +/- 4.3578      921.8196 +/- 4.6295
                          [ -129.3821 +/- 7.0907      -44.1033 +/- 4.3751      937.6829 +/- 4.4913
                          [ -151.0046 +/- 6.6905      -27.3256 +/- 4.1339      884.2787 +/- 4.3925
                          [ -174.9496 +/- 6.7056      -24.3502 +/- 4.1606      886.4959 +/- 4.6680
                          [ -134.3092 +/- 7.8887     -103.4984 +/- 4.8925     1042.4551 +/- 4.8184
                          [ -173.9842 +/- 7.6891      -73.1694 +/- 4.7812     1017.2383 +/- 4.8120
                          [ -202.9443 +/- 7.4327      -87.9094 +/- 4.6482      983.6955 +/- 4.9072
                          [ -319.8858 +/- 6.3213     -119.8901 +/- 4.0922      829.4580 +/- 4.9591
```

Version History

Introduced in R2013b

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Objects

`cameraCalibrationErrors` | `stereoCalibrationErrors` | `intrinsicsEstimationErrors`

Functions

`estimateCameraParameters`

Topics
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“Evaluating the Accuracy of Single Camera Calibration”

BRISKPoints

Object for storing BRISK interest points

Description

This object provides the ability to pass data between the `detectBRISKFeatures` and `extractFeatures` functions. You can also use it to manipulate and plot the data returned by these functions. You can use the object to fill the points interactively in situations where you might want to mix a non-BRISK interest point detector with a BRISK descriptor.

Creation

Syntax

```
points = BRISKPoints(location)
points = BRISKPoints(location,Name=Value)
```

Description

`points = BRISKPoints(location)` constructs a `BRISKPoints` object from `location`, an *M*-by-2 array of [x y] point coordinates.

`points = BRISKPoints(location,Name=Value)` sets properties using one or more name-value arguments, in addition to all input arguments from the previous syntax.

Input Arguments

`location` — Point locations

M-by-2 array

Point locations, specified as an *M*-by-2 array of [*x* *y*] point coordinates.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `points = BRISKPoints(location,Metric=12.0)` specifies a radius of `12.0` (in pixels) of the BRISK sampling pattern used during feature extraction.

`Scale` — Scale

`12.0` (default) | scalar

Scale at which the interest points are detected, specified by the radius (in pixels) of the BRISK sampling pattern used during feature extraction.

`Metric` — Strength of detected feature

`0.0` (default) | numeric scalar

Strength of detected feature, specified as a numeric value. The BRISK algorithm uses the FAST corner score as a metric.

`Orientation` — Orientation of detected feature

`0.0` (default) | angle in radians

Orientation of the detected feature, specified as an angle, in radians. The angle is measured counterclockwise from the *x*-axis with the origin specified by the `Location` property. Do not set this property manually. Use the call to `extractFeatures` to fill in this value. The `extractFeatures` function modifies the default value of `0.0`. Using BRISK interest points to extract a non-BRISK descriptor (e.g. SURF, FREAK, MSER, etc.) can alter `Orientation` values. The `Orientation` is mainly useful for visualization purposes.

Properties

`Count` — Number of points

`0` (default) | integer

This property is read-only.

Number of points held by the BRISK object, specified as a numeric value.

`Location` — Point locations

M-by-2 array

Point locations, specified as an *M*-by-2 array of [x y] point coordinates.

Object Functions

plot	Plot detected feature points
isempty	Determine if points object is empty
length	Number of stored points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation

Examples

Detect BRISK Features in an Image

Read an image and detect the BRISK interest points.

```
I = imread('cameraman.tif');
points = detectBRISKFeatures(I);
```

Select and plot the 10 strongest interest points.

```
  strongest = selectStrongest(points, 10);
  imshow(I); hold on;
  plot(strongest);
```



Display the [x y] coordinates.

```
strongest.Location
```

```
ans = 10×2 single matrix

  136.8434  114.7649
  143.7372  126.8453
  135.2803  123.5254
  134.8544  125.0714
  114.8065   47.3621
  105.1597  162.2291
  129.8372  108.4799
  114.6876  131.6573
  135.2267  127.3643
   92.4255   63.2960
```

Tips

Although `BRISKPoints` can hold many points, it is a scalar object. Therefore, `numel(BRISKPoints)` always returns `1`. This value can differ from `length(BRISKPoints)`, which returns the true number of points held by the object.

Version History

Introduced in R2014a

References

[1] Leutenegger, S., M. Chli, and R. Siegwart. *BRISK: Binary Robust Invariant Scalable Keypoints*, Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2011.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To index locations with this object, use the syntax: `points.Location(idx,:)`, for `points` object. See visionRecovertformCodeGeneration_kernel.m, which is used in the “Introduction to Code Generation with Feature Matching and Registration” example.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`detectBRISKFeatures` | `detectMSERFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `extractFeatures` | `matchFeatures` | `detectSURFFeatures` | `MSERRegions` | `SURFPoints` | `cornerPoints` | `KAZEPoints` | `detectORBFeatures` | `ORBPoints`

imageSet

(Removed) Define collection of images

Note The `imageSet` object has been removed. To manage collections of image data use an `imageDatastore` object instead. For more information, see “Version History”.

Description

An `imageSet` object stores information about an image data set or a collection of image data sets. You can use this object to manage your image data. The object contains image descriptions, locations of images, and the number of images in your collection.

Creation

Syntax

```
imgSet = imageSet(imageLocation)
imgSetVector = imageSet(imgFolder,'recursive')
```

Description

`imgSet = imageSet(imageLocation)` creates an `imageSet` object.

`imgSetVector = imageSet(imgFolder,'recursive')` returns a vector of image sets found through a recursive search starting from `imgFolder`. The `imgSetVector` output is a 1-by-*NumFolders* vector of `imageSet` objects, where *NumFolders* is the number of folders that contain at least one image.

Input Arguments

`imageLocation` — Image file location

character vector | cell array

Image file location, specified as a character vector or a cell array. The vector must specify the folder name that contains the images. The image files name extensions must be supported by `imread`. The cell array must contain image locations.

Example: {'*imagePath1*','*imagePath2*', ..., '*imagePathX*'}, where each *imagePath* represents the path to an image.

`imgFolder` — Start recursive image search folder

character vector

Start recursive image search folder, specified as a character vector. The function searches the folder structure recursively, starting from `imgFolder`.

Properties

`Description` — Information about image set
character vector

Information about the image set, specified as a character vector. When you create an image set by recursively searching folders or by specifying a single folder location, the `Description` property is set to the folder name. When you specify individual image files, the `Description` property is not set. You can set the property manually.

Data Types: `char`

`Count` — Number of images in image set
positive integer

Number of images in the image set, specified as a positive integer.

Data Types: `double` | `single`

`ImageLocation` — Image locations
cell array of character vectors

Image locations, specified as a cell array of character vectors.

Data Types: `cell`

Object Functions

partition	(Removed) Divide image set into subsets
read	(Removed) Read image at specified index
select	(Removed) Select subset of images from image set

Version History

Introduced in R2014b

R2024b: Removed
Errors starting in R2024b

`imageSet` has been removed.

R2023b: `imageSet` will be removed in a future release
Not recommended starting in R2023b

`imageSet` will be removed in a future release. Use `imageDatastore` instead, which offers more functionality.

R2016b: `imageSet` is not recommended
Not recommended starting in R2016b

`imageSet` is not recommended. Use `imageDatastore` instead, which offers more functionality.

See Also

`imgetfile` | `imageCategoryClassifier` | `bagOfFeatures` | `trainImageCategoryClassifier`

Topics
“Image Category Classification Using Bag of Features”

partition

(Removed) Divide image set into subsets

Note The `partition` function of the `imageSet` object is not recommended. Instead, use the `imageDatastore` object and its `partition` function. For more information, see "Version History".

Syntax

```
[set1,set2,...,setN] = partition(imgSet,groupSizes)
[set1,set2,...,setN] = partition(imgSet,groupPercentages)
[set1,set2,...,setN] = partition( ___ ,method)
```

Description

`[set1,set2,...,setN] = partition(imgSet,groupSizes)` partitions the input image set, `imgSet`, into subsets of size `groupSizes`.

`[set1,set2,...,setN] = partition(imgSet,groupPercentages)` partitions the input image set, `imgSet`, in terms of percentages.

`[set1,set2,...,setN] = partition( ___ ,method)` additionally specifies a method, `'sequential'` or `'randomized'`.

Input Arguments

`imgSet` — Image set

scalar `imageSet` object | array of `imageSet` objects

Image set, specified as a scalar `imageSet` object or an array of `imageSet` objects.

`groupSizes` — Group size

positive integer | vector of positive integers

Group size of images, specified as a positive integer or vector of positive integers. The number of output arguments must be between 1 and `length(groupSizes) + 1`.

Example: `[20 60]` returns 20 images in `set1`, 60 images in `set2`, and any remaining images in `set3`.

`groupPercentages` — Group size percentage

scalar

Group size of images by percentage.

Example: `[0.1 0.5]` returns 10% of images in `set1`, 50% in `set2`, and the remainder in `set3`.

`method` — Image selection method

`'sequential'` (default) | `'randomized'`

Image selection method, specified as either `method` or `'randomized'`. When you set `method` to `'randomized'` the images are randomly selected to form the new sets. When you set `method` to `'sequential'` the images are selected sequentially.

Output Arguments

`set1,set2,...,setN` — Partitioned image sets
array of `imageSet` objects

Partitioned image sets, returned as an array of `imageSet` objects.

Version History

Introduced in R2014b

R2024b: Removed
Errors starting in R2024b

`partition` has been removed.

R2016b: `partition` is not recommended
Not recommended starting in R2016b

The `imageSet` object and its `partition` function are not recommended. Instead, manage collections of image data using an `imageDatastore` object, and partition data in the image datastore using the `partition` function. There are no plans to remove the `partition` function of the `imageSet` object.

See Also

`imageDatastore` | `partition`

read

(Removed) Read image at specified index

Note The `read` function of the `imageSet` object is not recommended. Instead, use an `imageDatastore` object and its `read` and `readimage` functions. For more information, see "Version History".

Syntax

```
im = read(imgSet,idx)
```

Description

`im = read(imgSet,idx)` returns an image from the image set, `imgSet`, located at the index `idx`.

Input Arguments

`imgSet` — Image set
`imageSet` object

Image set, specified as an `imageSet` object.

`idx` — Image location index
positive integer

Image location index, specified as a positive integer.

Output Arguments

`im` — Image
numeric array

Image, returned as a numeric array.

Version History

Introduced in R2014b

R2024b: Removed
Errors starting in R2024b

`partition` has been removed.

R2016b: `read` is not recommended
Not recommended starting in R2016b

The `imageSet` object and its `read` function are not recommended. Instead, manage collections of image data using an `imageDatastore` object, and read images from the image datastore using the

`read` and `readimage` functions. There are no plans to remove the `read` function of the `imageSet` object.

See Also

`imageDatastore` | `read` | `readimage`

select

(Removed) Select subset of images from image set

Note The `select` function of the `imageSet` object is not recommended. Instead, use an `imageDatastore` object and its `subset` function. For more information, see "Version History".

Syntax

```
imgSetOut = select(imgSet,idx)
```

Description

`imgSetOut = select(imgSet,idx)` returns a subset of image set, `imgSet`, using the selection of images specified by the index `idx`.

Input Arguments

`imgSet` — Image set

`imageSet` object

Image set, specified as an `imageSet` object.

`idx` — Image location index

scalar | vector of linear indices | vector of logical indices

Image location index, specified as a scalar, vector of linear indices, or a vector of logical indices. The function uses the `idx` index to select the subset of images.

Output Arguments

`imgSetOut` — Subset of input image set

`imageSet` object

Subset of the input image set, returned as an `imageSet` object.

Version History

Introduced in R2014b

R2024b: Removed

Errors starting in R2024b

`partition` has been removed.

R2016b: `select` is not recommended

Not recommended starting in R2016b

The `imageSet` object and its `select` function are not recommended. Instead, manage collections of image data using an `imageDatastore` object, and select subsets of images from the image datastore

using the `subset` function. There are no plans to remove the `select` function of the `imageSet` object.

See Also

`imageDatastore` | `subset`

invertedImageIndex

Search index that maps visual words to images

Description

An `invertedImageIndex` object is a search index that stores a visual word-to-image mapping. You can use this object with the `retrieveImages` function to search for an image.

Creation

Syntax

```
imageIndex = invertedImageIndex(bag)
imageIndex = invertedImageIndex(bag,'SaveFeatureLocations',tf)
```

Description

`imageIndex = invertedImageIndex(bag)` returns a search index object that stores the visual word-to-image mapping based on the input bag of visual words, `bag`. The `bag` argument sets the bagOfFeatures property.

`imageIndex = invertedImageIndex(bag,'SaveFeatureLocations',tf)` specifies whether or not to save the feature location data in `imageIndex`. To save image feature locations in the `imageIndex` object, specify the logical value `tf` as `true`. You can use location data to verify the spatial or geometric image search results. If you do not require feature locations, you can reduce memory consumption by specifying `tf` as `false`.

Properties

`ImageLocation` — Indexed image locations
cell array

Indexed image locations, specified as a cell array that contains the path and folder locations of images.

`ImageWords` — Visual words
M-element vector of `visualWords` objects

Visual words, specified as a *M*-element vector of `visualWords` objects. *M* is the number of indexed images in the `invertedImageIndex` object. Each `visualWords` object contains the `WordIndex`, `Location`, `VocabularySize`, and `Count` properties for the corresponding indexed image.

`WordFrequency` — Word occurrence
N-element column

Word occurrence, specified as an *N*-element column. The column contains the percentage of images in which each visual word occurs. These percentages are analogous to document frequency in text retrieval applications.

To reduce the search set, when looking for the most relevant images, you can suppress the most common words. You can also suppress rare words that you suspect come from outliers in the image set.

You can control how much the top and bottom end of the visual word distribution affects the search results by tuning the `WordFrequencyRange` property.

`BagOfFeatures` — Bag of visual words
`bagOfFeatures` object

Bag of visual words, specified as the `bagOfFeatures` object used to create the index.

`MatchThreshold` — Required similarity percentage for potential image match
`0.01` (default) | numeric value in range [0,1]

Required similarity percentage for potential image match, specified as a numeric value in the range [0,1]. To obtain more search results, lower this threshold.

`WordFrequencyRange` — Word frequency range
`[0.01 0.9]` (default) | two-element vector

Word frequency range, specified as a two-element vector of a lower and an upper percentage, [*lower upper*]. Percentages must be in the range [0, 1], and the value of *lower* must be less than the value of *upper*. Use the word frequency range to ignore common words (the upper percentage range) or rare words (the lower percentage range) within the image index. These words often occur as repeated patterns or outliers, respectively, and can reduce search accuracy. To find potential values for this property, before you set this value, plot the sorted `WordFrequency` values.

`ImageID` — Indexed image identifiers
vector of integers

Indexed image identifiers, specified as a vector of integers that uniquely identify indexed images. Using the same identifier for `invertedImageIndex` and `imageviewset` eliminates the need to index the same image differently in each object.

Object Functions

addImages	Add new images to image index
removeImages	Remove images from image index
addImageFeatures	Add features of image to image index

Examples

Search ROI for Object

Define a set of images to search.

```
imageFiles = ...
  {'elephant.jpg', 'cameraman.tif', ...
   'peppers.png',  'saturn.png', ...
   'pears.png',    'stapleRemover.jpg', ...
   'football.jpg', 'mandi.tif', ...
   'kids.tif',     'liftingbody.png', ...
   'office_5.jpg', 'gantrycrane.png', ...
```

```
    'moon.tif',     'circuit.tif', ...
    'tape.png',     'coins.png'};

imds = imageDatastore(imageFiles);
```

Learn the visual vocabulary of the image view set.

```
numLevels = 1;
branchingFactor = 1000;
bag = bagOfFeatures(imds,'PointSelection','Detector', ...
  'TreeProperties',[numLevels,branchingFactor]);

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 16 images...done. Extracted 3680 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 2944.
** Using the strongest 2944 features from each of the other image categories.

* Creating a 1000 word visual vocabulary.
* Number of levels: 1
* Branching factor: 1000
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 2944
* Number of clusters          : 1000
* Initializing cluster centers...100.00%.
* Clustering...completed 6/100 iterations (~0.11 seconds/iteration)...converged in 6 iterations.

* Finished creating Bag-Of-Features
```

Create an image search index and add the image view set images.

```
imageIndex = invertedImageIndex(bag);

addImages(imageIndex,imds);

Encoding images using Bag-Of-Features.
--------------------------------------

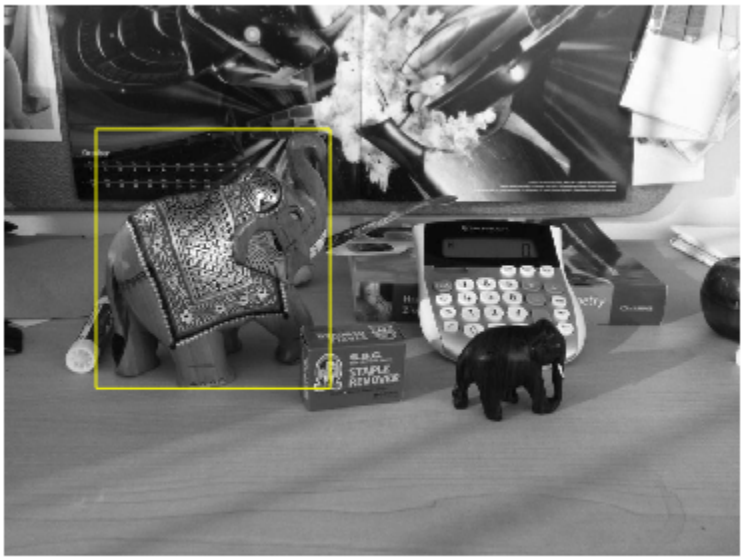
* Encoding 16 images...done.
```

Specify a query image and an ROI in which to search for the target object, an elephant. You can also use the `imrect` function to select an ROI interactively. For example, `queryROI = getPosition(imrect)`.

```
queryImage = imread('clutteredDesk.jpg');
queryROI = [130 175 330 365];

figure
```

```
imshow(queryImage)
rectangle('Position',queryROI,'EdgeColor','yellow')
```



Find images that contain the object.

```
imageIDs = retrieveImages(queryImage,imageIndex,'ROI',queryROI)
```

```
imageIDs = 15×1 uint32 column vector

    1
   11
    2
    6
   12
    3
    8
   16
   14
    5
    7
   10
   13
   15
    9
```

```
bestMatch = imageIDs(1);

figure
imshow(imageIndex.ImageLocation{bestMatch})
```



Version History

Introduced in R2015a

References

[1] Sivic, J. and A. Zisserman. *Video Google: A text retrieval approach to object matching in videos*. ICCV (2003) pg 1470-1477.

[2] Philbin, J., O. Chum, M. Isard, J. Sivic, and A. Zisserman. *Object retrieval with large vocabularies and fast spatial matching*. CVPR (2007).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`imageSet` | `bagOfFeatures` | `bagOfFeaturesDBoW` | `dbowLoopDetector`

Functions

`retrieveImages` | `evaluateImageRetrieval` | `indexImages`

Topics
“Image Retrieval Using Customized Bag of Features”
“Image Retrieval with Bag of Visual Words”

addImages

Add new images to image index

Syntax

```
addImages(imageIndex,imds)
addImages(imageIndex,I,imageId)
addImages( ___ ,Verbose=tf)
```

Description

`addImages(imageIndex,imds)` adds the images in `imds` to `imageIndex`.

`addImages(imageIndex,I,imageId)` adds an image, `I`, to `imageIndex`, specified by the image identifier `imageId`.

`addImages( ___ ,Verbose=tf)` specifies whether or not to display progress information, in addition to any combination of input arguments from previous syntaxes. By default, the `addImages` function displays progress information. To hide the display, specify `tf` as `false`.

Examples

Add Image to Image Index

Define a set of images to search

```
imageFiles = ...
  {'elephant.jpg', 'cameraman.tif', ...
   'peppers.png',  'saturn.png',...
   'pears.png',    'stapleRemover.jpg', ...
   'football.jpg', 'mandi.tif', ...
   'kids.tif',     'liftingbody.png', ...
   'office_5.jpg', 'gantrycrane.png', ...
   'moon.tif',     'circuit.tif', ...
   'tape.png',     'coins.png'};

imds = imageDatastore(imageFiles);
```

Learn the visual vocabulary of the image view set.

```
numlevels = 1;
branchingFactor = 1000;
bag = bagOfFeatures(imds,'PointSelection','Detector', ...
        'TreeProperties',[numlevels,branchingFactor],'Verbose',false);
```

Create an image search index.

```
imageIndex = invertedImageIndex(bag);
```

Add to the image view set images to the image search index.

```
addImages(imageIndex,imds);

Encoding images using Bag-Of-Features.
--------------------------------------

* Encoding 16 images...done.
```

Input Arguments

`imageIndex` — Image search index

`invertedImageIndex` object

Image search index, specified as an `invertedImageIndex` object.

`imds` — Images

`ImageDatastore` object

Images, specified as an `ImageDatastore` object. `imds` contains new images to add to an existing index. The function does not ignore duplicate images.

`I` — Input image

M-by-*N*-by-3 array | *M*-by-*N* 2-D matrix

Input image, specified as either an *M*-by-*N*-by-3 array, representing a truecolor image, or an *M*-by-*N* array, representing a 2-D grayscale image.

`imageId` — Indexed image identifier

positive integer

Indexed image identifier, specified as an positive integer.

Tips

- The `addImages` function supports parallel computing using multiple MATLAB workers. Enable parallel computing from the “Computer Vision Toolbox Preferences” dialog box. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, select **Preferences**. Then select **Computer Vision Toolbox**.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

For code generation, the `addImages` object does not support the `ImageDatastore` object as input. Instead, specify the input as a structure with fields `Images` and `Locations`. The `Images` field must contain a cell array of grayscale or RGB color images. The `Locations` field must contain a cell array of the file paths to the locations of input images.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`removeImages` | `addImageFeatures` | `invertedImageIndex`

removeImages

Remove images from image index

Syntax

```
removeImages(imageIndex,imageIds)
```

Description

`removeImages(imageIndex,imageIds)` removes the images from the `imageIndex` object that correspond to the `imageIds` input.

Examples

Remove Indexed Image

Create image set.

```
dataDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets','cups');
imds = imageDatastore(dataDir);
```

Index the image set.

```
imageIndex = indexImages(imds)
```

```
Creating an inverted image index using Bag-Of-Features.
-------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 6 images...done. Extracted 1708 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 1366.
** Using the strongest 1366 features from each of the other image categories.

* Creating a 1366 word visual vocabulary.
* Number of levels: 1
* Branching factor: 1366
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 1366
* Number of clusters          : 1366
```

```
* Initializing cluster centers...100.00%.
* Clustering...completed 1/100 iterations (~0.06 seconds/iteration)...converged in 1 iterations.

* Finished creating Bag-Of-Features


Encoding images using Bag-Of-Features.
---------------------------------------

* Encoding 6 images...done.
Finished creating the image index.

imageIndex =
  invertedImageIndex with properties:

         ImageLocation: {6×1 cell}
            ImageWords: [6×1 vision.internal.visualWords]
         WordFrequency: [0.1667 0.1667 0.1667 0.3333 0.1667 0.1667 0.1667 0.5000 0.3333 0.1667 0
         BagOfFeatures: [1×1 bagOfFeatures]
               ImageID: [1 2 3 4 5 6]
        MatchThreshold: 0.0100
    WordFrequencyRange: [0.0100 0.9000]
```

```
imageIndex.ImageLocation
```

```
ans = 6×1 cell
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\bigMug.jpg'    }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\blueCup.jpg'   }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\handMade.jpg'  }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\holdingCup.jpg'}
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\plaid.jpg'     }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\plainWhite.jpg'}
```

Remove first and third image.

```
removeImages(imageIndex,[1 3]);
imageIndex.ImageLocation
```

```
ans = 4×1 cell
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\blueCup.jpg'   }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\holdingCup.jpg'}
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\plaid.jpg'     }
    {'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\plainWhite.jpg'}
```

Input Arguments

`imageIndex` — Image search index

`invertedImageIndex` object

Image search index, specified as an `invertedImageIndex` object.

`imageIds` — Image identifiers

vector of positive integers

Image identifiers, specified as a vector of positive integers. The image identifiers correspond to the identifiers in `imageIndex.ImageID`, where `ImageID` is a property of the `invertedImageIndex` object.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`addImages` | `addImageFeatures` | `invertedImageIndex`

addImageFeatures

Add features of image to image index

Syntax

```
addImageFeatures(imageIndex,features,imageId)
```

Description

`addImageFeatures(imageIndex,features,imageId)` adds image `features` to `imageIndex` from the image specified by `imageId`.

Examples

Add Image Features of Image to Image Index

Define a set of images.

```
imageFiles = ...
   {'elephant.jpg','cameraman.tif', ...
    'peppers.png','saturn.png', ...
    'pears.png','stapleRemover.jpg', ...
    'football.jpg','mandi.tif', ...
    'kids.tif','liftingbody.png', ...
    'office_5.jpg','gantrycrane.png', ...
    'moon.tif','circuit.tif', ...
    'tape.png'};
```

Create an image datastore containing the images.

```
imds = imageDatastore(imageFiles);
```

Learn the visual vocabulary of the images.

```
bag = bagOfFeatures(imds);
```

```
Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Grid method.
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 15 images...done. Extracted 655092 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1
```

```
* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 524074
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 53/100 iterations (~6.84 seconds/iteration)...converged in 53 iterations

* Finished creating Bag-Of-Features
```

Create an image search index.

```
imageIndex = invertedImageIndex(bag,'SaveFeatureLocations',false);
```

Load an image and extract features from it.

```
I = imread('coins.png');
```

Detect SURF points from the image.

```
points = detectSURFFeatures(I);
```

Extract image features from the SURF points.

```
features = extractFeatures(I,points);
```

Add the image features into the image index.

```
imageId = 1;
addImageFeatures(imageIndex,features,imageId)
```

Input Arguments

`imageIndex` — Image search index

`invertedImageIndex` object

Image search index, specified as an `invertedImageIndex` object.

`features` — Feature descriptors

M-by-*N* matrix | `binaryFeatures` object

Feature descriptors, specified as an *M*-by-*N* matrix or a `binaryFeatures` object. Each descriptor is of length *N*. To add image features to the image index using `addImageFeatures`, you must set the `SaveFeatureLocations` property of the `invertedImageIndex` object to `false`.

`imageId` — Indexed image identifier

positive integer

Indexed image identifier, specified as a positive integer.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`invertedImageIndex` | `removeImages` | `addImages`

cameraCalibrationErrors

Object for storing standard errors of estimated camera parameters

Description

`cameraCalibrationErrors` contains the standard errors of estimated camera parameters. You can access the intrinsics and extrinsics standard errors using the object properties. You can display the standard errors using the object's `displayErrors` object function.

Creation

Create a `cameraCalibrationErrors` object by calling the `estimateCameraParameters` function.

Properties

`IntrinsicsErrors` — Standard intrinsics error
`intrinsicsEstimationErrors` object

Standard error of the estimated intrinsics for a camera, specified as a `intrinsicsEstimationErrors` object.

`ExtrinsicsErrors` — Standard extrinsics error
`extrinsicsEstimationErrors` object

Standard error of the estimate rotations and translations for a camera relative to the calibration pattern, specified as a `extrinsicsEstimationErrors` object.

Object Functions

displayErrors Display standard errors of camera parameter estimates

Examples

Estimate and Display Camera Calibration Standard Errors

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
 "calibration","mono"));
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the corners of the squares. The units of the square size are in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1),size(I,2)];
[params,~,errors] = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Display the standard errors.

```
displayErrors(errors,params);
```

```
            Standard Errors of Estimated Camera Parameters
            ----------------------------------------------

Intrinsics
----------
Focal length (pixels):   [  714.1884 +/- 3.3219      710.3783 +/- 4.0579  ]
Principal point (pixels):[  563.6478 +/- 5.3967      355.7255 +/- 3.3036  ]
Radial distortion:       [   -0.3536 +/- 0.0091        0.1730 +/- 0.0488  ]

Extrinsics
----------
Rotation vectors:
                         [   -0.6096 +/- 0.0054       -0.1789 +/- 0.0073       -0.3835 +/- 0.0024
                         [   -0.7283 +/- 0.0050       -0.0996 +/- 0.0072        0.1964 +/- 0.0027
                         [   -0.6722 +/- 0.0051       -0.1444 +/- 0.0074       -0.1329 +/- 0.0026
                         [   -0.5836 +/- 0.0056       -0.2901 +/- 0.0074       -0.5622 +/- 0.0025
                         [   -0.3157 +/- 0.0065       -0.1441 +/- 0.0075       -0.1067 +/- 0.0011
                         [   -0.7581 +/- 0.0052        0.1947 +/- 0.0072        0.4324 +/- 0.0030
                         [   -0.7515 +/- 0.0051        0.0767 +/- 0.0072        0.2070 +/- 0.0029
                         [   -0.6223 +/- 0.0053        0.0231 +/- 0.0073        0.3663 +/- 0.0024
                         [    0.3443 +/- 0.0063       -0.2226 +/- 0.0073       -0.0437 +/- 0.0014

Translation vectors (mm):
                         [ -146.0514 +/- 6.0391      -26.8688 +/- 3.7318      797.9024 +/- 3.9002
                         [ -209.4354 +/- 6.9637      -59.4568 +/- 4.3578      921.8196 +/- 4.6295
                         [ -129.3821 +/- 7.0907      -44.1033 +/- 4.3751      937.6829 +/- 4.4913
                         [ -151.0046 +/- 6.6905      -27.3256 +/- 4.1339      884.2787 +/- 4.3925
                         [ -174.9496 +/- 6.7056      -24.3502 +/- 4.1606      886.4959 +/- 4.6680
                         [ -134.3092 +/- 7.8887     -103.4984 +/- 4.8925     1042.4551 +/- 4.8184
                         [ -173.9842 +/- 7.6891      -73.1694 +/- 4.7812     1017.2383 +/- 4.8120
                         [ -202.9443 +/- 7.4327      -87.9094 +/- 4.6482      983.6955 +/- 4.9072
                         [ -319.8858 +/- 6.3213     -119.8901 +/- 4.0922      829.4580 +/- 4.9591
```

Version History

Introduced in R2014b

See Also

Apps

Stereo Camera Calibrator | Camera Calibrator

Classes
`cameraParameters` | `stereoParameters` | `stereoCalibrationErrors` | `intrinsicsEstimationErrors` | `extrinsicsEstimationErrors`

Functions
`estimateCameraParameters` | `showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `undistortPoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints`

Topics
“Evaluating the Accuracy of Single Camera Calibration”
“Using the Single Camera Calibrator App”

stereoCalibrationErrors

Object for storing standard errors of estimated stereo parameters

Description

The `stereoCalibrationErrors` object contains the standard errors of the estimated calibration parameters of a pair of stereo cameras. The object stores the estimated intrinsic parameters for camera one and camera two, the extrinsic parameters for camera one, and the rotation and translation of camera two relative to camera one. You can access the standard error values by using the object properties. To display the standard errors, use the `displayErrors` object function.

Creation

Create a stereo camera calibration errors object by using the `estimateCameraParameters` function to calibrate a pair of stereo cameras. The `stereoCalibrationErrors` object is the third output argument of the `estimateCameraParameters` function. For example:

```
[params,imgsUsed,Errors] = estimateCameraParameters(imagePoints,worldPoints)
```

Properties

`Camera1IntrinsicsErrors` — Standard errors of estimated intrinsic parameters and distortion coefficients for camera one
`intrinsicsEstimationErrors` object

Standard errors of estimated intrinsic parameters and distortion coefficients for camera one, specified as an `intrinsicsEstimationErrors` object.

`Camera1ExtrinsicsErrors` — Standard errors of estimated extrinsic parameters and distortion coefficients for camera one
`extrinsicsEstimationErrors` object

Standard errors of estimated extrinsic parameters and distortion coefficients for camera one, specified as an `extrinsicsEstimationErrors` object.

`Camera2IntrinsicsErrors` — Standard errors of estimated intrinsic parameters and distortion coefficients for camera two
`intrinsicsEstimationErrors` object

Standard errors of estimated intrinsic parameters and distortion coefficients for camera two, specified as an `intrinsicsEstimationErrors` object.

`RotationOfCamera2Error` — Standard errors of rotated vector of camera two relative to camera one
three-element vector

Standard errors of the rotated vector of camera two relative to camera one, specified as a three-element vector.

`TranslationOfCamera2Error` — Standard errors of translation of camera two relative to camera one
three-element vector

Standard errors of the translation of camera two relative to camera one, specified as a three-element vector.

Object Functions

displayErrors Display standard errors of stereo camera parameter estimation

Examples

Estimate and Display Stereo Calibration Standard Errors

Specify two sets of calibration images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata", ...
  "calibration","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = detectCheckerboardPoints( ...
  leftImages.Files,rightImages.Files);
```

Generate the world coordinates of the corners of the squares. Square size is in millimeters.

```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Here, both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
[params, ~, errors] = estimateCameraParameters(imagePoints,worldPoints, ...
                                  ImageSize=imageSize);
```

Display standard stereo calibration errors.

```
displayErrors(errors,params);
```

```
            Standard Errors of Estimated Stereo Camera Parameters
            -----------------------------------------------------

Camera 1 Intrinsics
-------------------
Focal length (pixels):   [ 1038.0285 +/- 0.6533     1037.9143 +/- 0.6389  ]
Principal point (pixels):[  656.0840 +/- 0.3408      485.5485 +/- 0.2639  ]
Radial distortion:       [   -0.3617 +/- 0.0008        0.1866 +/- 0.0026  ]

Camera 1 Extrinsics
-------------------
Rotation vectors:
                       [    0.1680 +/- 0.0007       -0.0271 +/- 0.0006        3.1125 +/- 0.0001
```

```
                          [    0.1995 +/- 0.0006       -0.0523 +/- 0.0005       -3.0991 +/- 0.0000
                          [    0.4187 +/- 0.0005       -0.1061 +/- 0.0004       -3.1113 +/- 0.0001
                          [    0.5239 +/- 0.0005       -0.0604 +/- 0.0004       -3.0552 +/- 0.0001
                          [    0.6807 +/- 0.0006       -0.0306 +/- 0.0005       -3.0331 +/- 0.0001
                          [    0.3513 +/- 0.0007       -0.0993 +/- 0.0006       -3.0334 +/- 0.0001
                          [    0.0212 +/- 0.0007       -0.1179 +/- 0.0007       -3.0833 +/- 0.0000
                          [   -0.2765 +/- 0.0008       -0.0847 +/- 0.0007       -3.0943 +/- 0.0001
                          [   -0.4407 +/- 0.0007       -0.1119 +/- 0.0006       -3.0652 +/- 0.0001
                          [   -0.2537 +/- 0.0008       -0.1334 +/- 0.0007       -3.1039 +/- 0.0001

Translation vectors (mm):
                          [  708.4193 +/- 0.4914      227.0500 +/- 0.4002     1492.8669 +/- 1.0127
                          [  368.4410 +/- 0.5228      191.7200 +/- 0.4094     1589.9144 +/- 0.9987
                          [  226.3712 +/- 0.5173      191.1430 +/- 0.4030     1578.4777 +/- 0.9575
                          [   49.5378 +/- 0.5183      196.7496 +/- 0.4030     1580.5402 +/- 0.9491
                          [ -172.4000 +/- 0.7003      150.9911 +/- 0.5406     2119.3251 +/- 1.2532
                          [   10.7779 +/- 0.6784      176.8786 +/- 0.5276     2066.8341 +/- 1.2907
                          [  295.4842 +/- 0.6616      167.8676 +/- 0.5158     2010.7710 +/- 1.2738
                          [  614.2340 +/- 0.6457      166.2017 +/- 0.5153     1968.1796 +/- 1.2722
                          [  767.0158 +/- 0.6106      165.5373 +/- 0.4991     1868.3331 +/- 1.2395
                          [  953.8135 +/- 0.7336      -14.7980 +/- 0.6039     2255.6166 +/- 1.5100

Camera 2 Intrinsics
-------------------
Focal length (pixels):   [ 1042.4815 +/- 0.6644     1042.2690 +/- 0.6534  ]
Principal point (pixels):[  640.5974 +/- 0.3305      479.0652 +/- 0.2633  ]
Radial distortion:       [   -0.3614 +/- 0.0007        0.1822 +/- 0.0022  ]

Position And Orientation of Camera 2 Relative to Camera 1
---------------------------------------------------------
Rotation of camera 2:         [   -0.0037 +/- 0.0002        0.0050 +/- 0.0004       -0.0002 +/- 0
Translation of camera 2 (mm): [ -119.8720 +/- 0.0401       -0.4005 +/- 0.0414       -0.0258 +/- 0
```

Version History

Introduced in R2014b

See Also

Apps

Stereo Camera Calibrator | Camera Calibrator

Classes

`cameraParameters` | `stereoParameters` | `cameraCalibrationErrors` | `intrinsicsEstimationErrors` | `extrinsicsEstimationErrors`

Functions

`estimateCameraParameters` | `showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `undistortPoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Using the Single Camera Calibrator App”

“Using the Stereo Camera Calibrator App”

displayErrors

Display standard errors of stereo camera parameter estimation

Syntax

```
displayErrors(estimationErrors,stereoParams)
```

Description

`displayErrors(estimationErrors,stereoParams)` displays the standard errors of stereo camera parameter estimation, as well as the estimated parameters, from the calibration of a pair of stereo cameras.

Examples

Estimate and Display Stereo Calibration Standard Errors

Specify two sets of calibration images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata", ...
  "calibration","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = detectCheckerboardPoints( ...
  leftImages.Files,rightImages.Files);
```

Generate the world coordinates of the corners of the squares. Square size is in millimeters.

```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Here, both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
[params, ~, errors] = estimateCameraParameters(imagePoints,worldPoints, ...
                                    ImageSize=imageSize);
```

Display standard stereo calibration errors.

```
displayErrors(errors,params);
```

```
            Standard Errors of Estimated Stereo Camera Parameters
            -----------------------------------------------------

Camera 1 Intrinsics
-------------------
Focal length (pixels):   [ 1038.0285 +/- 0.6533     1037.9143 +/- 0.6389  ]
Principal point (pixels):[  656.0840 +/- 0.3408      485.5485 +/- 0.2639  ]
```

```
Radial distortion:       [   -0.3617 +/- 0.0008        0.1866 +/- 0.0026  ]

Camera 1 Extrinsics
-------------------
Rotation vectors:
                         [    0.1680 +/- 0.0007       -0.0271 +/- 0.0006        3.1125 +/- 0.0001
                         [    0.1995 +/- 0.0006       -0.0523 +/- 0.0005       -3.0991 +/- 0.0000
                         [    0.4187 +/- 0.0005       -0.1061 +/- 0.0004       -3.1113 +/- 0.0001
                         [    0.5239 +/- 0.0005       -0.0604 +/- 0.0004       -3.0552 +/- 0.0001
                         [    0.6807 +/- 0.0006       -0.0306 +/- 0.0005       -3.0331 +/- 0.0001
                         [    0.3513 +/- 0.0007       -0.0993 +/- 0.0006       -3.0334 +/- 0.0001
                         [    0.0212 +/- 0.0007       -0.1179 +/- 0.0007       -3.0833 +/- 0.0000
                         [   -0.2765 +/- 0.0008       -0.0847 +/- 0.0007       -3.0943 +/- 0.0001
                         [   -0.4407 +/- 0.0007       -0.1119 +/- 0.0006       -3.0652 +/- 0.0001
                         [   -0.2537 +/- 0.0008       -0.1334 +/- 0.0007       -3.1039 +/- 0.0001

Translation vectors (mm):
                         [  708.4193 +/- 0.4914      227.0500 +/- 0.4002     1492.8669 +/- 1.0127
                         [  368.4410 +/- 0.5228      191.7200 +/- 0.4094     1589.9144 +/- 0.9987
                         [  226.3712 +/- 0.5173      191.1430 +/- 0.4030     1578.4777 +/- 0.9575
                         [   49.5378 +/- 0.5183      196.7496 +/- 0.4030     1580.5402 +/- 0.9493
                         [ -172.4000 +/- 0.7003      150.9911 +/- 0.5406     2119.3251 +/- 1.2532
                         [   10.7779 +/- 0.6784      176.8786 +/- 0.5276     2066.8341 +/- 1.2907
                         [  295.4842 +/- 0.6616      167.8676 +/- 0.5158     2010.7710 +/- 1.2738
                         [  614.2340 +/- 0.6457      166.2017 +/- 0.5153     1968.1796 +/- 1.2722
                         [  767.0158 +/- 0.6106      165.5373 +/- 0.4991     1868.3331 +/- 1.2395
                         [  953.8135 +/- 0.7336      -14.7980 +/- 0.6039     2255.6166 +/- 1.5100

Camera 2 Intrinsics
-------------------
Focal length (pixels):   [ 1042.4815 +/- 0.6644     1042.2690 +/- 0.6534  ]
Principal point (pixels):[  640.5974 +/- 0.3305      479.0652 +/- 0.2633  ]
Radial distortion:       [   -0.3614 +/- 0.0007        0.1822 +/- 0.0022  ]

Position And Orientation of Camera 2 Relative to Camera 1
---------------------------------------------------------
Rotation of camera 2:         [   -0.0037 +/- 0.0002        0.0050 +/- 0.0004       -0.0002 +/- 0
Translation of camera 2 (mm): [ -119.8720 +/- 0.0401       -0.4005 +/- 0.0414       -0.0258 +/- 0
```

Input Arguments

`estimationErrors` — Estimation errors
`stereoCalibrationErrors` object

Estimation errors, specified as a `stereoCalibrationErrors` object.

`stereoParams` — Stereo parameters
`stereoParameters` object

Stereo parameters, specified as a `stereoParameters` object.

Version History

Introduced in R2014b

See Also

Objects
`stereoParameters` | `stereoCalibrationErrors`

MSERRegions

Object for storing MSER regions

Description

This object describes MSER regions and corresponding ellipses that have the same second moments as the regions. It passes data between the `detectMSERFeatures` and `extractFeatures` functions. The object can also be used to manipulate and plot the data returned by these functions.

Creation

Syntax

```
regions = MSERRegions(pixellist)
```

Description

`regions = MSERRegions(pixellist)` constructs an MSER regions object, `regions` regions from the pixel list,`pixellist`.

Input Arguments

`pixellist` — Point coordinates for detected MSER regions

M-by-1 cell array

Point coordinates for detected MSER regions, specified as a *M*-by-1 cell array. Each cell contains a *P*-by-2 array of [x y] coordinates for the detected MSER regions, where *P* varies based on the number of pixels in a region.

Properties

`Location` — Locations of ellipses

M-by-2 array (default)

This property is read-only.

Locations of ellipses, stored as an *M*-by-2 array of [*x y*] coordinates. The ellipses that have the same second moments as the MSER regions.

`Axes` — Major and minor axis

two-element vector (default)

This property is read-only.

Major and minor axis, stored as a two-element vector, [majorAxis minorAxis]. This vector specifies the major and minor axis of the ellipse that have the same second moments as the MSER regions.

`Orientation` — Ellipse orientation

scalar in the range -pi/2 to +pi/2

This property is read-only.

Ellipse orientation, stored as a value in the range from -pi/2 to +pi/2 radians. This value represents the orientation of the ellipse as measured from the *X*-axis to the major axis of the ellipse. You can use this property for visualization purposes.

`Count` — Number of stored regions

`0` (default) | integer

Number of stored regions, specified as an integer.

Object Functions

isempty	Determine if points object is empty
length	Number of stored points
size	Return size of points object
plot	Plot MSER regions
select	Select point or region features during code generation

Examples

Detect MSER Features in an Image

Load an image.

```
I = imread('cameraman.tif');
```

Detect and store regions.

```
regions = detectMSERFeatures(I);
```

Display the centroids and axes of detected regions.

```
imshow(I); hold on;
plot(regions);
```



Display MSER Feature Regions from the MSERRegions Object

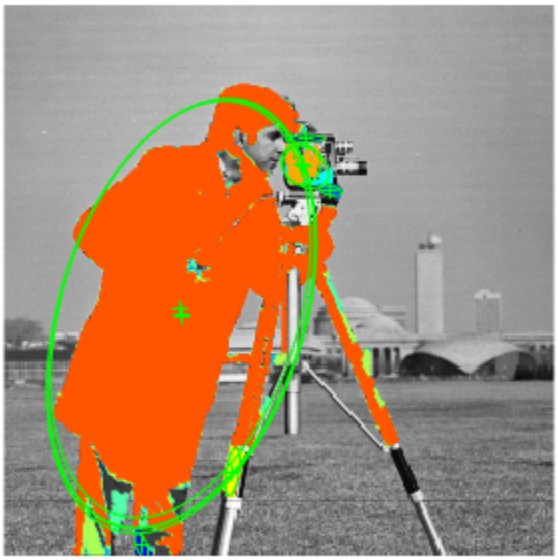
Detect and display the first 10 regions contained in the MSERRegions object.

Detect MSER features.

```
I = imread('cameraman.tif');
regions = detectMSERFeatures(I);
```

Display the first 10 regions in the MSERRegions object.

```
imshow(I); hold on;
plot(regions(1:10),'showPixelList', true);
```



Combine MSER Region Detector with SURF Descriptors

Extract and display SURF descriptors at locations identified by MSER detector.

Read image.

```
I = imread('cameraman.tif');
```

Detect MSER features.

```
regionsObj = detectMSERFeatures(I);
```

Extract and display SURF descriptors.

```
[features, validPtsObj] = extractFeatures(I, regionsObj);
imshow(I); hold on;
plot(validPtsObj,'showOrientation',true);
```



Tips

Although `MSERRegions` may hold many regions, it is a scalar object. Therefore, `numel(MSERRegions)` always returns `1`. This value may differ from `length(MSERRegions)`, which returns the true number of regions held by the object.

Version History

Introduced in R2012a

References

[1] Nister, D., and H. Stewenius, "Linear Time Maximally Stable Extremal Regions", *Lecture Notes in Computer Science*. 10th European Conference on Computer Vision, Marseille, France: 2008, no. 5303, pp. 183–196.

[2] Matas, J., O. Chum, M. Urba, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions. "*Proceedings of British Machine Vision Conference*, pages 384-396, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For code generation, you must specify both the `pixellist` cell array and the `length` of each array, as the second input. The object outputs, `regions.PixelList` as an array. The region sizes are defined in `regions.Lengths`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`detectMSERFeatures` | `detectSURFFeatures` | `edge` | `extractFeatures` | `matchFeatures` | `SURFPoints` | `cornerPoints` | `BRISKPoints` | `KAZEPoints` | `ORBPoints`

Topics
“Find MSER Regions in an Image” on page 3-603
“Detect SURF Interest Points in a Grayscale Image” on page 3-615
“Segment and Read Text in Image”

cornerPoints

Object for storing corner points

Description

This object stores information about point features detected from a 2-D grayscale image.

Creation

Syntax

```
points = cornerPoints(location)
points = cornerPoints(location,Name,Value)
```

Description

`points = cornerPoints(location)` constructs a `cornerPoints` object from an *M*-by-2 array [*x y*] of location coordinates.

`points = cornerPoints(location,Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `points = cornerPoints('Metric',0.0)`

Input Arguments

`location` — Location coordinates

M-by-2 array

Location of points, specified as an *M*-by-2 array of [*x y*] coordinates.

Properties

`Location` — Location of keypoints

`[]` (default) | *M*-by-2 matrix

This property is read-only.

Location of keypoints, specified as an *M*-by-2 matrix. Each row is of the form [*x y*] and represents the location of a keypoint. *M* is the number of keypoints. You cannot set this property, use the `location` input argument instead.

`Count` — Number of points

`0` (default) | integer

Number of points held by the corner points object, specified as a numeric value.

`Metric` — Strength of detected feature

`0.0` (default) | numeric scalar

Strength of detected feature, specified as a numeric value. The algorithm uses a determinant of an approximated Hessian.

Object Functions

plot	Plot detected feature points
isempty	Determine if points object is empty
length	Number of stored points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation
gather	Retrieve cornerPoints from the GPU

Examples

Plot Strongest Features from Detected Point Features

Read an image.

```
I = imread('cameraman.tif');
```

Detect point features.

```
points = detectHarrisFeatures(I);
```

Display the ten strongest points.

```
strongest = selectStrongest(points,10);
imshow(I)
hold on
plot(strongest)
```



Display the (x,y) coordinates of the strongest points.

```
strongest.Location
```

```
ans = 10×2 single matrix

  112.4516  208.4412
  108.6510  228.1681
  136.6969  114.7962
  181.4160  205.9876
  135.5823  123.4529
  100.4951  174.3253
  146.7581   94.7393
  135.2899   92.6485
  129.8439  110.0350
  130.5716   91.0424
```

Create Corner Points Object and Display Points

Create a checkerboard image.

```
I = checkerboard(50,2,2);
```

Load the locations of corner points.

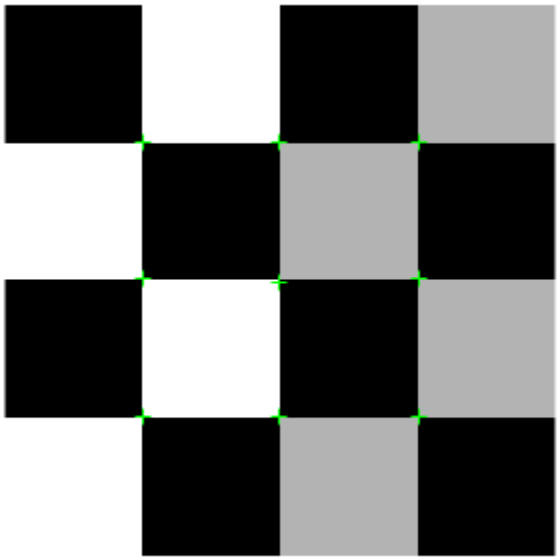
```
location = [51    51    51   100   100   100   151   151   151; ...
            50   100   150    50   101   150    50   100   150]';
```

Save the points in a `cornerPoints` object.

```
points = cornerPoints(location);
```

Display the points on the checkerboard.

```
imshow(I)
hold on
plot(points)
```



Tips

Although `cornerPoints` may hold many points, it is a scalar object. Therefore, `numel(cornerPoints)` always returns `1`. This value may differ from `length(cornerPoints)`, which returns the true number of points held by the object.

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To index locations with this object, use the syntax: `points.Location(idx,:)`, for `points` object. See visionRecovertformCodeGeneration_kernel.m, which is used in the “Introduction to Code Generation with Feature Matching and Registration” example.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

binaryFeatures | detectHarrisFeatures | detectFASTFeatures | detectMinEigenFeatures | detectBRISKFeatures | detectSURFFeatures | detectMSERFeatures | extractFeatures | extractHOGFeatures | matchFeatures | MSERRegions | detectORBFeatures | ORBPoints | SURFPoints | BRISKPoints | KAZEPoints

SURFPoints

Object for storing SURF interest points

Description

This object provides the ability to pass data between the `detectSURFFeatures` and `extractFeatures` functions. It can also be used to manipulate and plot the data returned by these functions. You can use the object to fill the points interactively. You can use this approach in situations where you might want to mix a non-SURF interest point detector with a SURF descriptor.

Creation

Syntax

```
points = SURFPoints(location)
```

Description

`points = SURFPoints(location)` constructs a `SURFPoints` object from an *M*-by-2 array of [*x y*] point coordinates.

Input Arguments

`location` — Point coordinates

M-by-2 array of [*x y*] point coordinates.

Point coordinates, specified as an *M*-by-2 array of form [*x y*] coordinates. *M* denotes the number of points.

Properties

`Count` — Number of points

`0` (default) | integer

Number of points held by the object, specified as a numeric value.

`Location` — Point locations

M-by-2 array

Point locations, specified as an *M*-by-2 array of [*x y*] point coordinates.

`Scale` — Scale

`1.6` (default) | numeric scalar | *M*-element vector

Scale at which the feature was detected, specified as a numeric scalar or as an *M*-element vector. Values must be greater than or equal to `1.6`.

`Metric` — Strength of detected feature

`0.0` (default) | numeric scalar | *M*-element vector

Strength of detected feature, specified as a numeric value or as an *M*-element vector. The SURF algorithm uses a determinant of approximated Hessian as the metric.

`Orientation` — Orientation

`0.0` (default) | angle in radians | numeric scalar | *M*-element vector

Orientation of the detected feature, specified as a numeric scalar or an *M*-element vector. Values must be an angle, in radians. The angle is measured counter-clockwise from the X-axis with the origin specified by the `Location` property. Do not set this property manually. Rely instead, on the call to `extractFeatures` to fill in this value. The `extractFeatures` function modifies the default value of `0.0`.The `Orientation` is mainly useful for visualization purposes.

`SignOfLaplacian` — Sign of Laplacian

`0` (default) | `-1` | `1`

Sign of the Laplacian determined during the detection process, specified as `-1`, `0`, or `1`. You can use this parameter to accelerate the feature matching process.

Blobs with identical metric values but different signs of Laplacian can differ by their intensity values. For example, a white blob on a background versus a black blob on a white background. You can use this parameter to quickly eliminate blobs that do not match.

For non-SURF detectors, this property is not relevant. For example, for corner features, you can simply use the default value of `0`.

Object Functions

plot	Plot detected feature points
isempty	Determine if points object is empty
length	Number of stored points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation

Examples

Detect SURF Features

Read in image.

```
I = imread('cameraman.tif');
```

Detect SURF features.

```
points = detectSURFFeatures(I);
```

Display location and scale for the 10 strongest points.

```
strongest = points.selectStrongest(10);
imshow(I); hold on;
plot(strongest);
```



Display [x y] coordinates for the 10 strongest points on command line.

```
strongest.Location
```

```
ans = 10×2 single matrix

  139.7482   95.9542
  107.4502  232.0347
  116.6112  138.2446
  105.5152  172.1816
  113.6975   48.7220
  104.4210   75.7348
  111.3914  154.4597
  106.2879  175.2709
  131.1298   98.3900
  124.2933   64.4942
```

Detect SURF Features and Display the Last 5 Points

Read in image.

```
I = imread('cameraman.tif');
```

Detect SURF feature.

```
points = detectSURFFeatures(I);
```

Display the last 5 points.

```
imshow(I); hold on;
plot(points(end-4:end));
```



Tips

Although `SURFPoints` may hold many points, it is a scalar object. Therefore, `numel(points)` always returns `1`. This value may differ from `length(points)`, which returns the true number of points held by the object.

Version History

Introduced in R2011b

References

[1] Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool. “SURF:Speeded Up Robust Features.” *Computer Vision and Image Understanding (CVIU)*.Vol. 110, No. 3, pp. 346–359, 2008.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To index locations with this object, use the syntax: `points.Location(idx,:)`, for `points` object. See visionRecovertformCodeGeneration_kernel.m, which is used in the “Introduction to Code Generation with Feature Matching and Registration” example.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects
`SIFTPoints` | `MSERRegions` | `BRISKPoints` | `cornerPoints` | `ORBPoints` | `KAZEPoints`

Functions
`detectSIFTFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectSURFFeatures` | `extractFeatures` | `matchFeatures`

Topics
“Create Panorama”
“Detect SURF Interest Points in a Grayscale Image” on page 3-615
“Display MSER Feature Regions from the MSERRegions Object” on page 2-827
“Find MSER Regions in an Image” on page 3-603
“Detect MSER Features in an Image” on page 2-826

ORBPoints

Object for storing ORB keypoints

Description

An `ORBPoints` object stores the Oriented FAST and rotated BRIEF (ORB) keypoints in an image. You can specify the keypoints and store them as an `ORBPoints` object. You can also use the `detectORBFeatures` function to detect the ORB keypoints in an image. The `detectORBFeatures` function stores the detected ORB keypoints as an `ORBPoints` object. Use “Object Functions” on page 2-842 to plot, select, and manipulate the detected ORB keypoints.

Creation

Syntax

```
points = ORBPoints
points = ORBPoints(location)
points = ORBPoints(location,Name,Value)
```

Description

`points = ORBPoints` creates an `ORBPoints` object with default property values.

`points = ORBPoints(location)` creates an `ORBPoints` object from a set of location coordinates specified by `location`. The location input sets the `Location` property.

`points = ORBPoints(location,Name,Value)` sets properties of the object using one or more name-value pair arguments. Enclose each property name in quotes. For example, `ORBPoints(location,'Count',15)` creates an `ORBPoints` object with `Count` property set to 15.

Properties

`Location` — Location of keypoints

[ ] (default) | *M*-by-2 matrix

This property is read-only.

Location of keypoints, specified as an *M*-by-2 matrix. Each row is of the form [*x* *y*] and represents the location of a keypoint. *M* is the number of keypoints. You cannot set this property as a name-value pair. Use the `location` input argument.

`Metric` — Strength of keypoints

[ ] (default) | scalar | *M*-element vector.

This property is read-only.

Strength of keypoints, specified as one of these values:

- A scalar — Detected keypoints have the same strength. In this case, the object assigns the same `Metric` value to all keypoints.
- An *M*-element vector — Detected keypoints have different strength values. In this case, the object assigns different `Metric` value to each keypoint. *M* is the number of keypoints.

Example: `ORBPoints(location,'Metric',0.5)`

Data Types: `single`

`Count` — Number of keypoints
`0` (default) | nonnegative integer

This property is read-only.

Number of keypoints held by the object, specified as a nonnegative integer.

`Scale` — Scale factor
`[]` (default) | scalar | *M*-element vector

This property is read-only.

Scale factor, specified as one of these values:

- A scalar — All keypoints are detected at the same level of decomposition. In this case, the object assigns the same `Scale` value to all keypoints.
- An *M*-element vector — The keypoints are detected at different levels of decomposition. In this case, the object assigns different `Scale` value to each keypoint. *M* is the number of keypoints.

The scale factor specifies the level of decomposition at which a keypoint is detected.

Example: `ORBPoints(location,'Scale',1.2)`

Data Types: `single`

`Orientation` — Angle of keypoints in radians
`[]` (default) | scalar | *M*-element vector

This property is read-only.

Angle of keypoints in radians, specified as one of these values:

- A scalar — Detected keypoints are of the same orientation. In this case, the object assigns the same `Orientation` value to all keypoints.
- An *M*-element vector — Detected keypoints are of different orientation. In this case, the object assigns different `Orientation` value to each keypoint. *M* is the number of keypoints.

The angle made by a keypoint is defined with reference to the horizontal axis of the image. The coordinate of the keypoint is set as the origin of the axis.

Example: `ORBPoints(location,'Orientation',0.7854)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Object Functions

isempty	Determine if points object is empty

length	Number of stored points
plot	Plot detected feature points
selectStrongest	Select points with strongest metrics
size	Return size of points object
selectUniform	Select uniformly distributed subset of point features
select	Select point or region features during code generation

Examples

Create and Modify Properties of ORBPoints

Read an image into the workspace.

```
I = imread('licensePlates.jpg');
```

Convert the image into a grayscale image.

```
I = im2gray(I);
```

Specify the location of keypoints in the image.

```
location = [400 398; ...
            485 343; ...
            274 323; ...
            274 367; ...
            241 313; ...
            302 213];
```

Create an `ORBPoints` object and display its properties.

```
points = ORBPoints(location)
```

```
points =
  6×1 ORBPoints array with properties:

       Location: [6×2 single]
         Metric: [6×1 single]
          Count: 6
          Scale: [6×1 single]
    Orientation: [6×1 single]
```

Inspect the `Scale` and `Orientation` properties of the `ORBPoints` object.

```
points.Scale
```

```
ans = 6×1 single column vector

     1
     1
     1
     1
     1
     1
```

```
points.Orientation
```

```
ans = 6×1 single column vector

     0
     0
     0
     0
     0
     0
```

Specify the scale value for each keypoint.

```
scale = [2.1 2.5 2.5 4 2.3 3.9];
```

Specify the angle values for the keypoints as 0.7854 radians.

```
angle = 0.7854;
```

Create an `ORBPoints` object with the keypoints and the updated scale and angle values. The output is an `ORBPoints` object containing keypoints with the modified `Scale` and `Orientation` properties.

```
points = ORBPoints(location,'Scale',scale,'Orientation',angle);
```

Inspect the updated `Scale` values.

```
points.Scale
```

```
ans = 6×1 single column vector

    2.1000
    2.5000
    2.5000
    4.0000
    2.3000
    3.9000
```

Inspect the updated `Orientation` values. Since the `Orientation` value is a scalar, the object assigns same value to all keypoints.

```
points.Orientation
```

```
ans = 6×1 single column vector

    0.7854
    0.7854
    0.7854
    0.7854
    0.7854
    0.7854
```

Display the image and plot the ORB keypoints on the image.

```
figure
imshow(I)
hold on
plot(points)
```



Detect and Store ORB Keypoints

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Use the `detectORBFeatures` function to detect ORB keypoints in the image. The function returns the detected ORB keypoints as an `ORBPoints` object.

```
points = detectORBFeatures(I);
```

Use the `selectUniform` object function to select 10 ORB keypoints. The output of the `selectUniform` object function is an `ORBPoints` object.

```
newPoints = selectUniform(points,10,size(I))
```

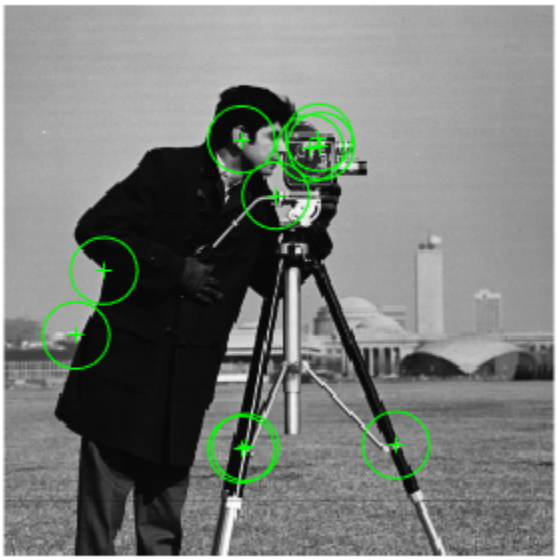
```
newPoints =
  10×1 ORBPoints array with properties:

       Location: [10×2 single]
         Metric: [10×1 single]
          Count: 10
          Scale: [10×1 single]
    Orientation: [10×1 single]
```

Display the location and scale of the selected keypoints on the image.

```
figure
imshow(I)
```

```
hold on
plot(newPoints)
```



Display the (*x*, *y*) coordinates of the selected keypoints.

```
x = newPoints.Location(:,1);
y = newPoints.Location(:,2);
[x y]
```

```
ans = 10×2 single matrix

   147    62
   111    63
   143    67
   127    89
    47   124
    34   154
   183   205
   113   206
   111   207
   148    66
```

Display the orientation of the selected keypoints.

```
newPoints.Orientation
```

```
ans = 10×1 single column vector

    5.4682
    0.7888
    5.3084
    6.2443
    2.8221
```

```
    3.8440
    6.1212
    6.0344
    4.8840
    5.5535
```

Tips

Although `ORBPoints` can hold many points, it is a scalar object. Therefore, `numel(ORBPoints)` always returns 1. This value can differ from `length(ORBPoints)`, which returns the true number of points held by the object.

Version History

Introduced in R2019a

References

[1] Rublee, E., V. Rabaud, K. Konolige, and G. Bradski. "ORB: An Efficient Alternative to SIFT or SURF." In *Proceedings of the 2011 International Conference on Computer Vision*, pp. 2564–2571. Barcelona, Spain: IEEE, 2011.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

This function supports C/C++ code generation with the limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`extractFeatures` | `detectORBFeatures` | `matchFeatures` | `SURFPoints` | `KAZEPoints` | `BRISKPoints` | `MSERRegions` | `cornerPoints` | `detectHarrisFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `detectMinEigenFeatures`

Topics

“Point Feature Types”
“Local Feature Detection and Extraction”

vision.AlphaBlender

Namespace: `vision`

(Not recommended) Combine images, overlay images, or highlight selected pixels

Note The `vision.AlphaBlender` System object™ is not recommended. Use the `imblend` function to blend two images, instead.

Description

The `AlphaBlender` System object combines two images, overlays one image over another, or highlights selected pixels.

To combine two images, overlay one image over another, or highlight selected pixels:

1. Create the `vision.AlphaBlender` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
alphablend = vision.AlphaBlender
alphablend = vision.AlphaBlender(Name,Value)
```

Description

`alphablend = vision.AlphaBlender` creates an alpha blender System object, `alphablend`, which combines two images, overlays one image over another, or highlights selected pixels.

`alphablend = vision.AlphaBlender(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `alphablend = vision.AlphaBlender('Operation','Blend')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Operation` — Operation

`'Blend'` | `'Binary mask'` | `'Highlight selected pixels'`

Operation, specified as `'Blend'`, `'Binary mask'`, or `'Highlight selected pixels'`.

Operation	Description
`'Blend'`	Linearly combine the pixels of one image with the another image.
`'Binary mask'`	Overwrite the pixel values of one image with the pixel values of another image.
`'Highlight selected pixels'`	Uses the `mask` binary image input to determine which pixels are set to the maximum value supported by their data type.

`OpacitySource` — Source of opacity factor
`Property` (default) | `'Input port'`

Source of opacity factor, specified as `'Property'` or `'Input port'`. This property applies when you set the `Operation` property to `Blend`.

`Opacity` — Pixel scale value
`0.75` (default) | scalar | matrix

Pixel scale value, specified as a scalar or matrix. The object scales each pixel before combining the images. This property applies when you set the `OpacitySource` property to `Property`.

Tunable: Yes

`MaskSource` — Binary mask source
`'Property'` (default) | `'Input port'`

Binary mask source, specified as `'Property'` or `'Input port'`. This property applies when you set the `Operation` property to `Binary mask`

`Mask` — Overwrite scalar
`1` (default) | `0` | `binary scalar` | matrix of binary scalars

Overwrite scalar, specified as `0` or `1` for every pixel, or a matrix of `0`s and `1`s. This property applies when you set the `MaskSource` property to `Property`.

Tunable: Yes

`LocationSource` — Source for upper-left location
`'Property'` (default) | `'Input port'`

Source of location of the upper-left corner of second input image

Source for the upper-left corner location of the second input image, specified as `Property` or `Input port`.

`Location` — Location
`[1 1]` | two-element vector

Location of the upper-left corner of the second input image relative to the location of the first input image, specified as a two-element vector in the format [x y]. This property applies when you set the `LocationSource` property to `Property`.

See “Coordinate Systems” for a discussion on pixel coordinates and spatial coordinates, which are the two main coordinate systems used in the Computer Vision Toolbox.

Tunable: Yes

Fixed-Point Properties

`RoundingMethod` — Rounding method
`'Floor'` (default) | `'Convergent'` | `'Floor'` | ...

Rounding method, specified as `'Ceiling'`,`'Convergent'`, `'Floor'`, `'Nearest'`, `'Round'`, `'Simplest'`, or `'Zero'`.

`OverflowAction` — Overflow action
`'Wrap'` (default) | `'Saturate'`

Overflow action for fixed-point operations, specified as `'Wrap'` or `'Saturate'`.

`OpacityDataType` — Opacity word and fraction lengths
`'Same word length as input'` (default) | `'Custom'`

Opacity word and fraction lengths for fixed-point operations, specified as `'Same word length as input'` or `'Custom'`.

`CustomOpacityDataType` — Opacity word and fraction lengths
`numerictype([],16)` (default) | unscaled numerictype object

Opacity word and fraction lengths factor for fixed-point operations, specified as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OpacityDataType` property to `Custom`

`ProductDataType` — Product word and fraction lengths
`'Custom'` (default) | `'Same as first input'`

Product word and fraction lengths, specified as `'Custom'` or `'Same as first input'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype([],32,10)` (default) | scaled numerictype object

Product word and fraction lengths, specified as a scaled `numerictype` object with a `'Signedness'` of `'Auto'`. This property applies when you set the `'ProductDataType'` property to `'Custom'`

`AccumulatorDataType` — Accumulator word and fraction lengths
`'Same as product'` (default) | `'Custom'`

Accumulator word and fraction lengths, specified as `'Same as product'`, `'Same as first input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype([],32,10)` (default) | scaled numerictype object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object with a `'Signedness'` of `'Auto'`.

`OutputDataType` — Output word and fraction lengths
`'Same as product'` (default) | `'Custom'`

Output word and fraction lengths, specified as `'Same as first input'` or `'Custom'`.

`CustomOutputDataType` — Output word and fraction lengths
`numerictype([],32,10)` (default) | scaled numerictype object

Output word and fraction lengths, specified as a scaled `numerictype` object with a `'Signedness'` of `'Auto'`.

Usage

Syntax

```
ab = alphablend(I1,I2)
ab = alphablend(I1,I2,opacity)
ab = alphablend(I1,I2,mask)
ab = alphablend(I1,mask)
ab = alphablend(I1,I2,___,location)
```

Description

`ab = alphablend(I1,I2)` returns an alpha blender object that blends images `I1` and `I2`.

`ab = alphablend(I1,I2,opacity)` uses the `opacity` input to combine pixel values of `I1` and `I2`. To use the `opacity` input, you must set the `Operation` property to '`Blend`' and the `OpacitySource` property to '`Input port`'

`ab = alphablend(I1,I2,mask)` uses the `mask` input to overlay `I1` over `I2`. To use the `mask` input with an overlay, you must set the `Operation` property to `'Binary mask'` and the `MaskSource` property to `'Input port'`.

`ab = alphablend(I1,mask)` uses the `mask` input to determine which pixels in `I1` are set to the maximum value supported by their data type. To use the `mask` input with an overlay, you must set the `Operation` property to `'Highlight selected pixels'` and the `MaskSource` property to `'Input port'`.

`ab = alphablend(I1,I2,___,location)` additionally sets the upper-left corner `location` for `I2`. To use the `location` input, you must set the `LocationSource` property to `'Input port'`.

Input Arguments

`I1` — Input image
M-by-*N*-by-3 truecolor | *M*-by-*N* 2-D grayscale image

Input image, specified in truecolor or 2-D grayscale.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`I2` — Input image
M-by-*N*-by-3 truecolor | *M*-by-*N* 2-D grayscale image

Input image, specified in truecolor or 2-D grayscale.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`opacity` — Scale
scalar | matrix

Pixel scale value, specified as a scalar or matrix. The object scales each pixel before combining the images.

`mask` — Overwrite scalar
`binary scalar` | matrix of binary scalars

Overwrite scalar, specified as `0` or `1` for every pixel, or a matrix of `0`s and `1`s.

`location` — Location
two-element vector

Location of the upper-left corner of the second input image relative to the location of the first input image, specified as a two-element vector in the format [x y].

Tunable: Yes

Output Arguments

`alphablend` — Blended output image
M-by-*N*-by-3 truecolor | *M*-by-*N* 2-D grayscale image

Output blended image, returned as a combined image, overlaid image, or an image with selected pixels highlighted.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

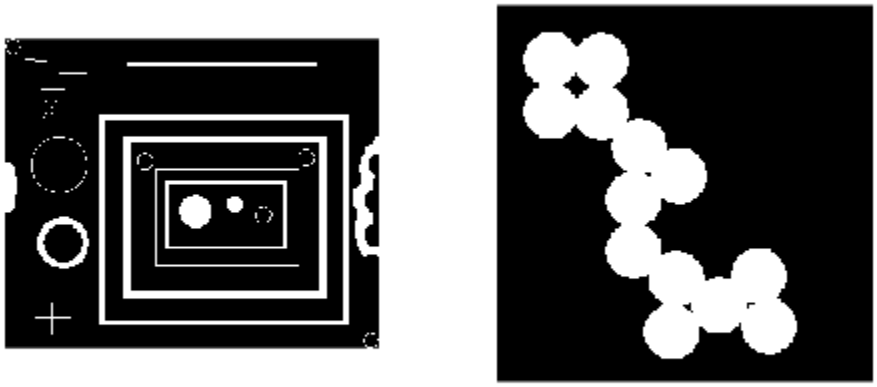
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Blend Two Images

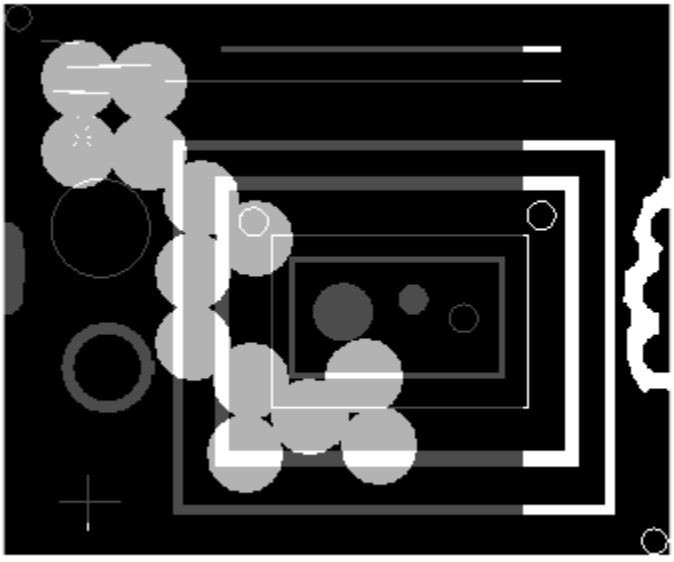
Display the two images.

```
I1 = im2single(imread("blobs.png"));
I2 = im2single(imread("circles.png"));
subplot(1,2,1);
imshow(I1);
subplot(1,2,2);
imshow(I2);
```



Blend the two images and display the result.

```
J = imblend(I2,I1);
figure;
imshow(J);
```



Version History

Introduced in R2012a

R2026a: vision.AlphaBlender system object warns
Warns starting in R2026a

The `vision.AlphaBlender` system object issues a warning that it will be removed in a future release. Use the `imblend` function to blend two images, instead.

R2024b: vision.AlphaBlender system object is not recommended
Not recommended starting in R2024b

The `vision.AlphaBlender` system object is not recommended and will be removed in a future release. Use the `imblend` function to blend two images, instead.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`insertText`

vision.BinaryFileReader

Namespace: `vision`

Read video data from binary files

Description

The `BinaryFileReader` object reads video data from binary files.

To read a binary file:

1. Create the `vision.BinaryFileReader` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
bFileReader = vision.BinaryFileReader
bFileReader = vision.BinaryFileReader(Name,Value)
bFileReader = vision.BinaryFileReader(file,Name,Value)
```

Description

`bFileReader = vision.BinaryFileReader` returns a binary file reader object, `bFileReader`, that reads binary video data from the specified file in `I420` Four Character Code (FOURCC) video format.

`bFileReader = vision.BinaryFileReader(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `bFileReader = vision.BinaryFileReader('Filename','vipmen.bin')`

`bFileReader = vision.BinaryFileReader(file,Name,Value)` returns a binary file reader System object, `bFileReader`, with the `Filename` property set to `file` and other specified properties set to the specified values.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Filename` — File name
`vipmen.bin` (default) | character vector

File name, specified as a character vector. The full path for the file needs to be specified only if the file is not on the MATLAB path.

`VideoFormat` — Video file format
`'Four character codes'` (default) | `'Custom'`

Video file format, specified as `'Four character codes'` or `'Custom'`.

`FourCharacterCode` — Four Character Code video format
binary file format

Four Character Code video format, specified as one of the available video formats. For more information on four character codes, see https://en.wikipedia.org/wiki/FourCC. This property applies when you set the `VideoFormat` property to `'Four character codes'`.

`BitstreamFormat` — Format of data
`'Planar'` (default) | `'Packed'`

Format of data, specified as `'Planar'` or `'Packed'`.This property applies when you set the `VideoFormat` property to `Custom`.

`OutputSize` — Size of output
M-by-*N* matrix

Size of output, specified as an *M*-by-*N* matrix. This property applies when you set the `BitstreamFormat` property to `'Packed'`.

`VideoComponentCount` — Number of video components in video stream
3 (default) | 1 | 2 | 4

Number of video components in video stream, specified as 1, 2, 3 or 4. This number corresponds to the number of video component outputs. This property applies when you set the `VideoFormat` property to `'Custom'`.

`VideoComponentBits` — Bit size of video components
`[8 8 8]` (default) | vector

Bit size of video components, specified as an integer vector of length *N*, where *N* is the value of the `VideoComponentCount` property. This property applies when you set the `VideoFormat` property to `'Custom'`.

`VideoComponentSizes` — Size of video components
`[120 160; 60 80; 60 80]` (default) | *N*-by-2 array

Size of video components, specified as an *N*-by-2 array, where *N* is the value of the `VideoComponentCount` property. Each row of the matrix corresponds to the size of that video component, with the first element denoting the number of rows and the second element denoting the number of columns. This property applies when you set the `VideoFormat` property to `'Custom'` and the `BitstreamFormat` property to `'Planar'`.

`VideoComponentOrder` — Order of video components
`[1 2 3]` (default) | 1-by-*N* vector

Order of video components, specified as a 1-by-*N* vector. This property must be set to a vector of length *N*, where *N* is set according to how you set the `BitstreamFormat` property. When you set the `BitStreamFormat` property to `'Planar'`, you must set *N* equal to the value of the `VideoComponentCount` property. Otherwise, you can set *N* equal to or greater than the value of the `VideoComponentCount` property. This property applies when you set the `VideoFormat` property to `'Custom'`.

`InterlacedVideo` — Interlaced video status
`false` (default) | `true`

Interlaced video status, specified as `true` or `false`. Set this property to `true` if the video stream represents interlaced video data. This property applies when you set the `VideoFormat` property to `'Custom'`.

`LineOrder` — Fill binary file format
`'Top line first'` (default) | `'Bottom line first'`

Fill binary file format, specified as `'Top line first'`, or `'Bottom line first'`. If you set this property to `'Top line first'`, the first row of the video frame gets filled first. If you set this property to `'Bottom line first'`, the last row of the video frame gets filled first.

`SignedData` — Signed data
`false` (default) | `true`

Signed data, specified as `true` or `false`. Set this property to `true` for signed input data. This property applies when you set the `VideoFormat` property to `'Custom'`

`ByteOrder` — Byte order
`'Little endian'` (default) | `'Big endian'`

Byte order, specified as `'Little endian'` or `'Big endian'`. This property applies when you set the `VideoFormat` property to `'Custom'`.

`PlayCount` — Play count
1 (default) | positive integer

Play count for the number of times to play the file, specified as a positive integer.

Usage

Syntax

```
[Y,Cb,Cr] = bFileReader()
Y = bFileReader()
[Y,Cb,Cr,Alpha] = bFileReader()
[___,EOF] = bFileReader()
```

Description

`[Y,Cb,Cr] = bFileReader()` reads the luminance, `Y` and chrominance, `Cb` and `Cr` components of a video stream from the specified binary file when you set the `VideoFormat` property to `'Four character codes'`. Alternatively, it reads the values when you set the `VideoFormat` property to `'Custom'` and the `VideoComponentCount` property to 2.

`Y = bFileReader()` reads the video component `Y` from the binary file when you set the `VideoFormat` property to `'Custom'` and the `VideoComponentCount` property to `1`.

`[Y,Cb,Cr,Alpha] = bFileReader()` additionally reads the `Alpha` when you set the `VideoFormat` property to `'Custom'` and the `VideoComponentCount` property to `4`.

`[ ___ ,EOF] = bFileReader()` also returns the end-of-file indicator, `EOF`. `EOF` is set to `true` when the output contains the last video frame in the file.

Output Arguments

`Y` — Luminance value
M-by-*N* matrix

Luminance value, returned as an *M*-by-*N* matrix.

`Cb` — Chrominance value
M-by-*N* matrix

Chrominance value, returned as an *M*-by-*N* matrix.

`Cr` — Chrominance value
M-by-*N* matrix

Chrominance value, returned as an *M*-by-*N* matrix.

`Alpha` — Transparency
scalar

Transparency value, returned as a scalar in the range [0,1].

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.BinaryFileReader

isDone	End-of-file status (logical)

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Read Binary Video File and Play Back on Screen

Create a binary file reader and video player object.

```
hbfr = vision.BinaryFileReader();
hvp = vision.VideoPlayer;
```

Use the while loop to play the default video.

```
while ~isDone(hbfr)
y = hbfr();
hvp(y);
end
```



Close the input file and the video display.

```
release(hbfr);
release(hvp);
```



Version History

Introduced in R2012a

See Also

`VideoReader` | `vision.BinaryFileWriter`

vision.BinaryFileWriter

Namespace: `vision`

Write binary video data to files

Description

The `BinaryFileWriter` object writes binary video data to files.

To write binary data to a file:

1. Create the `vision.BinaryFileWriter` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
bFileWriter = vision.BinaryFileWriter
bFileWriter = vision.BinaryFileWriter(Name,Value)
```

Description

`bFileWriter = vision.BinaryFileWriter` returns a binary writer object that writes binary video data to an output file, `output.bin` in the `I420` Four Character Code format.

`bFileWriter = vision.BinaryFileWriter(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `bFileWriter = vision.BinaryFileWriter('Filename','output.bin')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Filename` — File name

`output.bin` (default) | character vector

File name, specified as a character vector.

`VideoFormat` — Video file format

`'Four character codes'` (default) | `'Custom'`

Video file format, specified as `'Four character codes'` or `'Custom'`.

`FourCharacterCode` — Four Character Code video format
binary file format

Four Character Code video format, specified as one of the available video formats. For more information on four character codes, see https://en.wikipedia.org/wiki/FourCC. This property applies when you set the `VideoFormat` property to `'Four character codes'`.

`BitstreamFormat` — Format of data
`'Planar'` (default) | `'Packed'`

Format of data, specified as `'Planar'` or `'Packed'`.This property applies when you set the `VideoFormat` property to `Custom`.

`VideoComponentCount` — Number of video components in video stream
3 (default) | 1 | 2 | 4

Number of video components in video stream, specified as 1, 2, 3 or 4. This number corresponds to the number of video component outputs. This property applies when you set the `VideoFormat` property to `'Custom'`.

`VideoComponentBitsSource` — Size of video components format
`'Auto'` (default) | `Property'`

Size of video components format, specified as `'Auto'` or `'Property'`. If this property is set to `'Auto'`, each component will have a `VideoComponentBits` property. This property applies when you set the `VideoFormat` property to `'Custom'`.

`VideoComponentBits` — Bit size of video components
`[8 8 8]` (default) | vector

Bit size of video components, specified as an integer vector of length *N*, where *N* is the value of the `VideoComponentCount` property. This property applies when you set the `VideoFormat` property to `'Custom'`.

`VideoComponentOrder` — Order of video components
`[1 2 3]` (default) | 1-by-*N* vector

Order of video components, specified as a 1-by-*N* vector. This property must be set to a vector of length *N*, where *N* is set according to how you set the `BitstreamFormat` property. When you set the `BitStreamFormat` property to `'Planar'`, you must set *N* equal to the value of the `VideoComponentCount` property. Otherwise, you can set *N* equal to or greater than the value of the `VideoComponentCount` property. This property applies when you set the `VideoFormat` property to `'Custom'`.

`InterlacedVideo` — Interlaced video status
`false` (default) | `true`

Interlaced video status, specified as `true` or `false`. Set this property to `true` if the video stream represents interlaced video data. This property applies when you set the `VideoFormat` property to `'Custom'`.

`LineOrder` — Fill binary file format
`'Top line first'` (default) | `'Bottom line first'`

Fill binary file format, specified as `'Top line first'`, or `'Bottom line first'`. If you set this property to `'Top line first'`, the first row of the video frame gets filled first. If you set this property to `'Bottom line first'`, the last row of the video frame gets filled first.

`SignedData` — Signed data
`false` (default) | `true`

Signed data, specified as `true` or `false`. Set this property to `true` for signed input data. This property applies when you set the `VideoFormat` property to `'Custom'`

`ByteOrder` — Byte order
`'Little endian'` (default) | `'Big endian'`

Byte order, specified as `'Little endian'` or `'Big endian'`. This property applies when you set the `VideoFormat` property to `'Custom'`.

Usage

Syntax

```
bFileWriter(Y,Cb,Cr)
bFileWriter(Y)
bFileWriter(Y,Cb)
bFileWriter(Y,Cb,Cr)
bFileWriter(Y,Cb,Cr,Alpha)
```

Description

`bFileWriter(Y,Cb,Cr)` writes one frame of video to the specified output file. `Y` , `Cb`, `Cr` represent the luma (Y) and chroma (Cb and Cr) components of a video stream. This option applies when you set the `VideoFormat` property to `'Four character codes'`.

`bFileWriter(Y)` writes video component `Y` to the output file when the `VideoFormat` property is set to `'Custom'` and the `VideoComponentCount` property is set to `1`.

`bFileWriter(Y,Cb)` writes video components `Y` and `Cb` to the output file when the `VideoFormat` property is `'Custom'` and the `VideoComponentCount` property is set to `2`.

`bFileWriter(Y,Cb,Cr)` writes video components `Y` , `Cb` and `Cr` to the output file when the `VideoFormat` property is set to `'Custom'` and the `VideoComponentCount` property is set to `3`.

`bFileWriter(Y,Cb,Cr,Alpha)` writes video components `Y` , `Cb`, `Cr` and `Alpha` to the output file when the `VideoFormat` property is set to `'Custom'`, and the `VideoComponentCount` property is set to `4`.

Input Arguments

`Y` — Luminance value
M-by-*N* matrix

Luminance value, returned as an *M*-by-*N* matrix.

`Cb` — Chrominance value
M-by-*N* matrix

Chrominance value, returned as an *M*-by-*N* matrix.

`Cr` — Chrominance value
M-by-*N* matrix

Chrominance value, returned as an *M*-by-*N* matrix.

`Alpha` — Transparency
scalar

Transparency value, returned as a scalar in the range [0,1].

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Write Video Data to a Binary File

Set the output file name and create binary file reader and writer objects.

```
filename = fullfile(tempdir,'output.bin');
bFileReader = vision.BinaryFileReader;
bFileWriter = vision.BinaryFileWriter(filename);
```

Write to the file.

```
while ~isDone(bFileReader)
    [y,cb,cr] = bFileReader();
    bFileWriter(y,cb,cr);
end
```

Close the files.

```
release(bFileReader);
release(bFileWriter);
```

Version History

Introduced in R2012a

See Also

`VideoReader` | `vision.BinaryFileReader`

vision.BlobAnalysis

Namespace: `vision`

Properties of connected regions

Description

To compute statistics for connected regions in a binary image

To track a set of points:

1. Create the `vision.BlobAnalysis` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
Hblob = vision.BlobAnalysis
Hblob = vision.BlobAnalysis(Name,Value)
```

Description

`Hblob = vision.BlobAnalysis` returns a blob analysis object, `H`, used to compute statistics for connected regions in a binary image.

`Hblob = vision.BlobAnalysis(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `Hblob = vision.BlobAnalysis('AreaOutputPort',true)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`AreaOutputPort` — Return blob area
`true` (default) | `false`

Return blob area, specified as `true` or `false`.

`CentroidOutputPort` — Return coordinates of blob centroids
`true` (default) | `false`

Return coordinates of blob centroids, specified as `true` or `false`.

`BoundingBoxOutputPort` — Return coordinates of bounding boxes
`true` (default) | `false`

Return coordinates of bounding boxes, specified as `true` or `false`.

`MajorAxisLengthOutputPort` — Return vector whose values represent lengths of ellipses' major axes
`false` (default) | `true`

Return vector whose values represent lengths of ellipses' major axes, specified as `true` or `false`. Set this property to `true` to output a vector whose values represent the lengths of the major axes of the ellipses that have the same normalized second central moments as the labeled regions. This property applies when you set the `OutputDataType` property to `double` or `single`.

`MinorAxisLengthOutputPort` — Return vector whose values represent lengths of ellipses' minor axes
`false` (default) | `true`

Return vector whose values represent lengths of ellipses' minor axes, specified as `true` or `false`. Set this property to `true` to output a vector whose values represent the lengths of the minor axes of the ellipses that have the same normalized second central moments as the labeled regions. This property is available when the `OutputDataType` property is `double` or `single`.

`OrientationOutputPort` — Return vector whose values represent angles between ellipses' major axes and x-axis
`false` (default) | `true`

Return vector whose values represent angles between ellipses' major axes and x-axis, specified as `true` or `false`. Set this property to true to output a vector whose values represent the angles between the major axes of the ellipses and the x-axis. This property applies when you set the `OutputDataType` property to `double` or `single`.

`EccentricityOutputPort` — Return vector whose values represent ellipses' eccentricities
`false` (default) | `true`

Return vector whose values represent ellipses' eccentricities, specified as `true` or `false`. Set this property to `true` to output a vector whose values represent the eccentricities of the ellipses that have the same second moments as the region. This property applies when you set the `OutputDataType` property to `double` or `single`.

`EquivalentDiameterSquaredOutputPort` — Return vector whose values represent equivalent diameters squared
`false` (default) | `true`

Return vector whose values represent equivalent diameters squared, specified as `true` or `false`. Set this property to `true` to output a vector whose values represent the equivalent diameters squared.

`ExtentOutputPort` — Return vector whose values represent results of dividing blob areas by bounding box areas
`false` (default) | `true`

Return vector whose values represent results of dividing blob areas by bounding box areas, specified as `true` or `false`.

`PerimeterOutputPort` — Return vector whose values represent estimates of blob perimeter lengths
`false` (default) | `true`

Return vector whose values represent estimates of blob perimeter lengths, specified as `true` or `false`.

`OutputDataType` — Output data type of statistics
`double` (default) | `single` | `Fixed point`

Output data type of statistics, specified as `double`,`single`, or `Fixed point`. Area and bounding box outputs are always an `int32` data type. `Major axis length`, `Minor axis length`, `Orientation` and `Eccentricity` do not apply when you set this property to `Fixed point`.

`Connectivity` — Connected pixels
`8` (default) | `4`

Connected pixels, specified as `4` or `8`.

`LabelMatrixOutputPort` — Maximum number of labeled regions in each input image
`50` (default) | positive scalar integer.

Maximum number of labeled regions in each input image, specified as a positive scalar integer. The maximum number of blobs the object outputs depends on both the value of this property, and on the size of the input image. The number of blobs the object outputs may be limited by the input image size.

`MinimumBlobArea` — Minimum blob area in pixels
`0` (default) | positive scalar integer.

Minimum blob area in pixels, specified as positive scalar integer.

Tunable: Yes

`MaximumBlobArea` — Maximum blob area in pixels
`intmax('uint32')` (default) | integer

Maximum blob area in pixels, specified as an integer.

Tunable: Yes

`ExcludeBorderBlobs` — Exclude blobs that contain at least one image border pixel
`false` (default) | `true`

Exclude blobs that contain at least one image border pixel, specified as `true` or `false`.

`MaximumCount` — Maximum number of labeled regions in each input image
`50` (default) | positive scalar integer

Maximum number of labeled regions in each input image, specified as a positive scalar integer. The maximum number of blobs the object outputs depends on both the value of this property, and on the size of the input image. The number of blobs the object outputs may be limited by the input image size

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[area,centroid,bbox] = Hblob(bw)
[___,majoraxis] = Hblob(bw)
[___,minoraxis] = Hblob(bw)
[bw___,orientation] = Hblob(bw)
[___,eccentricity] = Hblob(bw)
[___,EQDIASQ] = Hblob(bw)
[___,EXTENT] = Hblob(bw)
[___,perimeter] = Hblob(bw)
[___,label] = Hblob(bw)
```

Description

`[area,centroid,bbox] = Hblob(bw)` returns the area, centroid, and the bounding box of the blobs when the `AreaOutputPort`, `CentroidOutputPort` and `BoundingBoxOutputPort` properties are set to `true`. These are the only properties that are set to true by default. If you set any

additional properties to `true`, the corresponding outputs follow the `area`,`centroid`, and `bbox` outputs.

`[ ___ ,majoraxis] = Hblob(bw)` computes the major axis length `majoraxis` of the blobs found in input binary image `bw` when you set the `MajorAxisLengthOutputPort` property to `true`.

`[ ___ ,minoraxis] = Hblob(bw)` computes the minor axis length `minoraxis` of the blobs found in input binary image `BW` when you set the `MinorAxisLengthOutputPort` property to `true`.

`[bw ___ ,orientation] = Hblob(bw)` computes the `orientation` of the blobs found in input binary image `bw` when you set the `OrientationOutputPort` property to `true`.

`[ ___ ,eccentricity] = Hblob(bw)` computes the `eccentricity` of the blobs found in input binary image `bw` when you set the `EccentricityOutputPort` property to `true`.

`[ ___ ,EQDIASQ] = Hblob(bw)` computes the equivalent diameter squared `EQDIASQ` of the blobs found in input binary image `bw` when you set the `EquivalentDiameterSquaredOutputPort` property to `true`.

`[ ___ ,EXTENT] = Hblob(bw)` computes the `EXTENT` of the blobs found in input binary image `bw` when the `ExtentOutputPort` property is set to `true`.

`[ ___ ,perimeter] = Hblob(bw)` computes the `perimeter` of the blobs found in input binary image `bw` when you set the `PerimeterOutputPort` property to `true`.

`[ ___ ,label] = Hblob(bw)` returns a label matrix `label` of the blobs found in input binary image `bw` when you set the `LabelMatrixOutputPort` property to true.

Input Arguments

`bw` — Binary image
vector | matrix

Binary image, specified as a vector or matrix.

Output Arguments

`area` — Number of pixels in labeled regions
vector

Number of pixels in labeled regions, specified as a vector

`centroid` — Centroid coordinates
M-by-2 matrix.

Centroid coordinates, specified as an *M*-by-2 matrix. *M* is the number of blobs.

`bbox` — Bounding box coordinates
M-by-4 matrix

M-by-4 matrix in the format [*x y width height*], where M represents the number of blobs and [*x*,*y*] represents the upper-left corner of the bounding box.

`majoraxis` — Lengths of major axes of ellipses
vector

Lengths of major axes of ellipses, specified as a vector.

`minoraxis` — Lengths of minor axes of ellipses
vector

Lengths of minor axes of ellipses, specified as a vector.

`orientation` — Angles between the major axes of the ellipses and the x-axis
vector

Angles between the major axes of the ellipses and the x-axis, specified as a vector.

`eccentricity` — Eccentricities of the ellipses
vector

Eccentricities of the ellipses, specified as a vector

`EQDIASQ` — Equivalent diameter squared
vector

Equivalent diameter squared, specified as a vector.

`extent` — Ratio between area and bounding box
vector

Ratio between area and bounding box, specified as a vector. The vector contains the results of dividing the areas of the blobs by the area of their bounding boxes

`perimeter` — Estimate of the perimeter length
vector

Estimate of the perimeter length, specified as a vector. The vector contains an estimate of the perimeter length, in pixels, for each blob.

`label` — Label matrix
matrix

Label matrix, specified as a matrix.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Find Centroid of Blob

Create the blob analysis object.

```
hBlob = vision.BlobAnalysis('AreaOutputPort',false,'BoundingBoxOutputPort',false);
```

Create the blob.

```
img = logical([0 0 0 0 0 0; ...
         0 1 1 1 1 0; ...
         0 1 1 1 1 0; ...
         0 1 1 1 1 0; ...
         0 0 0 0 0 0]);
```

Find the coordinates for the centroid.

```
centroid = hBlob(img);
```

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`graythresh` | `multithresh`

Topics

“Cell Counting”

vision.BlockMatcher

Namespace: `vision`

Estimate motion between images or video frames

Description

To estimate motion between images or video frames.

1. Create the `vision.BlockMatcher` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
blkMatcher = vision.BlockMatcher
blkMatcher = vision.BlockMatcher(Name,Value)
```

Description

`blkMatcher = vision.BlockMatcher` returns an object, `blkMatcher`, that estimates motion between two images or two video frames. The object performs this estimation using a block matching method by moving a block of pixels over a search region.

`blkMatcher = vision.BlockMatcher(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `blkMatcher = vision.BlockMatcher('ReferenceFrameSource','Input port')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ReferenceFrameSource` — Reference frame source
`'Property'` (default) | `'Input port'`

Reference frame source, specified as `'Input port'` or `'Property'`. When you set the `ReferenceFrameSource` property to `'Input port'`, a reference frame input must be specified to the `step` method of the block matcher object.

`ReferenceFrameDelay` — Number of frames between reference and current frames
`1` (default) | scalar integer

Number of frames between reference and current frames, specified as a scalar integer greater than or equal to zero. This property applies when you set the `ReferenceFrameSource` property to `'Property'`.

`SearchMethod` — Best match search method
`'Exhaustive'` (default) | `'Three-step'`

Best match search method, specified as `'Exhaustive'` or `'Three-step'`. Specify how to locate the block of pixels in frame k+1 that best matches the block of pixels in frame k. If you set this property to `'Exhaustive'`, the block matcher object selects the location of the block of pixels in frame k+1. The block matcher does so by moving the block over the search region one pixel at a time, which is computationally expensive.

If you set this property to `'Three-step'`, the block matcher object searches for the block of pixels in frame k+1 that best matches the block of pixels in frame k using a steadily decreasing step size. The object begins with a step size approximately equal to half the maximum search range. In each step, the object compares the central point of the search region to eight search points located on the boundaries of the region and moves the central point to the search point whose values is the closest to that of the central point. The object then reduces the step size by half, and begins the process again. This option is less computationally expensive, though sometimes it does not find the optimal solution.

`BlockSize` — Size of block in pixels
`[17 17]` (default) | two-element vector

Size of block, specified in pixels as a two-element vector.

`MaximumDisplacement` — Maximum displacement search
`[7 7]` (default) | two-element vector

Maximum displacement search, specified as a two-element vector. Specify the maximum number of pixels that any center pixel in a block of pixels can move, from image to image or from frame to frame. The block matcher object uses this property to determine the size of the search region.

`MatchCriteria` — Match criteria between blocks
`'Mean square error (MSE)'` (default) | `'Mean absolute difference (MAD')`

Match criteria between blocks, specified as `'Mean square error (MSE)'` or `'Mean absolute difference (MAD')`.

`OutputValue` — Motion output form
`'Magnitude-squared'` (default) | `'Horizontal and vertical components in complex form'`

Motion output form, specified as `'Magnitude-squared'` or `'Horizontal and vertical components in complex form'`.

`Overlap` — Input image subdivision overlap
`[0 0]` (default) | two-element vector

Input image subdivision overlap, specified in pixels as a two-element vector.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
V = blkMatcher(I)
C = blkMatcher(I)
Y = blkMatcher(I,iref)
```

Description

`V = blkMatcher(I)` computes the motion of input image `I` from one video frame to another, and returns `V` as a matrix of velocity magnitudes.

`C = blkMatcher(I)` computes the motion of input image `I` from one video frame to another, and returns `C` as a complex matrix of horizontal and vertical components, when you set the `OutputValue` property to `Horizontal and vertical components in complex form`.

`Y = blkMatcher(I,iref)` computes the motion between input image `I` and reference image `iref` when you set the `ReferenceFrameSource` property to `Input port`.

Input Arguments

`I` — Input data
scalar | vector | matrix

Input data, specified as a scalar, vector, or matrix of intensity values.

`iref` — Input reference data
scalar | vector | matrix

Input reference data, specified as a scalar, vector, or matrix of intensity values.

Output Arguments

`V` — Velocity magnitudes
matrix

Velocity magnitudes, returned as a matrix.

`C` — Horizontal and vertical components
matrix

Horizontal and vertical components, returned as a complex matrix.

`Y` — Motion between inputs
matrix

Motion between image and reference image, returned as a matrix.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Estimate Motion Using BlockMatcher

Read and convert an RGB image to grayscale.

```
img1 = im2double(im2gray(imread("onion.png")));
```

Create a block matcher and alpha blender object.

```
hbm = vision.BlockMatcher("ReferenceFrameSource",...
        "Input port","BlockSize",[35 35]);
hbm.OutputValue = "Horizontal and vertical components in complex form";
```

Offset the first image by [5 5] pixels to create a second image.

```
img2 = imtranslate(img1,[5,5]);
```

Compute motion for the two images.

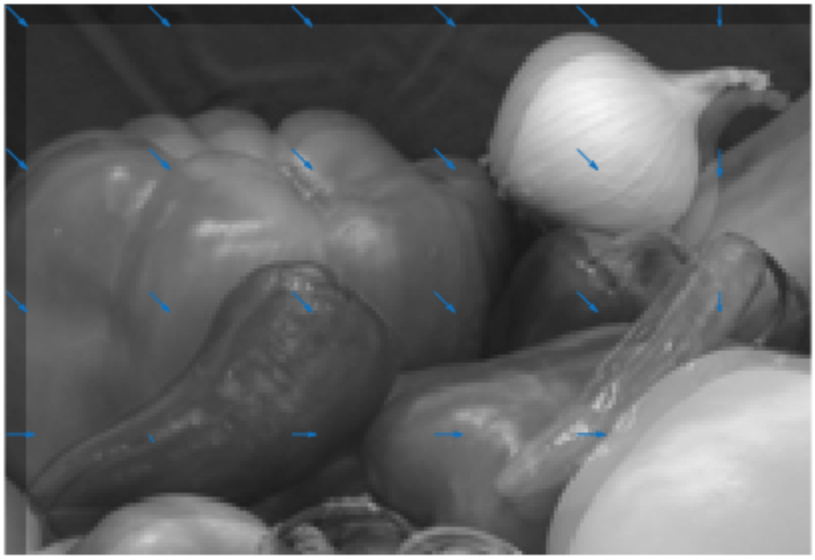
```
motion = hbm(img1,img2);
```

Blend the two images.

```
img12 = imblend(img1,img2);
```

Use a quiver plot to show the direction of motion on the images.

```
[X,Y] = meshgrid(1:35:size(img1,2),1:35:size(img1,1));
imshow(img12)
hold on
quiver(X(:),Y(:),real(motion(:)),imag(motion(:)),0)
hold off
```



Version History

Introduced in R2012a

See Also

`opticalFlow` | `opticalFlowFarneback` | `opticalFlowHS` | `opticalFlowLK` | `opticalFlowLKDoG`

vision.CascadeObjectDetector

Namespace: `vision`

Detect objects using the Viola-Jones algorithm

Description

The cascade object detector uses the Viola-Jones algorithm to detect people's faces, noses, eyes, mouth, or upper body. You can also use the Image Labeler to train a custom classifier to use with this System object. For details on how the function works, see "Get Started with Cascade Object Detector".

To detect facial features or upper body in an image:

1. Create the `vision.CascadeObjectDetector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
detector = vision.CascadeObjectDetector
detector = vision.CascadeObjectDetector(model)
detector = vision.CascadeObjectDetector(XMLFILE)
detector = vision.CascadeObjectDetector(Name,Value)
```

Description

`detector = vision.CascadeObjectDetector` creates a detector to detect objects using the Viola-Jones algorithm.

`detector = vision.CascadeObjectDetector(model)` creates a detector configured to detect objects defined by the input character vector, `model`.

`detector = vision.CascadeObjectDetector(XMLFILE)` creates a detector and configures it to use the custom classification model specified with the `XMLFILE` input.

`detector = vision.CascadeObjectDetector(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `detector = vision.CascadeObjectDetector('ClassificationModel','UpperBody')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ClassificationModel` — Trained cascade classification model

`'FrontalFaceCART'` (default) | `character string`

Trained cascade classification model, specified as a character vector. The `ClassificationModel` property controls the type of object to detect. By default, the detector is configured to detect faces.

You can set this character vector to an XML file containing a custom classification model, or to one of the valid model character vectors listed below. You can train a custom classification model using the `trainCascadeObjectDetector` function. The function can train the model using Haar-like features, histograms of oriented gradients (HOG), or local binary patterns (LBP). For details on how to use the function, see “Get Started with Cascade Object Detector”.

Classification Model	Image Size Used to Train Model	Model Description
`'FrontalFaceCART'`(Default)	[20 20]	Detects faces that are upright and forward facing. This model is composed of weak classifiers, based on the classification and regression tree analysis (CART). These classifiers use Haar features to encode facial features. CART-based classifiers provide the ability to model higher-order dependencies between facial features. [1]
`'FrontalFaceLBP'`	[24 24]	Detects faces that are upright and forward facing. This model is composed of weak classifiers, based on a decision stump. These classifiers use local binary patterns (LBP) to encode facial features. LBP features can provide robustness against variation in illumination. [2]
`'UpperBody'`	[18 22]	Detects the upper-body region, which is defined as the head and shoulders area. This model uses Haar features to encode the details of the head and shoulder region. Because it uses more features around the head, this model is more robust against pose changes, e.g. head rotations/tilts. [3]
`'EyePairBig'` `'EyePairSmall'`	[11 45] [5 22]	Detects a pair of eyes. The `'EyePairSmall'` model is trained using a smaller image. This enables the model to detect smaller eyes than the `'EyePairBig'` model can detect.[4]
`'LeftEye'` `'RightEye'`	[12 18]	Detects the left and right eye separately. These models are composed of weak classifiers, based on a decision stump. These classifiers use Haar features to encode details.[4]
`'LeftEyeCART'` `'RightEyeCART'`	[20 20]	Detects the left and right eye separately. The weak classifiers that make up these models are CART-trees. Compared to decision stumps, CART-tree-based classifiers are better able to model higher-order dependencies. [5]

Classification Model	Image Size Used to Train Model	Model Description
`'ProfileFace'`	[20 20]	Detects upright face profiles. This model is composed of weak classifiers, based on a decision stump. These classifiers use Haar features to encode face details.
`'Mouth'`	[15 25]	Detects the mouth. This model is composed of weak classifiers, based on a decision stump, which use Haar features to encode mouth details.[4]
`'Nose'`	[15 18]	This model is composed of weak classifiers, based on a decision stump, which use Haar features to encode nose details.[4]

`MinSize` — Size of smallest detectable object

`[ ]` (default) | two-element vector

Size of smallest detectable object, specified as a two-element vector [*height width*]. Set this property in pixels for the minimum size region containing an object. The value must be greater than or equal to the image size used to train the model. Use this property to reduce computation time when you know the minimum object size prior to processing the image. When you do not specify a value for this property, the detector sets it to the size of the image used to train the classification model.

For details explaining the relationship between setting the size of the detectable object and the `ScaleFactor` property, see “Algorithms” on page 2-885 section.

Tunable: Yes

`MaxSize` — Size of largest detectable object

`[ ]` (default) | two-element vector

Size of largest detectable object, specified as a two-element vector [*height width*]. Specify the size in pixels of the largest object to detect. Use this property to reduce computation time when you know the maximum object size prior to processing the image. When you do not specify a value for this property, the detector sets it to `size(I)`.

For details explaining the relationship between setting the size of the detectable object and the `ScaleFactor` property, see the “Algorithms” on page 2-885 section.

`ScaleFactor` — Scaling for multiscale object detection

`1.1` (default) | scalar

Scaling for multiscale object detection, specified as a value greater than `1.0001`. The scale factor incrementally scales the detection resolution between `MinSize` and `MaxSize`. You can set the scale factor to an ideal value using:
`size(I)/(size(I)-0.5)`

The detector scales the search region at increments between `MinSize` and `MaxSize` using the following relationship:

search region = `round`((*Training Size*)*(`ScaleFactor`^{N}))

N is the current increment, an integer greater than zero, and *Training Size* is the image size used to train the classification model.

Tunable: Yes

`MergeThreshold` — Detection threshold
4 (default) | integer

Detection threshold, specified as an integer. The threshold defines the criteria needed to declare a final detection in an area where there are multiple detections around an object. Groups of colocated detections that meet the threshold are merged to produce one bounding box around the target object. Increasing this threshold may help suppress false detections by requiring that the target object be detected multiple times during the multiscale detection phase. When you set this property to `0`, all detections are returned without performing thresholding or merging operation. This property is tunable.

`UseROI` — Use region of interest
`false` (default) | `true`

Use region of interest, specified as `false` or `true`. Set this property to `true` to detect objects within a rectangular region of interest within the input image.

Usage

Syntax

```
bbox = detector(I)
bbox = detector(I,roi)
detectionResults = detector(ds)
```

Description

`bbox = detector(I)` returns an *M*-by-4 matrix, `bbox`, that defines *M* bounding boxes containing the detected objects. The detector performs multiscale object detection on the input image, `I`.

`bbox = detector(I,roi)` detects objects within the rectangular search region specified by `roi`. Set the `'UseROI'` property to `true` to use this syntax.`I` is a grayscale or truecolor (RGB) image.

`detectionResults = detector(ds)` detects objects within all the images returned by the `read` function of the input datastore.

Input Arguments

`I` — Input image
grayscale | truecolor (RGB)

Input image, specified as grayscale or truecolor (RGB).

`ds` — Datastore
`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be grayscale or RGB. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns. Therefore, datastore `read` function must return image data in the first column.

`model` — Classification model
`'FrontalFaceCART'` (default) | `character string`

Classification model, specified as a character vector. The `model` input describes the type of object to detect. There are several valid `model` character vectors, such as '`FrontalFaceCART`', '`UpperBody`', and '`ProfileFace`'. See the `ClassificationModel` property description for a full list of available models.

`XMLFILE` — Custom classification model
XML file

Custom classification model, specified as an XML file. The `XMLFILE` can be created using the `trainCascadeObjectDetector` function or OpenCV (Open Source Computer Vision) training functionality. You must specify a full or relative path to the `XMLFILE`, if it is not on the MATLAB path.

`roi` — Rectangular region of interest
four-element vector (default)

Rectangular region of interest within image `I`, specified as a four-element vector, [*x y width height*].

Output Arguments

`bbox` — Detections
M-by-4 matrix (default)

Detections, returned as an *M*-by-4 element matrix. Each row of the output matrix contains a four-element vector, [*x y width height*], that specifies in pixels, the upper-left corner and size of a bounding box.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column contains *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Detect Faces in an Image Using the Frontal Face Classification Model

Create a face detector object.

```
faceDetector = vision.CascadeObjectDetector;
```

Read the input image.

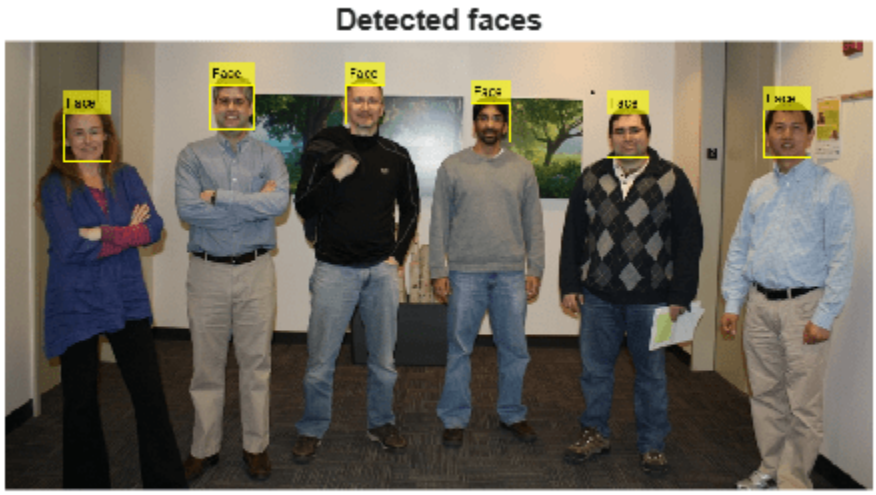
```
I = imread('visionteam.jpg');
```

Detect faces.

```
bboxes = faceDetector(I);
```

Annotate detected faces.

```
IFaces = insertObjectAnnotation(I,'rectangle',bboxes,'Face');
figure
imshow(IFaces)
title('Detected faces');
```



Detect Upper Body in Image Using Upper Body Classification Model

Create a body detector object and set properties.

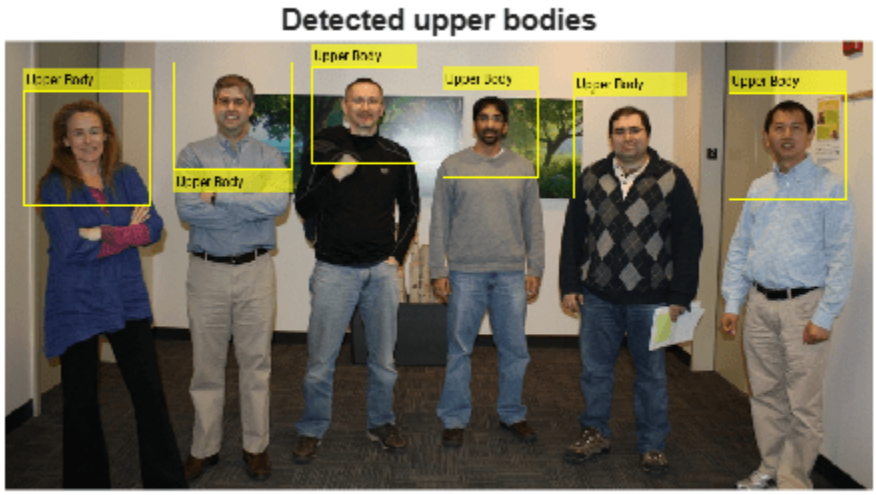
```
bodyDetector = vision.CascadeObjectDetector('UpperBody');
bodyDetector.MinSize = [60 60];
bodyDetector.MergeThreshold = 10;
```

Read input image and detect upper body.

```
I2 = imread('visionteam.jpg');
bboxBody = bodyDetector(I2);
```

Annotate detected upper bodies.

```
IBody = insertObjectAnnotation(I2,'rectangle',bboxBody,'Upper Body');
figure
imshow(IBody)
title('Detected upper bodies');
```



Algorithms

Classification Model Training

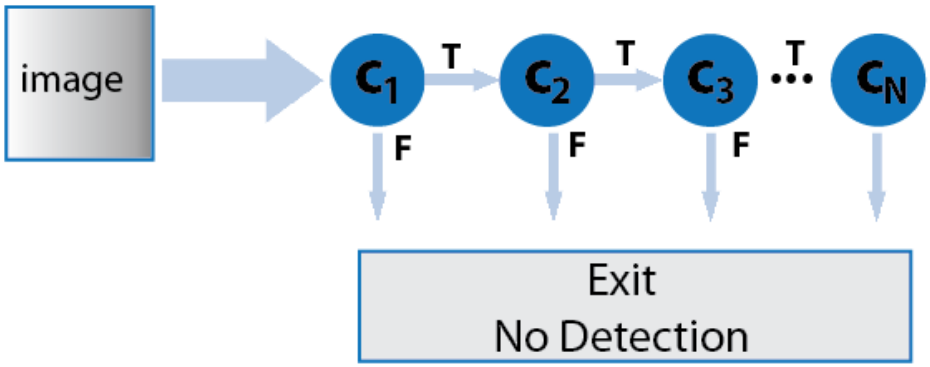
Each model is trained to detect a specific type of object. The classification models are trained by extracting features from a set of known images. These extracted features are then fed into a learning algorithm to train the classification model. Computer Vision Toolbox software uses the Viola-Jones cascade object detector. This detector uses HOG [7], LBP [8], and Haar-like [6] features and a cascade of classifiers trained using boosting.

The image size used to train the classifiers defines the smallest region containing the object. Training image sizes vary according to the application, type of target object, and available positive images. You must set the `MinSize` property to a value greater than or equal to the image size used to train the model.

Cascade of Classifiers

This object uses a cascade of classifiers to efficiently process image regions for the presence of a target object. Each stage in the cascade applies increasingly more complex binary classifiers, which allows the algorithm to rapidly reject regions that do not contain the target. If the desired object is not found at any stage in the cascade, the detector immediately rejects the region and processing is

terminated. By terminating, the object avoids invoking computation-intensive classifiers further down the cascade.



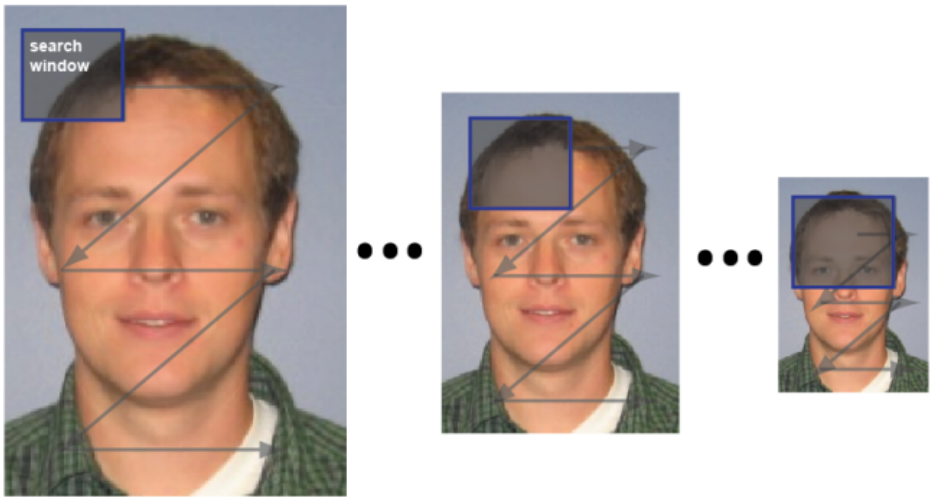
Multiscale Object Detection

The detector incrementally scales the input image to locate target objects. At each scale increment, a sliding window, whose size is the same as the training image size, scans the scaled image to locate objects. The `ScaleFactor` property determines the amount of scaling between successive increments.

The search region size is related to the `ScaleFactor` in the following way:

search region = `round`((*ObjectTrainingSize*)*(`ScaleFactor`^{N}))

N is the current increment, an integer greater than zero, and *ObjectTrainingSize* is the image size used to train the classification model.



The search window traverses the image for each scaled increment.

Relationship Between MinSize, MaxSize, and ScaleFactor

Understanding the relationship between the size of the object to detect and the scale factor will help you set the properties accordingly. The `MinSize` and `MaxSize` properties limit the size range of the

object to detect. Ideally, these properties are modified to reduce computation time when you know the approximate object size prior to processing the image. They are not designed to provide precise filtering of results, based on object size. The behavior of these properties is affected by the ScaleFactor. The scale factor determines the quantization of the search window sizes.

search region = `round`((*Training Size*)*(`ScaleFactor`^{N}))

The actual range of returned object sizes may not be exactly what you select for the `MinSize` and `MaxSize` properties. For example,
For a `ScaleFactor` value of `1.1` with a 24x24 training size, for `5` increments, the search region calculation would be:

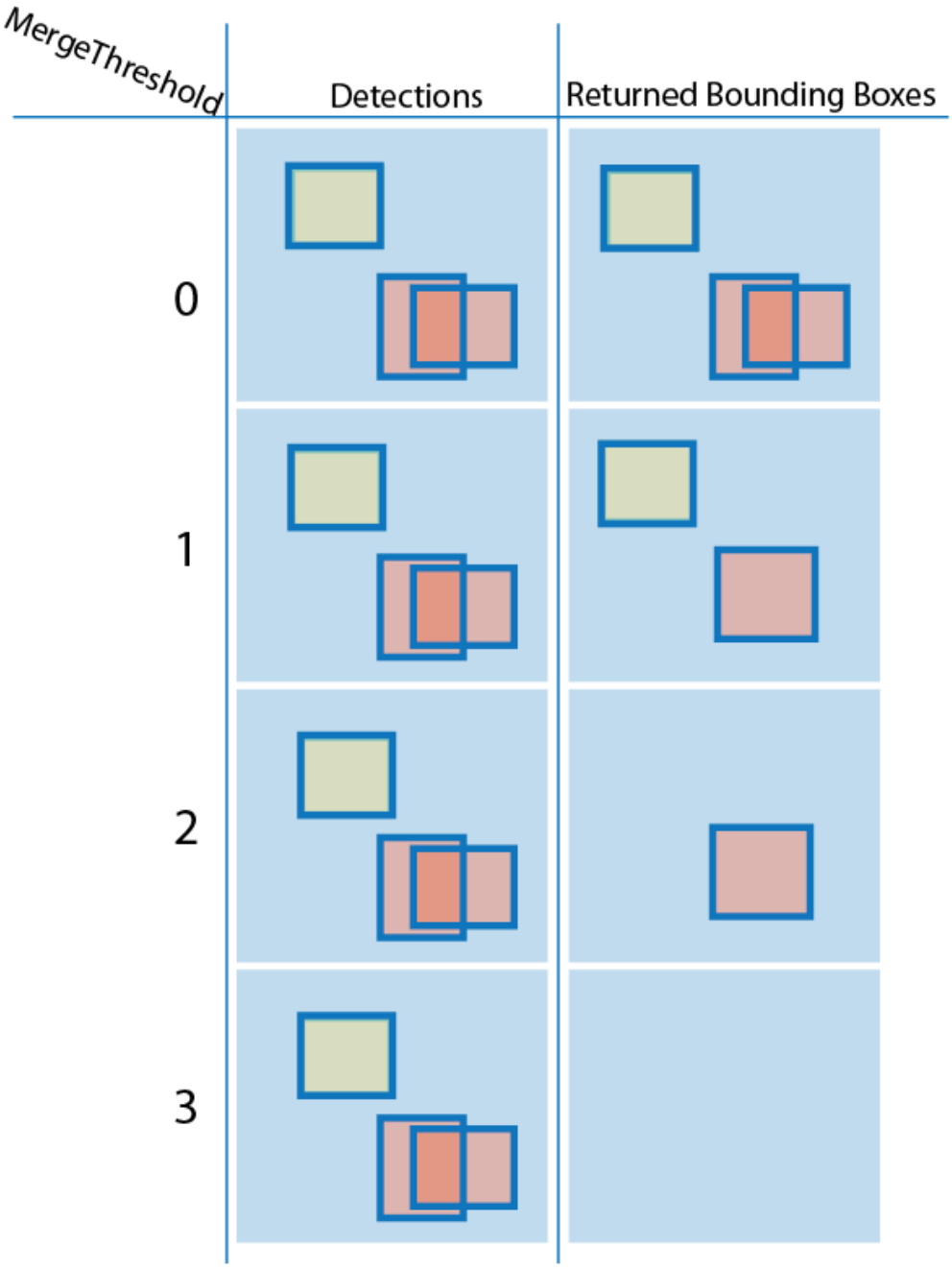
```
>> search region = round(24*1.1.^(1:5))

>> 26 29 32 35 39
```

If you were to set `MaxSize` to `34`, due to the search region quantization, the actual maximum object size used by the algorithm would be `32`.

Merge Detection Threshold

For each increment in scale, the search window traverses over the image producing multiple detections around the target object. The multiple detections are merged into one bounding box per target object. You can use the `MergeThreshold` property to control the number of detections required before combining or rejecting the detections. The size of the final bounding box is an average of the sizes of the bounding boxes for the individual detections and lies between `MinSize` and `MaxSize`.



Version History

Introduced in R2012a

References

[1] Lienhart R., Kuranov A., and V. Pisarevsky "Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection." *Proceedings of the 25th DAGM Symposium on Pattern Recognition.* Magdeburg, Germany, 2003.

[2] Ojala Timo, Pietikäinen Matti, and Mäenpää Topi, "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns". In *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002*. Volume 24, Issue 7, pp. 971-987.

[3] Kruppa H., Castrillon-Santana M., and B. Schiele. "Fast and Robust Face Finding via Local Context". *Proceedings of the Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, 2003, pp. 157–164.

[4] Castrillón Marco, Déniz Oscar, Guerra Cayetano, and Hernández Mario, "ENCARA2: Real-time detection of multiple faces at different resolutions in video streams". In *Journal of Visual Communication and Image Representation*, 2007 (18) 2: pp. 130-140.

[5] Yu Shiqi "Eye Detection." Shiqi Yu’s Homepage. http://yushiqi.cn/research/eyedetection.

[6] Viola, Paul and Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features" , *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. Volume: 1, pp.511–518.

[7] Dalal, N., and B. Triggs, "Histograms of Oriented Gradients for Human Detection". *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Volume 1, (2005), pp. 886–893.

[8] Ojala, T., M. Pietikainen, and T. Maenpaa, "Multiresolution Gray-scale and Rotation Invariant Texture Classification With Local Binary Patterns". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Volume 24, No. 7 July 2002, pp. 971–987.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV (Version 3.4.0) libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).
- `XMLFILE` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Image Labeler | `trainCascadeObjectDetector` | `insertShape` | `vision.PeopleDetector` | `integralImage` | `faceDetector`

Topics

“Face Detection and Tracking Using CAMShift”
“Face Detection and Tracking Using the KLT Algorithm”
“Face Detection and Tracking Using Live Video Acquisition”

“Code Generation for Face Tracking with PackNGo”
“Detect and Track Face”
“Get Started with the Image Labeler”
“Get Started with Cascade Object Detector”
“Multiple Object Tracking”

External Websites
Detect and Track Multiple Faces in a Live Video Stream

vision.Deinterlacer

Namespace: `vision`

Remove motion artifacts by deinterlacing input video signal

Description

To remove motion artifacts by deinterlacing input video signal.

To track a set of points:

1. Create the `vision.Deinterlacer` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
Hdeinterlacer = vision.Deinterlacer
Hdeinterlacer = vision.Deinterlacer(Name,Value)
```

Description

`Hdeinterlacer = vision.Deinterlacer` returns a deinterlacing System object, `Hdeinterlacer`, that removes motion artifacts from images composed of weaved top and bottom fields of an interlaced signal.

`Hdeinterlacer = vision.Deinterlacer(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `varObj = vision.Deinterlacer('Method','Line repetition')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Method` — Method used to deinterlace input video

`'Line repetition'` (default) | `'Linear interpolation'` | `'Vertical temporal median filtering'`

Method used to deinterlace input video, specified as `'Line repetition'`, `'Linear interpolation'`, `'Vertical temporal median filtering'`.

`TransposedInput` — Indicate if input data is in row-major order
`false` (default) | `true`

Indicate if input data is in row-major order, specified as `true` or `false`. Set this property to `true` if the input buffer contains data elements from the first row first, then the second row second, and so on.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
value = Hdeinterlacer(input)
```

Description

`value = Hdeinterlacer(input)` deinterlaces the input according to the algorithm set in the Method property

Input Arguments

`input` — Input data
top and bottom fields of interlaced video

Input data, specified as a combination of top and bottom fields of interlaced video.

Output Arguments

`value` — Frames of deinterlaced video
top and bottom fields of interlaced video

Frames of deinterlaced video, returned as the same data type as the input

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Remove Motion Artifacts from Image

Create a deinterlacer object.

```
hdinterlacer = vision.Deinterlacer;
```

Read an image with motion artifacts.

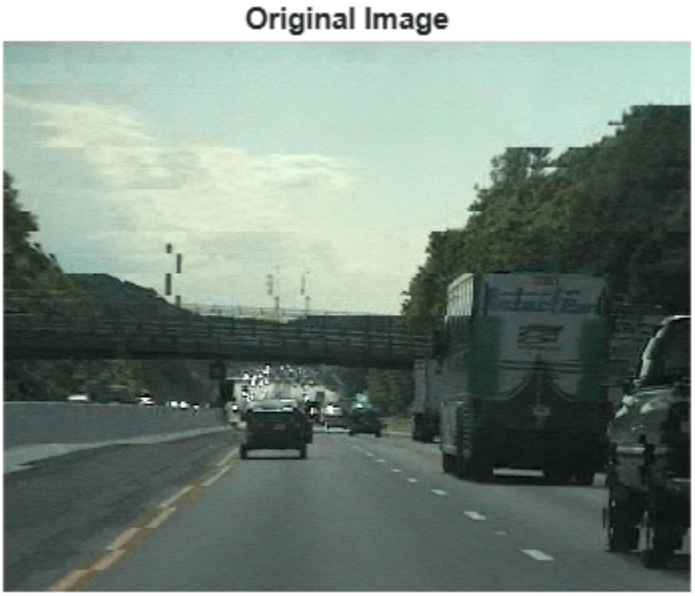
```
I = imread('vipinterlace.png');
```

Apply the deinterlacer to the image.

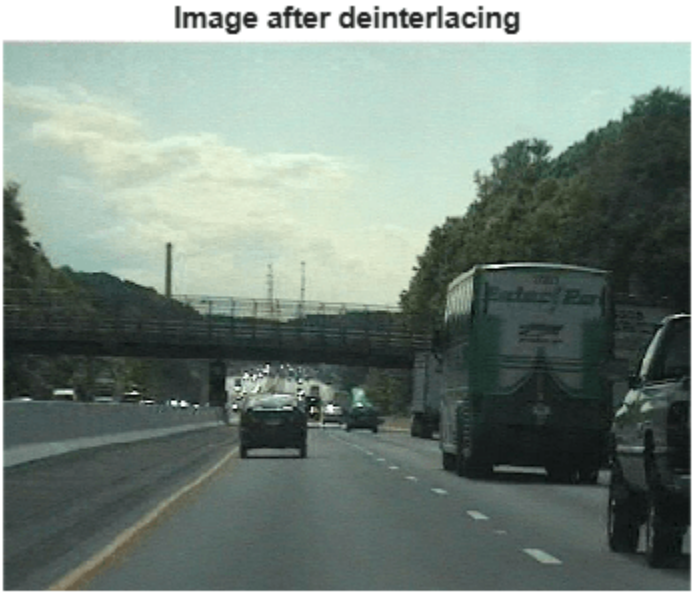
```
clearimage = hdinterlacer(I);
```

Display the results.

```
imshow(I);
title('Original Image');
```




```
figure, imshow(clearimage);
title('Image after deinterlacing');
```



Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Deinterlacing

vision.ForegroundDetector

Namespace: `vision`

Foreground detection using Gaussian mixture models

Description

The `ForegroundDetector` compares a color or grayscale video frame to a background model to determine whether individual pixels are part of the background or the foreground. It then computes a foreground mask. By using background subtraction, you can detect foreground objects in an image taken from a stationary camera.

To detect foreground in an image :

1. Create the `vision.ForegroundDetector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
detector = vision.ForegroundDetector
detector = vision.ForegroundDetector(Name,Value)
```

Description

`detector = vision.ForegroundDetector` computes and returns a foreground mask using the Gaussian mixture model (GMM).

`detector = vision.ForegroundDetector(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `detector = vision.ForegroundDetector('LearningRate',0.005)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`AdaptLearningRate` — Adapt learning rate
`'true'` (default) | `'false'`

Adapt learning rate, specified as `'true'` or `'false'`. This property enables the object to adapt the learning rate during the period specified by the `NumTrainingFrames` property. When you set this property to `true`, the object sets the `LearningRate` property to 1/(current frame number). When you set this property to `false`, the `LearningRate` property must be set at each time step.

`NumTrainingFrames` — Number of initial video frames for training background model
`150` (default) | integer

Number of initial video frames for training background model, specified as an integer. When you set the `AdaptLearningRate` to `false`, this property will not be available.

`LearningRate` — Learning rate for parameter updates
`0.005` (default) | numeric scalar

Learning rate for parameter updates, specified as a numeric scalar. Specify the learning rate to adapt model parameters. This property controls how quickly the model adapts to changing conditions. Set this property appropriately to ensure algorithm stability.

The learning rate specified by this property can only be implemented when you set the `AdaptLearningRate` to `true` and after the training period specified by `NumTrainingFrames` is over.

Tunable: Yes

`MinimumBackgroundRatio` — Threshold to determine background model
`0.7` (default) | numeric scalar

Threshold to determine background model, specified as a numeric scalar. Set this property to represent the minimum possibility for pixels to be considered background values. Multimodal backgrounds cannot be handled if this value is too small.

`NumGaussians` — Number of Gaussian modes in the mixture model
`5` (default) | positive integer

Number of Gaussian modes in the mixture model, specified as a positive integer. Typically, you would set this value to 3, 4 or 5. Set the value to 3 or greater to be able to model multiple background modes.

`InitialVariance` — Initial mixture model variance
`'Auto'` (default) | numeric scalar

Initial mixture model variance, specified as a numeric scalar or the `'Auto'` character vector.

Image Data Type	Initial Variance
`double/single`	(30/255)^2
`uint8`	30^2

This property applies to all color channels for color inputs.

Usage

Syntax

```
foregroundMask = detector(I)
foregroundMask = detector(I,learningRate)
```

Description

`foregroundMask = detector(I)` computes the foreground mask for input image `I`, and returns a logical mask. Values of `1` in the mask correspond to foreground pixels.

`foregroundMask = detector(I,learningRate)` computes the foreground mask using the `LearningRate`.

Input Arguments

`I` — Input image
grayscale | truecolor (RGB)

Input image, specified as grayscale or truecolor (RGB).

`learningRate` — Learning rate for parameter updates
`0.005` (default) | numeric scalar

Learning rate for parameter updates, specified as a numeric scalar. Specify the learning rate to adapt model parameters. This property controls how quickly the model adapts to changing conditions. Set this property appropriately to ensure algorithm stability.

The learning rate specified by this property can only be implemented when you set the `AdaptLearningRate` to `true` and after the training period specified by `NumTrainingFrames` is over.

Tunable: Yes

Output Arguments

`foregroundMask` — Foreground mask
binary mask

Foreground mask computed using a Gaussian mixture model, returned as a binary mask.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Detect Moving Cars In Video

Create a video source object to read file.

```
videoSource = VideoReader("viptraffic.avi");
```

Create a detector object and set the number of training frames to 5 (because it is a short video.) Set initial standard deviation.

```
detector = vision.ForegroundDetector(...
       NumTrainingFrames=5, ...
       InitialVariance=30*30);
```

Perform blob analysis.

```
blob = vision.BlobAnalysis(...
       CentroidOutputPort=false,...
       AreaOutputPort=false, ...
       BoundingBoxOutputPort=true, ...
       MinimumBlobAreaSource="Property",...
       MinimumBlobArea=250);
```

Play results. Draw bounding boxes around cars.

```
videoPlayer = vision.VideoPlayer();
while hasFrame(videoSource)
     frame  = readFrame(videoSource);
     fgMask = detector(frame);
     bbox   = blob(fgMask);
     out    = insertShape(frame,"rectangle",bbox,Color="green");
     videoPlayer(out);
     pause(0.1);
end
```



Release objects.

```
release(videoPlayer);
```



Version History

Introduced in R2011a

References

[1] Kaewtrakulpong, P. and R. Bowden. *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection*. In Proc. 2nd European Workshop on Advanced

Video Based Surveillance Systems, AVBS01, VIDEO BASED SURVEILLANCE SYSTEMS: Computer Vision and Distributed Processing (September 2001)

[2] Stauffer, C. and W.E.L. Grimson. *Adaptive Background Mixture Models for Real-Time Tracking*, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, Vol. 2 (06 August 1999), pp. 2246-252 Vol. 2.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates platform-dependent library for MATLAB host target.
- Generates portable C code for non MATLAB host target.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Topics

“Multiple Object Tracking”

vision.GammaCorrector

Namespace: `vision`

(To be removed) Apply or remove gamma correction from images or video streams

Note `vision.GammaCorrector` function will be removed in a future release. Use the `imwarp` function to apply a linear geometric transformation to an image, instead.

Description

To apply gamma correction to input images or a video stream:

1. Create the `vision.GammaCorrector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
gammaCorr = vision.GammaCorrector
gammaCorr = vision.GammaCorrector(Name,Value)
gammaCorr = vision.GammaCorrector(gamma,Name,Value)
```

Description

`gammaCorr = vision.GammaCorrector` returns a gamma corrector object, `gammaCorr`. You can use the gamma corrector to apply or remove gamma correction from images or video streams.

`gammaCorr = vision.GammaCorrector(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `gammaCorr = vision.GammaCorrector('Correction','Gamma')`

`gammaCorr = vision.GammaCorrector(gamma,Name,Value)` additionally sets the `Gamma` property.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Correction` — Specify gamma correction or linearization
`'Gamma'` (default) | `'De-gamma'`

Specify gamma correction or linearization, specified as `'Gamma'` or `'De-gamma'`.

`Gamma` — Gamma value of output or input
`2.2` (default) | numeric scalar

Gamma value of output or input, specified as numeric scalar greater than or equal to `1`. When you set the `Correction` property to `Gamma`, this property gives the desired gamma value of the output video stream. When you set the `Correction` property to `De-gamma`, this property indicates the gamma value of the input video stream.

`LinearSegment` — Enable gamma curve to have linear portion near origin
`true` (default) | `false`

Enable gamma curve to have linear portion near origin, specified as `true` or `false`.

`BreakPoint` — I-axis value of the end of gamma correction linear segment
`0.018` (default) | numeric scalar

I-axis value of the end of gamma correction linear segment, specified as a numeric scalar value in the range (0,1). This property applies when you set the `LinearSegment` property to `true`.

Usage

Syntax

```
y = gammaCorr(input)
```

Description

`y = gammaCorr(input)` applies or removes gamma correction from input `I`, and returns the gamma corrected or linearized output `y`.

Input Arguments

`input` — Input
M-by-*N* matrix of intensity values | *M*-by-*N*-*P* color video

Input, specified as an *M*-by-*N* matrix of intensity values or *M*-by-*N*-*P* color video, where *P* is the number of color planes.

Output Arguments

`y` — Corrected input
same as input (default)

Output, returned as an *M*-by-*N* matrix of intensity values or *M*-by-*N*-*P* color video, where *P* is the number of color planes.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Improve Image Contrast Using Gamma Correction

Create a gamma corrector object with the gamma property set to `2.0`.

```
hgamma = vision.GammaCorrector(2.0,'Correction','De-gamma');
```

Read an image.

```
img = imread('pears.png');
```

Apply gamma correction.

```
imgCor = hgamma(img);
```

Show the original and corrected images.

```
imshow(img); title('Original Image');
```




```
figure,
imshow(imgCor);
title('Enhanced Image after De-gamma Correction');
```



Version History

Introduced in R2012a

R2024a: `vision.GammaCorrector` function will be removed

Not recommended starting in R2024a

`vision.GammaCorrector` function will be removed in a future release. Use the `imwarp` function to apply a linear geometric transformation to an image, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`gamma` | `imwarp`

vision.HistogramBasedTracker

Namespace: `vision`

Histogram-based object tracking

Description

The histogram-based tracker incorporates the continuously adaptive mean shift (CAMShift) algorithm for object tracking. It uses the histogram of pixel values to identify the tracked object.

To track an object:

1. Create the `vision.HistogramBasedTracker` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
hbtracker = vision.HistogramBasedTracker
hbtracker = vision.HistogramBasedTracker(Name,Value)
```

Description

`hbtracker = vision.HistogramBasedTracker` returns a tracker that tracks an object by using the CAMShift algorithm. It uses the histogram of pixel values to identify the tracked object. To initialize the tracking process, you must use the `initializeObject` function to specify an exemplar image of the object.

`hbtracker = vision.HistogramBasedTracker(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `hbtracker = vision.HistogramBasedTracker('ObjectHistogram',[])`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ObjectHistogram` — Normalized pixel value histogram
`[ ]` (default) | *N*-element vector.

Normalized pixel value histogram, specified as an *N*-element vector. This vector specifies the normalized histogram of the object's pixel values. Histogram values must be normalized to a value between `0` and `1`. You can use the `initializeObject` method to set the property.

Tunable: Yes

Usage

Syntax

```
bbox = hbtracker(I)
[bbox,orientation] = hbtracker(I)
[bbox,orientation,score] = hbtracker(I)
```

Description

`bbox = hbtracker(I)` returns a bounding box, of the tracked object. Before using the tracker, you must identify the object to track, and set the initial search window. Use the `initializeObject` function to do this.

`[bbox,orientation] = hbtracker(I)` additionally returns the angle between the *x*-axis and the major axis of the ellipse that has the same second-order moments as the object. The returned angle is between –pi/2 and pi/2.

`[bbox,orientation,score] = hbtracker(I)` additionally returns the confidence score for the returned bounding box that contains the tracked object.

Input Arguments

`I` — Video frame

grayscale | 2-D feature map

Video frame, specified as grayscale or any 2-D feature map that distinguishes the object from the background. For example, I can be a hue channel of the HSV color space.

Output Arguments

`bbox` — Bounding box

[*x y width height*]

Bounding box, returned as a four-element vector in the format, [*x y width height*].

`orientation` — Orientation

angle

Orientation, returned as an angle between –pi/2 and pi/2. The angle is measured from the *x*-axis and the major axis of the ellipse that has the same second-order moments as the object.

`score` — Score

scalar

Score, returned as a scalar in the range [0 1]. A value of `1` corresponds to the maximum confidence. `1`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.HistogramBasedTracker

initializeObject	Set object to track
initializeSearchWindow	Set initial search window

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Track a Face

Track and display a face in each frame of an input video.

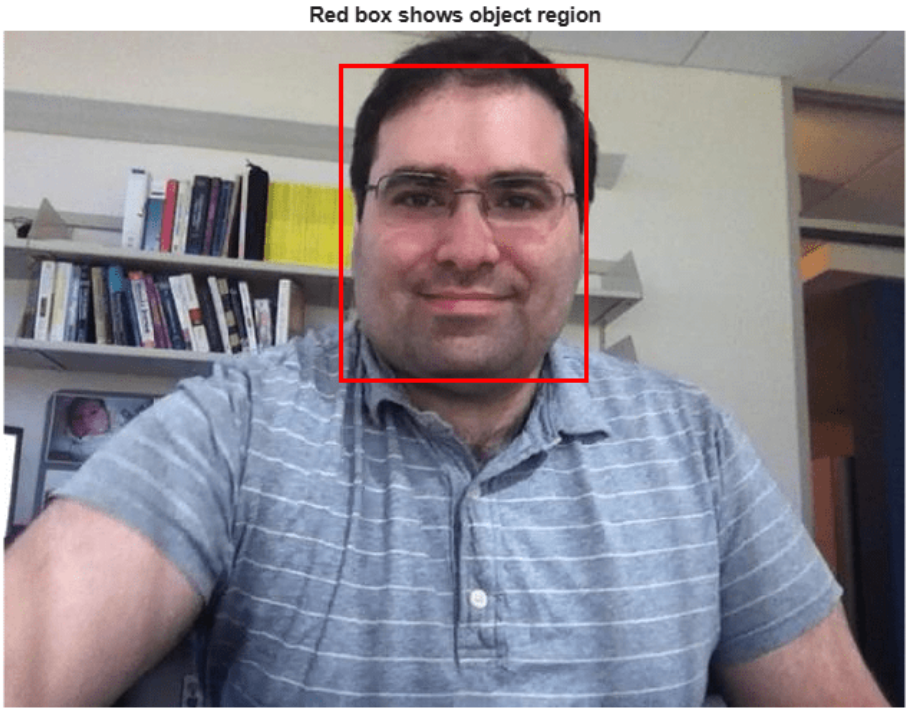
Load in the input video and create a System object displaying video.

```
videoReader = VideoReader("tilted_face.avi");
videoPlayer = vision.VideoPlayer();
```

Read the first video frame, which contains the object. Convert the image to HSV color space. Then define and display the object region.

```
objectFrame = im2single(readFrame(videoReader));
objectHSV = rgb2hsv(objectFrame);
objectRegion = [240, 25, 175, 225];
objectImage = insertShape(objectFrame,"rectangle",objectRegion,LineWidth=3,Color="red");

figure;
imshow(objectImage);
title("Red box shows object region")
```



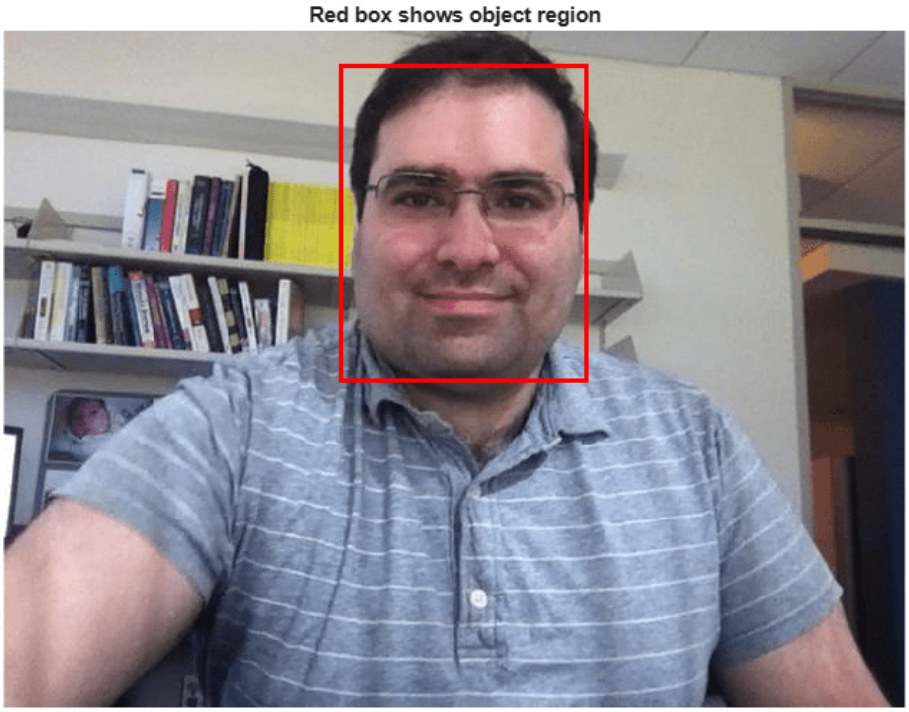
Set the object, based on the hue channel of the first video frame.

```
tracker = vision.HistogramBasedTracker;
initializeObject(tracker, objectHSV(:,:,1),objectRegion);
```

Track and display the object in each video frame. The while loop reads each image frame, converts the image to HSV color space, then tracks the object in the hue channel where it is distinct from the background. Finally, the example draws a box around the object and displays the results.

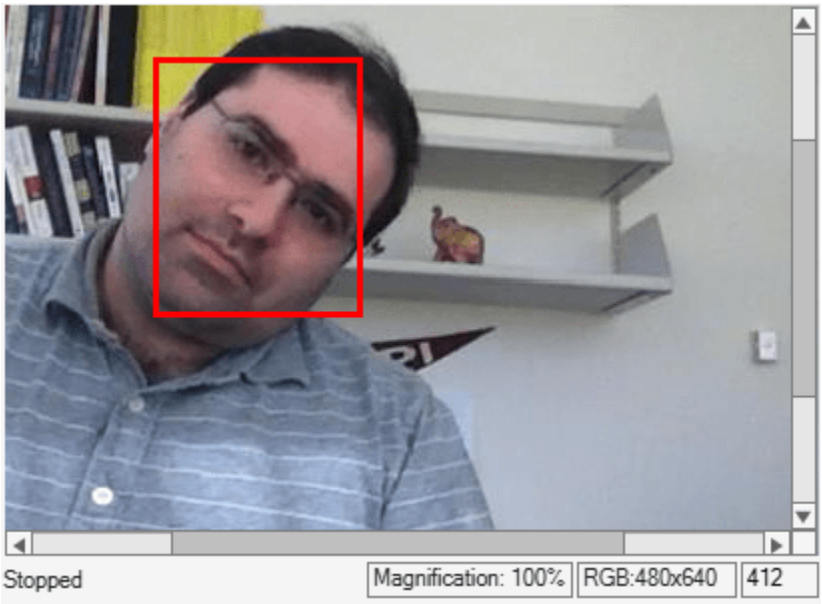
```
while hasFrame(videoReader)
  frame = im2single(readFrame(videoReader));
  hsvFrame = rgb2hsv(frame);
  bbox = tracker(hsvFrame(:,:,1));
  out = insertShape(frame,"rectangle",bbox,LineWidth=3,Color="red");
  videoPlayer(out);
end
```





Release the video player.

```
release(videoPlayer);
```



Copyright 2012-2025 The MathWorks, Inc.

Version History

Introduced in R2012a

References

[1] Bradsky, G.R. "Computer Vision Face Tracking For Use in a Perceptual User Interface." *Intel Technology Journal*. January 1998.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`size` | `imrect` | `rgb2hsv`

integralKernel

Define filter for use with integral images

Description

An `integralKernel` object describes box filters for use with integral images.

Creation

Syntax

```
intKernel = integralKernel(bbox,weights)
intKernel = integralKernel(bbox,weights,orientation)
```

Description

`intKernel = integralKernel(bbox,weights)` creates an upright box filter from bounding boxes, `bbox`, and their corresponding weights, `weights`. The bounding boxes set the "BoundingBoxes" on page 2-0 property and the weights set the "Weights" on page 2-0 property.

For example, a conventional filter with the coefficients:

```
h    = [1  1  1  1;
        1  1  1  1;
       -1 -1 -1 -1;
       -1 -1 -1 -1]
```

and two regions:
region 1: x=1, y=1, width = 4, height = 2
region 2: x=1, y=3, width = 4, height = 2
can be specified as

```
boxH = integralKernel([1 1 4 2; 1 3 4 2],[1, -1])
```

`intKernel = integralKernel(bbox,weights,orientation)` creates a box filter with an upright or rotated orientation. The specified orientation sets the "Orientation" on page 2-0 property.

Properties

BoundingBoxes — Bounding boxes

4-element vector | *M*-by-4 matrix

Bounding boxes, specified as a 4-element vector of the form [*x*,*y*,*width*, *height*] representing a single bounding box or an *M*-by-4 matrix representing *M* bounding boxes. The bounding boxes define the

filter. The (*x*,*y*) coordinates represent the top-most corner of the kernel. The (*width*, *height*) elements represent the width and height accordingly. Specifying the bounding boxes as an *M*-by-4 matrix is particularly useful for constructing Haar-like features composed of multiple rectangles.

Sums are computed over regions defined by `BoundingBoxes`. The bounding boxes can overlap. See “Define an 11-by-11 Average Filter” on page 2-917 for an example of how to specify a box filter.

`Weights` — Weights
numeric vector

Weights, specified as an *M*-element numeric vector containing a weight for each bounding box. The weights are used to define the coefficients of the filter.

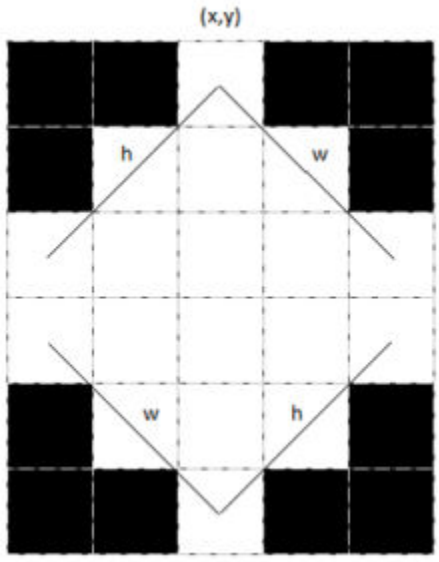
`Coefficients` — Filter coefficients
numeric

Filter coefficients, specified as a numeric value.

`Center` — Filter center
[*x*,*y*] coordinates

Filter center, specified as [*x*,*y*] coordinates. The filter center represents the center of the bounding rectangle. It is calculated by halving the dimensions of the rectangle. For even dimensional rectangles, the center is placed at subpixel locations. Hence, it is rounded up to the next integer.

For example, for this filter, the center is at [3,3].

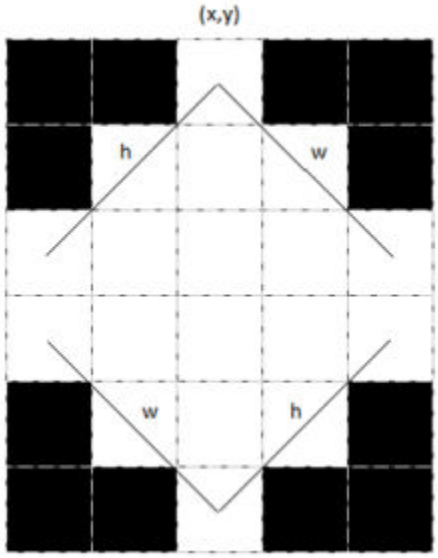


These coordinates are in the kernel space, where the top-left corner is (1,1). To place the center in a different location, provide the appropriate bounding box specification. For this filter, the best workflow would be to construct the upright kernel and then call the `rot45` method to provide the rotated version.

`Size` — Filter size
2-element vector

Filter size, specified as a 2-element vector. The size of the kernel is computed to be the dimensions of the rectangle that bounds the kernel. For a single bounding box vector [*x*,*y*,*width*, *height*], the kernel is bounded within a rectangle of dimensions [(*width*+*height*) (*width*+*height*)-1].

For cascaded rectangles, the lowest corner of the bottom-most rectangle defines the size. For example, a filter with a bounding box specification of [3 1 3 3], with weights set to 1, produces a 6-by-5 filter with this kernel:



`Orientation` — Filter orientation
`'upright'` (default) | `'rotated'`

Filter orientation, specified as `'upright'` or `'rotated'`. When you specify the orientation as `'rotated'`, the (*x*,*y*) components refer to the location of the top-left corner of the bounding box. Also, the (*width*,*height*) components refer to a 45-degree line from the top-left corner of the bounding box.

More About

Computing an Integral Image and Using it for Filtering with Box Filters

The `integralImage` function together with the `integralKernel` object and `integralFilter` function complete the workflow for box filtering based on integral images. You can use this workflow for filtering with box filters.

- Use the `integralImage` function to compute the integral images
- Use the `integralFilter` function for filtering
- Use the `integralKernel` object to define box filters

The `integralKernel` object allows you to transpose the filter. You can use this to aim a directional filter. For example, you can turn a horizontal edge detector into vertical edge detector.

Object Functions

rot45	Rotate upright kernel clockwise by 45 degrees
transpose	Transpose integral kernel

Examples

Define an 11-by-11 Average Filter

```
avgH = integralKernel([1 1 11 11], 1/11^2);
```

Define a Filter to Approximate a Gaussian Second Order Partial Derivative in Y Direction

```
ydH = integralKernel([1,1,5,9;1,4,5,3], [1, -3]);
```

You can also define this filter as `integralKernel`([1,1,5,3;1,4,5,3;1,7,5,3], [1, -2, 1]);|. This filter definition is less efficient because it requires three bounding boxes.

Visualize the filter.

```
ydH.Coefficients
```

```
ans = 9×5

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
    -2    -2    -2    -2    -2
    -2    -2    -2    -2    -2
    -2    -2    -2    -2    -2
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

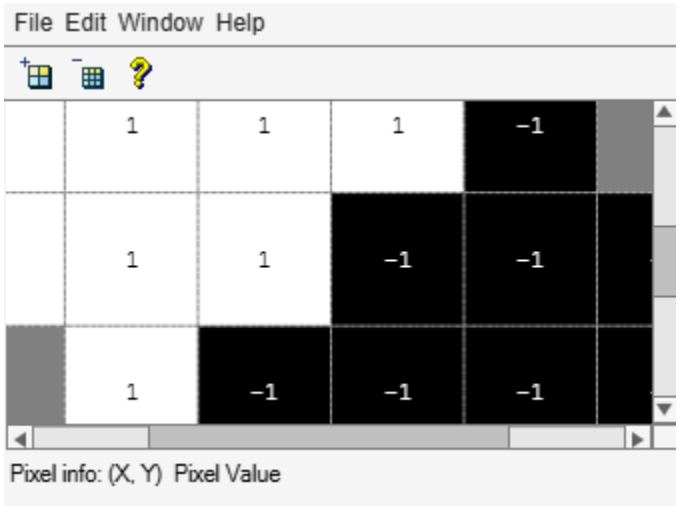
Create a Haar-like Wavelet to Detect 45-Degree Edges

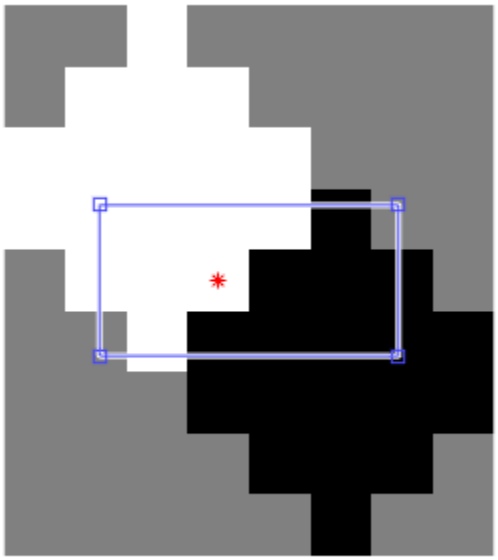
Create the filter.

```
K = integralKernel([3,1,3,3;6 4 3 3], [1 -1], "rotated");
```

Visualize the filter and mark the center.

```
imshow(K.Coefficients, []);
hold on;
plot(K.Center(2),K.Center(1), "r*");
impixelregion;
```





Blur an Image Using an Average Filter

Read and display the input image.

```
I = imread('pout.tif');
imshow(I);
```



Compute the integral image.

```
intImage = integralImage(I);
```

Apply a 7-by-7 average filter.

```
avgH = integralKernel([1 1 7 7], 1/49);
J = integralFilter(intImage, avgH);
```

Cast the result back to the same class as the input image.

```
J = uint8(J);
figure
imshow(J);
```



Version History

Introduced in R2012a

References

[1] Viola, Paul, and Michael J. Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features”. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Vol. 1, 2001, pp. 511–518.

See Also

`detectMSERFeatures` | `integralImage` | `integralFilter` | `detectSURFFeatures` | `SURFPoints`

transpose

Transpose integral kernel

Syntax

```
transposedKernel = transpose(intKernel)
```

Description

`transposedKernel = transpose(intKernel)` transposes the integral kernel. You can use this operation to change the direction of an oriented filter.

Examples

Construct Haar-like Wavelet Filters

Horizontal filter

```
horiH = integralKernel([1 1 4 3; 1 4 4 3], [-1, 1]);
```

Using the dot and apostrophe create a vertical filter.

```
vertH = horiH.';
```

Using the transpose method.

```
verticalH = transpose(horiH);
```

Input Arguments

`intKernel` — Integral kernel

`integralKernel` object

Integral kernel, specified as an `integralKernel` object.

Output Arguments

`transposedKernel` — Transposed integral kernel

`integralKernel` object

Transposed integral kernel, returned as an `integralKernel` object.

Version History

Introduced in R2012a

See Also

`rot45`

rot45

Rotate upright kernel clockwise by 45 degrees

Syntax

```
rotKernel = rot45(intKernel)
```

Description

`rotKernel = rot45(intKernel)` rotates upright kernel `intKernel` clockwise by 45 degrees.

Examples

Construct and Rotate a Haar-like Wavelet Filter

Create a horizontal filter.

```
H = integralKernel([1 1 4 3; 1 4 4 3], [-1, 1]);
```

Rotate the filter 45 degrees.

```
rotH = rot45(H);
```

Input Arguments

`intKernel` — Integral kernel

`integralKernel` object

Integral kernel, specified as an `integralKernel` object.

Output Arguments

`rotKernel` — Rotated integral kernel

`integralKernel` object

Rotated integral kernel, returned as an `integralKernel` object.

Version History

Introduced in R2012a

See Also

`transpose`

vision.KalmanFilter

Correction of measurement, state, and state estimation error covariance

Description

The Kalman filter object is designed for tracking. You can use it to predict a physical object's future location, to reduce noise in the detected location, or to help associate multiple physical objects with their corresponding tracks. A Kalman filter object can be configured for each physical object for multiple object tracking. To use the Kalman filter, the object must be moving at constant velocity or constant acceleration.

Creation

The Kalman filter algorithm involves two steps, prediction and correction (also known as the update step). The first step uses previous states to predict the current state. The second step uses the current measurement, such as object location, to correct the state. The Kalman filter implements a discrete time, linear State-Space System.

Note To make configuring a Kalman filter easier, you can use the `configureKalmanFilter` object to configure a Kalman filter. It sets up the filter for tracking a physical object in a Cartesian coordinate system, moving with constant velocity or constant acceleration. The statistics are the same along all dimensions. If you need to configure a Kalman filter with different assumptions, do not use the function, use this object directly.

In the state space system, the state transition model, A, and the measurement model, H, are set as follows:

Variable	Value
A	`[1 1 0 0; 0 1 0 0; 0 0 1 1; 0 0 0 1]`
H	`[1 0 0 0; 0 0 1 0]`

Syntax

```
kalmanFilter = vision.KalmanFilter
kalmanFilter = vision.KalmanFilter(StateTransitionModel,MeasurementModel)
kalmanFilter = vision.KalmanFilter(StateTransitionModel,MeasurementModel,
ControlModel,Name,Value)
```

Description

`kalmanFilter = vision.KalmanFilter` returns a kalman filter for a discrete time, constant velocity system.

`kalmanFilter = vision.KalmanFilter(StateTransitionModel,MeasurementModel)` additionally configures the control model, B.

`kalmanFilter = vision.KalmanFilter(StateTransitionModel,MeasurementModel, ControlModel,Name,Value)` configures the Kalman filter object properties, specified as one or more `Name,Value` pair arguments. Unspecified properties have default values.

Properties

`StateTransitionModel` — Model describing state transition between time steps (*A*)

`[1 1 0 0; 0 1 0 0; 0 0 1 1; 0 0 0 1]` (default) | *M*-by-*M* matrix

Model describing state transition between time steps (*A*), specified as an *M*-by-*M* matrix. After the object is constructed, this property cannot be changed. This property relates to the *A* variable in the state-space model.

`MeasurementModel` — Model describing state to measurement transformation (*H*)

`[1 0 0 0; 0 0 1 0]` (default) | *N*-by-*M* matrix

Model describing state to measurement transformation (*H*) , specified as an *N*-by-*M* matrix. After the object is constructed, this property cannot be changed. This property relates to the *H* variable in the state-space model.

`ControlModel` — Model describing control input to state transformation (*B*)

`[]` (default) | *M*-by-*L* matrix

Model describing control input to state transformation (*B*) , specified as an *M*-by-*L* matrix. After the object is constructed, this property cannot be changed. This property relates to the *B* variable in the state-space model.

`State` — State (*x*)

`[0]` (default) | scalar | *M*-element vector.

State (*x*), specified as a scalar or an *M*-element vector. If you specify `State` as a scalar, it will be extended to an *M*-element vector. This property relates to the *x* variable in the state-space model.

`StateCovariance` — State estimation error covariance (*P*)

`[1]` (default) | scalar | *M*-by-*M* matrix

State estimation error covariance (*P*), specified as a scalar or an *M*-by-*M* matrix. If you specify `StateCovariance` as a scalar it will be extended to an *M*-by-*M* diagonal matrix. This property relates to the *P* variable in the state-space system.

`ProcessNoise` — Process noise covariance (*Q*)

`[1]` (default) | scalar | *M*-by-*M* matrix

Process noise covariance (*Q*) , specified as a scalar or an *M*-by-*M* matrix. If you specify `ProcessNoise` as a scalar it will be extended to an *M*-by-*M* diagonal matrix. This property relates to the *Q* variable in the state-space model.

`MeasurementNoise` — Measurement noise covariance (*R*)

`[1]` (default) | scalar | *N*-by-*N* matrix

Measurement noise covariance (*R*) , specified as a scalar or an *N*-by-*N* matrix. If you specify `MeasurementNoise` as a scalar it will be extended to an *N*-by-*N* diagonal matrix. This property relates to the *R* variable in the state-space model.

Object Functions

Use the `predict` and `correct` functions based on detection results. Use the `distance` function to find the best matches.

- When the tracked object is detected, use the `predict` and `correct` functions with the Kalman filter object and the detection measurement. Call the functions in the following order:

  ```
  [...] = predict(kalmanFilter);
  [...] = correct(kalmanFilter,measurement);
  ```

- When the tracked object is not detected, call the `predict` function, but not the `correct` function. When the tracked object is missing or occluded, no measurement is available. Set the functions up with the following logic:

  ```
  [...] = predict(kalmanFilter);
  If measurement exists
      [...] = correct(kalmanFilter,measurement);
  end
  ```

- If the tracked object becomes available after missing for the past *t*-1 contiguous time steps, you can call the `predict` function *t* times. This syntax is particularly useful to process asynchronous video.. For example,

  ```
  for i = 1:k
    [...] = predict(kalmanFilter);
  end
  [...] = correct(kalmanFilter,measurement)
  ```

correct	Correction of measurement, state, and state estimation error covariance
predict	Prediction of measurement
distance	Confidence value of measurement

Examples

Track Location of An Object

Track the location of a physical object moving in one direction.

Generate synthetic data which mimics the 1-D location of a physical object moving at a constant speed.

```
detectedLocations = num2cell(2*randn(1,40) + (1:40));
```

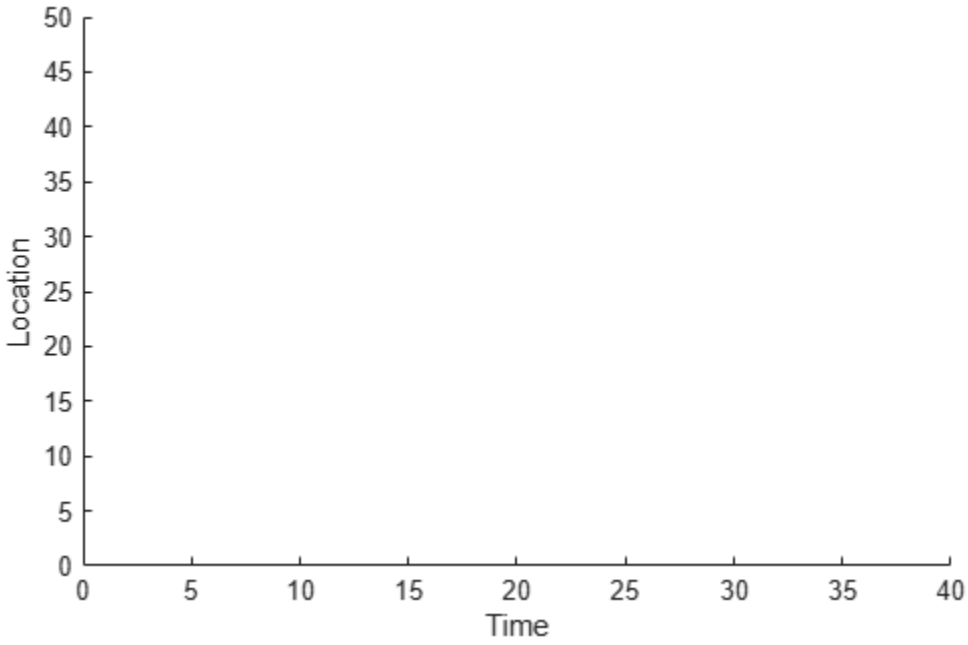
Simulate missing detections by setting some elements to empty.

```
detectedLocations{1} = [];
  for idx = 16: 25
      detectedLocations{idx} = [];
  end
```

Create a figure to show the location of detections and the results of using the Kalman filter for tracking.

```
figure;
hold on;
```

```
ylabel('Location');
ylim([0,50]);
xlabel('Time');
xlim([0,length(detectedLocations)]);
```

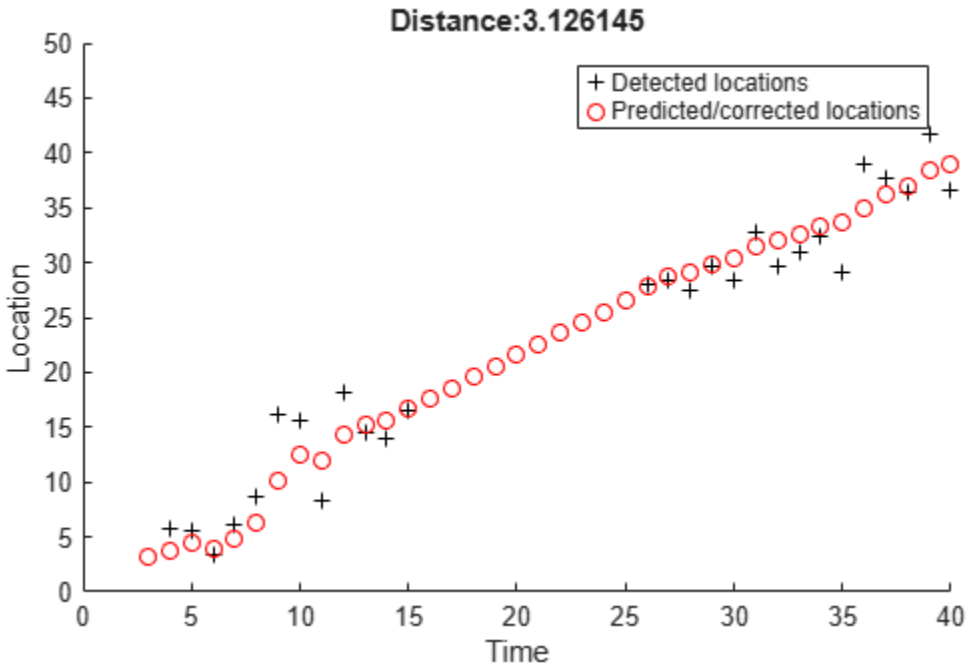


Create a 1-D, constant speed Kalman filter when the physical object is first detected. Predict the location of the object based on previous states. If the object is detected at the current time step, use its location to correct the states.

```
kalman = [];
for idx = 1: length(detectedLocations)
   location = detectedLocations{idx};
   if isempty(kalman)
     if ~isempty(location)

       stateModel = [1 1;0 1];
       measurementModel = [1 0];
       kalman = vision.KalmanFilter(stateModel,measurementModel,'ProcessNoise',1e-4,'MeasurementN
      kalman.State = [location, 0];
     end
   else
     trackedLocation = predict(kalman);
     if ~isempty(location)
       plot(idx, location,'k+');
      d = distance(kalman,location);
       title(sprintf('Distance:%f', d));
       trackedLocation = correct(kalman,location);
     else
       title('Missing detection');
     end
     pause(0.2);
     plot(idx,trackedLocation,'ro');
   end
```

```
 end
legend('Detected locations','Predicted/corrected locations');
```



Remove Noise from a Signal

Use Kalman filter to remove noise from a random signal corrupted by a zero-mean Gaussian noise.

Synthesize a random signal that has value of 1 and is corrupted by a zero-mean Gaussian noise with standard deviation of 0.1.

```
x = 1;
len = 100;
z = x + 0.1 * randn(1,len);
```

Remove noise from the signal by using a Kalman filter. The state is expected to be constant, and the measurement is the same as state.

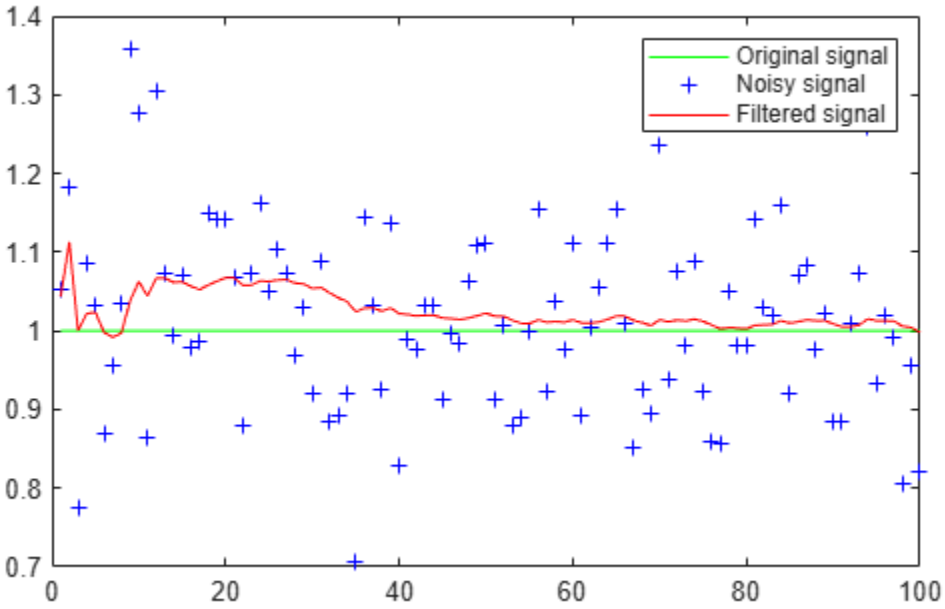
```
stateTransitionModel = 1;
measurementModel = 1;
obj = vision.KalmanFilter(stateTransitionModel,measurementModel,'StateCovariance',1,'ProcessNoise

z_corr = zeros(1,len);
for idx = 1: len
 predict(obj);
 z_corr(idx) = correct(obj,z(idx));
end
```

Plot results.

```
figure, plot(x * ones(1,len),'g-');
hold on;
```

```
plot(1:len,z,'b+',1:len,z_corr,'r-');
legend('Original signal','Noisy signal','Filtered signal');
```



Algorithms

State Space Model

This object implements a discrete time, linear state-space system, described by the following equations.

State equation:	$x(k) = Ax(k-1) + Bu(k-1) + w(k-1)$
Measurement equation:	$z(k) = Hx(k) + v(k)$

Variable Definition

Variable	Description	Dimension
k	Time.	Scalar
x	State. Gaussian vector with covariance P. [$x \sim N(\bar{x}, P)$]	M-element vector
P	State estimation error covariance.	M-by-M matrix
A	State transition model.	M-by-M matrix
B	Control model.	M-by-L matrix
u	Control input.	L-element vector
w	Process noise; Gaussian vector with zero mean and covariance Q. [$w \sim N(0, Q)$]	M-element vector
Q	Process noise covariance.	M-by-M matrix
z	Measurement. For example, location of detected object.	N-element vector
H	Measurement model.	N-by-M matrix
v	Measurement noise; Gaussian vector with zero mean and covariance R. [$v \sim N(0, R)$]	N-element vector
R	Measurement noise covariance.	N-by-N matrix

Version History

Introduced in R2012b

References

[1] Welch, Greg, and Gary Bishop, *An Introduction to the Kalman Filter*, TR 95–041. University of North Carolina at Chapel Hill, Department of Computer Science.

[2] Blackman, S. *Multiple-Target Tracking with Radar Applications.* Artech House, Inc., pp. 93, 1986.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`configureKalmanFilter` | `assignDetectionsToTracks`

vision.LocalMaximaFinder

Namespace: `vision`

(Removed) Find local maxima in matrices

Note `vision.LocalMaximaFinder` function has been removed. Use the `imregionalmax` to find the local maxima, instead.

Description

To find local maxima in matrices.

1. Create the `vision.LocalMaximaFinder` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
LMaxFinder = vision.LocalMaximaFinder
LMaxFinder = vision.LocalMaximaFinder(maxnum,neighborsize)
LMaxFinder = vision.LocalMaximaFinder(Name,Value)
```

Description

`LMaxFinder = vision.LocalMaximaFinder` returns a local maxima finder System object, `H`, that finds local maxima in input matrices.

`LMaxFinder = vision.LocalMaximaFinder(maxnum,neighborsize)` returns a local maxima finder object with the `MaximumNumLocalMaxima` property set to `maxnum`, `NeighborhoodSize` property set to `neighborsize`, and other specified properties set to the specified values.

`LMaxFinder = vision.LocalMaximaFinder(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `LMaxFinder = vision.LocalMaximaFinder('ThresholdSource','Property')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MaximumNumLocalMaxima` — Maximum number of maxima
`2` (default) | positive scalar integer

Maximum number of maxima to find, specified as a positive scalar integer.

`NeighborhoodSize` — Neighborhood size for zero-ing out values
`[5 7]` (default) | two-element vector

Neighborhood size for zero-ing out values, specified as a two-element vector

`ThresholdSource` — Source of threshold
`'Property'` (default) | `'Input port'`

Source of threshold, specified as `'Property'` or `'Input port'`.

`Threshold` — Value that all maxima should match or exceed
`10` (default) | built-in numeric data type

Value that all maxima should match or exceed, specified as a scalar of MATLAB built-in numeric data type. This property applies when you set the `ThresholdSource` property to `'Property'`.

Tunable: Yes

`HoughMatrixInput` — Indicator of Hough transform matrix input
`false` (default) | `true`

Indicator of Hough transform matrix input, specified as `true` or `false`. The block applies additional processing, specific to Hough transform on the right and left boundaries of the input matrix. Set this property to `true` if the input is antisymmetric about the rho axis and the theta value ranges from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ radians, which correspond to a Hough matrix.

`IndexDataType` — Data type of index values
`uint32` (default) | `double` | `single` | `uint8` | `uint16`

Data type of index values, specified as `double`, `single` , `uint8`, `uint16`, or `uint32`.

Usage

Syntax

```
idx = LMaxFinder(I)
idx = LMaxFinder(I,threshold)
```

Description

`idx = LMaxFinder(I)` returns [*x y*] coordinates of the local maxima in an *M*-by-2 matrix, `idx`. *M* represents the number of local maximas found. The maximum value of *M* may not exceed the value set in the `MaximumNumLocalMaxima` property.

`idx = LMaxFinder(I,threshold)` finds the local maxima in the input image `I`, using the threshold value `threshold`, when you set the `ThresholdSource` property to `'Input port'`.

Input Arguments

`I` — Video frame
grayscale | truecolor (RGB)

Video frame, specified as grayscale or truecolor (RGB).

`threshold` — Threshold
scalar

Value that all maxima should match or exceed, specified as a scalar of MATLAB built-in numeric data type.

Output Arguments

`idx` — Local maxima
M-by-2 matrix

Local maxima, returned as an *M*-by-2 matrix of one-based [*x* *y*] coordinates, where *M* represents the number of local maximas found.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.LocalMaximaFinder

initialize	Initialize video frame and points to track

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Find Local Maxima of an Input

1 Create an example input.

```
I = [0 0 0 0 0 0 0 0 0 0 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 3 5 7 5 3 1 0 0; ...
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 0 0 0 0 0 0 0 0 0];
```

2 Create a local maxima finder object.

```
hLocalMax = vision.LocalMaximaFinder('MaximumNumLocalMaxima',1, ...
                                     'NeighborhoodSize',[3,3], ...
                                     'Threshold',1);
```

3 Find local maxima.

```
location = hLocalMax(I)
```

```
location = 1x2 uint32 row vector

   7   4
```

Version History

Introduced in R2012b

R2024b: `vision.LocalMaximaFinder` function has been removed

Errors starting in R2024b

`vision.LocalMaximaFinder` function has been removed. Use the `imregionalmax` to find the local maxima, instead. For example,

```
I = [0 0 0 0 0 0 0 0 0 0 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 3 5 7 5 3 1 0 0; ...
     0 0 0 1 2 3 4 3 2 1 0 0; ...
     0 0 0 1 1 2 3 2 1 1 0 0; ...
     0 0 0 0 0 0 0 0 0 0 0 0];

BW = imregionalmax(I,8);
threshold = 1;
BW(I<threshold) = false;
BW = bwmorph(BW,"shrink",inf);
[row,col] = find(BW);
```

R2024a: `vision.LocalMaximaFinder` function will be removed

Warns starting in R2024a

`vision.LocalMaximaFinder` function will be removed in a future release. Use the `houghpeaks` or `max` functions to find the local maxima, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also
`hough` | `houghpeaks` | `max` | `vision.Maximum`

vision.Maximum

Namespace: `vision`

(Removed) Find maximum values in input or sequence of inputs

Note `vision.Maximum` function has been removed. Use the `max` and `movmax` function to find maximum values, instead.

Description

Find maximum values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.Maximum` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
maxObj = vision.Maximum
maxObj = vision.Maximum(Name,Value)
```

Description

`maxObj = vision.Maximum` returns an object, `maxObj`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`maxObj = vision.Maximum(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `maxObj = vision.Maximum('RunningMaximum',false)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ValueOutputPort` — Output maximum value
`true` (default) | `false`

Output maximum value, specified as `true` or `false`. Set this property to `true` to output the maximum value of the input. This property applies when you set the `RunningMaximum` property to `false`.

`RunningMaximum` — Calculate over single input or multiple inputs
`false` (default) | `true`

Calculate over single input or multiple inputs, specified as `true` or `false`. When you set this property to `true`, the object computes the maximum value over a sequence of inputs. When you set this property to `false`, the object computes the maximum value over the current input.

`IndexOutputPort` — Output the index of the maximum value
`true` (default) | `false`

Output the index of the maximum value, specified as `true` or `false`. This property applies only when you set the `RunningMaximum` property to `false`.

`ResetInputPort` — Additional input to enable resetting of running maximum
`false` (default) | `true`

Additional input to enable resetting of running maximum, specified as `true` or `false`. When you set this property to `true`, a reset input must be specified to reset the running maximum. This property applies only when you set the `RunningMaximum` property to `true`.

`ResetCondition` — Condition that triggers resetting of running maximum
`'Non-zero'` (default) | `'Rising edge'` | `'Falling edge'` | `'Either edge'`

Condition that triggers resetting of running maximum, specified as `'Rising edge'`, `'Falling edge'`, `'Either edge'`, or `'Non-zero'`. This property applies only when you set the `ResetInputPort` property to `true`.

`IndexBase` — Numbering base for index of maximum value
`'One'` (default) | `'Zero'`

Numbering base for index of maximum value, specified as `'One'` or `'Zero'`. This property applies only when you set the `IndexOutputPort` property to true.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningMaximum` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
1 (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

`ROIProcessing` — Enable region-of-interest processing
`false` (default) | `true`

Enable region-of-interest processing, specified as `true` or `false`. This property applies when you set the `Dimension` property to `'All'` and the `RunningMaximum` property to `false`.

`ROIForm` — Type of ROI
`'Rectangles'` (default) | `Lines'` | `'Label matrix''Binary mask'`

Type of ROI, specified as `'Rectangles'`, `'Lines'`, `'Label matrix'`, or `'Binary mask'`. This property applies only when you set the `ROIProcessing` property to `true`.

`ROIPortion` — ROI or perimeter calculation
`'Entire ROI'` (default) | `'ROI perimeter'`

ROI or perimeter calculation, specified as `'Entire ROI'` or `'ROI perimeter'`. This property applies only when you set the `ROIForm` property to `'Rectangles'`.

`ROIStatistics` — Calculate statistics for each ROI or one for all ROIs
`'Individual statistics for each ROI'` (default) | `'Single statistic for all ROIs'`

Calculate statistics for each ROI or one for all ROIs, specified as `'Individual statistics for each ROI'` or `'Single statistic for all ROIs'`. This property applies only when you set the `'ROIForm'` property to `'Rectangles'`, `'Lines'`, or `'Label matrix'`.

`ValidityOutputPort` — Flag to indicate if any part of ROI is outside input image
`false` (default) | `true`

Flag to indicate if any part of ROI is outside input image, specified as `true` or `false`.

This applies when you set the `ROIForm` property to `'Lines'` or `'Rectangles'`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `'Label matrix'`.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[value,index] = maxObj(input)
index = maxObj(input)

[___] = maxObj(I,ROI)
[___,flag] = maxObj(I,ROI)

[___] = maxObj(I,label,labelNumbers)
[___,flag] = maxObj(I,label,labelNumbers)
```

Description

`[value,index] = maxObj(input)` returns the maximum value and index of the input.

`index = maxObj(input)` returns the one-based index of the maximum value when you set the `IndexOutputPort` property to `true` and the `ValueOutputPort` property to `false`. The `RunningMaximum` property must be set to `false`.

`[___] = maxObj(I,ROI)` returns the maximum value in the input image within the given region of interest.

`[___,flag] = maxObj(I,ROI)`additionally returns a flag to indicate whether the given ROI is within the bounds of the image.

`[___] = maxObj(I,label,labelNumbers)` returns the maximum of the input image for a region the labels specified in the `labelNumbers` vector. The regions are defined and labeled in the `label` matrix.

`[___,flag] = maxObj(I,label,labelNumbers)` additionally returns a flag to indicate whether the input label numbers are valid.

Input Arguments

`input` — Input data
vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

`ROI` — Region of interest
four-element vector

Region of interest, specified as a four-element vector, [*x y width height*]. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Lines'`, `'Rectangles'`, or `'Binary Mask'`.

`label` — Label numbers
matrix

Label numbers, specified as a matrix. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

`labelNumbers` — Label numbers
vector

Label numbers, specified as a vector. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

Output Arguments

`value` — Maximum value
same as input

Maximum value, returned as the same data type as the input

`index` — Index to maximum value
one-based index

Index to maximum value, returned as a one-based index.

`flag` — Flag for valid data
`true` | `false`

Flag for valid data, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Find Index to Maximum Value in Grayscale Image

1 Read a color image.

```
img = imread('peppers.png');
```

2 Convert the image to grayscale.

```
img = im2single(im2gray(img));
```

3 Find the maximum.

```
hMax = vision.Maximum
```

```
hMax =
  vision.Maximum with properties:

    ValueOutputPort: true
     RunningMaximum: false
    IndexOutputPort: true
          Dimension: 'All'
      ROIProcessing: false

  Use get to show all properties
```

```
[m,ind] = hMax(img);
```

Version History

Introduced in R2012a

R2024b: `vision.Maximum` function has been removed
Errors starting in R2024b

`vision.Maximum` function has been removed. Use the `max` and `movmax` function to find maximum values, instead.

R2024a: `vision.Maximum` function will be removed
Warns starting in R2024a

`vision.Maximum` function will be removed in a future release. Use the `max` and `movmax` function to find maximum values, instead.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`max` | `movmax`

vision.Mean

Namespace: `vision`

(Removed) Find mean values in input or sequence of inputs

Note `vision.Mean` function has been removed. Use the `mean` and `movmean` functions to find mean values, instead.

Description

Find the mean values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.Mean` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
minObj = vision.Mean
minObj = vision.Mean(Name,Value)
```

Description

`minObj = vision.Mean` returns an object, `minObj`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`minObj = vision.Mean(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `minObj = vision.Maximum('RunningMean',false)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`RunningMean` — Calculate over single input or multiple inputs

`false` (default) | `true`

Calculate over single input or multiple inputs, specified as `true` or `false`. When you set this property to `true`, the object computes the mean value over a sequence of inputs. When you set this property to `false`, the object computes the mean value over the current input.

`ResetInputPort` — Additional input to enable resetting of running mean
`false` (default) | `true`

Additional input to enable resetting of running mean, specified as `true` or `false`. When you set this property to `true`, a reset input must be specified to reset the running mean. This property applies only when you set the `RunningMean` property to `true`.

`ResetCondition` — Condition that triggers resetting of running mean
`'Non-zero'` (default) | `'Rising edge'` | `'Falling edge'` | `'Either edge'`

Condition that triggers resetting of running mean, specified as `'Rising edge'`, `'Falling edge'`, `'Either edge'`, or `'Non-zero'`. This property applies only when you set the `ResetInputPort` property to `true`.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningMean` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
`1` (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

`ROIProcessing` — Enable region-of-interest processing
`false` (default) | `true`

Enable region-of-interest processing, specified as `true` or `false`. This property applies when you set the `Dimension` property to `'All'` and the `RunningMean` property to `false`.

`ROIForm` — Type of ROI
`'Rectangles'` (default) | `Lines'` | `'Label matrix''Binary mask'`

Type of ROI, specified as `'Rectangles'`, `'Lines'`, `'Label matrix'`, or `'Binary mask'`. This property applies only when you set the `ROIProcessing` property to `true`.

`ROIPortion` — ROI or perimeter calculation
`'Entire ROI'` (default) | `'ROI perimeter'`

ROI or perimeter calculation, specified as `'Entire ROI'` or `'ROI perimeter'`. This property applies only when you set the `ROIForm` property to `'Rectangles'`.

`ROIStatistics` — Calculate statistics for each ROI or one for all ROIs
`'Individual statistics for each ROI'` (default) | `'Single statistic for all ROIs'`

Calculate statistics for each ROI or one for all ROIs, specified as `'Individual statistics for each ROI'` or `'Single statistic for all ROIs'`. This property applies only when you set the `'ROIForm'` property to `'Rectangles'`, `'Lines'`, or `'Label matrix'`.

`ValidityOutputPort` — Flag to indicate if any part of ROI is outside input image
`false` (default) | `true`

Flag to indicate if any part of ROI is outside input image, specified as `true` or `false`.

This applies when you set the `ROIForm` property to `'Lines'` or `'Rectangles'`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `'Label matrix'`.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[value,index] = minObj(input)
index = minObj(input)

[___] = minObj(I,ROI)
[___,flag] = minObj(I,ROI)
```

```
[ ___ ] = minObj(I,label,labelNumbers)
[ ___ ,flag] = minObj(I,label,labelNumbers)
```

Description

`[value,index] = minObj(input)` returns the mean value and index of the input.

`index = minObj(input)` returns the one-based index of the mean value when you set the `IndexOutputPort` property to `true` and the `ValueOutputPort` property to `false`. The `RunningMean` property must be set to `false`.

`[ ___ ] = minObj(I,ROI)` returns the mean value in the input image within the given region of interest.

`[ ___ ,flag] = minObj(I,ROI)`additionally returns a flag to indicate whether the given ROI is within the bounds of the image.

`[ ___ ] = minObj(I,label,labelNumbers)` returns the mean of the input image for a region the labels specified in the `labelNumbers` vector. The regions are defined and labeled in the `label` matrix.

`[ ___ ,flag] = minObj(I,label,labelNumbers)` additionally returns a flag to indicate whether the input label numbers are valid.

Input Arguments

`input` — Input data
vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

`ROI` — Region of interest
four-element vector

Region of interest, specified as a four-element vector, [*x y width height*]. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Lines'`, `'Rectangles'`, or `'Binary Mask'`.

`label` — Label numbers
matrix

Label numbers, specified as a matrix. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

`labelNumbers` — Label numbers
vector

Label numbers, specified as a vector. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

Output Arguments

`value` — Mean value
same as input

Mean value, returned as the same data type as the input

`index` — Index to mean value
one-based index

Index to mean value, returned as a one-based index.

`flag` — Flag for valid data
`true` | `false`

Flag for valid data, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Determine Mean Value in Grayscale Image

1 Read a color image.

```
img = imread('peppers.png');
```

2 Convert the image to grayscale.

```
img = im2single(im2gray(img));
```

3 Find the mean.

```
hMean = vision.Mean;
mean = hMean(img);
```

Version History

Introduced in R2012a

R2024b: `vision.Mean` function has been removed
Errors starting in R2024b

`vision.Mean` function has been removed. Use the `mean` and `movmean` functions to find mean values, instead.

R2024a: `vision.Mean` function will be removed
Warns starting in R2024a

`vision.Mean` function will be removed in a future release. Use the `mean` and `movmean` functions to find mean values, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`mean` | `movmean`

vision.Median

Namespace: `vision`

(Removed) Find median values in input or sequence of inputs

Note `vision.Median` function has been removed. Use the `median` and `movmedian` functions to find median values, instead.

Description

Find the median values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.Median` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
medObj = vision.Median
medObj = vision.Median(Name,Value)
```

Description

`medObj = vision.Median` returns an object, `medObj`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`medObj = vision.Median(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `medObj = vision.Median('Dimension','Column')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SortMethod` — Sort method

`'Quick sort'` | `'Insertion sort'`

Sort method for calculating the median value, specified as `'Quick sort'` or `'Insertion sort'`.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningMean` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
1 (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[value,index] = medObj(input)
```

Description

`[value,index] = medObj(input)` returns the median value and index of the input.

Input Arguments

`input` — Input data

vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

Output Arguments

`value` — Median value

same as input

Median value, returned as the same data type as the input

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Determine Median Value in Grayscale Image

1. Read a color image.

```
img = imread('peppers.png');
```

2. Convert the image to grayscale.

```
img = im2single(rgb2gray(img));
```

3. Find the median.

```
hmed = vision.Median;
medValue = hmed(img);
```

Version History

Introduced in R2012a

R2024b: `vision.Median` function has been removed

Errors starting in R2024b

`vision.Median` function has been removed. Use the `median` and `movmedian` functions to find median values, instead.

R2024a: `vision.Median` function will be removed
Warns starting in R2024a

`vision.Median` function will be removed in a future release. Use the `median` and `movmedian` functions to find median values, instead.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also
`median` | `movmedian`

vision.Minimum

Namespace: `vision`

(Removed) Find minimum values in input or sequence of inputs

Note `vision.Minimum` function has been removed. Use the `min` and `movmin` functions to find minimum values, instead.

Description

Find the minimum values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.Minimum` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
minObj = vision.Minimum
minObj = vision.Minimum(Name,Value)
```

Description

`minObj = vision.Minimum` returns an object, `minObj`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`minObj = vision.Minimum(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `minObj = vision.Maximum('RunningMinimum',false)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ValueOutputPort` — Output Minimum value
`true` (default) | `false`

Output minimum value, specified as `true` or `false`. Set this property to `true` to output the Minimum value of the input. This property applies when you set the `RunningMinimum` property to `false`.

`RunningMinimum` — Calculate over single input or multiple inputs
`false` (default) | `true`

Calculate over single input or multiple inputs, specified as `true` or `false`. When you set this property to `true`, the object computes the minimum value over a sequence of inputs. When you set this property to `false`, the object computes the minimum value over the current input.

`IndexOutputPort` — Output the index of the minimum value
`true` (default) | `false`

Output the index of the minimum value, specified as `true` or `false`. This property applies only when you set the `RunningMinimum` property to `false`.

`ResetInputPort` — Additional input to enable resetting of running minimum
`false` (default) | `true`

Additional input to enable resetting of running minimum, specified as `true` or `false`. When you set this property to `true`, a reset input must be specified to reset the running minimum. This property applies only when you set the `RunningMinimum` property to `true`.

`ResetCondition` — Condition that triggers resetting of running minimum
`'Non-zero'` (default) | `'Rising edge'` | `'Falling edge'` | `'Either edge'`

Condition that triggers resetting of running minimum, specified as `'Rising edge'`, `'Falling edge'`, `'Either edge'`, or `'Non-zero'`. This property applies only when you set the `ResetInputPort` property to `true`.

`IndexBase` — Numbering base for index of minimum value
`'One'` (default) | `'Zero'`

Numbering base for index of minimum value, specified as `'One'` or `'Zero'`. This property applies only when you set the `IndexOutputPort` property to true.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningMinimum` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
1 (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

`ROIProcessing` — Enable region-of-interest processing
`false` (default) | `true`

Enable region-of-interest processing, specified as `true` or `false`. This property applies when you set the `Dimension` property to `'All'` and the `RunningMinimum` property to `false`.

`ROIForm` — Type of ROI
`'Rectangles'` (default) | `Lines'` | `'Label matrix''Binary mask'`

Type of ROI, specified as `'Rectangles'`, `'Lines'`, `'Label matrix'`, or `'Binary mask'`. This property applies only when you set the `ROIProcessing` property to `true`.

`ROIPortion` — ROI or perimeter calculation
`'Entire ROI'` (default) | `'ROI perimeter'`

ROI or perimeter calculation, specified as `'Entire ROI'` or `'ROI perimeter'`. This property applies only when you set the `ROIForm` property to `'Rectangles'`.

`ROIStatistics` — Calculate statistics for each ROI or one for all ROIs
`'Individual statistics for each ROI'` (default) | `'Single statistic for all ROIs'`

Calculate statistics for each ROI or one for all ROIs, specified as `'Individual statistics for each ROI'` or `'Single statistic for all ROIs'`. This property applies only when you set the `'ROIForm'` property to `'Rectangles'`, `'Lines'`, or `'Label matrix'`.

`ValidityOutputPort` — Flag to indicate if any part of ROI is outside input image
`false` (default) | `true`

Flag to indicate if any part of ROI is outside input image, specified as `true` or `false`.

This applies when you set the `ROIForm` property to `'Lines'` or `'Rectangles'`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `'Label matrix'`.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[value,index] = minObj(input)
index = minObj(input)

[ ___ ] = minObj(I,ROI)
[ ___ ,flag] = minObj(I,ROI)

[ ___ ] = minObj(I,label,labelNumbers)
[ ___ ,flag] = minObj(I,label,labelNumbers)
```

Description

`[value,index] = minObj(input)` returns the minimum value and index of the input.

`index = minObj(input)` returns the one-based index of the minimum value when you set the `IndexOutputPort` property to `true` and the `ValueOutputPort` property to `false`. The `RunningMinimum` property must be set to `false`.

`[ ___ ] = minObj(I,ROI)` returns the minimum value in the input image within the given region of interest.

`[ ___ ,flag] = minObj(I,ROI)`additionally returns a flag to indicate whether the given ROI is within the bounds of the image.

`[ ___ ] = minObj(I,label,labelNumbers)` returns the minimum of the input image for a region the labels specified in the `labelNumbers` vector. The regions are defined and labeled in the `label` matrix.

`[ ___ ,flag] = minObj(I,label,labelNumbers)` additionally returns a flag to indicate whether the input label numbers are valid.

Input Arguments

`input` — Input data
vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

`ROI` — Region of interest
four-element vector

Region of interest, specified as a four-element vector, [*x y width height*]. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Lines'`, `'Rectangles'`, or `'Binary Mask'`.

`label` — Label numbers
matrix

Label numbers, specified as a matrix. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

`labelNumbers` — Label numbers
vector

Label numbers, specified as a vector. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

Output Arguments

`value` — Minimum value
same as input

Minimum value, returned as the same data type as the input

`index` — Index to minimum value
one-based index

Index to minimum value, returned as a one-based index.

`flag` — Flag for valid data
`true` | `false`

Flag for valid data, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Find Index to Minimum Value in Grayscale Image

1. Read a color image.

```
img = imread('peppers.png');
```

2. Convert the image to grayscale.

```
img = im2single(im2gray(img));
```

3 Find the minimum.

```
hMin = vision.Minimum

hMin =
  vision.Minimum with properties:

    ValueOutputPort: true
     RunningMinimum: false
    IndexOutputPort: true
          Dimension: 'All'
      ROIProcessing: false

  Use get to show all properties


[m, ind] = hMin(img);
```

Version History

Introduced in R2012a

R2024b: `vision.Minimum` function has been removed

Errors starting in R2024b

`vision.Minimum` function has been removed. Use the `min` and `movmin` functions to find minimum values, instead.

R2024a: `vision.Minimum` function will be removed

Warns starting in R2024a

`vision.Minimum` function will be removed in a future release. Use the `min` and `movmin` functions to find minimum values, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`min` | `movmin`

vision.VideoFileReader

Namespace: `vision`

(To be removed) Read video frames and audio samples from video file

Note The `vision.VideoFileReader` system object will be removed in a future release. Use the `VideoReader` object instead to read video files or the `audioread` or `dsp.AudioFileReader` object to read audio files.

Description

The `VideoFileReader` object reads video frames, images, and audio samples from a video file. The object can also read image files.

Platforms	File Formats
All Platforms	AVI, including uncompressed, indexed, grayscale, and Motion JPEG-encoded video (`.avi`) Motion JPEG 2000 (`.mj2`) **Note** `VideoReader` does not support remote reading for formats exclusive to Microsoft DirectShow®.
All Windows	MPEG-1 (`.mpg`) Windows Media Video (`.wmv`, `.asf`) Any format supported by Microsoft DirectShow
Windows 7 or later	MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (`.mov`) Any format supported by Microsoft Media Foundation
Macintosh	Most formats supported by QuickTime Player, including: MPEG-1 (`.mpg`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (`.mov`) 3GPP 3GPP2 AVCHD DV **Note:** For OS X Yosemite (Version 10.10) and later, MPEG-4/H.264 files written using `VideoWriter`, play correctly, but display an inexact frame rate.

Platforms	File Formats
Linux	Any format supported by your installed plug-ins for GStreamer 1.0 or higher, as listed on https://gstreamer.freedesktop.org/documentation/plugins_doc.html, including Ogg Theora (`.ogg`).

To read a file:

1. Create the `vision.VideoFileReader` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
videoFReader = vision.VideoFileReader(Filename)
videoFReader = vision.VideoFileReader( ___ ,Name,Value)
```

Description

`videoFReader = vision.VideoFileReader(Filename)` returns a video file reader System object, `videoFReader`, that sequentially reads video frames or audio samples from an input file, `Filename`.

`videoFReader = vision.VideoFileReader( ___ ,Name,Value)` additionally sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `videoFReader = vision.VideoFileReader('PlayCount',1)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Filename` — File name

`vipmen.avi` (default) | character vector

File name, specified as a character vector. The full path for the file needs to be specified only if the file is not on the MATLAB path.

`PlayCount` — Number of times to play file

`1` (default) | positive integer | `inf`

Number of times to play file, specified as a positive integer.

`AudioOutputPort` — Output audio data
`false` (default) | `true`

Output audio data, specified as `true` or `false`. Use this property to control the audio output only when the input file contains audio and video streams.

`ImageColorSpace` — Image format
`RGB` (default) | `YCbCr 4:2:2` | intensity video

Image format, specified as `RGB`, `YCbCr 4:2:2`, or intensity video frames when the input file contains video.

`VideoOutputDataType` — Output video data type
`single` (default)

Output video data type, specified as one of the data types listed or `'inherit'`. This property applies when the input file contains video. When you set this property to `'inherit'`, the object sets the output data type to the native data type of the input video.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`AudioOutputDataType` — Output audio samples data type
`int16` (default)

Output audio samples data type, specified as one of the data types listed and `inherit`. This property applies when the input file contains audio.

Data Types: `double` | `single` | `int16` | `uint8`

Usage

Syntax

```
I = videoFReader()
[Y,Cb,Cr] = videoFReader()
[___,audio] = videoFReader()
[___,EOF] = videoFReader()
```

Description

`I = videoFReader()` returns the next video frame.

`[Y,Cb,Cr] = videoFReader()` returns the next frame of `YCbCr 4:2:2` format video in the color components `Y`, `Cb`, and `Cr`. This syntax requires that you set the `'ImageColorSpace'` property to `'YCbCr 4:2:2'`

`[___,audio] = videoFReader()` also returns one frame of audio samples, `audio`. This syntax requires that you set the `AudioOutputPort` property to `true`.

`[___,EOF] = videoFReader()` also returns the end-of-file indicator, `EOF`. The object sets `EOF` to `true` each time the output contains the last audio sample and/or video frame.

Output Arguments

`I` — Video frame

truecolor | 2-D grayscale image

Video frame, returned as a truecolor or 2-D grayscale image.

`Y`, `Cb`, `Cr` — YCbCr color channels

numeric matrices

YCbCr color channels of a `YCbCr 4:2:2` format image, returned as numeric matrices.

`audio` — One frame of audio samples

`.wav` | `.mp3` | `.mp3` | `.mp4` | `.ogg` | `.flac` | `.au` | `.aiff` | `.aif` | `.aifc`

One frame of audio samples, returned in one of the following formats.

Platform	Supported File Name Extensions
All Platforms	AVI (`.avi`)
Windows	**Image**: `.jpg`,`.bmp`
	Video: MPEG (`.mpeg`) MPEG-2 (`.mp2`) MPEG-1`.mpg` MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Motion JPEG 2000 (`.mj2`) Windows Media Video (`.wmv`,`.asf`, `.asx`, .asx) and any format supported by Microsoft DirectShow® 9.0 or higher.
	Audio: WAVE (`.wav`) Windows Media Audio File (`.wma`) Audio Interchange File Format (`.aif`, `.aiff`) Compressed Audio Interchange File Format(`.aifc`), MP3 (`.mp3`) Sun Audio (`.au`) Apple (`.snd`)
Macintosh	**Video**: `.avi` Motion JPEG 2000 (`.mj2`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (.mov) and any format supported by QuickTime as listed on http://support.apple.com/kb/HT3775.
	Audio: Uncompressed `.avi`
Linux	Motion JPEG 2000 (`.mj2`) Any format supported by your installed plug-ins for GStreamer 0.1 or higher, as listed on https://gstreamer.freedesktop.org/documentation/plugins_doc.html?gi-language=c, including Ogg Theora (.ogg).

Windows XP and Windows 7 x64 platform ships with a limited set of 64-bit video and audio codecs. If a compressed multimedia file fails to play, try saving the multimedia file to a supported file format listed in the table above.

If you use Windows, use Windows Media player Version 11 or later.

Note MJ2 files with bit depth higher than 8-bits are not supported by `vision.VideoFileReader`. Use `VideoReader` and `VideoWriter` for higher bit depths.

Reading audio from compressed MP4 files with video and audio is not supported by `vision.VideoFileReader`.

EOF — End-of-file

`true` | `false`

End-of-file indicator, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.VideoFileReader

info	Information about specified video file
isDone	End-of-file status (logical)

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Read and Play a Video File

Load the video using a video reader object.

```
vidReader = VideoReader('ecolicells.avi');
```

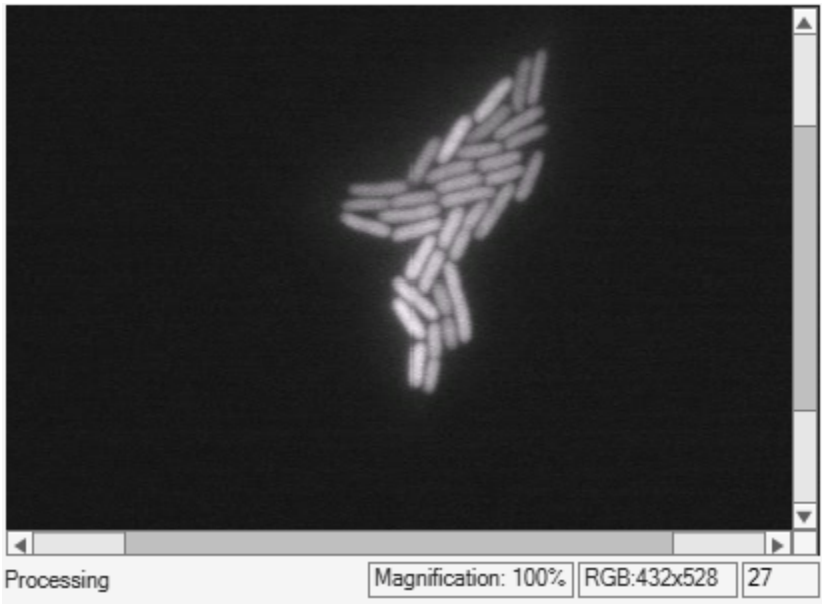
Create a video player object to play the video file.

```
videoPlayer = vision.VideoPlayer;
```

Use a while loop to read and play the video frames. Pause for 0.1 seconds after displaying each frame.

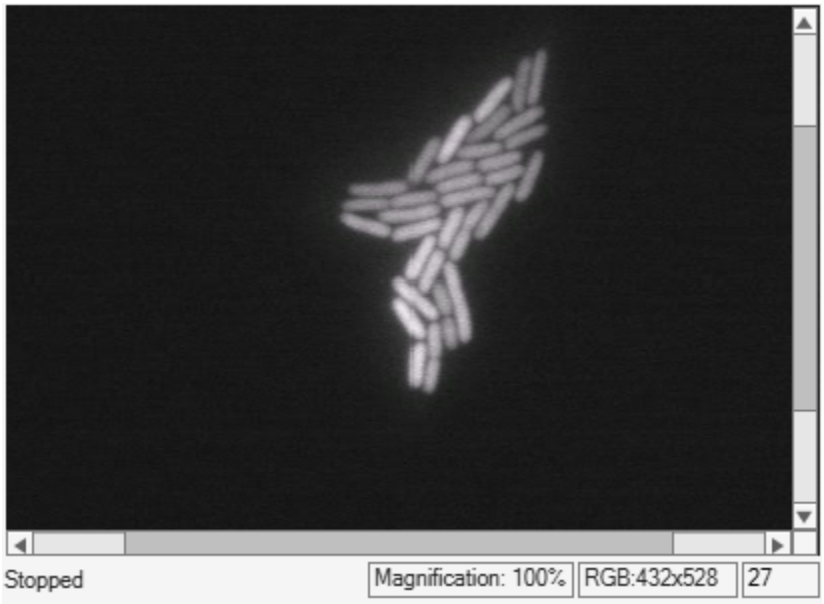
```
while hasFrame(vidReader)
```

```
    videoFrame = readFrame(vidReader);
    videoPlayer(videoFrame);
    pause(0.1)
end
```



Release the objects.

```
release(videoPlayer);
```



Tips

- Video Reading Performance on Windows Systems:To achieve better video reader performance on Windows for MP4 and MOV files, MATLAB uses the system's graphics hardware for decoding.

However, in some cases using the graphics card for decoding can result in poorer performance depending on the specific graphics hardware on the system. If you notice slower video reader performance on your system, turn off the hardware acceleration by typing:

```
matlab.video.read.UseHardwareAcceleration('off')
```

Hardware acceleration can be reenabled by typing:

```
matlab.video.read.UseHardwareAcceleration('on')
```

Version History

Introduced in R2012a

R2022b: `vision.VideoFileReader` system object will be removed in future release
Warns starting in R2022b

The `vision.VideoFileReader` system object will be removed in a future release. Use the `VideoReader` object instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generated code for this function uses a precompiled platform-specific shared library.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`vision.VideoFileWriter` | `VideoWriter` | `VideoReader` | `vision.VideoPlayer` | Video Viewer

Topics

“Video Display in a Custom User Interface”

vision.VideoFileWriter

Namespace: `vision`

Write video frames and audio samples to video file

Description

The `VideoFileWriter` object writes video frames and audio samples to a video file. The video and audio can be compressed. The available compression types depend on the encoders installed on the platform.

Note This block supports code generation for platforms that have file I/O available. You cannot use this block with Simulink Desktop Real-Time software, because that product does not support file I/O.

This object performs best on platforms with Version 11 or later of Windows Media Player software. This object supports only uncompressed RGB24 AVI files on Linux and Mac platforms.

The generated code for this object relies on prebuilt library files. You can run this code outside the MATLAB environment, or redeploy it, but be sure to account for these extra library files when doing so. The packNGo function creates a single zip file containing all of the pieces required to run or rebuild this code. See `packNGo` for more information.

To run an executable file that was generated from an object, you may need to add precompiled shared library files to your system path. See “MATLAB Coder” and “Simulink Shared Library Dependencies” for details.

Creation

Syntax

```
videoFWriter = vision.VideoFileWriter
videoFWriter = vision.VideoFileWriter(Filename)
videoFWriter = vision.VideoFileWriter( ___ ,Name,Value)
```

Description

`videoFWriter = vision.VideoFileWriter` returns a video file writer System object, `videoFWriter`. It writes video frames to an uncompressed '`output.avi`' video file. Every call to the `step` method writes a video frame.

`videoFWriter = vision.VideoFileWriter(Filename)` returns a video file writer object, `videoFWriter` that writes video to a file, `Filename`. The file type can be `.avi`, `.mj2`, `.mp4`, and `.m4v` specified by the `FileFormat` property.

`videoFWriter = vision.VideoFileWriter( ___ ,Name,Value)` configures the video file writer properties, specified as one or more name-value pair arguments. Unspecified properties have default values.

`Name` is a property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `videoFWriter = vision.VideoFileWriter('myFile.avi','FrameRate',... videoFReader.info.VideoFrameRate);`

Properties

`Filename` — Video output file name

`output.avi` (default) | character vector

Video output file name, specified as a character vector. The file extension you give for `Filename` must match the `FileFormat`.

`FileFormat` — Output file format

`'AVI'` (default) | `'MJ2000'` | `'MPEG4'`

Output file format, specified as one of the supported file formats shown in the table.

File Format	Description	File Extension	Supported Platform
`'AVI'`	Audio-Video Interleave file	`.avi`	All platforms
`'MJ2000'`	Motion JPEG 2000 file	`.mj2`	All platforms
`'MPEG4'`	MPEG-4/H.264 Video	`.mp4` , `.m4v`	Windows and Mac

`AudioInputPort` — Write audio data

`false` (default) | `true`

Write audio data, specified as `false` or `true`. Use this property to control whether the object writes audio samples to the video file. Set this value to `true` to write audio data. To write audio and video to a file, you must use the `.avi` format.

`FrameRate` — Video frame rate

`30` (default) | positive numeric scalar

Video frame rate in frames per second, specified as a positive numeric scalar. For videos which also contain audio data, the rate of the audio data will be determined as the rate of the video multiplied by the number of audio samples passed in. For example, if you use a frame rate of `30`, and pass 1470 audio samples, the object sets the audio sample to `44100`, (`1470 x 30 = 44100`).

`AudioCompressor` — Audio compression encoder

`None (uncompressed)` (default) | system compressors

Specify the type of compression algorithm to implement for audio data. This compression reduces the size of the video file. Choose `None (uncompressed)` to save uncompressed audio data to the video file. The other options reflect the available audio compression algorithms installed on your system. This property applies only when writing `AVI` files on Windows platforms.

`VideoCompressor` — Video compression encoder

`None (uncompressed)` (default) | system compressors

Specify the type of compression algorithm to use to compress the video data. This compression reduces the size of the video file. Choose `None (uncompressed)` to save uncompressed video data

to the video file. The `VideoCompressor` property can also be set to one of the compressors available on your system. To obtain a list of available video compressors, you can use tab completion. Follow these steps:

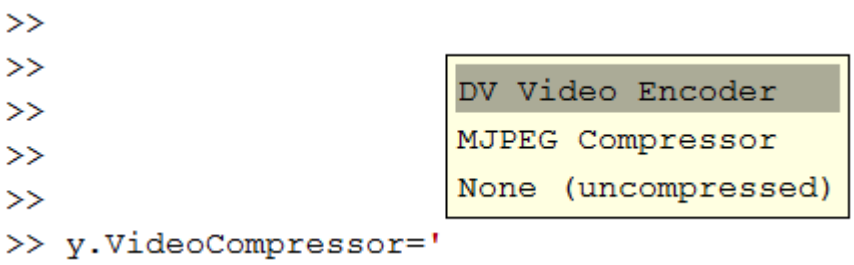
1. Instantiate the object:

   ```
   y = vision.VideoFileWriter
   ```

2. To launch the tab completion functionality, type the following up to the open quote.

   ```
   y.VideoCompressor='
   ```

   A list of compressors available on your system will appear after you press the `Tab` key. For example:



This property applies only when writing `AVI` files on Windows platforms.

`AudioDataType` — Uncompressed audio data type
`WAV` (default)

Specify the compressed output audio data type. This property only applies when you write uncompressed `WAV` files.

`FileColorSpace` — Color space for output file
`RGB` (default) | `YCbCr 4:2:2`

Color space for output AVI file, specified as `RGB` or `YCbCr 4:2:2`. This property applies when you set the `FileFormat` property to `AVI` and only on Windows platforms.

`Quality` — Control size of output video file
`75` (default) | integer

Control size of output video file, specified as an integer in the range `[0,100]`. Increase this value for greater video quality. However, doing so increases the file size. Decrease the value to lower video quality with a smaller file size.

The `Quality` property only applies when you are writing MPEG4 video files (on Windows or Mac) or when you are writing MJPEG-AVI video only files on a Mac or Linux.

`CompressionFactor` — Target ratio between number of bytes in input image and compressed image
`10` (default) | integer

Target ratio between number of bytes in input image and compressed image, specified as an integer greater than `1`. The `CompressionFactor` indicates the target ratio between the number of bytes in the input image and the compressed image. The data is compressed as much as possible, up to the specified target. This property applies only when writing Lossy MJ2000 files.

Usage

Syntax

```
Filename = videoFWriter(videoFrame)
Filename = videoFWriter(videoFrame,audio)
Filename = videoFWriter(videoFrame,Y,Cb,Cr,audio)
```

Description

`Filename = videoFWriter(videoFrame)` writes one frame of video, `videoFrame`, to the output file. The input video can be an *M*-by-*N*-by-3 truecolor RGB video frame, or an *M*-by-*N* grayscale video frame..

`Filename = videoFWriter(videoFrame,audio)` writes one frame of the input video, `videoFrame`, and one frame of audio samples, `audio`, to the output file. This applies when you set the `AudioInputPort` property to `true`.

`Filename = videoFWriter(videoFrame,Y,Cb,Cr,audio)` writes one frame of YCbCr 4:2:2 video, and one frame of audio samples, `audio`, to the output file. This applies when you set the `AudioInputPort` to `true` and the `FileColorSpace` property to '`YCbCr 4:2:2`'. The width of Cb and Cr color components must be half of the width of Y.

Input Arguments

`videoFrame` — Video frame

truecolor | 2-D grayscale image

Video frame, returned as a truecolor or 2-D grayscale image.

`Y,Cb,Cr` — YCbCr color format

`'YCbCr 4:2:2'`

YCbCr color format, returned in the `YCbCr 4:2:2` format.

`audio` — One frame of audio samples

`.wav` | `.mp3` | `.mp3` | `.mp4` | `.ogg` | `.flac` | `.au` | `.aiff` | `.aif` | `.aifc`

One frame of audio samples, returned in one of the following formats.

Platform	Supported File Name Extensions
All Platforms	AVI (`.avi`)
Windows	**Image**: `.jpg`,`.bmp`
	Video: MPEG (`.mpeg`) MPEG-2 (`.mp2`) MPEG-1`.mpg` MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Motion JPEG 2000 (`.mj2`) Windows Media Video (`.wmv`,`.asf`, `.asx`, .asx) and any format supported by Microsoft DirectShow® 9.0 or higher.

Platform	Supported File Name Extensions
	Audio: WAVE (`.wav`) Windows Media Audio File (`.wma`) Audio Interchange File Format (`.aif`, `.aiff`) Compressed Audio Interchange File Format(`.aifc`), MP3 (`.mp3`) Sun Audio (`.au`) Apple (`.snd`)
Macintosh	**Video**: `.avi` Motion JPEG 2000 (`.mj2`) MPEG-4, including H.264 encoded video (`.mp4`, `.m4v`) Apple QuickTime Movie (.mov) and any format supported by QuickTime as listed on http://support.apple.com/kb/HT3775.
	Audio: Uncompressed `.avi`
Linux	Motion JPEG 2000 (`.mj2`) Any format supported by your installed plug-ins for GStreamer 0.1 or higher, as listed on http://gstreamer.freedesktop.org/documentation/plugins.html, including Ogg Theora (.ogg).

Windows XP and Windows 7 x64 platform ships with a limited set of 64-bit video and audio codecs. If a compressed multimedia file fails to play, try saving the multimedia file to a supported file format listed in the table above.

If you use Windows, use Windows Media player Version 11 or later.

Note Use `VideoReader` and `VideoWriter` for higher bit depths.

Output Arguments

`Filename` — Video output file name
`'.avi'` (default) | `'.mj2'` | `'.mp4'` | `'.m4v'`

Video output file name, specified as `'.avi'`, `'.mj2'`, `'.mp4'`, and `'.m4v'`. The file extension you give for `Filename` must match the `FileFormat`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.VideoFileWriter

info Information about specified video file

isDone End-of-file status (logical)

Common to All System Objects

step Run System object algorithm
release Release resources and allow changes to System object property values and input characteristics
reset Reset internal states of System object

Examples

Write a Video to an AVI File

Load a video file and write the file to a writer object.

```
vidReader = VideoReader('viplanedeparture.mp4');
videoFWriter = vision.VideoFileWriter('myFile.avi', ...
    'FrameRate',vidReader.FrameRate);
```

Write the first 50 frames from original file into a newly created `myFile.avi` file.

```
for i=1:50
  videoFrame = readFrame(vidReader);
  videoFWriter(videoFrame);
end
```

Close the input and output files.

```
release(videoFWriter);
```

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generated code for this function uses a precompiled platform-specific shared library.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`VideoWriter` | `VideoReader` | `vision.VideoPlayer`

ocrText

Store OCR results

Description

The `ocrText` object contains recognized text and metadata collected during optical character recognition (OCR). You can locate text that matches a specific pattern with the `locateText` function.

Creation

Create an `ocrText` object or array using the `ocr` function.

Properties

`Text` — Text recognized by OCR
cell array of characters

Text recognized by OCR, specified as a cell array of characters. The text includes white space and new line characters.

`CharacterBoundingBoxes` — Character bounding box locations and sizes
M-by-4 matrix

Character bounding box locations and sizes, specified as an *M*-by-4 matrix. Each row of the matrix is of the form, [*x y width height*]. The [*x y*] elements correspond to the upper-left corner of the bounding box. The [*width height*] elements correspond to the horizontal and vertical size, respectively, of the rectangular region in pixels. The bounding boxes enclose the text found in an image by the `ocr` function. Widths and heights of bounding boxes that correspond to new line characters have values of zero. Bounding boxes for character modifiers found in languages such as Hindi, Tamil, and Bengali, also have values of zero.

`CharacterConfidences` — Character recognition confidences
array

Character recognition confidence, specified as an array. Confidence values are in the range [0, 1]. Interpret each confidence value set by the `ocr` function, as a probability. The `ocr` function sets confidence values for spaces between words and new line characters to `NaN`, as OCR does not explicitly recognize spaces and new line characters. You can use the confidence values to identify the location of misclassified text within the image by extracting characters with low confidence.

`Words` — Recognized words
cell array of character vectors

Recognized words, specified as a cell array of character vectors.

`WordBoundingBoxes` — Word bounding box location and size of words
N-by-4 matrix

Word bounding box locations and sizes, specified as an *N*-by-4 matrix. Each row of the matrix is of the form, [*x y width height*], and specifies the upper-left corner position and size of a rectangular region in pixels.

`WordConfidences` — Word recognition confidences
vector

Word recognition confidences, specified as a vector of probability values in the range [0,1]. The `ocr` function sets confidence values for spaces between words and new line characters to `NaN`, as OCR does not explicitly recognize spaces and new line characters. You can use confidence values to identify the location of misclassified text within the image by extracting words with low confidence.

`TextLines` — Recognized text lines
cell array of character vectors

Recognized text lines, specified as a cell array of character vectors.

`TextLineBoundingBoxes` — Text line bounding box location and size
N-by-4 matrix

Text line bounding box location and size, specified as an *N*-by-4 matrix. Each row of the matrix is of the form, [*x y width height*], and specifies the upper-left corner position and size of a rectangular region in pixels.

`TextLineConfidences` — Text line confidences
vector

Text line confidences, specified as a vector of probability values in the range [0,1]. Use confidence values to identify the location of misclassified text lines within the image by extracting text lines with low confidence.

Object Functions

locateText Locate text pattern

Examples

Find and Highlight Text in Image

Load an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults = ocr(businessCard);
bboxes = locateText(ocrResults,"Math",IgnoreCase=true);
Iocr = insertShape(businessCard,"FilledRectangle",bboxes);
figure
imshow(Iocr)
```



Find Text Using Regular Expressions

Load an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults   = ocr(businessCard);
bboxes = locateText(ocrResults, "www.*com","UseRegexp", true);
img    = insertShape(businessCard, "FilledRectangle", bboxes);
figure
imshow(img)
```



Version History

Introduced in R2014a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.
- The `Words` property cannot be accessed in code generation. Use the `Text` property in place of the `Words` property to access the OCR results.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Image Labeler

Objects

`ocrTrainingOptions`

Functions
ocr | insertShape | regexp | strfind | quantizeOCR | evaluateOCR | trainOCR | ocrTrainingData

locateText

Locate text pattern

Syntax

```
bboxes = locateText(ocrText,pattern)
bboxes = locateText(ocrText,pattern,Name=Value)
```

Description

`bboxes = locateText(ocrText,pattern)` returns the location and size of bounding boxes stored in the `ocrText` object. The `locateText` function returns only the locations of bounding boxes which correspond to text within an image that exactly match the input `pattern`.

`bboxes = locateText(ocrText,pattern,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `IgnoreCase=true` specifies case-insensitive text locations.

Examples

Find and Highlight Text in Image

Load an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults = ocr(businessCard);
bboxes = locateText(ocrResults,"Math",IgnoreCase=true);
Iocr = insertShape(businessCard,"FilledRectangle",bboxes);
figure
imshow(Iocr)
```



Find Text Using Regular Expressions

Load an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults   = ocr(businessCard);
bboxes = locateText(ocrResults, "www.*com","UseRegexp", true);
img    = insertShape(businessCard, "FilledRectangle", bboxes);
figure
imshow(img)
```



Input Arguments

`ocrText` — Object containing OCR results

`ocrText` object

Recognized text and metrics, returned as an `ocrText` object. The object contains the recognized text, the location of the recognized text within the input image, and the metrics indicating the confidence of the results. The confidence values, representing the probabilities, range between `0` and `1`. When you specify an *M*-by-4 `roi`, the function returns `ocrText` as an *M*-by-1 array of `ocrText` objects.

`pattern` — OCR character vector pattern

single character vector | cell array of character vectors | string scalar | string array

OCR character vector pattern, specified as a single character vector, string scalar, cell array of character vectors, or a string array. The object function returns only the locations of bounding boxes which correspond to text within an image that exactly match the input `pattern`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `IgnoreCase=true` specifies case-insensitive text locations

`UseRegexp` — Logical to treat pattern as regular expression

`false` (default) | logical scalar

Logical to treat pattern as regular expression, specified as a logical scalar. When you set this property to `true`, the object function treats the pattern as a regular expression. For more information about regular expressions, see `regexp`.

`IgnoreCase` — Case sensitivity
`false` (default) | logical scalar

Case sensitivity, specified as a logical scalar. When you set this property to `true`, the object function performs case-insensitive text location.

Output Arguments

`bboxes` — Text bounding boxes
M-by-4 matrix

Text bounding boxes, returned as an *M*-by-4 matrix. Each row of the matrix contains a four-element vector, [*x y width height*]. The [*x y*] elements correspond to the upper-left corner of the bounding box. The [*width height*] elements correspond to the size of the rectangular region in pixels. The bounding boxes enclose text found in an image using the `ocr` function. The `ocr` function stores OCR results in the `ocrText` object.

Version History

Introduced in R2014a

pointCloud

Object for storing 3-D point cloud

Description

The `pointCloud` object creates point cloud data from a set of points in 3-D coordinate system. The points generally represent the *x*,*y*, and *z* geometric coordinates for samples on a surface or of an environment. Each point can also be represented with additional information, such as the RGB color. The point cloud data is stored as an object with the properties listed in “Properties” on page 2-984. Use “Object Functions” on page 2-985 to retrieve, select, and remove desired points from the point cloud data.

Creation

Syntax

```
ptCloud = pointCloud(xyzPoints)
ptCloud = pointCloud( ___ ,Name=Value)
```

Description

`ptCloud = pointCloud(xyzPoints)` returns a point cloud object with coordinates specified by `xyzPoints`.

`ptCloud = pointCloud( ___ ,Name=Value)` sets options using one or more name-value arguments in addition to the previous syntax. For example, `Color=[1 0 0]` sets the color of the point cloud to red.

Input Arguments

`xyzPoints` — 3-D coordinate points
M-by-3 list of points | *M*-by-*N*-by-3 array for organized point cloud

3-D coordinate points, specified as an *M*-by-3 list of points or an *M*-by-*N*-by-3 array for an organized point cloud. The 3-D coordinate points specify the *x*, *y*, and *z* positions of a point in the 3-D coordinate space. The first two dimensions of an organized point cloud correspond to the scanning order from sensors such as RGBD or lidar. This argument sets the `Location` property.

Data Types: `single` | `double`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

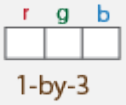
Example: `Color=[1 0 0]` sets the color of the point cloud to red.

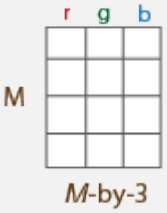
`Color` — Point cloud color

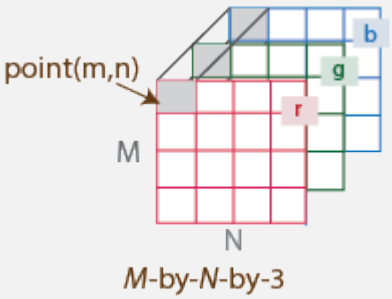
`[ ]` (default) | color string | 1-by-3 vector | *M*-by-3 matrix | *M*-by-*N*-by-3 array

Point cloud color, specified as an RGB value as one of, a color string, a 1-by-3 vector, or an *M*-by-3 or *M*-by-*N*-by-3 array. *M*-by- *N* specifies the dimensions of the point cloud. Each entry specifies the RGB color of a point in the point cloud data. Therefore, you can specify the same color for all points or a different color for each point.

- For `single` or `double` datatype, the `Color` RGB value must be specified in the range [0, 1].
- For `uint8` datatype, the `Color` RGB value must be specified in the range [0, 255].
- For `uint16` datatype, the `Color` RGB value must be specified in the range [0, 65535].
- The function stores the `Color` value as either an *M*-by-3 or *M*-by-*N*-by-3. If you specify the color as a string or a 1-by-3 vector, the function converts it to one of these dimensions.

Color	Format	Example
Specify one color for all points	string scalar or character vector	`"r"` `"red"`
	1-by-3 vector	 `[255 0 0]`

Specify a color for each point	*M*-by-3 matrix, as a list of RGB values	`255 0 0`
		`255 0 0`
		`0 255 255`
		

	M-by-*N*-by-3 array for an organized point cloud containing RGB values for each point, where *M*-by-*N* is the dimension of the point cloud.	

Data Types: `uint8` | `uint16` | `single` | `double`

`Normal` — Surface normals

`[ ]` (default) | *M*-by-3 array | *M*-by-*N*-by-3 array

Surface normals, specified as a *M*-by-3 or *M*-by-*N*-by-3 array. Use this property to specify the normal vector with respect to each point in the point cloud. Each entry in the surface normals specifies the *x*, *y*, and *z* component of a normal vector.

Coordinates	Surface Normals
M-by-3 array	*M*-by-3 array, where each row contains a corresponding normal vector.
M-by-*N*-by-3 array	*M*-by-*N*-by-3 array containing a 1-by-1-by-3 normal vector for each point.

Data Types: `single` | `double`

`Intensity` — Grayscale intensities
`[ ]` (default) | *M*-by-1 vector | *M*-by-*N* matrix

Grayscale intensities at each point, specified as a *M*-by-1 vector or *M*-by-*N* matrix. The function maps each intensity value to a color value in the current colormap.

Coordinates	Intensity
M-by-3 array	*M*-by-1 vector, where each row contains a corresponding intensity value.
M-by-*N*-by-3 array	*M*-by-*N* matrix containing intensity value for each point.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, returned as a `pointCloud` object with the properties listed in “Properties” on page 2-984.

Properties

`Location` — Position of the points in 3-D coordinate space
M-by-3 array | *M*-by-*N*-by-3 array

This property is read-only.

Position of the points in 3-D coordinate space or geographic coordinate space, represented as an *M*-by-3 or *M*-by-*N*-by-3 array. Each entry specifies the coordinates of a point, which can be either:

The `xyzPoints` input argument sets this property. To reduce memory use, you can store point locations as a `single` data type.

- For unorganized point clouds, `Location` must be specified as an *M*-by-3 array, where *M* is the total number of points, and the array provides the coordinates for each point.
- For organized point clouds, `Location` must be specified as an *M*-by-*N*-by-3 array, where *M***N* is the total number of points, and the array provides the coordinates for each point. Points obtained from a projective camera, such as Kinect or a lidar sensor, are stored as an organized point cloud. An organized point cloud is laid out as a 2-D array of points that resemble an image-like structure.

Data Types: `single` | `double`

`Count` — Number of points
positive integer

This property is read-only.

Number of points in the point cloud, represented as a positive integer.

`XLimits` — Range of *x* coordinates
2-element vector

This property is read-only.

Range of coordinates along *x*-axis, represented as a 2-element vector.

`YLimits` — Range of *y* coordinates
2-element vector

This property is read-only.

Range of coordinates along *y*-axis, represented as a 2-element vector.

`ZLimits` — Range of *z* coordinates
2-element vector

This property is read-only.

Range of coordinates along *z*-axis, represented as a 2-element vector.

Object Functions

findNearestNeighbors	Find nearest neighbors of query points in point cloud
findNeighborsInRadius	Find neighbors within radius for query points in point cloud
findPointsInROI	Find points within a region of interest in the point cloud
findPointsInCylinder	Find points within a cylindrical region in a point cloud
removeInvalidPoints	Remove invalid points from point cloud
select	Select points in point cloud
copy	Copy array of point cloud objects

Examples

Create a Point Cloud Object and Modify Properties

Read the 3-D coordinate points into the workspace.

```
load("xyzPoints");
```

Create a point cloud object from the input point coordinates.

```
ptCloud = pointCloud(xyzPoints);
```

Inspect the properties of the point cloud object.

```
ptCloud
```

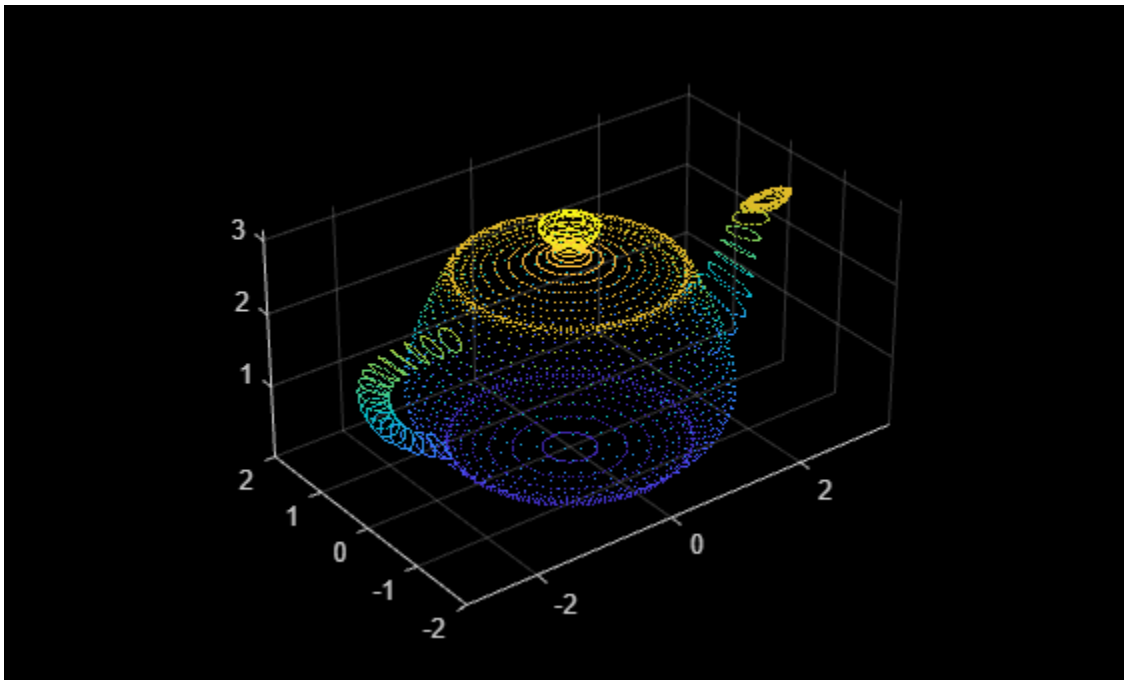
```
ptCloud =
  pointCloud with properties:

     Location: [5184×3 single]
        Count: 5184
      XLimits: [-3 3.4338]
```

```
      YLimits: [-2 2]
      ZLimits: [0.0016 3.1437]
        Color: []
       Normal: []
    Intensity: []
```

Display the point cloud by using `pcshow`.

```
pcshow(ptCloud)
```



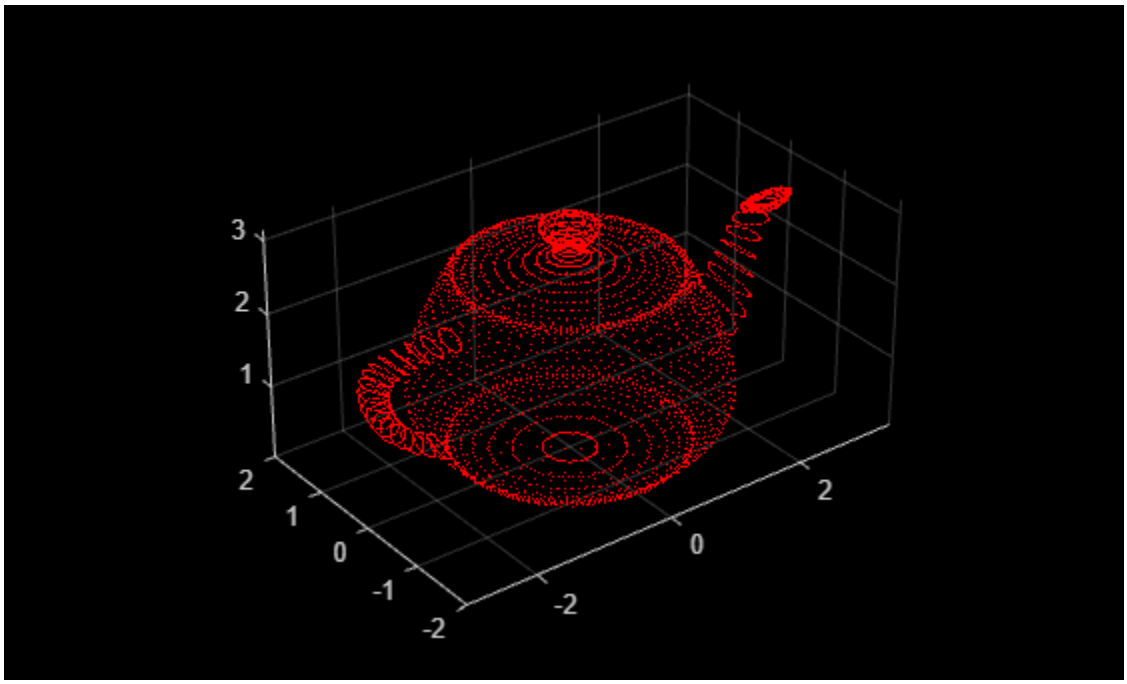
Modify Color of Point Cloud Data

Create an RGB color array of size same as the size of the point cloud data. Set the point colors to Red.

```
cmatrix = ones(size(ptCloud.Location)).*[1 0 0];
```

Create the point cloud object with the color property set to the RGB color array.

```
ptCloud = pointCloud(xyzPoints,Color=cmatrix);
pcshow(ptCloud)
```



Add Surface Normals to Point Cloud Data

Compute surface normals corresponding to the point cloud data using `pcnormals`.

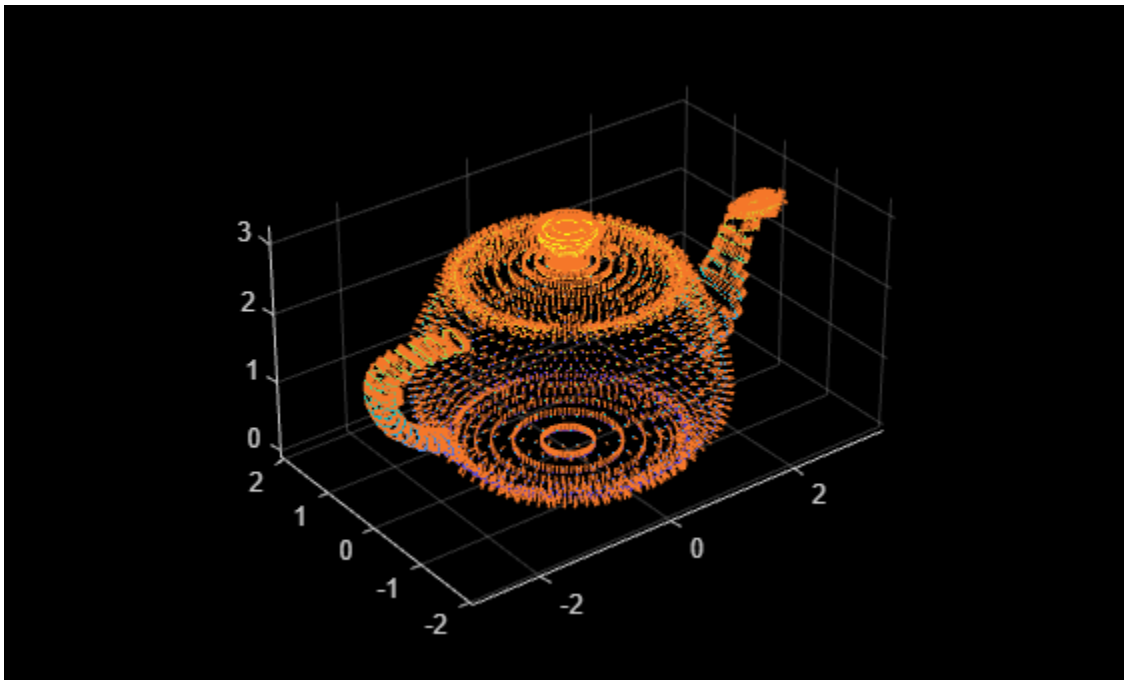
```
normals = pcnormals(ptCloud);
```

Create point cloud object from input point coordinates. Add the computed surface normals to point cloud object.

```
ptCloud = pointCloud(xyzPoints,Normal=normals);
```

Display the point cloud and plot the surface normals.

```
pcshow(ptCloud)
x = ptCloud.Location(:,1);
y = ptCloud.Location(:,2);
z = ptCloud.Location(:,3);
u = normals(:,1);
v = normals(:,2);
w = normals(:,3);
hold on
quiver3(x,y,z,u,v,w);
hold off
```



Tips

The `pointCloud` object is a `handle` object. If you want to create a separate copy of a point cloud, you can use the MATLAB `copy` method.
`ptCloudB = copy(ptCloudA)`

If you want to preserve a single copy of a point cloud, which can be modified by point cloud functions, use the same point cloud variable name for the input and output.
`ptCloud = pcFunction(ptCloud)`

Version History

Introduced in R2015a

R2022b: Color support `uint16` datatype

`uint16` datatype support for the `Color` property

R2026a: Store `Color` property using `single` or `double` data type

The object now supports storing the `Color` property using the `single` or `double` data type.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- GPU code generation for variable input sizes is not optimized. Consider using constant size inputs for an optimized code generation.
- GPU code generation supports the `Color`, `Normal`, and `Intensity` name-value pairs.
- GPU code generation supports the `findNearestNeighbors`, `findNeighborsInRadius`, `findPointsInROI`, `removeInvalidPoints`, and `select` methods.
- For very large inputs, the memory requirements of the algorithm may exceed the GPU device limits. In such cases, consider reducing the input size to proceed with code generation.

See Also

Functions

`pccat` | `pcalign` | `pcshow` | `pcwrite` | `pcread` | `pcmerge` | `pcdenoise` | `pcnormals` | `pcfromkinect` | `reconstructScene` | `pcdownsample` | `pctransform` | `pcregistericp`

Objects

`pcplayer`

Topics

“3-D Point Cloud Registration and Stitching”
“Coordinate Systems”

findNearestNeighbors

Find nearest neighbors of query points in point cloud

Syntax

```
[indices,dists] = findNearestNeighbors(ptCloud,points,K)
[indices,dists] = findNearestNeighbors(ptCloud,points,K,camMatrix)
[indices,dists] = findNearestNeighbors( ___ ,Name,Value)
```

Description

`[indices,dists] = findNearestNeighbors(ptCloud,points,K)` returns the `indices` for the K-nearest neighbors of one or more query points in the input point cloud. `ptCloud` can be an unorganized or organized point cloud. The K-nearest neighbors of the query points are computed by using the Kd-tree based search algorithm.

`[indices,dists] = findNearestNeighbors(ptCloud,points,K,camMatrix)` returns the K-nearest neighbors of one or more query points in the input point cloud. The input point cloud is an organized point cloud generated by a depth camera. The K-nearest neighbors of the query points are determined using fast approximate K-nearest neighbor search algorithm.

The function uses the camera projection matrix `camMatrix` to know the relationship between adjacent points and hence, speeds up the nearest neighbor search. However, the results have lower accuracy as compared to the Kd-tree based approach.

Note

- This syntax only supports organized point cloud data produced by RGB-D sensors.
- You can use `estimateCameraMatrix` to estimate camera projection matrix for the given point cloud data.

`[indices,dists] = findNearestNeighbors( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the preceding syntaxes.

Examples

Find K-Nearest Neighbors in Point Cloud

Load a set of 3-D coordinate points into the workspace.

```
load("xyzPoints.mat");
```

Create a point cloud object.

```
ptCloud = pointCloud(xyzPoints);
```

Specify two query points and the number of nearest neighbors to identify.

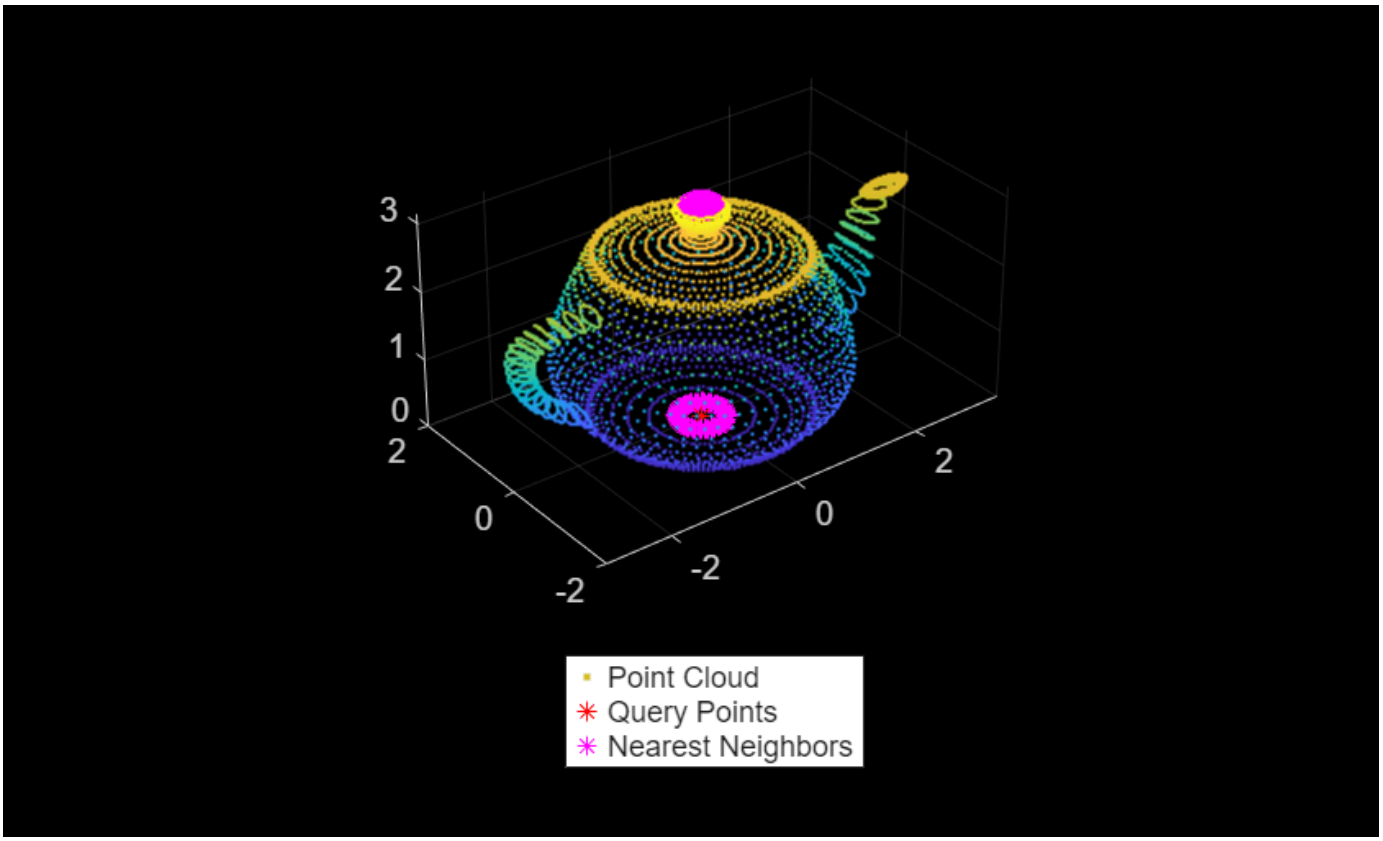
```
points = [0 0 0; 0 0 3];
K = 70;
```

Get the indices and the distances of the K nearest neighbors for each query point.

```
[indices,dists] = findNearestNeighbors(ptCloud,points,K);
```

Display the point cloud. Plot the query points and their nearest neighbors.

```
figure
pcshow(ptCloud)
hold on
plot3(points(:,1),points(:,1),points(:,3),"*r")
plot3(ptCloud.Location(indices,1), ...
    ptCloud.Location(indices,2), ...
    ptCloud.Location(indices,3),"*m")
legend("Point Cloud","Query Points", ...
    "Nearest Neighbors", ...
    Location="southoutside", ...
    Color=[1 1 1])
hold off
```



Find K-Nearest Neighbors in Organized Point Cloud

Find the K-nearest neighbors of a query point in the organized point cloud data by using the camera projection matrix. Compute the camera projection matrix from sampled point cloud data points and their corresponding image point coordinates.

Load an organized point cloud data into the workspace. The point cloud is generated by using the Kinect depth sensor.

```
ld = load('object3d.mat');
ptCloud = ld.ptCloud;
```

Specify the step size for sampling the point cloud data.

```
stepSize = 100;
```

Sample the input point cloud and store the sampled 3-D point coordinates as a point cloud object.

```
indices = 1:stepSize:ptCloud.Count;
tempPtCloud = select(ptCloud,indices);
```

Remove invalid points from the sampled point cloud.

```
[tempPtCloud,validIndices] = removeInvalidPoints(tempPtCloud);
```

Define the 3-D world point coordinates of input point cloud.

```
worldPoints = tempPtCloud.Location;
```

Find the 2-D image coordinates corresponding to the 3-D point coordinates of input point cloud.

```
[Y,X] = ind2sub([size(ptCloud.Location,1),size(ptCloud.Location,2)],indices);
imagePoints = [X(validIndices)' Y(validIndices)'];
```

Estimate camera projection matrix from the image and the world point coordinates.

```
camMatrix = estimateCameraMatrix(imagePoints,worldPoints);
```

Specify a query point and the number of nearest neighbors to be identified.

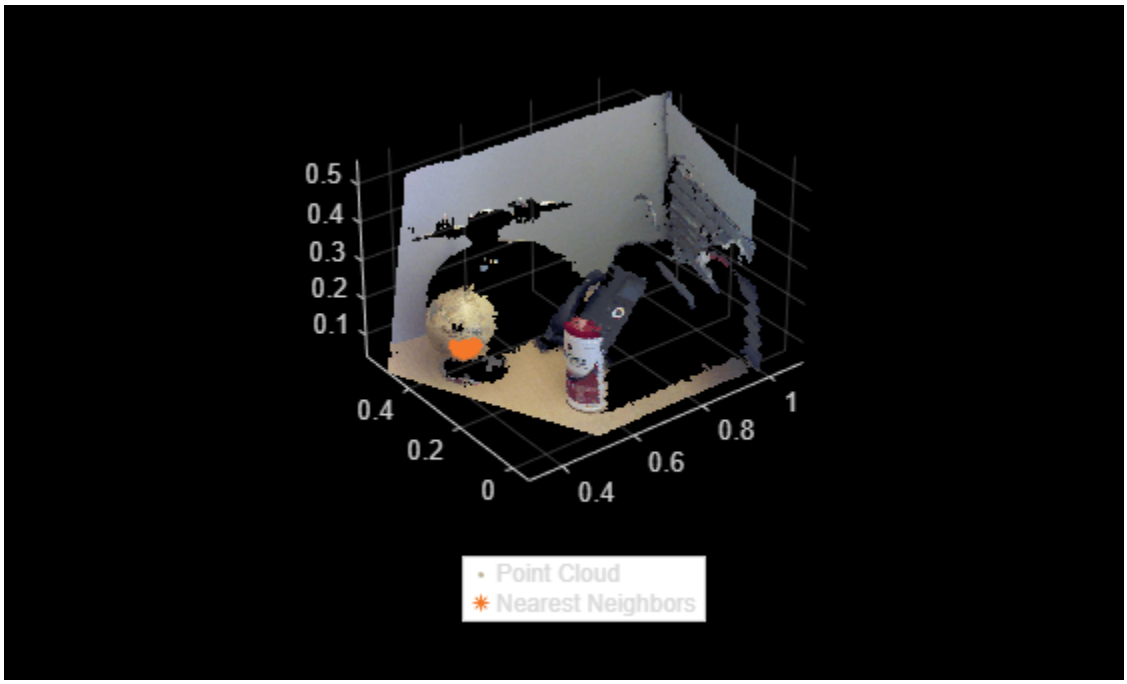
```
point = [0.4 0.3 0.2];
K = 1000;
```

Find the indices and distances of K nearest neighboring points by using the camera projection matrix. Use the point cloud method `select` to get the point cloud data of nearest neighbors.

```
[indices,dists] = findNearestNeighbors(ptCloud,point,K,camMatrix);
ptCloudB = select(ptCloud,indices);
```

Display the point cloud and the nearest neighbors of the query point.

```
figure
pcshow(ptCloud)
hold on
plot3(ptCloudB.Location(:,1),ptCloudB.Location(:,2),ptCloudB.Location(:,3),'*')
legend('Point Cloud','Nearest Neighbors','Location','southoutside','Color',[1 1 1])
hold off
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

Note The function supports organized point cloud data generated only from RGB-D sensors.

`points` — Query points

1-by-3 vector | *N*-by-3 matrix

Query points, specified as a 1-by-3 vector or *N*-by-3 matrix.

- To find the nearest neighbors of a single query point, specify its coordinates as a 1-by-3 vector of the form `[x y z]`.
- To find the nearest neighbors of multiple query points, specify them as an *N*-by-3 matrix. *N* is the number of query points. Each row of the matrix is of the form `[x y z]`, representing the coordinates of a query point.

`K` — Number of nearest neighbors

positive integer

Number of nearest neighbors, specified as a positive integer.

`camMatrix` — Camera projection matrix

4-by-3 matrix

Camera projection matrix, specified as a 4-by-3 matrix that maps 3-D world points to 2-D image points. You can compute the `camMatrix` by using the `estimateCameraMatrix` function.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `findNearestNeighbors(ptCloud,point,k,'Sort',true)`

`Sort` — Sort indices

`false` (default) | `true`

Sort indices, specified as a comma-separated pair of `'Sort'` and a logical scalar. When you set `Sort` to `true`, the returned indices are sorted in the ascending order based on the distance from a query point. To turn off sorting, set `Sort` to `false`.

`MaxLeafChecks` — Number of leaf nodes to check

`Inf` (default) | integer

Number of leaf nodes to check, specified as a comma-separated pair consisting of `'MaxLeafChecks'` and an integer. When you set this value to `Inf`, the entire tree is searched. When the entire tree is searched, it produces exact search results. Increasing the number of leaf nodes to check increases accuracy, but reduces efficiency.

Note The name-value argument `'MaxLeafChecks'` is valid only with Kd-tree based search method.

Output Arguments

`indices` — Indices of nearest neighbor points

K-by-1 vector | *K*-by-*N* matrix

Indices of nearest neighbor points, returned as a *K*-by-1 vector or *K*-by-*N* matrix. *K* is the number of nearest neighbors.

- For a single query point, the function returns linear indices of the *K* nearest neighbors as a *K*-by-1 vector.
- For multiple query points, the function returns a *K*-by-*N* matrix. *K* is the number of nearest neighbors and *N* is the number of query points. Each column of the matrix contains the linear indices of the *K* nearest neighbors of the query point specified by the corresponding row of the `points` input argument.

`dists` — Distances between query points and their nearest neighbors

K-by-1 vector | *K*-by-*N* matrix

Distances between query points and their nearest neighbors, returned as a *K*-by-1 vector or *K*-by-*N* matrix. *K* is the number of nearest neighbors.

- For a single query point, the function returns a *K*-by-1 vector. The vector contains the Euclidean distances between the query point and its *K* nearest neighbors.

- For multiple query points, the function returns a *K*-by-*N* matrix. *K* is the number of nearest neighbors and *N* is the number of query points. Each column of the matrix contains the Euclidean distances to the *K* nearest neighbors of a query point. This query point is specified by the corresponding row in the `points` input argument.

Version History

Introduced in R2015a

R2025a: Find nearest neighbors of multiple query points in point cloud

Starting in R2025a, the `points` input argument value supports an *N*-by-3 matrix. Using an *N*-by-3 matrix enables you to simultaneously find the nearest neighbors of multiple query points in the input point cloud. To find the nearest neighbors of a single query point, specify the `points` argument as a 1-by-3 vector.

References

[1] Muja, M. and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". *In VISAPP International Conference on Computer Vision Theory and Applications*. 2009. pp. 331–340.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For code generation in non-host platforms, the value for `'MaxLeafChecks'` must be set to the default value `Inf`. If you specify values other than `Inf`, the function generates a warning and automatically assigns the default value for `'MaxLeafChecks'`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- For GPU code generation, the `camMatrix` input argument is ignored.
- For GPU code generation, the `'MaxLeafChecks'` name-value pair option is ignored.

See Also

`pointCloud` | `findNeighborsInRadius` | `estimateCameraMatrix` | `cameraMatrix`

findNeighborsInRadius

Find neighbors within radius for query points in point cloud

Syntax

```
[indices,dists,numNeighbors] = findNeighborsInRadius(ptCloud,points,radius)
[indices,dists,numNeighbors] = findNeighborsInRadius(ptCloud,points,radius,
camMatrix)
[indices,dists,numNeighbors] = findNeighborsInRadius( ___ ,Name,Value)
```

Description

`[indices,dists,numNeighbors] = findNeighborsInRadius(ptCloud,points,radius)` returns the `indices` of neighbors within a radius of one or more query points in the input point cloud. `ptCloud` can be an unorganized or organized point cloud. The neighbors within a radius of the query points are computed by using the Kd-tree based search algorithm.

`[indices,dists,numNeighbors] = findNeighborsInRadius(ptCloud,points,radius, camMatrix)` returns the neighbors within a radius of one or more query points in the input point cloud. The input point cloud is an organized point cloud generated by a depth camera. The neighbors within a radius of the query points are determined using fast approximate neighbor search algorithm.

The function uses the camera projection matrix `camMatrix` to know the relationship between adjacent points and hence, speeds up the search. However, the results have lower accuracy as compared to the Kd-tree based approach.

Note

- This syntax only supports organized point cloud data produced by RGB-D sensors.
- You can use `estimateCameraMatrix` to estimate camera projection matrix for the given point cloud data.

`[indices,dists,numNeighbors] = findNeighborsInRadius( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the preceding syntaxes.

Examples

Find Radial Neighbors in Point Cloud

Load a set of 3-D coordinate points into the workspace.

```
load("xyzPoints.mat")
```

Create a point cloud object.

```
ptCloud = pointCloud(xyzPoints);
```

Specify two query points and a search radius.

```
points = [0 0 0; 0 0 3];
radius = 0.5;
```

For each query point, find the neighbors within the specified radius.

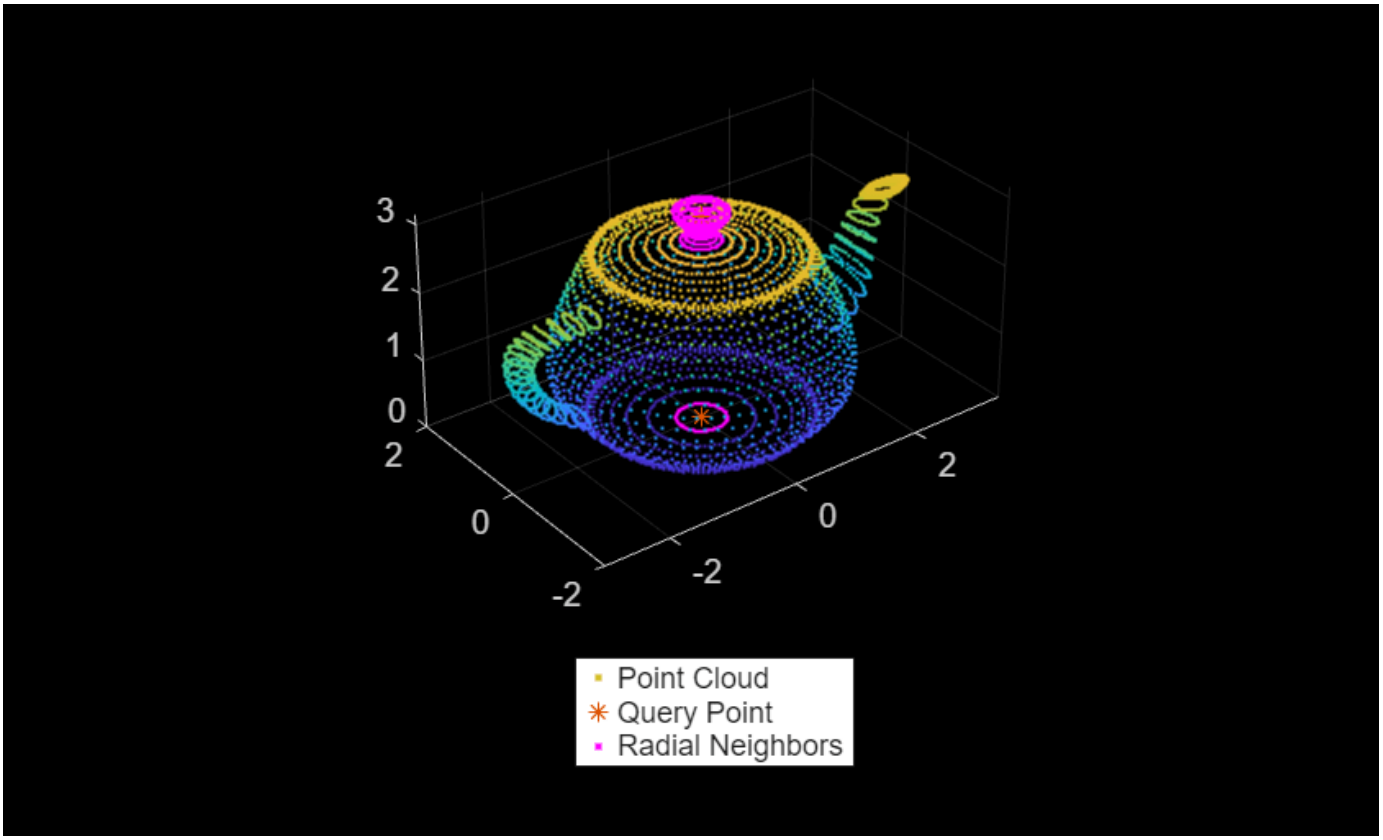
```
[indices,dists,numNeighbors] = findNeighborsInRadius(ptCloud,points,radius);
```

Convert the cell array of indices to an ordinary array. Then, select radial neighbors using these indices from the input point cloud.

```
indices = cell2mat(indices);
ptCloudB = select(ptCloud,indices);
```

Display the point cloud and overlay the query points and radial neighbors.

```
figure
pcshow(ptCloud)
hold on
plot3(points(:,1),points(:,2),points(:,3),"*")
pcshow(ptCloudB.Location,"m")
legend("Point Cloud", ...
    "Query Point", ...
    "Radial Neighbors", ...
    Location="southoutside", ...
    Color=[1 1 1])
hold off
```



Find Radial Neighbors in Organized Point Cloud

Find radial neighbors of a query point in the organized point cloud data by using the camera projection matrix. Compute the camera projection matrix from sampled point cloud data points and their corresponding image point coordinates.

Load an organized point cloud data into the workspace. The point cloud is generated by using the Kinect depth sensor.

```
ld = load('object3d.mat');
ptCloud = ld.ptCloud;
```

Specify the step size for sampling the point cloud data.

```
stepSize = 100;
```

Sample the input point cloud and store the sampled 3-D point coordinates as a point cloud object.

```
indices = 1:stepSize:ptCloud.Count;
tempPtCloud = select(ptCloud,indices);
```

Remove invalid points from the sampled point cloud.

```
[tempPtCloud,validIndices] = removeInvalidPoints(tempPtCloud);
```

Define the 3-D world point coordinates of input point cloud.

```
worldPoints = tempPtCloud.Location;
```

Find the 2-D image coordinates corresponding to the 3-D point coordinates of input point cloud.

```
[Y,X] = ind2sub([size(ptCloud.Location,1),size(ptCloud.Location,2)],indices);
imagePoints = [X(validIndices)' Y(validIndices)'];
```

Estimate camera projection matrix from the image and the world point coordinates.

```
camMatrix = estimateCameraMatrix(imagePoints,worldPoints);
```

Specify a query point and the radius within which the neighbors are to be identified.

```
point = [0.4 0.3 0.2];
radius = 0.05;
```

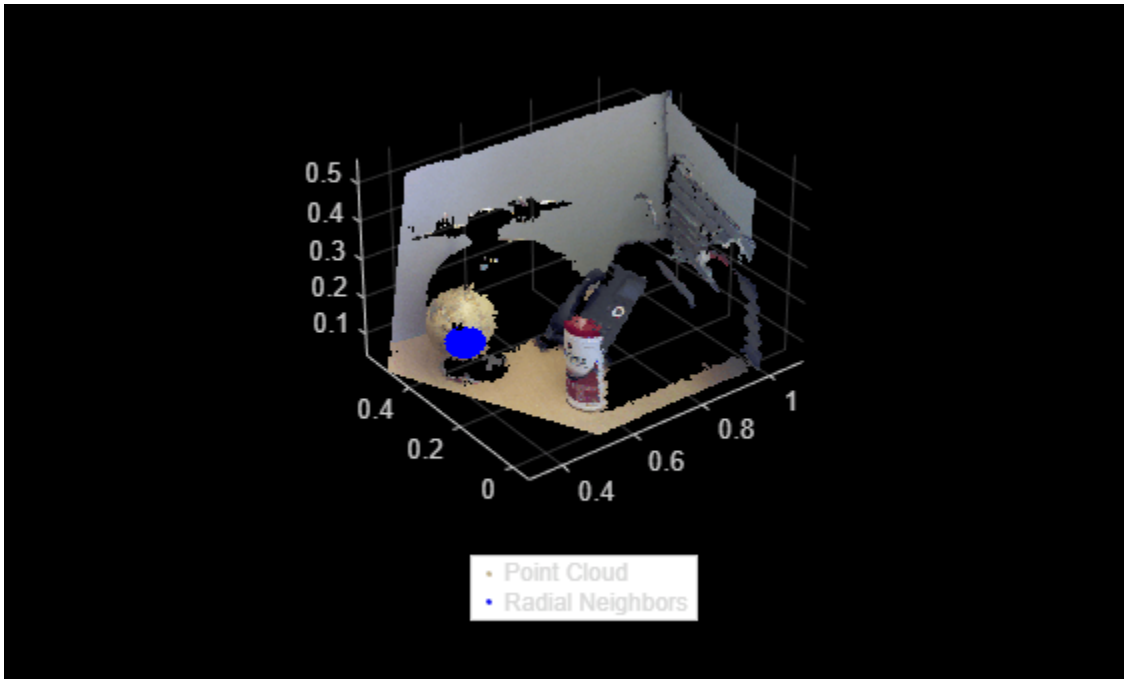
Get the indices and the distances of radial neighbors. Use the point cloud method `select` to get the point cloud data of neighboring points.

```
[indices,dists] = findNeighborsInRadius(ptCloud,point,radius,camMatrix);
ptCloudB = select(ptCloud,indices);
```

Display the point cloud and the radial neighbors found around a query point.

```
figure
pcshow(ptCloud);
hold on;
pcshow(ptCloudB.Location, 'b');
```

```
legend('Point Cloud','Radial Neighbors','Location','southoutside','Color',[1 1 1] )
hold off;
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`points` — Query points

1-by-3 vector | *N*-by-3 matrix

Query points, specified as a 1-by-3 vector or *N*-by-3 matrix.

- To find the neighbors within a radius for a single query point, specify its coordinates as a 1-by-3 vector of the form `[x y z]`.
- To find the neighbors within a radius for multiple query points, specify them as an *N*-by-3 matrix. *N* is the number of query points. Each row of the matrix is of the form `[x y z]`, representing the coordinates of a query point.

`radius` — Search radius

scalar

Search radius, specified as a scalar. The function finds the neighbors within the specified `radius` around a query point in the input point cloud.

`camMatrix` — Camera projection matrix

4-by-3 matrix

Camera projection matrix, specified as a 4-by-3 matrix that maps 3-D world points to 2-D image points. You can find `camMatrix` by using the `estimateCameraMatrix` function.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `findNeighborsInRadius(ptCloud,point,radius,'Sort',true)`

`Sort` — Sort indices

`false` (default) | `true`

Sort indices, specified as a comma-separated pair of `'Sort'` and a logical scalar. When you set `Sort` to `true`, the returned indices are sorted in the ascending order based on the distance from a query point. To turn off sorting, set `Sort` to `false`.

`MaxLeafChecks` — Number of leaf nodes

`Inf` (default) | integer

Number of leaf nodes, specified as a comma-separated pair consisting of `'MaxLeafChecks'` and an integer. When you set this value to `Inf`, the entire tree is searched. When the entire tree is searched, it produces exact search results. Increasing the number of leaf nodes to check increases accuracy, but reduces efficiency.

Output Arguments

`indices` — Indices of stored points

M-by-1 vector | *N*-by-1 cell array

Indices of stored points, returned as an *M*-by-1 vector or *N*-by-1 cell array.

- For a single query point, the function returns linear indices of *M* radial neighbors as an *M*-by-1 vector.
- For multiple query points, the function returns an *N*-by-1 cell array. *N* is the number of query points. Each cell contains an *M*-by-1 column vector of linear indices for the radial neighbors of a query point. *M* is the number of radial neighbors for that query point.

`dists` — Distances between query points and their radial neighbors

M-by-1 vector | *N*-by-1 cell array

Distances between query points and their radial neighbors, returned as an *M*-by-1 vector or *N*-by-1 cell array.

- For a single query point, the function returns an *M*-by-1 vector. The vector contains the Euclidean distances between the query point and its *M* radial neighbors.
- For multiple query points, the function returns an *N*-by-1 cell array. *N* is the number of query points. Each cell contains an *M*-by-1 column vector of the Euclidean distances between the query point and its *M* radial neighbors.

`numNeighbors` — Number of identified radial neighbors
nonnegative integer | *N*-by-1 vector

Number of identified radial neighbors, returned as a nonnegative integer for a single query point or an *N*-by-1 vector of nonnegative integers for *N* query points.

Version History

Introduced in R2015a

R2025a: Find neighbors within radius for multiple query points in point cloud

Starting in R2025a, the `points` input argument value supports an *N*-by-3 matrix. Using an *N*-by-3 matrix enables you to simultaneously find the radial neighbors of multiple query points in the input point cloud. To find the radial neighbors of a single query point, specify the `points` argument as a 1-by-3 vector.

The function now also returns a `numNeighbors` argument, which indicates the number of neighbors found within the specified radius of each query point.

References

[1] Muja, M. and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". *In VISAPP International Conference on Computer Vision Theory and Applications*. 2009. pp. 331–340.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For code generation in non-host platforms, the value for `'MaxLeafChecks'` must be set to the default value `Inf`. If you specify values other than `Inf`, the function generates a warning and automatically assigns the default value for `'MaxLeafChecks'`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- For GPU code generation, the `camMatrix` input argument is ignored.
- For GPU code generation, the `'MaxLeafChecks'` name-value pair option is ignored.

See Also

`pointCloud` | `findNearestNeighbors` | `estimateCameraMatrix` | `cameraMatrix`

findPointsInROI

Find points within a region of interest in the point cloud

Syntax

```
indices = findPointsInROI(ptCloud,roi)
indices = findPointsInROI(ptCloud,roi,camMatrix)
```

Description

`indices = findPointsInROI(ptCloud,roi)` returns the points within a region of interest (ROI) in the input point cloud. The points within the specified ROI are obtained using a Kd-tree based search algorithm.

`indices = findPointsInROI(ptCloud,roi,camMatrix)` returns the points within the ROI area of the input point cloud. The input point cloud is an organized point cloud generated by a depth camera. The points within the specified ROI are obtained using fast approximate neighbor search algorithm.

The function uses the camera projection matrix `camMatrix` to know the relationship between adjacent points and hence, speeds up the search. However, the results have lower accuracy as compared to the Kd-tree based approach.

Note

- This syntax only supports organized point cloud data produced by RGB-D sensors.
- You can use `estimateCameraMatrix` to estimate camera projection matrix for the given point cloud data.

Examples

Find Points Within a Cuboid ROI in Point Cloud

Read a point cloud data into the workspace.

```
ptCloud = pcread('teapot.ply');
```

Define a cuboid ROI within the range of the *x*, *y* and *z* coordinates of the input point cloud.

```
roi = [-2 2 -2 2 2.4 3.5];
```

Find the indices of the points that lie within the cuboid ROI.

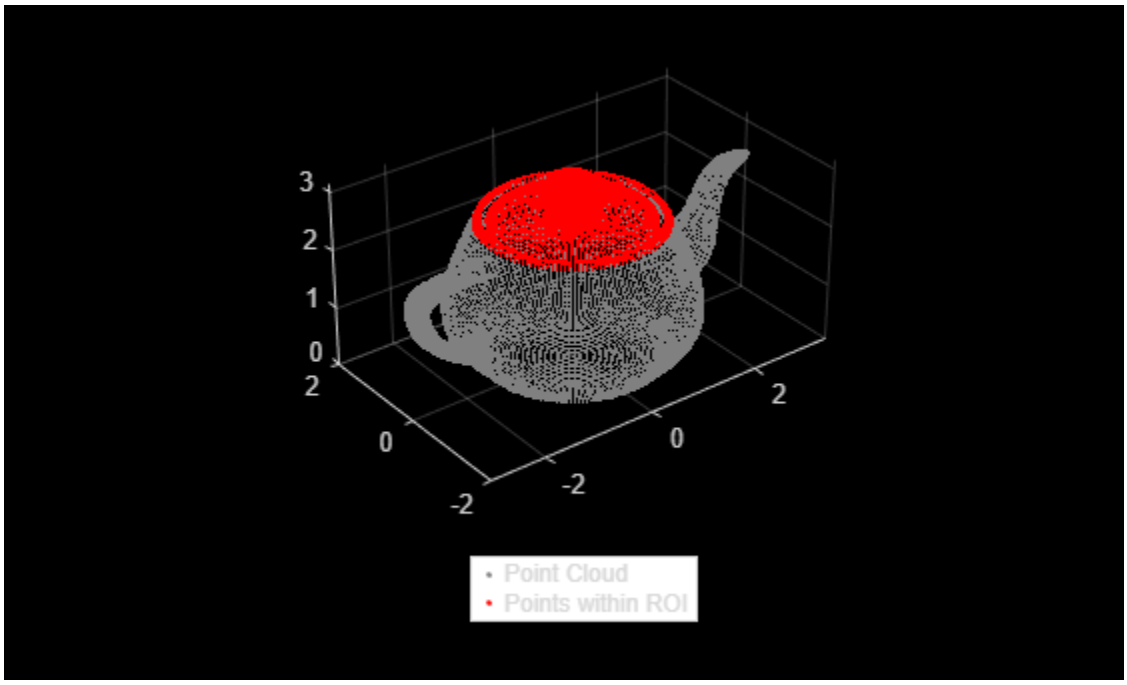
```
indices = findPointsInROI(ptCloud,roi);
```

Select the points that lie within the cuboid ROI and store as a point cloud object.

```
ptCloudB = select(ptCloud,indices);
```

Display the input point cloud and the point cloud within the specified ROI.

```
figure
pcshow(ptCloud.Location,[0.5 0.5 0.5])
hold on
pcshow(ptCloudB.Location,'r');
legend('Point Cloud','Points within ROI','Location','southoutside','Color',[1 1 1])
hold off
```



Find Points Within a Cuboid ROI in Organized Point Cloud

Find points within a cuboid ROI in the organized point cloud data by using the camera projection matrix. Compute the camera projection matrix from sampled point cloud data points and their corresponding image point coordinates.

Load an organized point cloud data into the workspace. The point cloud is generated by using the Kinect depth sensor.

```
ld = load('object3d.mat');
ptCloud = ld.ptCloud;
```

Specify the step size for sampling the point cloud data.

```
stepSize = 100;
```

Sample the input point cloud and store the sampled 3-D point coordinates as a point cloud object.

```
indices = 1:stepSize:ptCloud.Count;
tempPtCloud = select(ptCloud,indices);
```

Remove invalid points from the sampled point cloud.

```
[tempPtCloud,validIndices] = removeInvalidPoints(tempPtCloud);
```

Get the 3-D world point coordinates from input point cloud.

```
worldPoints = tempPtCloud.Location;
```

Find the 2-D image coordinates corresponding to the 3-D point coordinates of input point cloud.

```
[Y,X] = ind2sub([size(ptCloud.Location,1),size(ptCloud.Location,2)],indices);
imagePoints = [X(validIndices)' Y(validIndices)'];
```

Estimate camera projection matrix from the image and the world point coordinates.

```
camMatrix = estimateCameraMatrix(imagePoints,worldPoints);
```

Specify a cuboid ROI within the range of the *x*, *y* and *z* coordinates of the input point cloud.

```
roi = [0.3 0.7 0 0.4 0.1 0.3];
```

Find the indices of the point cloud data that lie within the cuboid ROI.

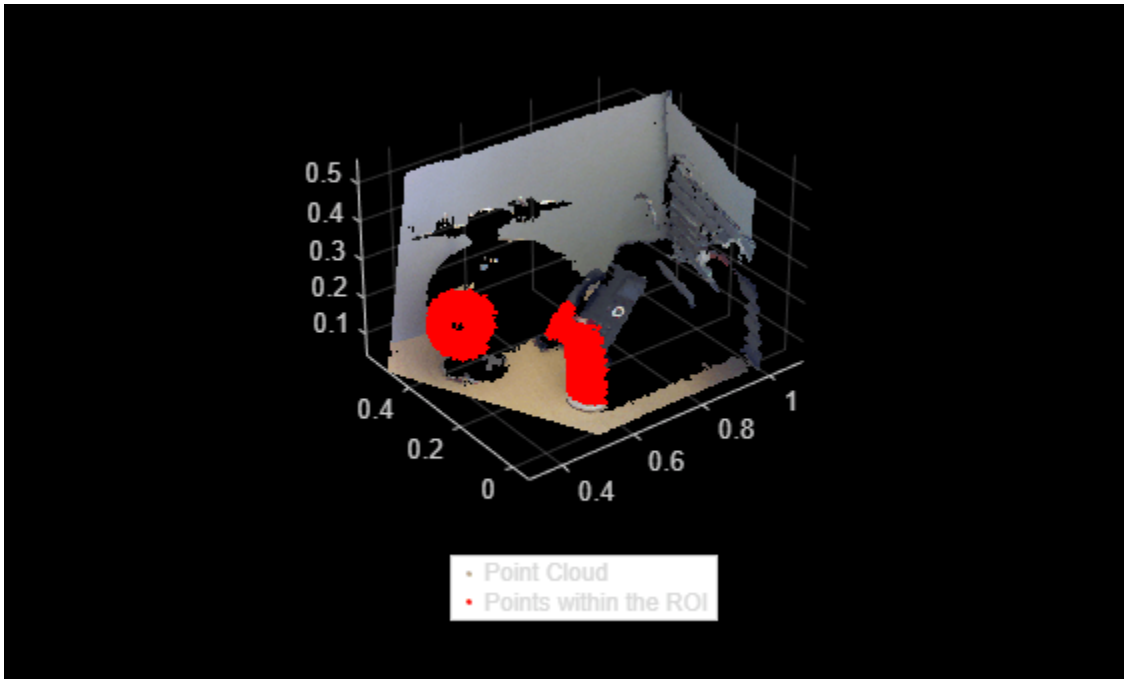
```
indices = findPointsInROI(ptCloud,roi,camMatrix);
```

Use the point cloud method `select` to get the point cloud data of points within the ROI.

```
ptCloudB = select(ptCloud,indices);
```

Display the input point cloud and the points within the cuboid ROI.

```
figure
pcshow(ptCloud)
hold on
pcshow(ptCloudB.Location,'r');
legend('Point Cloud','Points within the ROI','Location','southoutside','Color',[1 1 1])
hold off
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`roi` — Region of interest

six-element vector

Region of interest, specified as a six-element vector of form [*xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*], where:

- *xmin* and *xmax* are the minimum and the maximum limits along the *x*-axis respectively.
- *ymin* and *ymax* are the minimum and the maximum limits along the *y*-axis respectively.
- *zmin* and *zmax* are the minimum and the maximum limits along the *z*-axis respectively.

`camMatrix` — Camera projection matrix

4-by-3 matrix

Camera projection matrix, specified as a 4-by-3 matrix that maps 3-D world points to 2-D image points. You can find `camMatrix` by using the `estimateCameraMatrix` function.

Output Arguments

`indices` — Indices of stored points

column vector

Indices of stored points, returned as a column vector. The vector contains the linear indices of the ROI points stored in the point cloud.

Version History

Introduced in R2015a

References

[1] Muja, M. and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". *In VISAPP International Conference on Computer Vision Theory and Applications*. 2009. pp. 331–340.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For GPU code generation, the `camMatrix` input argument is ignored.

See Also

Functions
`estimateCameraMatrix` | `findPointsInModel`

Objects
`pointCloud` | `cameraMatrix`

findPointsInCylinder

Find points within a cylindrical region in a point cloud

Syntax

```
indices = findPointsInCylinder(ptCloud,radius)
indices = findPointsInCylinder( ___ ,Name=Value)
```

Description

`indices = findPointsInCylinder(ptCloud,radius)` finds the points within a cylindrical neighborhood of the specified `radius` in a point cloud, ptCloud.

`indices = findPointsInCylinder( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `Height=10`, sets the height of the cylinder to 10.

Examples

Find Points in Hollow Cylinder

Load a point cloud from a saved MAT file.

```
load("drivingLidarPoints.mat","ptCloud")
```

Specify a hollow cylindrical neighborhood with an inner radius of 2 meters and an outer radius of 15 meters, and find the points in the point cloud within the neighborhood.

```
minRadius = 2;
maxRadius = 15;

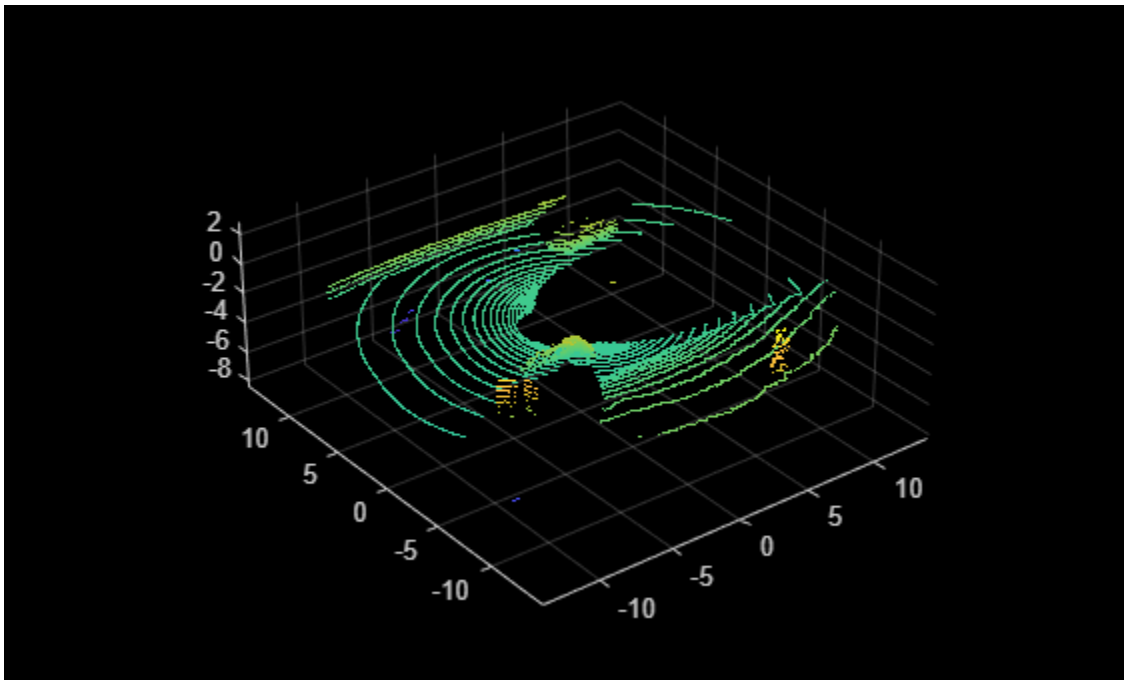
indices = findPointsInCylinder(ptCloud,[minRadius maxRadius]);
```

Select the cylindrical neighborhood in the point cloud, preserving the organized format.

```
ptCloudCylinder = select(ptCloud,indices,OutputSize="full");
```

Visualize the selected points.

```
figure
pcshow(ptCloudCylinder)
```



Apply Solid Cylindrical Filter Centered Around Point of Interest

Load a point cloud from a saved MAT file.

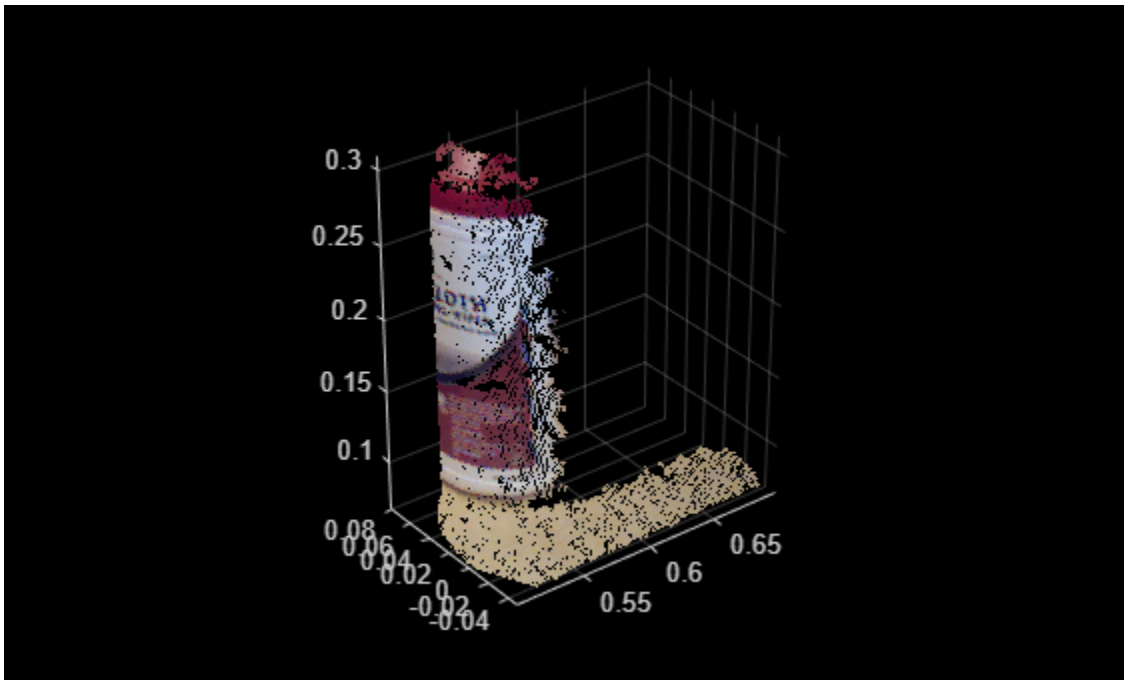
```
load("object3d.mat","ptCloud")
maxRadius = 0.1;
cylinderHeight = 0.5;
```

Offset the center of the cylinder 0.6 units in the *X*-direction and 0.2 units in the *Z*-direction. Then, find and select the points of the point cloud within the cylinder.

```
shiftedCenter = [0.6 0 0.2];
indices = findPointsInCylinder(ptCloud,maxRadius,Height=cylinderHeight,Center=shiftedCenter,Verti
ptCloudCylinder = select(ptCloud,indices);
```

Visualize the selected points.

```
figure
pcshow(ptCloudCylinder)
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`radius` — Cylinder radius

scalar | 2-element vector

Cylinder radius, specified as a scalar or a 2-element vector of the form [*minRadius maxRadius*] that specifies the inner and outer radii of the cylindrical neighborhood. The radius is along the Z-axis of the point cloud and with `[0 0 0]` as its origin. The function finds the neighbors within the specified `radius` along the defined `VerticalAxis` with its origin at the `Center` of the cylinder.

Name-Value Pair Arguments

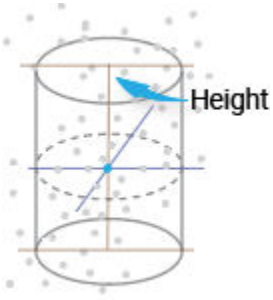
Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Height=10` sets the height of the cylinder to 10.

`Height` — Total height of cylinder

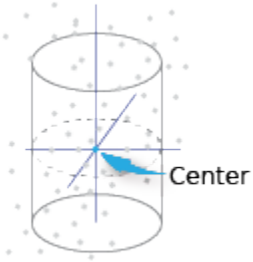
`Inf` (default) | positive scalar

Total height of cylinder, specified as a positive scalar. The height is measured along the cylinder's vertical axis, `VerticalAxis`. The cylinder is centered at its midpoint, the `Center`, with half of the height extending in one direction along the vertical axis and the other half extending in the opposite direction.



`Center` — Center of cylinder
`[0 0 0]` (default) | 3-element vector

Center of the cylinder, specified as a 3-element vector of the form [*x y z*] in Cartesian coordinates. The center point is halfway between the top and bottom of the cylinder, and the axis of the cylinder passes through it.



`VerticalAxis` — Vertical axis of cylinder
`"Z"` (default) | `"X"` | `"Y"`

Vertical axis of the cylinder, specified as `"X"`, `"Y"`, or `"Z"`.

Output Arguments

`indices` — Logical indices of points within cylindrical neighborhood
column vector | matrix

Logical indices of the points within a cylindrical neighborhood, returned as a column vector for unorganized point clouds or a matrix for organized point clouds. For organized point clouds. `indices` is the same size as the input point cloud, and each element with a value of `1` (true) indicates that the corresponding index of the input point cloud references a point inside the specified cylindrical neighborhood.

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`findPointsInROI` | `pcfitcylinder`

Objects
`pointCloud`

removeInvalidPoints

Remove invalid points from point cloud

Syntax

```
[ptCloudOut,indices] = removeInvalidPoints(ptCloud)
```

Description

`[ptCloudOut,indices] = removeInvalidPoints(ptCloud)` removes points with `Inf` or `NaN` coordinate values from point cloud and returns the indices of valid points.

Note The output is always an unorganized (*X*-by-3) point cloud. If the input `ptCloud` is an organized point cloud (*M*-by-*N*-by-3), the function returns the output as an unorganized point cloud.

Examples

Remove Invalid Points from Point Cloud

Create a point cloud object with `NaN` and `Inf` values.

```
xyzpoints = abs(randn(10,3)).*100;
xyzpoints(1:2:4,:) = nan('single');
xyzpoints(6:2:10,:) = inf('single');
ptCloud = pointCloud(xyzpoints);
```

Inspect the `Location` property of point cloud data to verify the occurrence of `NaN` and `Inf` values.

```
ptCloud.Location
```

```
ans = 10×3

       NaN       NaN       NaN
  183.3885  303.4923  120.7487
       NaN       NaN       NaN
   86.2173    6.3055  163.0235
   31.8765   71.4743   48.8894
       Inf       Inf       Inf
   43.3592   12.4144   72.6885
       Inf       Inf       Inf
  357.8397  140.9034   29.3871
       Inf       Inf       Inf
```

Remove points with `NaN` and `Inf` values from the point cloud.

```
ptCloudOut = removeInvalidPoints(ptCloud);
```

Inspect the `Location` property of point cloud data to verify that the invalid points are removed.

```
ptCloudOut.Location

ans = 5×3

  183.3885  303.4923  120.7487
   86.2173    6.3055  163.0235
   31.8765   71.4743   48.8894
   43.3592   12.4144   72.6885
  357.8397  140.9034   29.3871
```

Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

Output Arguments

`ptCloudOut` — Point cloud with points removed

`pointCloud` object

Point cloud, returned as a `pointCloud` object with `Inf` or `NaN` coordinates removed.

Note The output is always an unorganized (*X*-by-3) point cloud. If the input `ptCloud` is an organized point cloud (*M*-by-*N*-by-3), the function returns the output as an unorganized point cloud.

`indices` — Indices of valid points

vector

Indices of valid points in the point cloud, specified as a vector.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`pointCloud` | `pcread`

select

Select points in point cloud

Syntax

```
ptCloudOut = select(ptCloud,indices)
ptCloudOut = select(ptCloud,row,column)
ptCloudOut = select( ___ ,OutputSize=outputSize)
```

Description

`ptCloudOut = select(ptCloud,indices)` returns a `pointCloud` object containing only the points that are selected using linear or logical indices.

`ptCloudOut = select(ptCloud,row,column)` returns a `pointCloud` object containing only the points that are selected using row and column subscripts. This syntax applies only if the input is an organized point cloud data of size *M*-by-*N*-by-3.

`ptCloudOut = select( ___ ,OutputSize=outputSize)` returns the selected points as a `pointCloud` object of size specified by `outputSize`.

Examples

Select Points from Point Cloud

Read a point cloud data into the workspace.

```
ptCloud = pcread("teapot.ply");
```

Read the number of points in the point cloud data.

```
Length = ptCloud.Count;
```

Select indices within the range [1, `Length`].

```
stepSize = 10;
indices = 1:stepSize:Length;
```

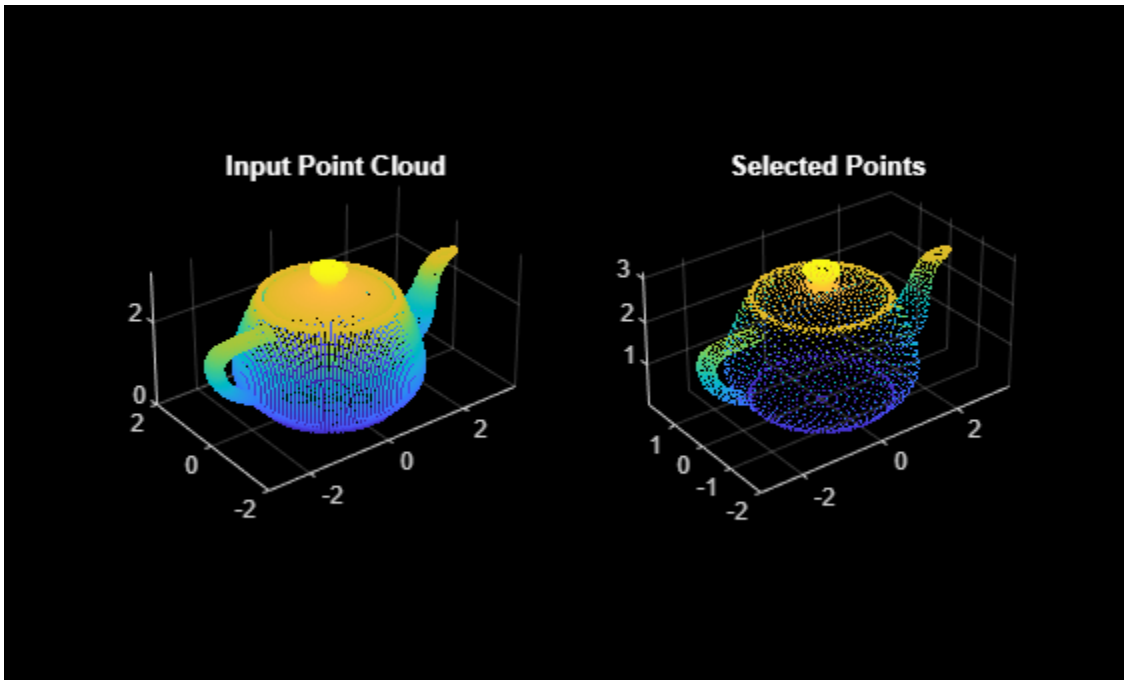
Select points specified by the indices from input point cloud. This selection of points downsamples the input point cloud by a factor of `stepSize`.

```
ptCloudB = select(ptCloud,indices);
```

Display the input and the selected point cloud data.

```
figure
subplot(1,2,1)
pcshow(ptCloud)
title("Input Point Cloud",Color=[1 1 1]);
subplot(1,2,2)
```

```
pcshow(ptCloudB)
title("Selected Points",Color=[1 1 1]);
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`indices` — Linear or logical indices of selected points

vector | matrix

Linear or logical indices of selected points, specified as a vector or a matrix. For organized point clouds, `indices` (linear or logical), refer only to the first two dimensions of the "Location" on page 2-0 property.

`row` — Row indices

vector

Row indices, specified as a vector. This argument applies only if the input is an organized point cloud with the "Location" on page 2-0 data of size *M*-by-*N*-by-3.

`column` — Column indices

vector

Column indices, specified as a vector. This argument applies only if the input is an organized point cloud with the "Location" on page 2-0 data of size *M*-by-*N*-by-3.

`outputSize` — Size of output point cloud

`"selected"` (default) | `"full"`

Size of the output point cloud, `ptCloudOut`, specified as `"selected"` or `"full"`.

- If the size is `"selected"`, then the output contains only the selected points from the input point cloud, `ptCloud`.
- If the size is `"full"`, then the output is same size as the input point cloud `ptCloud`. Cleared points are filled with `NaN` and the color is set to [0 0 0].
- When working with organized point clouds and to return an organized point cloud, set `outputSize` to `"full"`.

Output Arguments

`ptCloudOut` — Selected point cloud
`pointCloud` object

Point cloud, returned as a `pointCloud` object.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The input argument `outputSize` must be compile-time constant. For example,

```
outputSize = coder.Constant('full');
```

If you use the coder app, you can choose the `outputSize` by right-clicking on it and selecting **Use constant**.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`pointCloud` | `pcread`

copy

Copy array of point cloud objects

Syntax

```
ptCloudOut = copy(ptCloudIn)
```

Description

`ptCloudOut = copy(ptCloudIn)` copies an array of point cloud objects at the input to the new array `ptCloudOut`.

Examples

Create Copy of Point Cloud

- `pointCloudOut = copy(ptCloudIn)`

Input Arguments

`ptCloudIn` — Input point cloud data

`pointCloud` object

Input point cloud data, specified as a `pointCloud` object or an array of `pointCloud` objects.

Output Arguments

`ptCloudOut` — Output point cloud data

`pointCloud` object

Output point cloud data, returned as a `pointCloud` object or an array of `pointCloud` objects. The output data contains a copy of the input point cloud data.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`findNearestNeighbors` | `findNeighborsInRadius` | `findPointsInROI` | `removeInvalidPoints` | `select` | `pointCloud`

cylinderModel

Parametric cylinder model

Description

The `cylinderModel` object stores a parametric cylinder model.

Creation

Syntax

```
model = cylinderModel(parameters)
```

Description

`model = cylinderModel(parameters)` constructs a parametric cylinder model from the specified `parameters`. The `parameters` argument is a 1-by-7 numeric vector that determines the value of the `Parameters` property.

Input Arguments

`parameters` — Cylinder parameters

1-by-7 numeric vector

Cylinder parameters, specified as a 1-by-7 numeric vector. This input specifies the “Parameters” on page 2-0 property. The vector is of the form [*x1 y1 z1 x2 y2 z2 r*].

- [*x1 y1 z1*] and [*x2 y2 z2*] are the 3-D centers of each end-cap surface of the cylinder.
- *r* is the radius of the cylinder.

Properties

`Parameters` — Cylinder model parameters

1-by-7 numeric vector

This property is read-only.

Cylinder model parameters, stored as a 1-by-7 numeric vector. The vector is of the form [*x1 y1 z1 x2 y2 z2 r*].

- [*x1 y1 z1*] and [*x2 y2 z2*] are the 3-D centers of each end-cap surface of the cylinder.
- *r* is the radius of the cylinder.

`Center` — Center of cylinder

1-by-3 numeric vector

This property is read-only.

Center of the cylinder, stored as a 1-by-3 numeric vector representing the x-, y-, and z-coordinates.

`Height` — Height of cylinder
scalar

This property is read-only.

Height of the cylinder, stored as a scalar.

`Radius` — Radius of cylinder
scalar

This property is read-only.

Radius of the cylinder, stored as a scalar.

`Orientation` — Orientation of cylinder
1-by-3 vector

This property is read-only.

Orientation of the cylinder, stored as a 1-by-3 vector. The orientation vector points from one center [*x1*,*y1*,*z1*] of the cylinder end-cap to the center [*x2*,*y2*,*z2*] of the other end-cap.

Object Functions

plot	Plot parametric model
findPointsInModel	Find points in or on surface of geometric model

Examples

Detect Cylinder in Point Cloud

Load a MAT file containing a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud.

```
figure
pcshow(ptCloud)
title("Detect a Cylinder in a Point Cloud")
```

Set the maximum point-to-cylinder distance for cylinder fitting to 5mm.

```
maxDistance = 0.005;
```

Specify a region of interest (ROI) to constrain the fitting function.

```
roi = [0.4 0.6; -inf 0.2; 0.1 inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set the orientation constraint for the fitting function

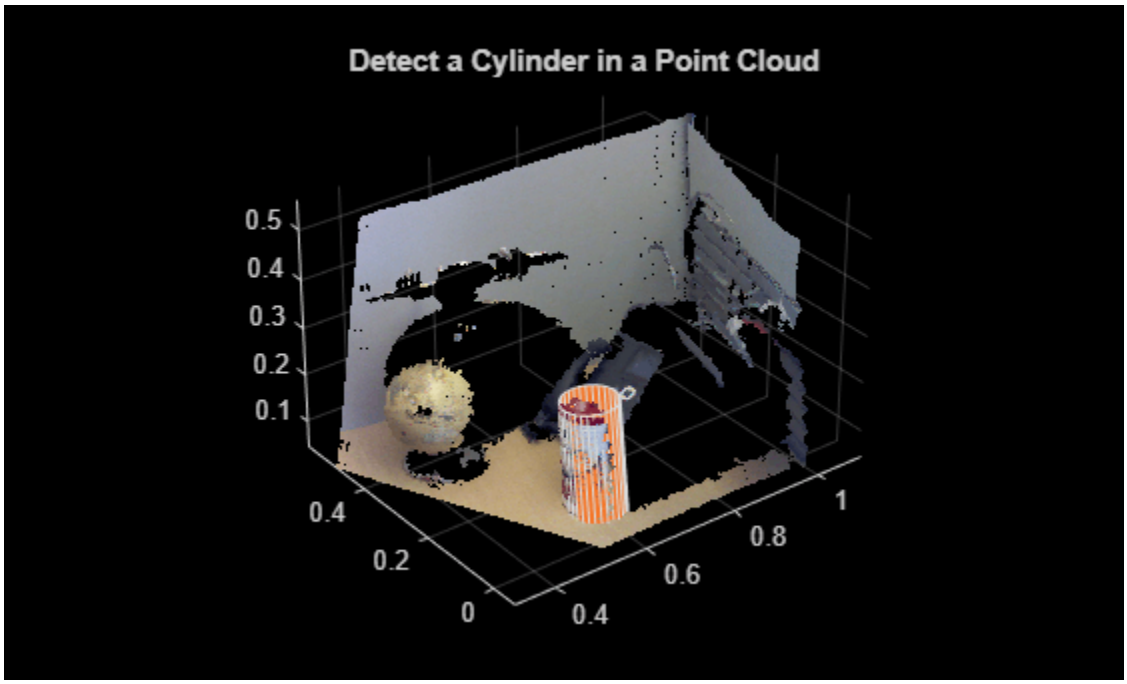
```
referenceVector = [0 0 1];
```

Detect the cylinder in the specified ROI of the point cloud and extract it.

```
model = pcfitcylinder(ptCloud,maxDistance,referenceVector, ...
        SampleIndices=sampleIndices);
```

Plot the model of the detected cylinder.

```
hold on
plot(model)
```



Version History

Introduced in R2015b

See Also

Objects

`pointCloud` | `planeModel` | `sphereModel` | `rigidtform3d`

Functions

`pcfitcylinder` | `pcfitplane` | `pcfitsphere` | `pcshow` | `pcplayer` | `pcwrite` | `pcread` | `pcmerge` | `pctransform` | `pcdownsample` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”

planeModel

Object for storing parametric plane model

Description

Construct and store a parametric plane model based on parameters that describe a plane.

Creation

Syntax

```
model = planeModel(parameters)
```

Description

`model = planeModel(parameters)` constructs a parametric plane model from the 1-by-4 `Parameters` input vector that describes a plane.

Input Arguments

`parameters` — Plane parameters
1-by-4 scalar vector

Plane parameters, specified as a 1-by-4 vector. This input specifies the `Parameters` property. The four parameters [a,b,c,d] describe the equation for a plane:

$$ax + by + cz + d = 0$$

Properties

These properties are read-only.

`Parameters` — Plane model parameters
1-by-4 vector

Plane model parameters, stored as a 1-by-4 vector. These parameters are specified by the parameters on page 2-0 input argument.

`Normal` — Normal vector of the plane
1-by-3 vector

Normal vector of the plane, stored as a 1-by-3 vector. The [a,b,c] vector specifies the unnormalized normal vector of the plane.

Object Functions

plot	Plot parametric model
normalRotation	Compute transform for rotation of normal to plane

Examples

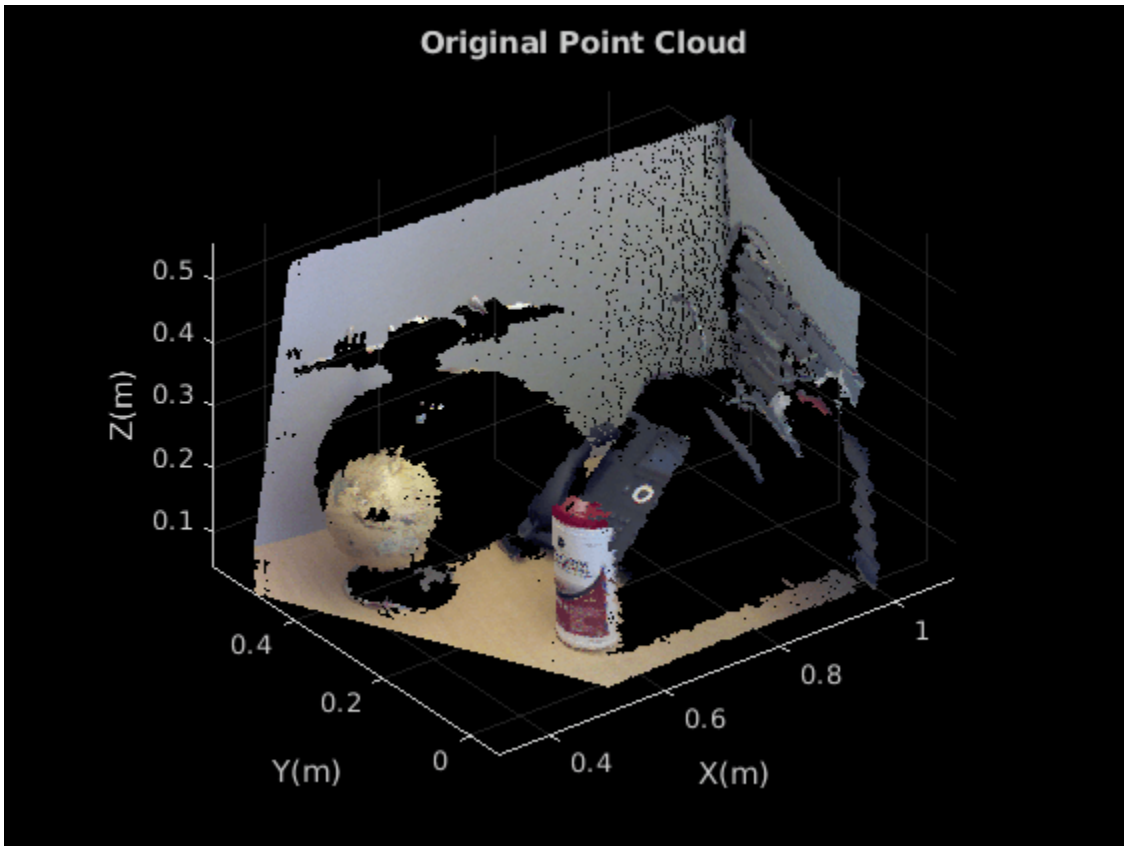
Detect Multiple Planes from Point Cloud

Load the point cloud.

```
load("object3d.mat")
```

Display the point cloud and label the figure.

```
figure
pcshow(ptCloud)
xlabel("X(m)")
ylabel("Y(m)")
zlabel("Z(m)")
title("Original Point Cloud")
```



Set the maximum point-to-plane distance (2cm) for plane fitting.

```
maxDistance = 0.02;
```

Set the normal vector of the plane.

```
referenceVector = [0,0,1];
```

Set the maximum angular distance to 5 degrees.

```
maxAngularDistance = 5;
```

Detect the first plane, the table, and extract it from the point cloud.

```
[model1,inlierIndices,outlierIndices] = pcfitplane(ptCloud,...
           maxDistance,referenceVector,maxAngularDistance);
plane1 = select(ptCloud,inlierIndices);
remainPtCloud = select(ptCloud,outlierIndices);
```

Set the region of interest to constrain the search for the second plane, left wall.

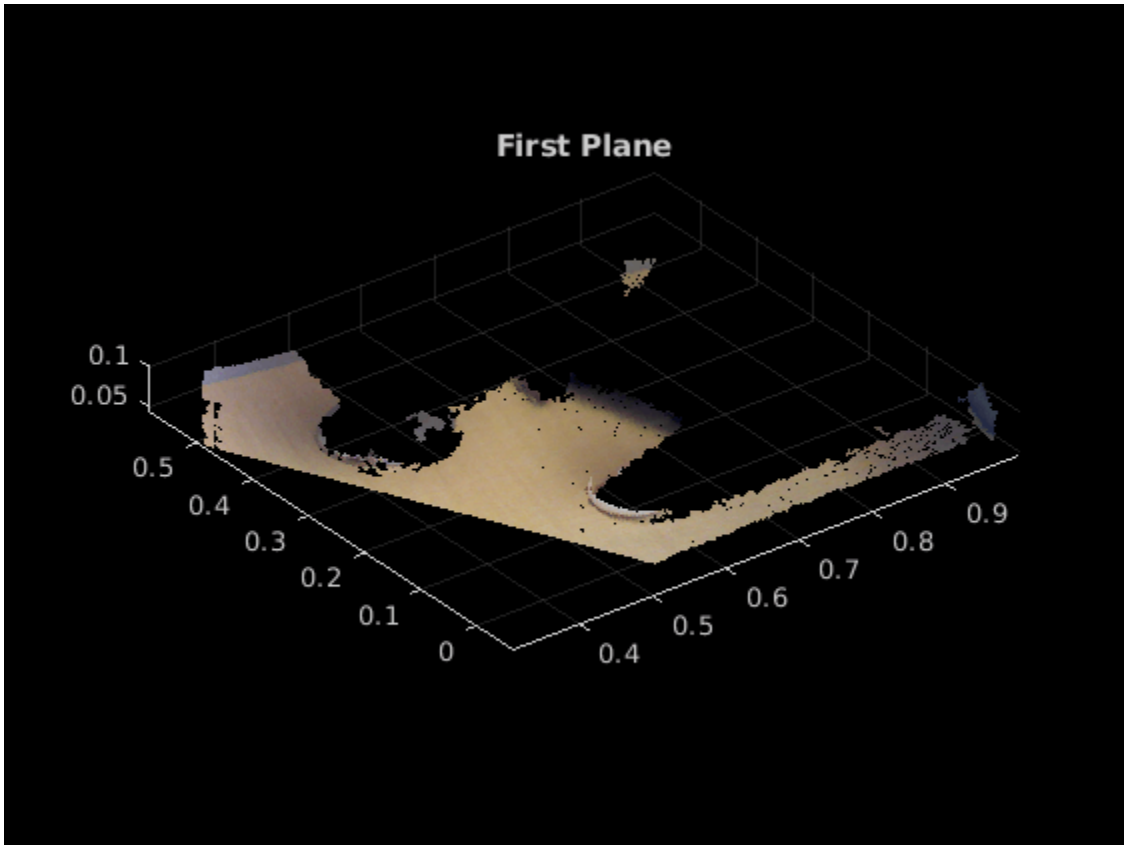
```
roi = [-inf,inf;0.4,inf;-inf,inf];
sampleIndices = findPointsInROI(remainPtCloud,roi);
```

Detect the left wall and extract it from the remaining point cloud.

```
[model2,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,...
           maxDistance,SampleIndices=sampleIndices);
plane2 = select(remainPtCloud,inlierIndices);
remainPtCloud = select(remainPtCloud,outlierIndices);
```

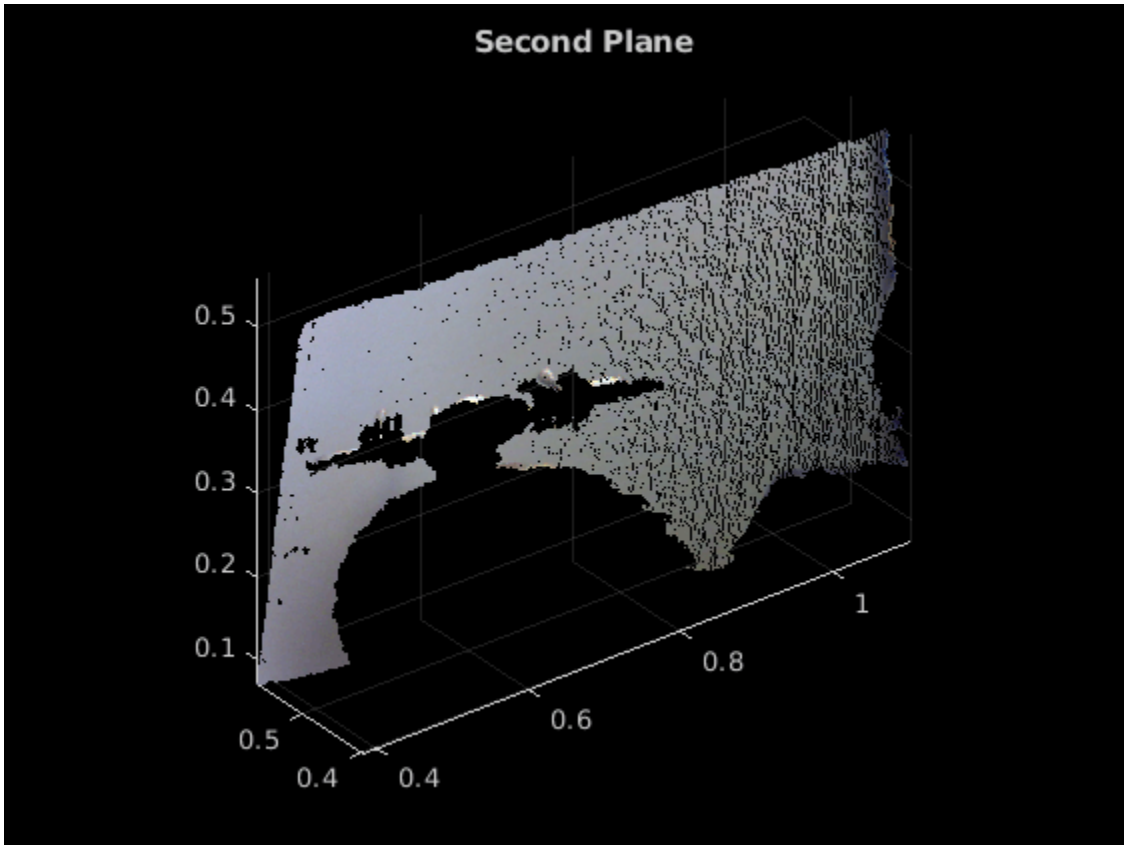
Plot the two planes and the remaining points.

```
figure
pcshow(plane1)
title("First Plane")
```

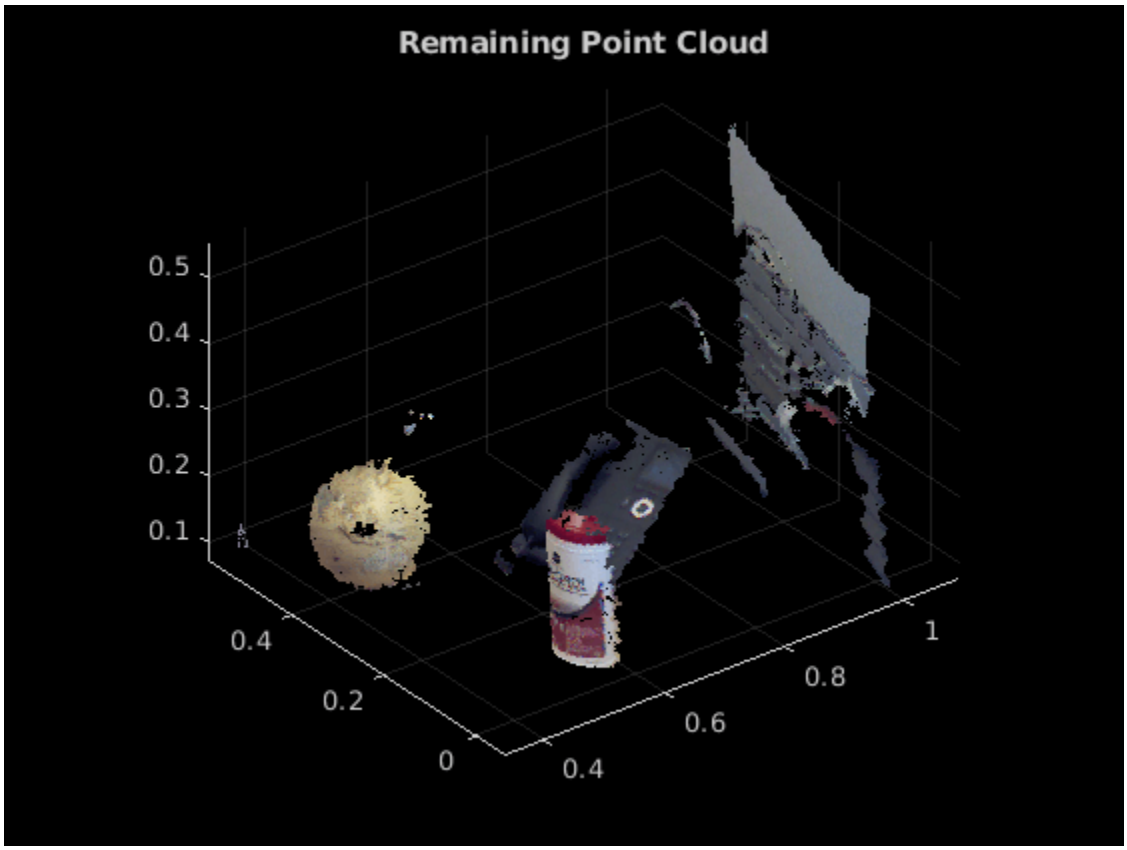



```
figure
```

```
pcshow(plane2)
title("Second Plane")
```




```
figure
pcshow(remainPtCloud)
title("Remaining Point Cloud")
```



Version History

Introduced in R2015b

See Also

Objects

`cylinderModel` | `sphereModel` | `pointCloud` | `pcplayer` | `rigidtform3d`

Functions

`pcfitcylinder` | `pcfitplane` | `pcfitsphere` | `pcshow` | `pcmerge` | `pctransform` | `pcdenoise` | `pcdownsample` | `pcwrite` | `pcread` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”

sphereModel

Object for storing a parametric sphere model

Description

The `sphereModel` object stores a parametric sphere model.

Creation

Syntax

```
model = sphereModel(parameters)
```

Description

`model = sphereModel(parameters)` constructs a parametric sphere model from the specified parameters. The `parameters` argument is a 1-by-4 numeric vector that determines the value of the `Parameters` property.

Input Arguments

`parameters` — Sphere parameters

1-by-4 scalar vector

Sphere parameters, specified as a 1-by-4 vector. This input specifies the `Parameters` property. The four parameters [*a b c d*] satisfy this equation for a sphere:

$$(x-a)^2+(y-b)^2+(z-c)^2=d^2$$

Properties

`Parameters` — Sphere model parameters

1-by-4 numeric vector

This property is read-only.

Sphere model parameters, stored as a 1-by-4 numeric vector. This property is set by the “parameters” on page 2-0 input. The four parameters [*a b c d*] satisfy this equation for a sphere:

$$(x-a)^2+(y-b)^2+(z-c)^2=d^2$$

`Center` — Center of sphere

1-by-3 vector

This property is read-only.

Center of the sphere, stored as a 1-by-3 vector of form [*xc yc zc*]. The elements *xc*,*yc*,*zc* specify the x-, y-, and z-coordinates of the center coordinates of the sphere, respectively.

Radius — Radius of sphere

scalar

This property is read-only.

Radius of the sphere, stored as a scalar value.

Object Functions

plot	Plot parametric model
findPointsInModel	Find points in or on surface of geometric model

Examples

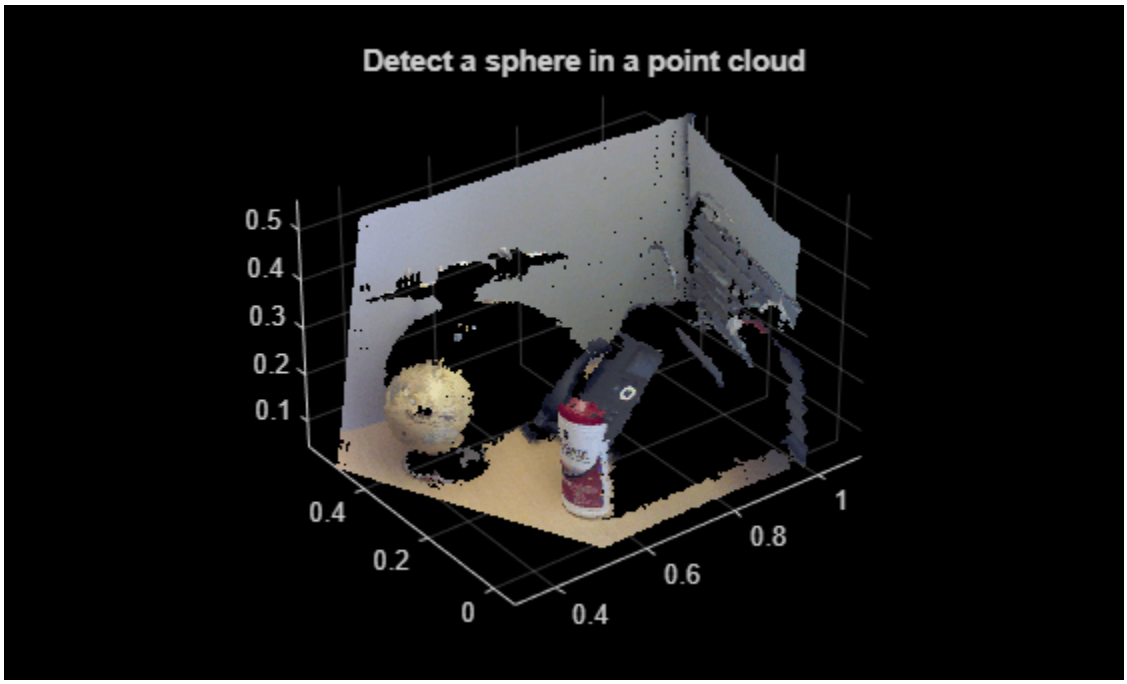
Detect Sphere in Point Cloud

Load a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud and label the figure.

```
figure
pcshow(ptCloud)
title("Detect a sphere in a point cloud")
```



Set the maximum point-to-sphere distance for sphere fitting to 1cm.

```
maxDistance = 0.01;
```

Set the region of interest to constrain the search.

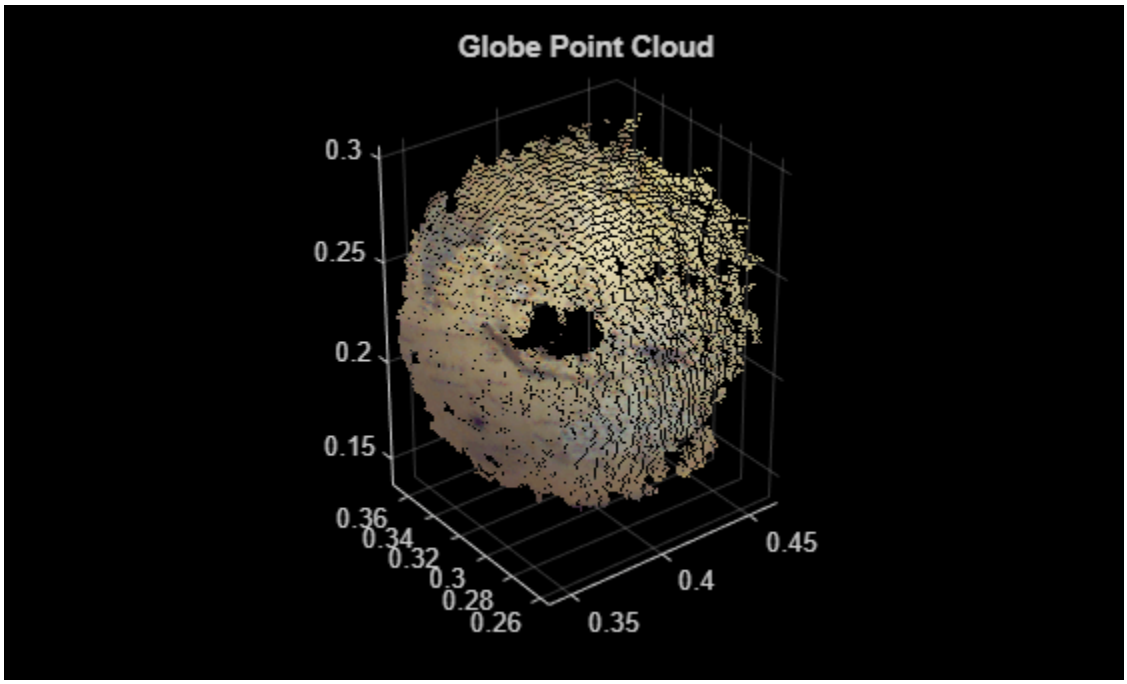
```
roi = [-inf,0.5;0.2,0.4;0.1,inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Detect the globe in the point cloud and extract it.

```
[model,inlierIndices] = pcfitsphere(ptCloud,maxDistance,SampleIndices=sampleIndices);
globe = select(ptCloud,inlierIndices);
```

Plot the extracted globe.

```
figure
pcshow(globe)
title("Globe Point Cloud")
```



Version History

Introduced in R2015b

See Also

Objects

`pointCloud` | `planeModel` | `rigidtform3d` | `cylinderModel`

Functions

`pcfitcylinder` | `pcfitplane` | `pcfitsphere` | `pcshow` | `pcplayer` | `pcwrite` | `pcread` | `pcmerge` | `pctransform` | `pcdownsample` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”

findPointsInModel

Find points in or on surface of geometric model

Syntax

```
indices = findPointsInModel(model,ptCloud)
```

Description

`indices = findPointsInModel(model,ptCloud)` finds the points in the point cloud `ptCloud` that are located inside or on the surface of the geometric shape specified by `model`, and returns their linear `indices`.

Examples

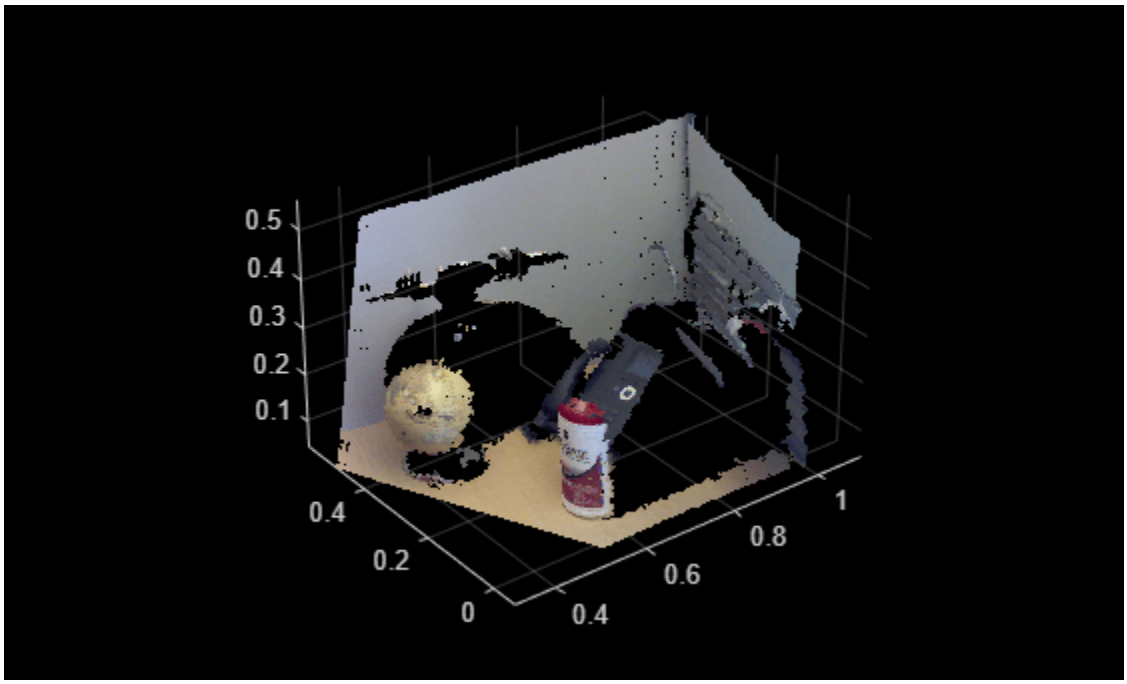
Find Points of Cylinder Model in Point Cloud

Load a MAT file containing point cloud data into the workspace.

```
load("object3d.mat")
```

Visualize the point cloud.

```
figure
pcshow(ptCloud)
```



Define a region of interest (ROI) in the point cloud in which to detect a cylinder model.

```
roi = [0.4 0.7; -0.1 0.2; 0 0.4];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set a maximum point-to-cylinder distance of 5 mm for cylinder fitting and the orientation constraint.

```
maxDistance = 0.005;
referenceVector = [0 0 1];
```

Detect a cylinder in the point cloud.

```
 model = pcfitcylinder(ptCloud,maxDistance,referenceVector, ...
        SampleIndices=sampleIndices);
```

Create a synthetic point cloud that is evenly distributed and covers the region of interest.

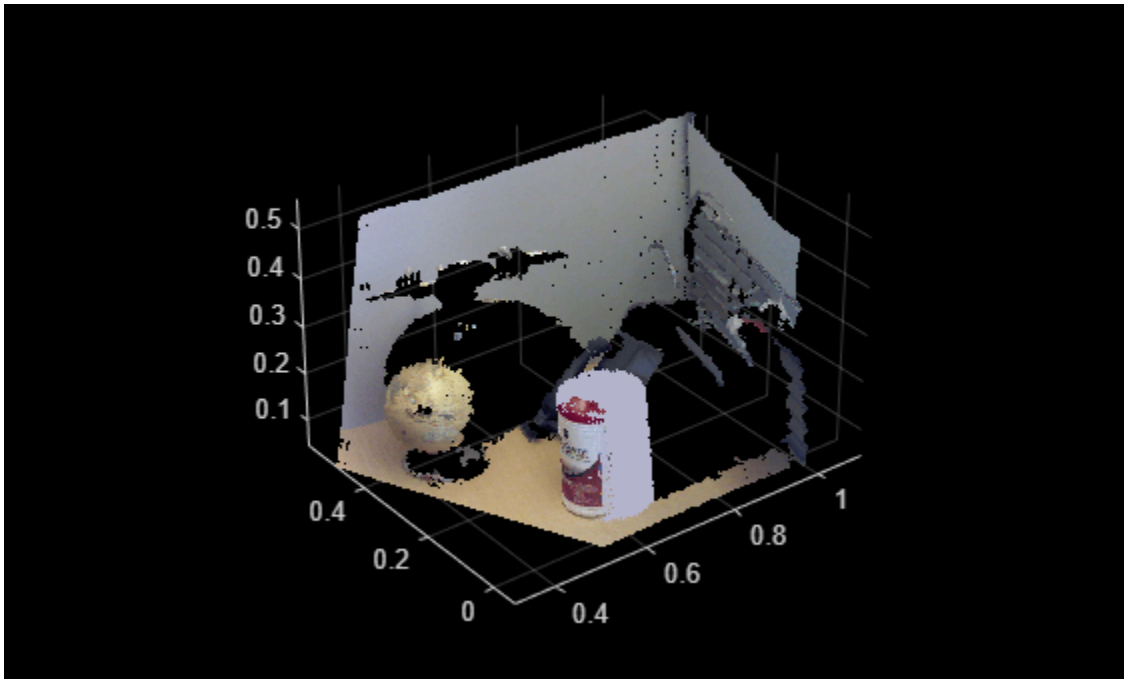
```
gridStep = 0.005;
[x,y,z] = meshgrid(roi(1,1):gridStep:roi(1,2),roi(2,1):gridStep:roi(2,2), ...
roi(3,1):gridStep:roi(3,2));
syntheticPtCloud = pointCloud([x(:) y(:) z(:)],Color=[0.7 0.7 0.8]);
```

Find the points in the synthetic point cloud that are in the cylinder.

```
idx = findPointsInModel(model,syntheticPtCloud);
cylinderPtCloud = select(syntheticPtCloud,idx);
```

Visualize the resulting point cloud, with the synthetic points in the cylinder modeling the object on the table.

```
figure
pcshow(cylinderPtCloud)
hold on
pcshow(ptCloud)
```



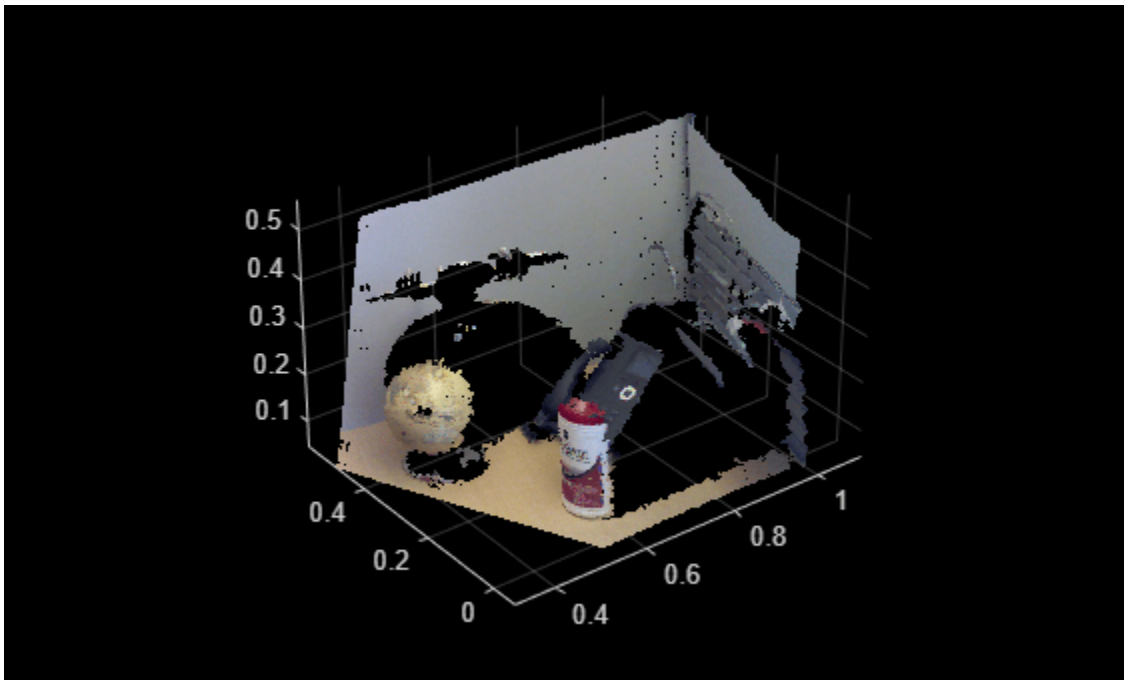
Find Points of Sphere Model in Point Cloud

Load a MAT file containing a point cloud into the workspace.

```
load("object3d.mat")
```

Visualize the point cloud.

```
figure
pcshow(ptCloud)
```



Define a region of interest (ROI) in the point cloud in which to detect a sphere model.

```
roi = [0.2 0.55; 0.2 0.5; 0 0.5];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set a maximum point-to-sphere distance of 5mm for sphere fitting.

```
maxDistance = 0.005;
```

Detect the sphere in the point cloud.

```
[model,inlierIdx] = pcfitsphere(ptCloud,maxDistance,SampleIndices=sampleIndices);
```

Find one of the colors in the detected sphere, to use for the synthetic point cloud.

```
inlierPtCloud = select(ptCloud,inlierIdx);
color = median(inlierPtCloud.Color);
```

Create a synthetic point cloud that is evenly distributed and covers the region of interest.

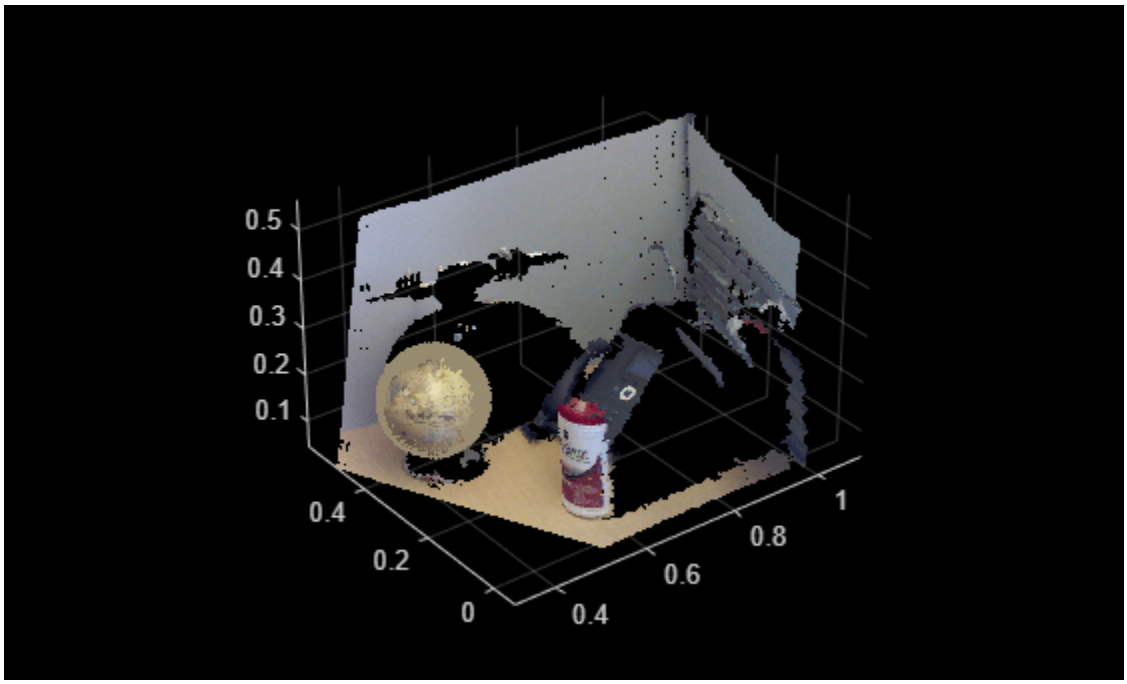
```
gridStep = 0.005;
[x,y,z] = meshgrid(roi(1,1):gridStep:roi(1,2),roi(2,1):gridStep:roi(2,2), ...
roi(3,1):gridStep:roi(3,2));
syntheticPtCloud = pointCloud([x(:) y(:) z(:)],Color=color);
```

Find the points in the synthetic point cloud that are in the sphere.

```
idx = findPointsInModel(model,syntheticPtCloud);
spherePtCloud = select(syntheticPtCloud,idx);
```

Visualize the resulting point cloud, with the synthetic points in the sphere modeling the object on the table.

```
figure
pcshow(spherePtCloud)
hold on
pcshow(ptCloud)
```



Input Arguments

`model` — Parametric model
`cylinderModel` object | `sphereModel` object

Parametric model, specified as a `cylinderModel` or `sphereModel` object.

`ptCloud` — Point cloud
`pointCloud` object

Point cloud in the sensor coordinate system, specified as a `pointCloud` object.

Output Arguments

`indices` — Linear indices of point cloud points
column vector

Linear indices of the point cloud points that are located inside or on the surface of the model, returned as a column vector.

Version History

Introduced in R2024a

See Also

Objects

`cylinderModel` | `sphereModel`

opticalFlow

Object for storing optical flow matrices

Description

The `opticalFlow` object stores the direction and speed of a moving object from one image or video frame to another. Use the object function `plot` to plot the optical flow vectors.

Creation

Syntax

```
flow = opticalFlow
flow = opticalFlow(Vx,Vy)
```

Description

`flow = opticalFlow` creates an optical flow object with default property values.

`flow = opticalFlow(Vx,Vy)` creates an optical flow object from two equal-sized matrices `Vx` and `Vy`. The matrices `Vx` and `Vy` are the *x* and *y* components of velocity respectively. The input velocity components set the “Properties” on page 2-1035 of optical flow object.

Input Arguments

Vx — *x* component of velocity

M-by-*N* matrix

x component of velocity in pixels per frame, specified as an *M*-by-*N* matrix.

Data Types: `single` | `double`

Vy — *y* component of velocity

M-by-*N* matrix

y component of velocity in pixels per frame, specified as an *M*-by-*N* matrix.

Data Types: `single` | `double`

Note `Vx` and `Vy` must be of the same size and data type.

Properties

Vx — *x* component of velocity

0-by-1 empty matrix (default) | *M*-by-*N* matrix

x component of velocity in pixels per frame, specified as an *M*-by-*N* matrix. If the input `Vx` is not specified, the default value of this property is set to a 0-by-1 empty matrix.

Data Types: `single` | `double`

`Vy` — *y* component of velocity

0-by-1 empty matrix (default) | *M*-by-*N* matrix

y component of velocity in pixels per frame, specified as an *M*-by-*N* matrix. If the input `Vy` is not specified, the default value of this property is set to a 0-by-1 empty matrix.

Data Types: `single` | `double`

`Orientation` — Phase angles of optical flow

0-by-1 empty matrix (default) | *M*-by-*N* matrix

This property is read-only.

Phase angles of optical flow in radians, specified as an *M*-by-*N* matrix of the same size and data type as the components of velocity. The phase angles of optical flow is calculated from the *x* and *y* components of velocity. If the inputs `Vx` and `Vy` are not specified, the default value of this property is set to 0-by-1 empty matrix.

Data Types: `single` | `double`

`Magnitude` — Magnitude of optical flow

0-by-1 empty matrix (default) | *M*-by-*N* matrix

This property is read-only.

Magnitude of optical flow in pixels per frame, specified as an *M*-by-*N* matrix of the same size and data type as the components of velocity. The magnitude of optical flow is calculated from the *x* and *y* components of velocity. If the inputs `Vx` and `Vy` are not specified, the default value of this property is set to 0-by-1 empty matrix.

Data Types: `single` | `double`

Object Functions

plot Plot optical flow vectors

Examples

Create Optical Flow Object

Create an optical flow object and view its properties.

```
flow = opticalFlow
```

```
flow =

  opticalFlow with properties:

            Vx: [0×1 double]
            Vy: [0×1 double]
    Orientation: [0×1 double]
      Magnitude: [0×1 double]
```

The default values of the properties are returned as an empty matrix of size 0-by-1 and data type `double`.

Create an Optical Flow Object and Plot Its Velocity

Create an optical flow object from two equal-sized velocity matrices.

```
Vx = randn(100,100);
Vy = randn(100,100);
opflow = opticalFlow(Vx,Vy);
```

Inspect the properties of the optical flow object. The orientation and the magnitude are computed from the velocity matrices.

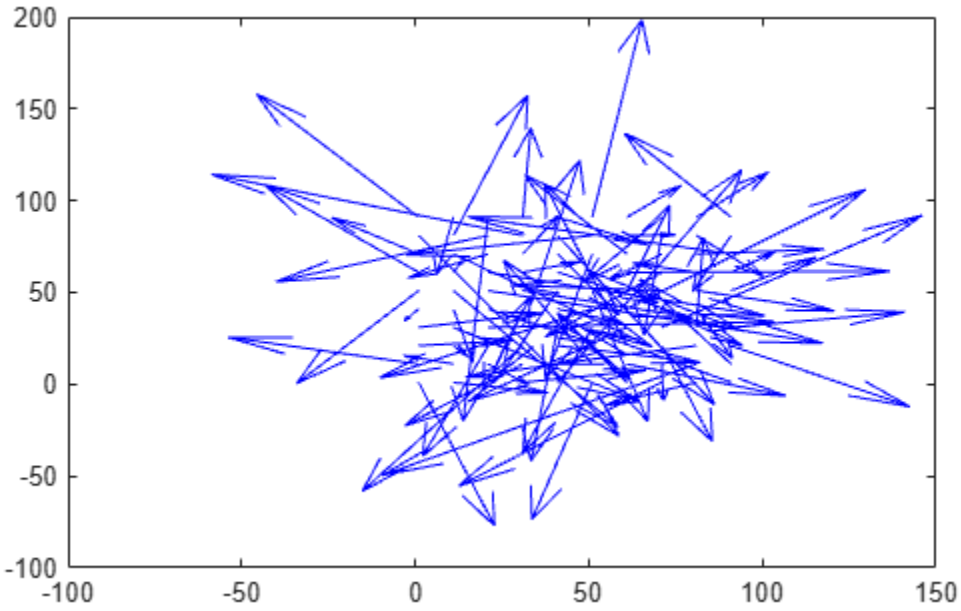
```
opflow
```

```
opflow =
  opticalFlow with properties:

            Vx: [100×100 double]
            Vy: [100×100 double]
   Orientation: [100×100 double]
     Magnitude: [100×100 double]
```

Plot the velocity of the object as a quiver plot.

```
plot(opflow,DecimationFactor=[10 10],ScaleFactor=10);
```



Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`quiver`

Objects

`opticalFlowRAFT` | `opticalFlowHS` | `opticalFlowLKDoG` | `opticalFlowLK` | `opticalFlowFarneback`

plot

Plot optical flow vectors

Syntax

```
plot(flow)
plot(flow,Name=Value)
```

Description

`plot(flow)` plots the optical flow vectors.

`plot(flow,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `plot(flow,Color="red")` plots the optical flow with red flow vectors.

Examples

Create an Optical Flow Object and Plot Its Velocity

Create an optical flow object from two equal-sized velocity matrices.

```
Vx = randn(100,100);
Vy = randn(100,100);
opflow = opticalFlow(Vx,Vy);
```

Inspect the properties of the optical flow object. The orientation and the magnitude are computed from the velocity matrices.

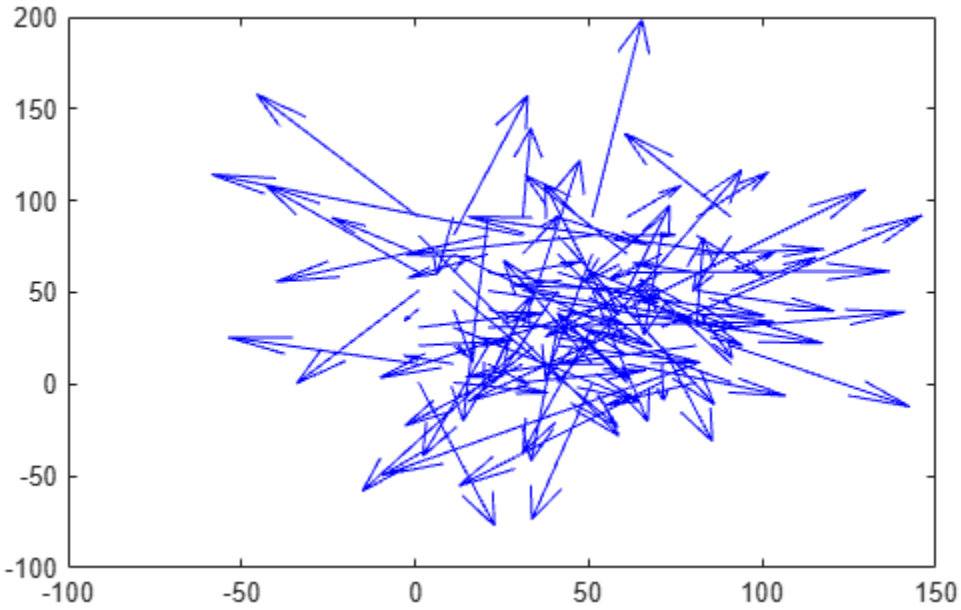
```
opflow
```

```
opflow =
  opticalFlow with properties:

            Vx: [100×100 double]
            Vy: [100×100 double]
    Orientation: [100×100 double]
      Magnitude: [100×100 double]
```

Plot the velocity of the object as a quiver plot.

```
plot(opflow,DecimationFactor=[10 10],ScaleFactor=10);
```



Input Arguments

`flow` — Object containing optical flow matrices

`opticalFlow` object

Object containing optical flow velocity matrices, specified as an `opticalFlow` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `plot(flow,Color="red")` plots the optical flow with red flow vectors.

`DecimationFactor` — Decimation factor of velocity vectors

`[1 1]` (default) | two-element vector

Decimation factor of velocity vectors, specified as a two-element vector. The two-element vector is of form [*XDecimFactor YDecimFactor*] and specifies the decimation factor for velocity vectors along the *x* and *y* directions respectively. *XDecimFactor* and *YDecimFactor* are positive scalar integers. Increase the value of this property to get a less cluttered quiver plot.

`ScaleFactor` — Scaling factor for velocity vector display

1 (default) | positive integer-valued scalar

Scaling factor for velocity vector display, specified as a positive integer-valued scalar. Increase this value to display longer vectors.

`Parent` — Figure axes
`gca` handle

Figure axes, specified as an axes object. The default is set to the current axes handle, `gca`.

`Color` — Color of optical flow vectors
"blue" (default) | RGB triplet | long or short color name

Color of optical flow vectors, specified as a 1-by-3 RGB triplet in the range [0,1] or a long or short color name.

Version History

Introduced in R2015a

See Also

`quiver` | `opticalFlow` | `opticalFlowRAFT`

opticalFlowHS

Object for estimating optical flow using Horn-Schunck method

Description

Create an optical flow object for estimating the direction and speed of a moving object using the Horn-Schunck method. Use the object function `estimateFlow` to estimate the optical flow vectors. Using the `reset` object function, you can reset the internal state of the optical flow object.

Creation

Syntax

```
opticFlow = opticalFlowHS
opticFlow = opticalFlowHS(Name,Value)
```

Description

`opticFlow = opticalFlowHS` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Horn-Schunck method.

`opticFlow = opticalFlowHS(Name,Value)` returns an optical flow object with properties specified as one or more `Name,Value` pair arguments. Any unspecified properties have default values. Enclose each property name in quotes.

For example, `opticalFlowHS('Smoothness',1.5)`

Properties

`Smoothness` — Expected smoothness
`1` (default) | positive scalar

Expected smoothness of optical flow, specified as a positive scalar. Increase this value when there is increased motion between consecutive frames. A typical value for `'Smoothness'` is around `1`.

`MaxIteration` — Maximum number of iterations
`10` (default) | positive integer-valued scalar

Maximum number of iterations, specified as a positive integer-valued scalar. Increase this value to estimate the optical flow of objects with low velocity.

The iterative computation stops when the number of iterations equals the value of `'MaxIteration'` or when the algorithm reaches the value set for `'VelocityDifference'`. To stop computation only by using `'MaxIteration'`, set the value of `'VelocityDifference'` to `0`.

`VelocityDifference` — Minimum absolute velocity difference
`0` (default) | positive scalar

Minimum absolute velocity difference, specified as a positive scalar. This value depends on the input data type. Decrease this value to estimate the optical flow of objects that have low velocity.

The iterative computation stops when the algorithm reaches the value set for `'VelocityDifference'` or the number of iterations equals `'MaxIteration'`. To use only `'VelocityDifference'` to stop computation, set `'MaxIteration'` to `Inf`.

Object Functions

estimateFlow	Estimate optical flow
reset	Reset the internal state of the optical flow estimation object

Examples

Estimate Optical Flow Using Horn-Schunck Method

Create a `VideoReader` object for the input video file, `visiontraffic.avi`. Specify the timestamp of the frame to read as 11.

```
vidReader = VideoReader('visiontraffic.avi',CurrentTime=11);
```

Specify the optical flow estimation method as `opticalFlowHS`. The output is an object specifying the optical flow estimation method and its properties.

```
opticFlow = opticalFlowHS
```

```
opticFlow =
  opticalFlowHS with properties:

            Smoothness: 1
          MaxIteration: 10
    VelocityDifference: 0
```

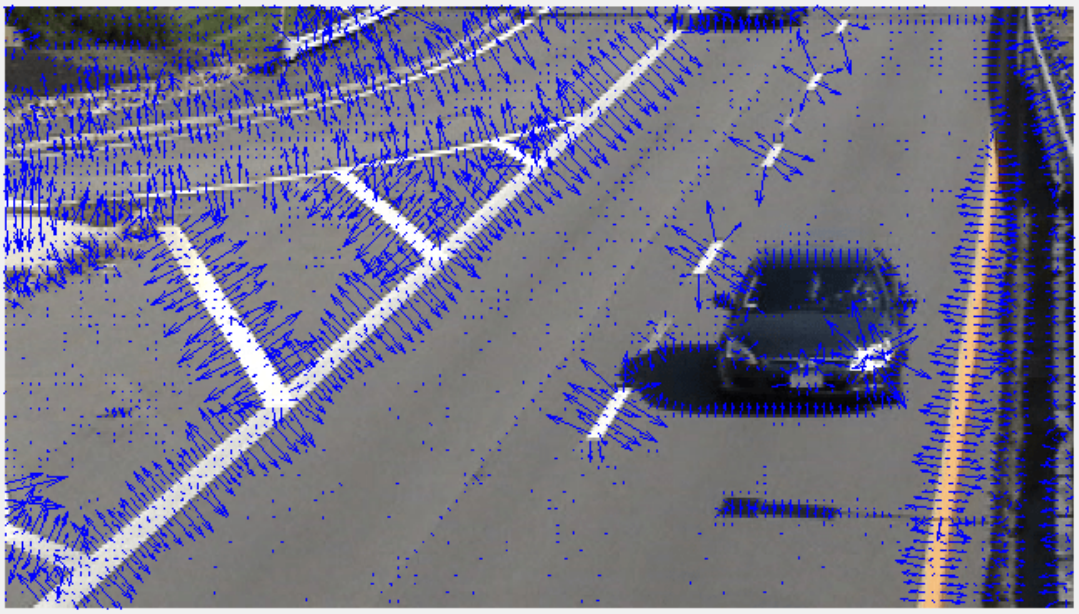
Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,Position=[0 0 1 1],Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

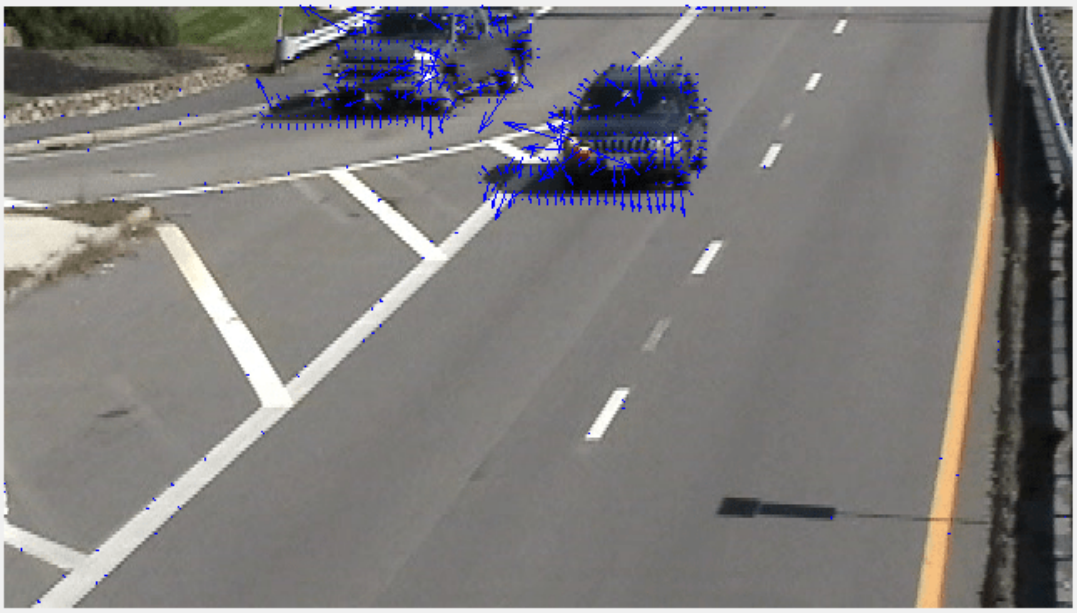
Read image frames from the `VideoReader` object and convert to grayscale images. Estimate the optical flow from consecutive image frames. Display the current image frame and plot the optical flow vectors as quiver plot.

```
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,DecimationFactor=[5 5],ScaleFactor=5,Parent=hPlot);
    hold off
    pause(10^-3)
end
```

Plot of Optical Flow Vectors



Plot of Optical Flow Vectors



Algorithms

To compute the optical flow between two images, you must solve this optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

.

- I_x, I_y, and I_t are the spatiotemporal image brightness derivatives.
- u is the horizontal optical flow.
- v is the vertical optical flow.

Horn-Schunck Method

By assuming that the optical flow is smooth across the entire image, the Horn-Schunck method estimates a velocity field, $[u\ v]^T$, that minimizes this equation:

$$E = \iint (I_x u + I_y v + I_t)^2 dx dy + \alpha \iint \left\{ \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right\} dx dy$$

.

In this equation, $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ are the spatial derivatives of the optical velocity component, u, and α scales the global smoothness term. The Horn-Schunck method minimizes the previous equation to obtain the velocity field, $[u\ v]$, for each pixel in the image. This method is given by the following equations:

$$u_{x,y}^{k+1} = \bar{u}_{x,y}^{k} - \frac{I_x [I_x \bar{u}^k_{x,y} + I_y \bar{v}^k_{x,y} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v_{x,y}^{k+1} = \bar{v}_{x,y}^{k} - \frac{I_y [I_x \bar{u}^k_{x,y} + I_y \bar{v}^k_{x,y} + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

.

In these equations, $\left[u_{x,y}^k\ v_{x,y}^k\right]$ is the velocity estimate for the pixel at (x,y), and $\left[\bar{u}_{x,y}^k\ \bar{v}_{x,y}^k\right]$ is the neighborhood average of $\left[u_{x,y}^k\ v_{x,y}^k\right]$. For $k = 0$, the initial velocity is 0.

To solve u and v using the Horn-Schunck method:

1. Compute I_x and I_y by using the Sobel convolution kernel, $[-1\ -2\ -1;\ 0\ 0\ 0;\ 1\ 2\ 1]$, and its transposed form for each pixel in the first image.
2. Compute I_t between images 1 and 2 using the $[-1\ 1]$ kernel.
3. Assume the previous velocity to be 0, and compute the average velocity for each pixel using $[0\ 1\ 0;\ 1\ 0\ 1;\ 0\ 1\ 0]$ as a convolution kernel.
4. Iteratively solve for u and v.

Version History

Introduced in R2015a

References

[1] Barron, J. L., D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. “ Performance of optical flow techniques.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),*236-242. Champaign, IL: CVPR, 1992.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`opticalFlowRAFT` | `quiver` | `opticalFlow` | `opticalFlowLKDoG` | `opticalFlowLK` | `opticalFlowFarneback`

Topics

“What Is Optical Flow?”

opticalFlowLK

Object for estimating optical flow using Lucas-Kanade method

Description

Create an optical flow object for estimating the direction and speed of a moving object using the Lucas-Kanade method. Use the object function `estimateFlow` to estimate the optical flow vectors. Using the `reset` object function, you can reset the internal state of the optical flow object.

Creation

Syntax

```
opticFlow = opticalFlowLK
opticFlow = opticalFlowLK('NoiseThreshold',threshold)
```

Description

`opticFlow = opticalFlowLK` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Lucas-Kanade method.

`opticFlow = opticalFlowLK('NoiseThreshold',threshold)` returns an optical flow object with the property `'NoiseThreshold'` specified as a `Name,Value` pair. Enclose the property name in quotes.

For example, `opticalFlowLK('NoiseThreshold',0.05)`

Properties

`NoiseThreshold` — Threshold for noise reduction

`0.0039` (default) | positive scalar

Threshold for noise reduction, specified as a positive scalar. As you increase this number, the movement of the objects has less impact on optical flow calculation.

Object Functions

estimateFlow	Estimate optical flow
reset	Reset the internal state of the optical flow estimation object

Examples

Compute Optical Flow Using Lucas-Kanade Algorithm

Read a video file. Specify the timestamp of the frame to be read.

```
vidReader = VideoReader('visiontraffic.avi','CurrentTime',11);
```

Create an optical flow object for estimating the optical flow using Lucas-Kanade method. Specify the threshold for noise reduction. The output is an optical flow object specifying the optical flow estimation method and its properties.

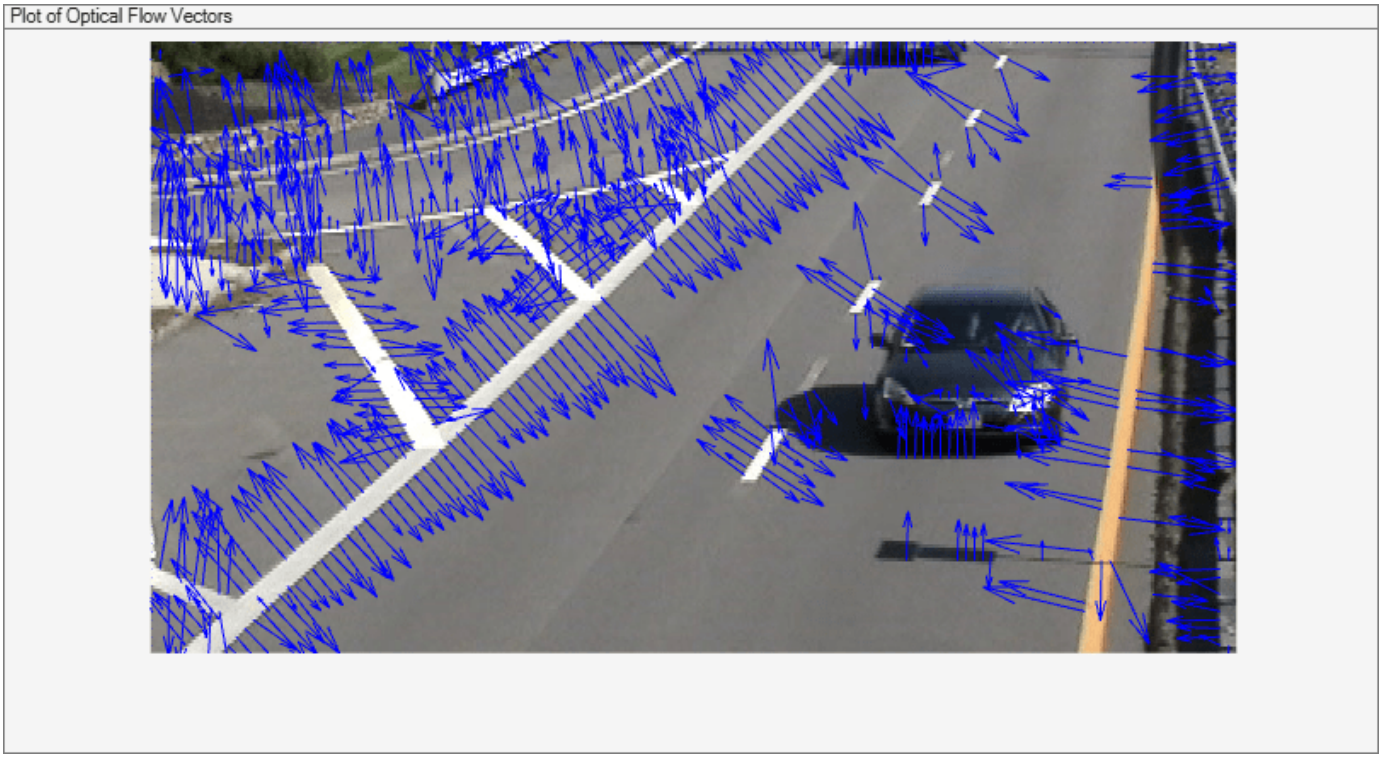
```
opticFlow = opticalFlowLK('NoiseThreshold',0.009);
```

Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of Optical Flow Vectors');
hPlot = axes(hViewPanel);
```

Read the image frames and convert to grayscale images. Estimate the optical flow from consecutive image frames. Display the current image frame and plot the optical flow vectors as quiver plot.

```
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',10,'Parent',hPlot);
    hold off
    pause(10^-3)
end
```





Algorithms

To compute the optical flow between two images, you must solve this optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

.

- I_x, I_y, and I_t are the spatiotemporal image brightness derivatives.
- u is the horizontal optical flow.
- v is the vertical optical flow.

Lucas-Kanade Method

To solve the optical flow constraint equation for u and v, the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity in each section. Then it performs a weighted, least-square fit of the optical flow constraint equation to a constant model for $[u\ v]^T$ in each section Ω. The method achieves this fit by minimizing this equation:

$$\sum_{x \in \Omega} W^2 [I_x u + I_y v + I_t]^2$$

W is a window function that emphasizes the constraints at the center of each section. The solution to the minimization problem is

$$\begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum W^2 I_x I_t \\ \sum W^2 I_y I_t \end{bmatrix}$$

.

The Lucas-Kanade method computes I_t using a difference filter, `[-1 1]`.

u and v are solved as follows:

1. Compute I_x and I_y using the kernel $[-1\ 8\ 0\ -8\ 1]/12$ and its transposed form.
2. Compute I_t between images 1 and 2 using the $[-1\ 1]$ kernel.
3. Smooth the gradient components, I_x, I_y, and I_t, using a separable and isotropic 5-by-5 element kernel whose effective 1-D coefficients are $[1\ 4\ 6\ 4\ 1]/16$.
4. Solve the 2-by-2 linear equations for each pixel using the following method:

   - If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

     Then the eigenvalues of A are $\lambda_i = \frac{a+c}{2} \pm \frac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$

   - The eigenvalues are compared to the threshold, τ, that corresponds to the value you enter for the threshold for noise reduction. The results fall into one of the following cases:

     Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

     A is nonsingular, the system of equations are solved using Cramer's rule.

     Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

     A is singular (noninvertible), the gradient flow is normalized to calculate u and v.

     Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

     The optical flow, u and v, is 0.

Version History

Introduced in R2015a

References

[1] Barron, J. L., D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. “ Performance of optical flow techniques.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),*236-242. Champaign, IL: CVPR, 1992.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`opticalFlowRAFT` | `quiver` | `opticalFlow` | `opticalFlowHS` | `opticalFlowFarneback` | `opticalFlowLKDoG`

Topics

“What Is Optical Flow?”

opticalFlowLKDoG

Object for estimating optical flow using Lucas-Kanade derivative of Gaussian method

Description

Create an optical flow object for estimating the direction and speed of moving objects using the Lucas-Kanade derivative of Gaussian (DoG) method. Use the object function `estimateFlow` to estimate the optical flow vectors. Using the `reset` object function, you can reset the internal state of the optical flow object.

Creation

Syntax

```
opticFlow = opticalFlowLKDoG
opticFlow = opticalFlowLKDoG(Name,Value)
```

Description

`opticFlow = opticalFlowLKDoG` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Lucas-Kanade derivative of Gaussian (DoG) method.

`opticFlow = opticalFlowLKDoG(Name,Value)` returns an optical flow object with properties specified as one or more `Name,Value` pair arguments. Any unspecified properties have default values. Enclose each property name in quotes.

For example, `opticalFlowLKDoG('NumFrames',3)`

Properties

`NumFrames` — Number of buffered frames
3 (default) | positive integer-valued scalar

Number of buffered frames for temporal smoothing, specified as a positive integer-valued scalar. As you increase this number, the optical flow estimation method becomes less robust to abrupt changes in the trajectory of the moving objects. The amount of delay in flow estimation depends on the value of `NumFrames`. The output flow corresponds to the image at $t_{flow} = t_{current} - 0.5(NumFrames\text{-}1)$, where $t_{current}$ is the time of the current image.

`ImageFilterSigma` — Standard deviation for image smoothing filter
`1.5` | positive scalar

Standard deviation for image smoothing filter, specified as a positive scalar.

`GradientFilterSigma` — Standard deviation for gradient smoothing filter
1 | positive scalar

Standard deviation for gradient smoothing filter, specified as a positive scalar.

`NoiseThreshold` — Threshold for noise reduction
`0.0039` (default) | positive scalar

Threshold for noise reduction, specified as a positive scalar. As you increase this number, the movement of the objects has less impact on optical flow calculation.

Object Functions

estimateFlow	Estimate optical flow
reset	Reset the internal state of the optical flow estimation object

Examples

Compute Optical Flow Using Lucas-Kanade DoG Method

Read a video file. Specify the timestamp of the frame to be read.

```
vidReader = VideoReader('visiontraffic.avi',CurrentTime=11);
```

Create an optical flow object for estimating the optical flow using Lucas-Kanade DoG method. Specify the threshold for noise reduction. The output is an optical flow object specifying the optical flow estimation method and its properties.

```
opticFlow = opticalFlowLKDoG(NoiseThreshold=0.0005)
```

```
opticFlow =
  opticalFlowLKDoG with properties:

              NumFrames: 3
       ImageFilterSigma: 1.5000
    GradientFilterSigma: 1
         NoiseThreshold: 5.0000e-04
```

Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,Position=[0 0 1 1],Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

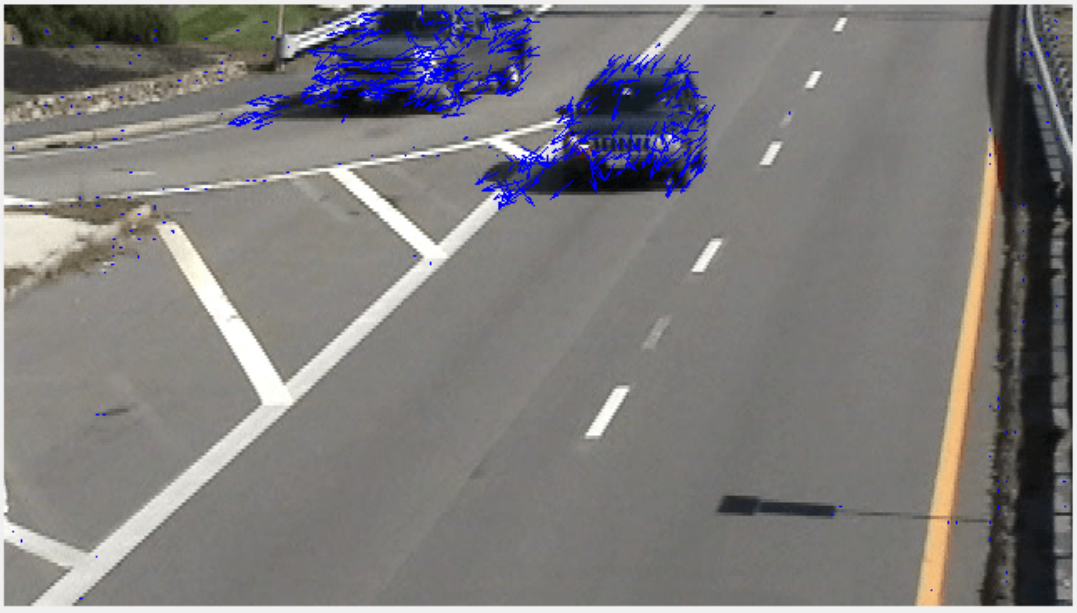
Read the image frames and convert to grayscale images. Estimate the optical flow from consecutive image frames. Display the current image frame and plot the optical flow vectors as quiver plot.

```
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
    plot(flow,DecimationFactor=[5 5],ScaleFactor=5,Parent=hPlot);
    hold off
    pause(10^-3)
end
```

Plot of Optical Flow Vectors



Plot of Optical Flow Vectors



Algorithms

To compute the optical flow between two images, you must solve this optical flow constraint equation:

$$I_x u + I_y v + I_t = 0$$

.

- I_x, I_y, and I_t are the spatiotemporal image brightness derivatives.
- u is the horizontal optical flow.
- v is the vertical optical flow.

Lucas-Kanade Derivative of Gaussian Method

The Lucas-Kanade method computes I_t using a derivative of Gaussian filter.

To solve the optical flow constraint equation for u and v:

1. Compute I_x and I_y using the following steps:
   a. Use a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the `NumFrames` property.
   b. Use a Gaussian filter and the derivative of a Gaussian filter to smooth the image using spatial filtering. Specify the standard deviation and length of the image smoothing filter using the `ImageFilterSigma` property.
2. Compute I_t between images 1 and 2 using the following steps:
   a. Use the derivative of a Gaussian filter to perform temporal filtering. Specify the temporal filter characteristics such as the standard deviation and number of filter coefficients using the `NumFrames` property.
   b. Use the filter described in step 1b to perform spatial filtering on the output of the temporal filter.
3. Smooth the gradient components, I_x, I_y, and I_t, using a gradient smoothing filter. Use the `GradientFilterSigma` property to specify the standard deviation and the number of filter coefficients for the gradient smoothing filter.
4. Solve the 2-by-2 linear equations for each pixel using the following method:
   - If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sum W^2 I_x^2 & \sum W^2 I_x I_y \\ \sum W^2 I_y I_x & \sum W^2 I_y^2 \end{bmatrix}$

     Then the eigenvalues of A are $\lambda_i = \frac{a+c}{2} \pm \frac{\sqrt{4b^2 + (a-c)^2}}{2}; i = 1, 2$
   - When the algorithm finds the eigenvalues, it compares them to the threshold, τ, that corresponds to the value you enter for the `NoiseThreshold` property. The results fall into one of the following cases:

     Case 1: $\lambda_1 \geq \tau$ and $\lambda_2 \geq \tau$

     A is nonsingular, so the algorithm solves the system of equations using Cramer's rule.

Case 2: $\lambda_1 \geq \tau$ and $\lambda_2 < \tau$

A is singular (noninvertible), so the algorithm normalizes the gradient flow to calculate u and v.

Case 3: $\lambda_1 < \tau$ and $\lambda_2 < \tau$

The optical flow, u and v, is 0.

Version History

Introduced in R2015a

References

[1] Barron, J. L., D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt. " Performance of optical flow techniques." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),*236-242. Champaign, IL: CVPR, 1992.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`opticalFlowRAFT` | `quiver` | `opticalFlow` | `opticalFlowHS` | `opticalFlowLK` | `opticalFlowFarneback`

Topics

"What Is Optical Flow?"

opticalFlowFarneback

Object for estimating optical flow using Farneback method

Description

Create an optical flow object for estimating the direction and speed of moving objects using the Farneback method. Use the object function `estimateFlow` to estimate the optical flow vectors. Using the `reset` object function, you can reset the internal state of the optical flow object.

Creation

Syntax

```
opticFlow = opticalFlowFarneback
opticFlow = opticalFlowFarneback(Name,Value)
```

Description

`opticFlow = opticalFlowFarneback` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Farneback method.

`opticFlow = opticalFlowFarneback(Name,Value)` returns an optical flow object with properties specified as one or more `Name,Value` pair arguments. Any unspecified properties have default values. Enclose each property name in quotes.

For example, `opticalFlowFarneback('NumPyramidLevels',3)`

Properties

`NumPyramidLevels` — Number of pyramid layers
3 (default) | positive scalar

Number of pyramid layers, specified as a positive scalar. The value includes the initial image as one of the layers. When you set this value to `1`, the function estimates optical flow only from the original image frame and does not perform pyramid decomposition. The recommended values are between `1` and `4`.

`PyramidScale` — Image scale
`0.5` (default) | positive scalar in the range (0,1)

Image scale, specified as a positive scalar in the range (0,1). The value specifies the rate of downsampling at each pyramid level. A value of `0.5` creates a classical pyramid, where the resolution of the pyramid reduces by a factor of two at each level. The lowest level in the pyramid has the highest resolution.

`NumIterations` — Number of search iterations per pyramid level
3 (default) | positive integer

Number of search iterations per pyramid level, specified as a positive integer. The Farneback algorithm performs an iterative search for the key points at each pyramid level, until convergence.

`NeighborhoodSize` — Size of the pixel neighborhood
5 (default) | positive integer

Size of the pixel neighborhood, specified as a positive integer. Increase the neighborhood size to increase blurred motion. The blur motion yields a more robust estimation of optical flow. A typical value for `NeighborhoodSize` is 5 or 7.

`FilterSize` — Averaging filter size
15 (default) | positive integer in the range [2, Inf)

Averaging filter size, specified as a positive integer in the range [2, Inf). After the algorithm computes the displacement (flow), the averaging over neighborhoods is done using a Gaussian filter of size (`FilterSize` * `FilterSize`). Also, the pixels close to the borders are given a reduced weight because the algorithm assumes that the polynomial expansion coefficients are less reliable there. Increasing the filter size increases the robustness of the algorithm to image noise. The larger the filter size, the greater the algorithm handles image noise and fast motion detection, making it more robust.

Object Functions

estimateFlow	Estimate optical flow
reset	Reset the internal state of the optical flow estimation object

Examples

Estimate Optical Flow Using Farneback Method

Read a video file. Specify the timestamp of the frame to be read.

```
vidReader = VideoReader('visiontraffic.avi','CurrentTime',11);
```

Create an optical flow object for estimating the optical flow using Farneback method. The output is an object specifying the optical flow estimation method and its properties.

```
opticFlow = opticalFlowFarneback
```

```
opticFlow =
  opticalFlowFarneback with properties:

    NumPyramidLevels: 3
        PyramidScale: 0.5000
       NumIterations: 3
    NeighborhoodSize: 5
          FilterSize: 15
```

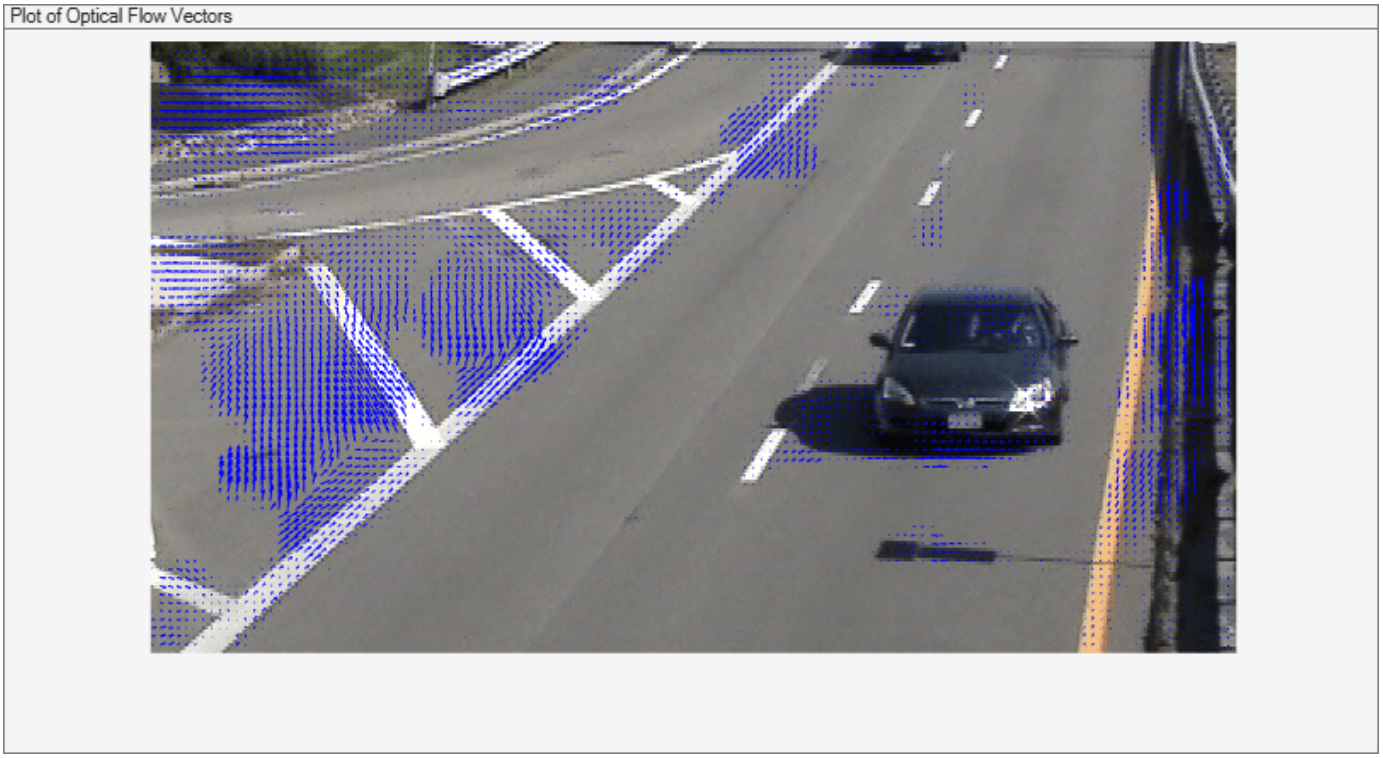
Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of Optical Flow Vectors');
hPlot = axes(hViewPanel);
```

Read the image frames and convert to grayscale images. Estimate the optical flow from consecutive image frames. Display the current image frame and plot the optical flow vectors as quiver plot.

```
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);

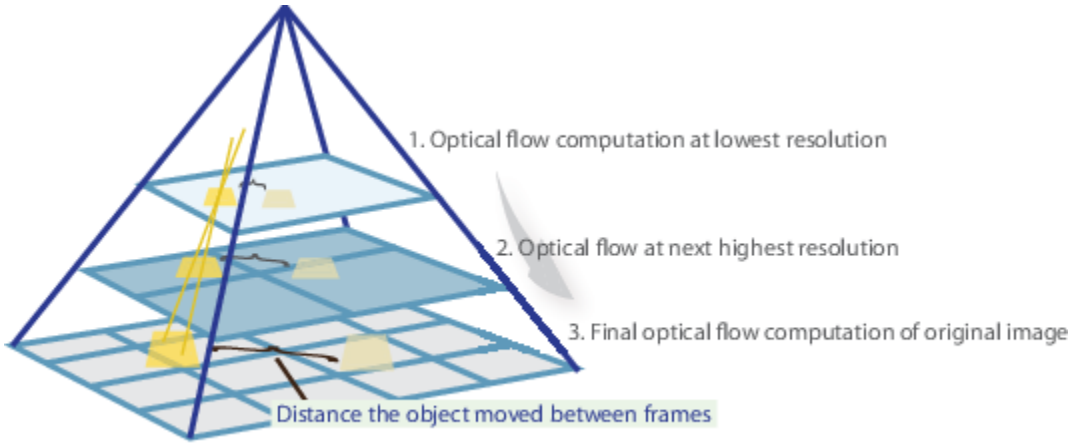
    imshow(frameRGB)
    hold on
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',2,'Parent',hPlot);
    hold off
    pause(10^-3)
end
```





Algorithms

The Farneback algorithm generates an image pyramid, where each level has a lower resolution compared to the previous level. When you select a pyramid level greater than 1, the algorithm can track the points at multiple levels of resolution, starting at the lowest level. Increasing the number of pyramid levels enables the algorithm to handle larger displacements of points between frames. However, the number of computations also increases. The diagram shows an image pyramid with three levels.



The tracking begins in the lowest resolution level, and continues until convergence. The point locations detected at a level are propagated as keypoints for the succeeding level. In this way, the algorithm refines the tracking with each level. The pyramid decomposition enables the algorithm to handle large pixel motions, which can be distances greater than the neighborhood size.

Version History

Introduced in R2015b

References

[1] Farneback, G. “Two-Frame Motion Estimation Based on Polynomial Expansion.” In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, 363 - 370. Halmstad, Sweden: SCIA, 2003.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`opticalFlowRAFT` | `opticalFlow` | `opticalFlowLKDoG` | `opticalFlowHS` | `opticalFlowLK`

Functions

`quiver`

Topics

“What Is Optical Flow?”

estimateFlow

Estimate optical flow

Syntax

```
flow = estimateFlow(opticFlow,I)
```

Description

`flow = estimateFlow(opticFlow,I)` estimates optical flow between two consecutive video frames.

Examples

Estimate Optical Flow Using Horn-Schunck Method

Create a `VideoReader` object for the input video file, `visiontraffic.avi`. Specify the timestamp of the frame to read as 11.

```
vidReader = VideoReader('visiontraffic.avi',CurrentTime=11);
```

Specify the optical flow estimation method as `opticalFlowHS`. The output is an object specifying the optical flow estimation method and its properties.

```
opticFlow = opticalFlowHS
```

```
opticFlow =
  opticalFlowHS with properties:

           Smoothness: 1
         MaxIteration: 10
    VelocityDifference: 0
```

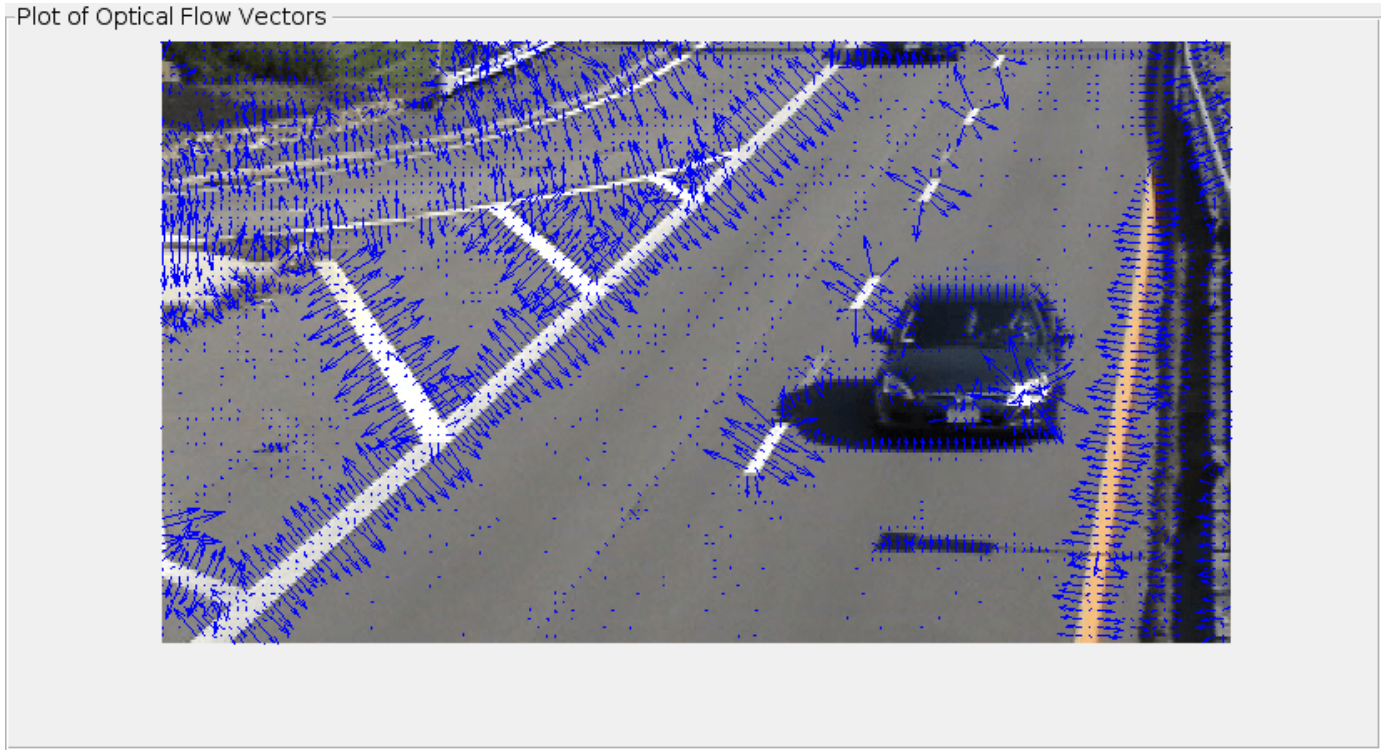
Create a custom figure window to visualize the optical flow vectors.

```
h = figure;
movegui(h);
hViewPanel = uipanel(h,Position=[0 0 1 1],Title="Plot of Optical Flow Vectors");
hPlot = axes(hViewPanel);
```

Read image frames from the `VideoReader` object and convert to grayscale images. Estimate the optical flow from consecutive image frames. Display the current image frame and plot the optical flow vectors as quiver plot.

```
while hasFrame(vidReader)
    frameRGB = readFrame(vidReader);
    frameGray = im2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    imshow(frameRGB)
    hold on
```

```
    plot(flow,DecimationFactor=[5 5],ScaleFactor=5,Parent=hPlot);
    hold off
    pause(10^-3)
end
```





Input Arguments

`opticFlow` — Object for optical flow estimation

`opticalFlowFarneback` object | `opticalFlowHS` | object | `opticalFlowLK` object | `opticalFlowLKDoG` object

Object for optical flow estimation, specified as one of the following:

- `opticalFlowFarneback` object
- `opticalFlowHS` object
- `opticalFlowLK` object
- `opticalFlowLKDoG` object
- `opticalFlowRAFT` object

The input `opticFlow` defines the optical flow estimation method and its properties used for estimating the optical flow velocity matrices.

`I` — Current video frame

2-D grayscale image

Current video frame, specified as a 2-D grayscale image of size *m*-by-*n*. The input image is generated from the current video frame read using the `VideoReader` object. The video frames in RGB format must be converted to 2-D grayscale images for estimating the optical flow.

Output Arguments

`flow` — Object for storing optical flow velocity matrices
`opticalFlow` object

Object for storing optical flow velocity matrices, returned as an `opticalFlow` object.

Algorithms

The function estimates optical flow of the input video using the method specified by the input object `opticFlow`. The optical flow is estimated as the motion between two consecutive video frames. The video frame *T* at the given instant $t_{current}$ is referred as current frame and the video frame *T-1* is referred as previous frame. The initial value of the previous frame at time $t_{current} = 0$ is set as a uniform image of grayscale value 0.

Note If you specify `opticFlow` as `opticalFlowLKDoG` object, then the estimation delays by an amount relative to the number of video frames. The amount of delay depends on the value of `NumFrames` defined in `opticalFlowLKDoG` object. The optic flow estimated for a video frame at $t_{current}$ corresponds to the video frame at time $t_{flow} = (t_{current} - (NumFrames - 1)/2)$. $t_{current}$ is the time of the current video frame.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`reset`

reset

Reset the internal state of the optical flow estimation object

Syntax

```
reset(opticFlow)
```

Description

`reset(opticFlow)` resets the internal state of the optical flow estimation object. The previous frame is reset to its initial value which is a uniform image of intensity value 0.

Input Arguments

`opticFlow` — Object for optical flow estimation

`opticalFlowFarneback` object | `opticalFlowHS` | object | `opticalFlowLK` object | `opticalFlowLKDoG` object

Object for optical flow estimation, specified as one of the following:

- `opticalFlowFarneback` object
- `opticalFlowHS` object
- `opticalFlowLK` object
- `opticalFlowLKDoG` object

The input `opticFlow` defines the optical flow estimation method and its properties used for estimating the optical flow velocity matrices.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`estimateFlow`

vision.PeopleDetector

(Removed) Detect upright people using HOG features

Note `vision.PeopleDetector` function has been removed. Use the `peopleDetectorACF` function to detect people, instead.

Description

The people detector object detects people in an input image using the Histogram of Oriented Gradient (HOG) features and a trained Support Vector Machine (SVM) classifier. The object detects unoccluded people in an upright position.

To detect people in an image:

1. Create the `vision.PeopleDetector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
peopleDetector = vision.PeopleDetector
peopleDetector = vision.PeopleDetector(model)
peopleDetector = vision.PeopleDetector(Name,Value)
```

Description

`peopleDetector = vision.PeopleDetector` returns a people detector object, `peopleDetector`, that tracks a set of points in a video.

`peopleDetector = vision.PeopleDetector(model)` creates a people detector object and sets the `ClassificationModel` property to `model`.

`peopleDetector = vision.PeopleDetector(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `peopleDetector = vision.PeopleDetector('ClassificationModel','UprightPeople_128x64')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ClassificationModel` — Name of classification model
`'UprightPeople_128x64'` (default) | `'UprightPeople_96x48'`

Name of classification model, specified as `'UprightPeople_128x64'` or `'UprightPeople_96x48'`. The pixel dimensions indicate the image size used for training.

The images used to train the models include background pixels around the person. Therefore, the actual size of a detected person is smaller than the training image size.

`ClassificationThreshold` — People classification threshold
1 (default) | nonnegative scalar value

People classification threshold, specified as a nonnegative scalar value. Use this threshold to control the classification of individual image subregions during multiscale detection. The threshold controls whether a subregion gets classified as a person. You can increase this value when there are many false detections. The higher the threshold value, the more stringent the requirements are for the classification. Vary the threshold over a range of values to find the optimum value for your data set. Typical values range from `0` to `4`.

Tunable: Yes

`MinSize` — Smallest region containing a person
`[]` (default) | two-element vector

Smallest region containing a person, specified in pixels as a two-element vector, [*height width*]. Set this property to the minimum size region containing a person. You can reduce computation time when you set this property. To do so, set this property to a value larger than the image size used to train the classification model. When you do not specify this property, the detector sets it to the image size used to train the classification model.

Tunable: Yes

`MaxSize` — Largest region containing a person
`[]` (default) | two-element vector

Largest region that contains a person, specified in pixels as a two-element vector, [*height width*]. Set this property to the largest region containing a person. You can reduce computation time when you set this property. To do so, set this property to a value smaller than the size of the input image. When you do not specify this property, the detector sets it to the input image size. This property is tunable.

`ScaleFactor` — Multiscale object detection scaling
`1.05` (default) | numeric value greater than `1.0001`

Multiscale object detection scaling, specified as a value greater than `1.0001`. The scale factor incrementally scales the detection resolution between `MinSize` and `MaxSize`. You can set the scale factor to an ideal value using:
`size(I)/(size(I)-0.5)`
The object calculates the detection resolution at each increment.

`round(`*TrainingSize*`*(ScaleFactor`^{N}`))`

In this case, the *TrainingSize* is `[128 64]` for the `'UprightPeople_128x64'` model and `[96 48]` for the `'UprightPeople_96x48'` model. *N* is the increment. Decreasing the scale factor can

increase the detection accuracy. However, doing so increases the computation time. This property is tunable.

`WindowStride` — Detection window stride
`[8 8]` (default) | scalar | two-element vector

Detection window stride in pixels, specified as a scalar or a two-element vector, [*x y*]. The detector uses the window stride to slide the detection window across the image. When you specify this value as a vector, the first and second elements are the stride size in the *x* and *y* directions. When you specify this value as a scalar, the stride is the same for both *x* and *y*. Decreasing the window stride can increase the detection accuracy. However, doing so increases computation time. Increasing the window stride beyond `[8 8]` can lead to a greater number of missed detections. This property is tunable.

`MergeDetections` — Merge detection control
`true` | `false`

Merge detection control, specified as `true` or `false`. This property controls whether similar detections are merged. Set this property to `true` to merge bounding boxes using a mean-shift based algorithm. Set this property to `false` to output the unmerged bounding boxes.

For more flexibility and control of merging parameters, you can use the `selectStrongestBbox` function in place of the `MergeDetections` algorithm. To do this, set the `MergeDetections` property to `false`. See the “Tracking Pedestrians from a Moving Car” example, which shows the use of the people detector and the `selectStrongestBbox` function.

`UseROI` — Use region of interest
`false` (default) | `true`

Use region of interest, specified as `true` or `false`. Set this property to `true` to detect objects within a rectangular region of interest within the input image.

Usage

Syntax

```
bboxes = peopleDetector(I)
[bboxes, scores] = peopleDetector(I)
[ ___ ] = peopleDetector(I,roi)
```

Description

`bboxes = peopleDetector(I)` performs multiscale object detection on the input image, `I` and returns an *M*-by-4 matrix defining *M* bounding boxes. *M* represents the number of detected people. Each row of the output matrix, `BBOXES`, contains a four-element vector, [*x y* width height]. This vector specifies, in pixels, the upper-left corner and size, of a bounding box. When no people are detected, the object returns an empty vector. The input image, `I`, must be a grayscale or truecolor (RGB) image.

`[bboxes, scores] = peopleDetector(I)` additionally returns a confidence value for the detections.

`[ ___ ] = peopleDetector(I,roi)` detects people within the rectangular search region, `roi`.

Input Arguments

`I` — Input image
grayscale | truecolor (RGB)

Input image, specified as grayscale or truecolor (RGB).

`roi` — Rectangular region of interest
four-element vector (default)

Rectangular region of interest within image `I`, specified as a four-element vector, [*x y width height*].

`model` — Classification model
`'UprightPeople_128x64'` (default) | `'UprightPeople_96x48'`

Classification model, specified as `'UprightPeople_128x64'` or `'UprightPeople_96x48'`.

Output Arguments

`peopleDetector` — People detector
object (default)

People detector object, returned as an object. The detector detects people in an input image using the Histogram of Oriented Gradient (HOG) features and a trained SVM classifier. The object detects unoccluded people in an upright position.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Detect People

1. Create a people detector and load the input image.

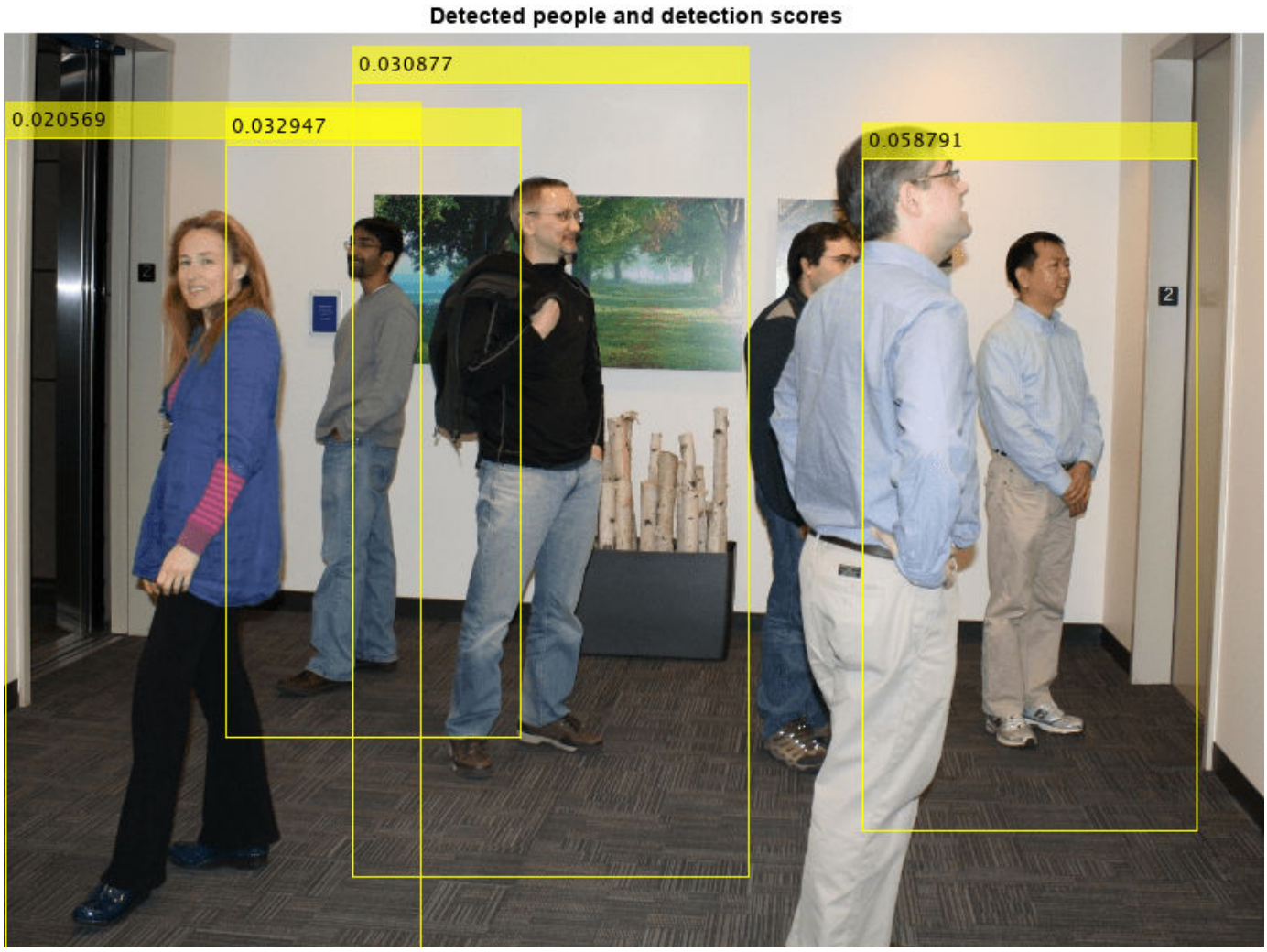
```
peopleDetector = vision.PeopleDetector;
I = imread('visionteam1.jpg');
```

2. Detect people using the people detector object.

```
[bboxes,scores] = peopleDetector(I);
```

3. Annotate detected people.

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure, imshow(I)
title('Detected people and detection scores');
```



Version History

Introduced in R2012b

R2024b: `vision.PeopleDetector` function has been removed

Errors starting in R2024b

`vision.PeopleDetector` function has been removed. Use the `peopleDetectorACF` function to detect people, instead.

R2024a: `vision.PeopleDetector` function will be removed

Warns starting in R2024a

`vision.PeopleDetector` function will be removed in a future release. Use the `peopleDetectorACF` function to detect people, instead.

References

[1] Dalal, N. and B. Triggs. “Histograms of Oriented Gradients for Human Detection,”*Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2005, pp. 886-893.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`peopleDetectorACF` | `vision.CascadeObjectDetector` | `insertObjectAnnotation` | `extractHOGFeatures`

Topics

“Tracking Pedestrians from a Moving Car”
“Multiple Object Tracking”

instanceSegmentationMetrics

Instance segmentation quality metrics

Description

Use the `instanceSegmentationMetrics` object and its object functions to evaluate the quality of instance segmentation results.

An `instanceSegmentationMetrics` object stores instance segmentation quality metrics for a set of images, such as the average precision (AP) and precision and recall, computed per class and per image. To compute the AP and precision recall metrics, pass the `instanceSegmentationMetrics` object to the `averagePrecision` or the `precisionRecall` object functions, respectively. To compute the confusion matrix, pass the `instanceSegmentationMetrics` object to the `confusionMatrix` object function. Evaluate the summary of all metrics across all classes and all images in the data set using the `summarize` object function.

Creation

Create an `instanceSegmentationMetrics` object using the `evaluateInstanceSegmentation` function.

Properties

`ClassMetrics` — Metrics per class
table

This property is read-only.

Metrics per class, specified as a table with *C* rows, where *C* is the number of classes in the instance segmentation. `ClassMetrics` has these columns, corresponding to these instance segmentation metrics:

- `mAP` — Mean average precision calculated by averaging over all thresholds specified in the `threshold` argument for a class.
- `AP` — Average precision calculated for a class at each overlap threshold in `OverlapThreshold`, returned as a *numThresh*-by-1 array, where *numThresh* is the number of overlap thresholds.
- `Precision` — Precision values, returned as a *numThresh*-by-(*numPredictions*+1) matrix, where *numPredictions* is the number of predicted object masks. Precision is the ratio of the number of true positives (*TP*) and the total number of predicted positives.
  Precision = *TP* / (*TP* + *FP*)

  FP is the number of false positives, or misclassified pixels that comprise object instances. Larger precision scores imply that most object instance classes match ground truth objects.
- `Recall` — Recall values, returned as a *numThresh*-by-(*numPredictions*+1) matrix, where *numPredictions* is the number of predicted object masks. Recall is the ratio of the number of true positives (*TP*) and the total number of ground truth positives.

Recall = *TP* / (*TP* + *FN*)

FN is the number of false negatives, or the number of pixels that comprise instances that have been misclassified as background pixels. Larger recall scores imply that most ground truth objects are detected.

`ImageMetrics` — Metrics for each image
table

This property is read-only.

Metrics for each image in the data set, specified as a table with *numImages* rows, where *numImages* is the number of images in the data set. `ImageMetrics` has these columns, corresponding to these instance segmentation metrics:

- `NumObjects` — Number of objects in the ground truth data in each image, stored as a positive integer.
- `mAP` — Mean average precision (mAP), calculated by averaging the average precision (AP) across all classes at each overlap threshold in the `OverlapThreshold` property, stored as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds. Specify the overlap thresholds for an image using the `threshold` argument.
- `mAPOverlapAvg` — Mean average precision (mAP), calculated by averaging the AP across all classes and all overlap thresholds specified by the `OverlapThreshold` property, stored as a numeric scalar.

`ClassNames` — Class names
array of strings | cell array of character vectors

Class names of segmented objects, specified as an array of strings.

Example: `["sky" "grass" "building" "sidewalk"]`

`OverlapThreshold` — Overlap threshold
numeric scalar | numeric vector

Overlap threshold, specified as a numeric scalar or numeric vector. When the intersection over union (IoU) of the pixels in the predicted object mask and ground truth object mask is equal to or greater than the overlap threshold, the prediction is considered a true positive.

IoU, or the Jaccard Index, is the number of pixels in the intersection of the binary masks divided by the number of pixels in the union of the masks. In other words, IoU is the ratio of correctly classified pixels to the total number of pixels that are assigned that class by the ground truth and the predictor.

Object Functions

averagePrecision	Evaluate average precision metric of instance segmentation results
confusionMatrix	Compute confusion matrix of instance segmentation results
precisionRecall	Get precision recall metrics of instance segmentation results
summarize	Summarize instance segmentation performance metrics at data set and class level
metricsByArea	Evaluate instance segmentation across object mask size ranges

Version History

Introduced in R2022b

R2024b: `ConfusionMatrix`, `NormalizedConfusionMatrix`, and `DatasetMetrics` properties have been removed
Not recommended starting in R2024b

The `ConfusionMatrix`, `NormalizedConfusionMatrix`, and `DatasetMetrics` properties of the `instanceSegmentationMetrics` object have been removed.

To update your code to compute the confusion matrix, replace instances of the `ConfusionMatrix` and `NormalizedConfusionMatrix` properties with the `confusionMatrix` object function.

To compute the summary of the instance segmentation quality metrics over the entire data set or over each class, use the `summarize` object function.

To compute precision, recall, and confidence scores for all classes in the data set, or at specified classes and overlap thresholds, use the `precisionRecall` object function.

To compute average precision (AP) for all classes and overlap thresholds in the data set, or specify the classes and overlap thresholds for which to compute AP, use the `averagePrecision` object function.

For an example that uses the new `instanceSegmentationMetrics` object functions to evaluate instance segmentation results, see the “Perform Instance Segmentation Using SOLOv2” example.

R2024b: Table columns in `ClassMetrics` and `ImageMetrics` properties have been renamed
Behavior change in future release

These table columns of the `ClassMetrics` and `ImageMetrics` properties have been renamed.

`instanceSegmentationMetrics` Property	Renamed Columns
`ClassMetrics`	• `mAP`, or the average precision (AP) averaged over all overlap thresholds for each class, has been renamed to `APOverlapAvg`. • `mLAMR`, or the log-average miss rate for each class averaged over all specified overlap thresholds, has been renamed to `LAMROverlapAvg`. • `mAOS`, or the average orientation similarity for each class averaged over all the specified overlap thresholds, has been renamed to `AOSOverlapAvg`.

instanceSegmentationMetrics Property	Renamed Columns
ImageMetrics	• AP, or the AP across all classes at each overlap threshold, has been renamed to mAP. • mAP, or the AP averaged across all classes and all overlap thresholds, has been renamed to mAPOverlapAvg. • mLAMR, or the log-average miss rate for each class averaged over all specified overlap thresholds, has been renamed to LAMROverlapAvg. • mAOS, or the average orientation similarity for each class averaged over all the specified overlap thresholds, has been renamed to AOSOverlapAvg.

See Also

solov2 | trainSOLOV2 | evaluateInstanceSegmentation | maskrcnn | trainMaskRCNN

Topics

“Get Started with SOLOv2 for Instance Segmentation”
“Getting Started with Mask R-CNN for Instance Segmentation”
“Perform Instance Segmentation Using Mask R-CNN”
“Get Started with Instance Segmentation Using Deep Learning”

External Websites

https://cocodataset.org/#detection-eval

metricsByArea

Evaluate instance segmentation across object mask size ranges

Syntax

```
areaMetrics = metricsByArea(metrics,objectAreaRanges)
areaMetrics = metricsByArea(metrics,objectAreaRanges,ClassNames)
```

Description

`areaMetrics = metricsByArea(metrics,objectAreaRanges)` evaluates the instance segmentation metrics detection performance for the object size range or ranges, specified as bounding box areas, `objectAreaRanges`. The function evaluates the specified bounding box size range within the existing instance segmentation `metrics`.

`areaMetrics = metricsByArea(metrics,objectAreaRanges,ClassNames)` also specifies the names of the classes for which to evaluate area-based metrics.

Input Arguments

`metrics` — Instance segmentation metrics

M-by-4 numeric matrix | vector

Instance segmentation metrics, specified as an `instanceSegmentationMetrics` object.

`objectAreaRanges` — Area of rectangular bounding boxes

M-by-2 numeric matrix | vector

Area of rectangular bounding boxes which are axis-aligned and surround the object mask, specified as one of the following:

- An *M*-by-2 numeric matrix, with each row specifying the upper (exclusive) and lower (inclusive) limits of an box area range in evaluation. For example, if the input is $[x_1\ x_2;\ x_3\ x_4]$. The evaluated bounding box area ranges are $x_1 \leq$ area $< x_2$ and $x_3 \leq$ area $< x_4$.
- A vector with $M+1$ elements, specifying the edge values of M contiguous area ranges. Each range is inclusive in the lower limit, and exclusive in the upper limit. For example, if the input is $[x_1\ x_2\ x_3]$, the area ranges evaluated are $x_1 \leq$ area $< x_2$ and $x_2 \leq$ area $< x_3$.

`ClassNames` — Names of classes

`''` (default) | string scalar | vector

Names of the classes for which to evaluate area-based metrics. If multiple classes are specified, the function computes averaged metrics across the classes. By default, all classes will be considered and the returned metrics will be the average of all classes.

Output Arguments

`areaMetrics` — Metrics by area

table

Metrics by area, returned as a table with *M* rows. Each row contains metrics using objects within an area range (for a total of *M* area ranges). The columns are specified by these table variables.

- `AreaRange`: The upper and lower limits of each area range, returned as a two-element row vector.
- `NumObjects`: The number of objects falling within each area range, returned as a positive integer.
- `mAP`: Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.
- `mAPOverlapAvg`: Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property, returned as a numeric scalar.

Version History

Introduced in R2023b

See Also

`instanceSegmentationMetrics` | `evaluateInstanceSegmentation`

averagePrecision

Evaluate average precision metric of instance segmentation results

Syntax

```
ap = averagePrecision(metrics)
ap = averagePrecision(metrics,Name=Value)
```

Description

`ap = averagePrecision(metrics)` evaluates the average precision (AP) for all classes and overlap thresholds of `metrics`.

AP aggregates the precision across different recall levels, providing a single metric to assess the overall ability of an object detector to identify objects accurately while minimizing false detections.

`ap = averagePrecision(metrics,Name=Value)` specifies options for the average precision evaluation using one or more name-value arguments. For example, `ClassNames=["cars" "people"]` specifies to evaluate the average precision metric for the cars and people classes.

Input Arguments

`metrics` — Instance segmentation performance metrics

`instanceSegmentationMetrics` object

Instance segmentation performance metrics, specified as an `instanceSegmentationMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `averagePrecision(metrics,ClassNames=[cars people])` specifies to evaluate the average precision metric for the cars and people classes.

`ClassNames` — Class names

string array | cell array of character vectors

Class names of segmented objects, specified as an array of strings or a cell array of character vectors. By default, the `averagePrecision` function returns the average precision metrics for all classes specified by the `ClassNames` property of the `instanceSegmentationMetrics` object `metrics`.

`OverlapThreshold` — Overlap threshold

numeric scalar | numeric vector

Overlap threshold to use for evaluating the average precision, specified as a numeric scalar or numeric vector of box overlap threshold values. To evaluate multiple overlap thresholds, specify this argument as a numeric vector. By default, the `averagePrecision` object function returns the

average precision metrics for all overlap thresholds specified by the `OverlapThreshold` property of the `instanceSegmentationMetrics` object `metrics`.

Output Arguments

ap — Average precision

M-by-*N* matrix

Average precision (AP) for specified classes and overlap thresholds, specified as an *M*-by-*N* matrix. *M* is the number of classes in the `ClassNames` property and *N* is the number of specified overlap thresholds `OverlapThreshold`.

The AP metric evaluates instance segmentation performance by quantifying the accuracy of the model in identifying object instances across different confidence thresholds, enabling you to assess both the precision (correctness of detections) and recall (completeness of detections).

Version History

Introduced in R2024b

See Also

`instanceSegmentationMetrics` | `evaluateInstanceSegmentation`

Topics

“Perform Instance Segmentation Using SOLOv2”

confusionMatrix

Compute confusion matrix of instance segmentation results

Syntax

```
[confMat,confusionClassNames] = confusionMatrix(metrics)
[confMat,confusionClassNames] = confusionMatrix(metrics,Name=Value)
```

Description

`[confMat,confusionClassNames] = confusionMatrix(metrics)` computes the confusion matrix for all overlap thresholds in the `OverlapThreshold` property of `metrics`, and returns the confusion matrix and the class names. The function considers all segmentations to compute the confusion matrix.

`[confMat,confusionClassNames] = confusionMatrix(metrics,Name=Value)` specifies one or more name-value arguments. For example, `ScoreThreshold=0.3` specifies to disregard segmentations with a confidence score value below `0.3` when computing the confusion matrix.

Input Arguments

`metrics` — Instance segmentation metrics

`instanceSegmentationMetrics` object

Instance segmentation performance metrics, specified as an `instanceSegmentationMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `confusionMatrix(metrics,ScoreThreshold=0.3)` specifies segmentations with a confidence score value below `0.3` to be disregarded when computing the confusion matrix.

`ScoreThreshold` — Confidence score threshold values

`0` (default) | numeric scalar in range `[0, 1]` | numeric vector

Confidence score threshold values to use for computing the confusion matrix, specified as a numeric scalar in the range `[0, 1]` or a numeric vector of values in the range `[0, 1]`. The function filters out predictions with confidence scores less than the threshold value when evaluating the confusion matrix.

Increase this value to reduce the number of false positives, at the possible expense of missing some true positives.

`OverlapThreshold` — Overlap threshold values

numeric scalar | numeric vector

Overlap threshold values to use for computing the confusion matrix, specified as a numeric scalar or numeric vector. Each overlap threshold value must be an element of the `OverlapThreshold` property of the `instanceSegmentationMetrics` object. By default, the function returns the confusion matrix computed at all the overlap thresholds specified by the `OverlapThreshold` property.

`Normalize` — Confusion matrix normalization
`false` or `0` (default) | `true` or `1`

Confusion matrix normalization option, specified as a numeric or logical `0` (or `false`) or `1` (or `true`). If you specify `Normalize` as `true`, the function normalizes the elements of the confusion matrix in the `confMat` argument by the number of objects known to belong to each class. For each overlap threshold, each element (*i*, *j*) in the normalized confusion matrix is the count of objects known to belong to class *i*, but predicted to belong to class *j*, divided by the total number of objects predicted in class *j*.

For other normalization options, see “Tips” on page 2-1082.

Output Arguments

`confMat` — Confusion matrix
M-by-*N* cell array

Confusion matrix, returned as an *M*-by-*N* cell array. *M* is the number of confidence score thresholds specified by the `ScoreThreshold` argument, and *N* is the number of overlap thresholds specified by the `OverlapThreshold` argument.

Each element of the `confMat` cell array is a square numeric matrix of size (*C*+1)-by-(*C*+1), where *C* is the number of classes. Each element (*i*, *j*) of the matrix is the count of *matched* bounding boxes predicted to belong to class *j*, but that have a ground truth annotation class of *i*.

Row and column *C*+1 in the confusion matrix correspond to these *unmatched* conditions:

- Column *C*+1 — Undetected objects, or *false negative* predictions. Each element in this column is the number of ground truth annotations of the corresponding class unmatched with any predicted bounding box.
- Row *C*+1 — Incorrect predictions, or *false positive* predictions. Each element in this row is the number of predicted bounding boxes of the corresponding class unmatched with any ground truth annotation.

`confusionClassNames` — Class names
string array

Class names corresponding to the returned confusion matrices, returned as a string array of size *C* +1. The elements of the string array correspond to the classes stored in the `ClassNames` property of the `instanceSegmentationMetrics` object `metrics` and an extra background class. The background class, `"background"`, corresponds to the row and column *C*+1 of the confusion matrix, which count the false positives and false negatives.

Tips

Normalize the confusion matrix output of the `confusionMatrix` function in these ways:

- Specify the `Normalize` name-value argument as `true` to divide each entry of the confusion matrix, `confMat`, by the sum of the corresponding column, so each value represents the proportion of predicted bounding boxes for a given class.
- For additional normalization options, use the `confusionchart` function. Pass the confusion matrix, `confMat`, to the `confusionchart` function, and specify the `Normalization` name-value argument. You can use this option to normalize the confusion matrix by row or by the total number of detected bounding boxes, in addition to column normalization.

Version History

Introduced in R2024b

See Also

`instanceSegmentationMetrics` | `evaluateInstanceSegmentation`

objectDetectionMetrics

Object detection quality metrics

Description

Use the `objectDetectionMetrics` object and its object functions to evaluate the quality of object detection results.

An `objectDetectionMetrics` object stores object detection quality metrics, such as the average precision (AP) and precision recall, computed per class. To compute the AP and precision recall metrics, pass the `objectDetectionMetrics` object to the `averagePrecision` or the `precisionRecall` object functions, respectively. To compute the confusion matrix, pass the `objectDetectionMetrics` object to the `confusionMatrix` object function. Evaluate the summary of all metrics across all classes and all images in the data set using the `summarize` object function.

To get started with evaluating object detector performance using the performance metrics, see "Evaluate Object Detector Performance".

Creation

Create an `objectDetectionMetrics` object by using the `evaluateObjectDetection` function.

Properties

`ClassMetrics` — Metrics per class
table

This property is read-only.

Metrics per class, stored as a table with *C* rows, where *C* is the number of classes in the object detection. If additional metrics are not specified through the `AdditionalMetrics` argument of the `evaluateObjectDetection` function, the `ClassMetrics` table has five columns, corresponding to these object detection metrics.

- `NumObjects` — Number of objects in the ground truth data for a class.
- `AP` — Average precision (AP) for each class at each overlap threshold in `OverlapThreshold`, stored as a *numThresh*-by-1 array, where *numThresh* is the number of overlap thresholds.
- `APOverlapAvg` — AP averaged over all overlap thresholds. Specify the overlap thresholds for a class using the `threshold` argument.
- `Precision` — Precision values, stored as a *numThresh*-by-(*numPredictions*+1) matrix, where *numPredictions* is the number of predicted bounding boxes. Precision is the ratio of the number of true positives (*TP*) and the total number of predicted positives.
  Precision = *TP* / (*TP* + *FP*)

  FP is the number of false positives. Larger precision scores imply that most detected objects match ground truth objects.

- `Recall` — Recall values, stored as a *numThresh*-by-(*numPredictions*+1) matrix, where *numPredictions* is the number of predicted boxes. Recall is the ratio of the number of true positives (*TP*) and the number of groundtruths – the sum of true positives (*TP*) and false negatives (*FN*).
  Recall = *TP* / (*TP* + *FN*)

  FN is the number of false negatives. Larger recall scores indicate that more of ground truth objects are detected.

  Note For each overlap threshold (row in the `Recall` matrix), the recall values (columns of the `Recall` matrix) are sorted in the order of decreasing confidence score associated with each detection.

For information on optional additional metrics for this table, see the `AdditionalMetrics` argument of the `evaluateObjectDetection` function.

`ClassNames` — Class names
array of strings | cell array of character vectors

Class names of detected objects, stored as an array of strings or a cell array of character vectors.

Example: `{"sky"} {"grass"} {"building"} {"sidewalk"}`

`OverlapThreshold` — Overlap threshold
numeric scalar | numeric vector

Overlap threshold, stored as a numeric scalar or numeric vector of box overlap threshold values over which the mean average precision is computed. When the intersection over union (IoU) of the pixels in the ground truth bounding box and the predicted bounding box is equal to or greater than the overlap threshold, the detection is considered a match to the ground truth (true positive). The IoU is the number of pixels in the intersection of the bounding boxes divided by the number of pixels in the union of the bounding boxes.

`NumClasses` — Number of classes in data set
non-negative integer

This property is read-only.

Number of classes in the data set, specified as a non-negative integer.

`NumImages` — Number of images in data set
non-negative integer

This property is read-only.

Number of images in the data set, specified as a non-negative integer.

Object Functions

imageMetrics	Evaluate per-image object detection performance metrics
averagePrecision	Evaluate average precision metric of object detection results
confusionMatrix	Compute confusion matrix of object detection results
precisionRecall	Get precision recall metrics of object detection results
summarize	Summarize object detection performance metrics at data set and class level

metricsByArea Evaluate detection performance across object size ranges

Examples

Plot Precision-Recall Curve for Object Detection

Load a table containing images and ground truth bounding box labels. The first column contains the images, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Display the first 5 rows of the training data table.

```
disp(trainingData(1:5,:))
```

```
                                imageFilename                                    vehicle
    ____________________________________________________________________    _______________

    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00001.jpg"    {[126 78 20 16]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00002.jpg"    {[100 72 35 26]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00003.jpg"    {[ 62 69 26 19]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00004.jpg"    {[ 71 64 22 21]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00005.jpg"    {[147 71 25 18]}
```

Create a `boxLabelDatastore` using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load Pretrained Object Detector

Load a pretrained YOLO v2 object detector trained to detect vehicles into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Evaluate and Plot Object Detection Metrics

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This helps you evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Use `evaluateObjectDetection` to compute metrics for evaluating the performance of an object detector.

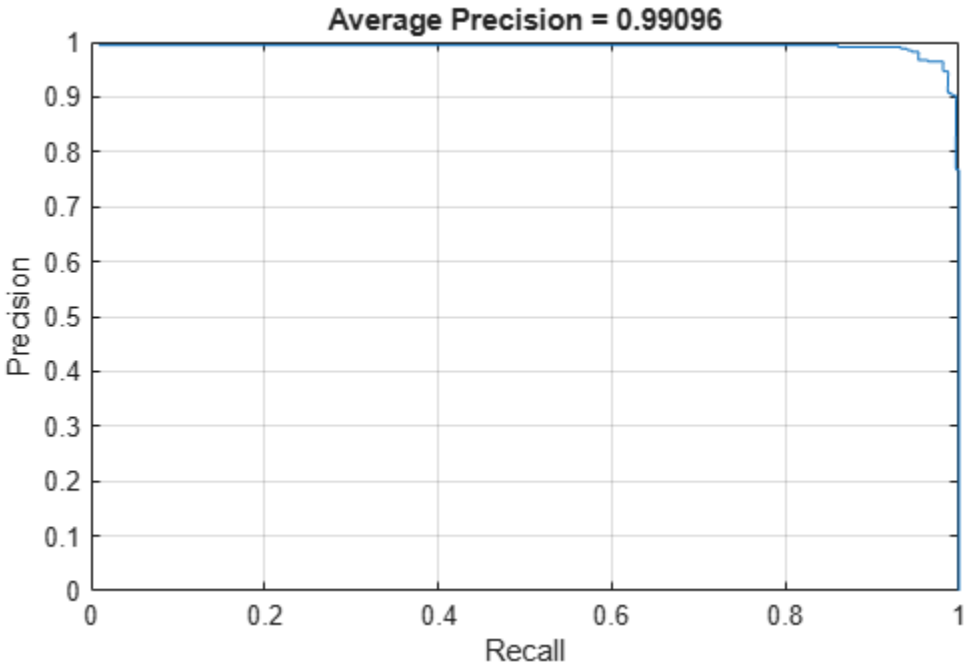
```
metrics = evaluateObjectDetection(results,blds);
```

Return the precision and recall metrics for the vehicle class using the `precisionRecall` object function.

```
[recall,precision,scores] = precisionRecall(metrics);
ap = averagePrecision(metrics);
```

Plot the precision-recall curve for the vehicle class, the only class in the data set. Compute the average precision (AP) using the `averagePrecision` object function.

```
figure
plot(recall{1},precision{1})
grid on
title("Average Precision = " + ap);
xlabel("Recall");
ylabel("Precision");
```



Compute the summary of the object detection metrics for the data set using the `summarize` object function.

```
[summaryDataset,summaryClass] = summarize(metrics);
disp(summaryDataset)
```

NumObjects	mAPOverlapAvg	mAP0.5
336	0.99096	0.99096

Version History

Introduced in R2023b

R2026a: Evaluate per-image object detection metrics

To evaluate the per-image object detection metrics for all images or a subset of images in the data set, use the `imageMetrics` object function.

To update your code to compute per-image metrics, replace instances of the `ImageMetrics` property with the `imageMetrics` object function. For an example, see the “Multiclass Object Detection Using YOLO v2 Deep Learning” example.

R2024b: `ConfusionMatrix`, `NormalizedConfusionMatrix`, and `DatasetMetrics` properties have been removed
Not recommended starting in R2024b

The `ConfusionMatrix`, `NormalizedConfusionMatrix`, and `DatasetMetrics` properties of the `objectDetectionMetrics` object have been removed.

To update your code to compute the confusion matrix, replace instances of the `ConfusionMatrix` and `NormalizedConfusionMatrix` properties with the `confusionMatrix` object function. For an example, see the "Evaluate Detector Errors Using Confusion Matrix" section of the “Multiclass Object Detection Using YOLO v2 Deep Learning” example.

To compute the summary of the object detection metrics over the entire data set or over each class, use the `summarize` object function.

To compute precision, recall, and confidence scores for all classes in the data set, or at specified classes and overlap thresholds, use the `precisionRecall` object function.

To compute average precision (AP) for all classes and overlap thresholds in the data set, or specify the classes and overlap thresholds for which to compute AP, use the `averagePrecision` object function.

R2024b: Table columns in `ClassMetrics` and `ImageMetrics` properties have been renamed
Behavior change in future release

These table columns of the `ClassMetrics` and `ImageMetrics` properties have been renamed.

`objectDetectionMetrics` property	Renamed Columns
`ClassMetrics`	• `mAP`, or the average precision (AP) averaged over all overlap thresholds for each class, has been renamed to `APOverlapAvg`. • `mLAMR`, or the log-average miss rate for each class averaged over all specified overlap thresholds, has been renamed to `LAMROverlapAvg`. • `mAOS`, or the average orientation similarity for each class averaged over all the specified overlap thresholds, has been renamed to `AOSOverlapAvg`.

objectDetectionMetrics property	Renamed Columns
ImageMetrics	• AP, or the AP across all classes at each overlap threshold, has been renamed to mAP. • mAP, or the AP averaged across all classes and all overlap thresholds, has been renamed to mAPOverlapAvg. • mLAMR, or the log-average miss rate for each class averaged over all specified overlap thresholds, has been renamed to LAMROverlapAvg. • mAOS, or the average orientation similarity for each class averaged over all the specified overlap thresholds, has been renamed to AOSOverlapAvg.

See Also

Apps
Object Detector Analyzer

Functions
evaluateObjectDetection | imageMetrics | averagePrecision | confusionMatrix | metricsByArea | precisionRecall

Topics
“Evaluate Object Detector Performance”
“Calibrate Object Detection Confidence Scores”
Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

External Websites
https://cocodataset.org/#detection-eval

precisionRecall

Get precision recall metrics of instance segmentation results

Syntax

```
[precision,recall,scores] = precisionRecall(metrics)
[precision,recall,scores] = precisionRecall(metrics,Name=Value)
```

Description

`[precision,recall,scores] = precisionRecall(metrics)` gets the precision, recall, and prediction score values for all classes in the `ClassNames` property and all overlap thresholds in the `OverlapThreshold` property of the `instanceSegmentationMetrics` object `metrics`.

`[precision,recall,scores] = precisionRecall(metrics,Name=Value)` specifies precision and recall evaluation options using one or more name-value arguments. For example, `ClassNames=["cars" "people"]` specifies to get the precision recall metrics for the cars and people classes.

Input Arguments

`metrics` — instance segmentation performance metrics

`instanceSegmentationMetrics` object

instance segmentation performance metrics, specified as an `instanceSegmentationMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[precision,recall,scores]=precisionRecall(metrics,ClassNames=[cars people])` specifies the precision recall metric to be evaluated for the cars and people classes.

`ClassNames` — Class names

string array | cell array of character vectors

Class names of detected objects, specified as an array of strings or a cell array of character vectors. By default, the `precisionRecall` function returns the precision recall metrics for all classes specified by the `ClassNames` property of the `instanceSegmentationMetrics` object.

`OverlapThresholds` — Overlap threshold

numeric scalar | numeric vector

Overlap threshold, or Intersection over Union (IoU) threshold, for which to get the precision recall metrics, specified as a numeric scalar or numeric vector of box overlap threshold values. By default, the `precisionRecall` object function returns the precision and recall metrics for all overlap

thresholds specified by the `OverlapThreshold` property of the `instanceSegmentationMetrics` object. To learn more about how to use overlap thresholds to evaluate instance segmentation results, see “Evaluate Network Performance Using Precision Recall at Multiple Overlap Thresholds” on page 2-1091.

Output Arguments

`precision` — Precision

M-by-*N* cell array

Precision, returned as an *M*-by-*N* cell array. *M* is the number of classes in the `ClassNames` property, and *N* is the number of overlap thresholds in the `OverlapThreshold` property of the `instanceSegmentationMetrics` object `metrics`. Each element of the cell array contains a (*numPredictions*+1)-element numeric vector of precision values, sorted in descending order of prediction confidence scores. *numPredictions* is the number of predicted object masks.

Precision is the ratio of the number of true positives (TP) and the total number of predicted positives. Larger precision scores indicate that more predicted object masks match ground truth objects. To learn more about the precision metric, see “Compute Precision and Recall” on page 2-1092.

`recall` — Recall

M-by-*N* cell array

Recall, returned as an *M*-by-*N* cell array. *M* is the number of classes in the `ClassNames` property, and *N* is the number of overlap thresholds in the `OverlapThreshold` property of the `instanceSegmentationMetrics` object `metrics`. Each element of the cell array contains a (*numPredictions*+1)-element numeric vector of recall values, sorted in order of prediction confidence scores. *numPredictions* is the number of predicted object masks.

Recall is the ratio of true positives (TP) to the sum of true positives and false negatives (FN), where false negatives represent instances in the image that the model incorrectly classified as background or missed entirely. Larger recall scores indicate that the instance segmentation model is effectively identifying most of the actual instances within the images. To learn more about the recall metric, see “Compute Precision and Recall” on page 2-1092.

`scores` — Confidence score

M-by-1 cell array

Confidence score for each instance segmentation, returned as an *M*-by-1 cell array. *M* is the number of classes in the `ClassNames` property. Each element of the array is a (*numPredictions*+1)-element numeric vector. *numPredictions* is the number of predicted object masks.

More About

Evaluate Network Performance Using Precision Recall at Multiple Overlap Thresholds

To evaluate how precisely a network can localize object instances, plot the precision recall curve for a range of overlap thresholds.

At higher overlap thresholds, the criteria for how well the object masks and ground truth masks must overlap to be a true positive (TP) are stricter. Therefore, average precision is lower at higher overlap thresholds, which corresponds to a decrease in the area under the precision-recall curve, which plots

the recall against precision. Plotting the precision-recall curves across different overlap thresholds can help you evaluate the instance segmentation network in these ways.

- Evaluate how well the network identifies object classes at higher overlap thresholds.
- Identify specific overlap thresholds where network performance is significantly lower, highlighting areas for model improvement.
- Choose an overlap threshold that balances the tradeoff between identifying as many object instances as possible (high recall) and ensuring high accuracy in those detections (high precision).
- Use precision-recall curves at different overlap thresholds for a more granular comparison between instance segmentation models than single metric evaluations. Nuances in performance can be obscured when considering only one overlap threshold or when relying solely on aggregated metrics such as mean average precision.

Algorithms

Compute Precision and Recall

Precision recall is a powerful tool you can use to evaluate instance segmentation network performance. The `precisionRecall` function computes the precision and recall in these steps.

1. Sorts object instances in descending order based on their confidence scores. Higher confidence scores indicate that the model is more certain that the instance segmentation is correct.
2. Identifies true positives (TP) and false positives (FP) according to the specified overlap threshold criteria. A segmentation is considered a TP if its IoU is greater to or equal to the specified overlap threshold. A segmentation is considered an FP if it does not match any ground truth object or if it matches a ground truth object but the IoU is below the specified overlap threshold.
3. Starting with the instance segmentation with the highest confidence score, the `precisionRecall` function iterates through each instance and calculates the cumulative number of TPs and FPs at each step. This step-by-step process enables the calculation of precision and recall at every confidence level. At each score, the function calculates the precision and recall.

   Precision measures the accuracy of positive predictions, consisting of the proportion of correctly detected instances from all object instances.
   $$\text{Precision} = TP / (TP + FP)$$

   Recall measures the ability of the network to detect all object instances, consisting of the proportion of actual instances that the network correctly identifies. False negatives (FNs) are ground truth objects the network misses or detects with a confidence score below the specified threshold.
   $$\text{Recall} = TP / (TP + FN)$$
4. For each instance segmentation, the set of TPs, FPs, and FNs changes, which enables you to recalculate precision and recall at each confidence level.

To visualize the tradeoff between precision and recall for different confidence thresholds, plot the calculated precision and recall values. For an example, see “Perform Instance Segmentation Using SOLOv2”.

Version History

Introduced in R2024b

See Also

`instanceSegmentationMetrics` | `evaluateInstanceSegmentation`

Topics

“Perform Instance Segmentation Using SOLOv2”

summarize

Summarize instance segmentation performance metrics at data set and class level

Syntax

```
[summaryDataset,summaryClass] = summarize(metrics)
[summaryDataset,summaryClass] = summarize(metrics,MetricName=metricName)
```

Description

`[summaryDataset,summaryClass] = summarize(metrics)` summarizes the object detection evaluation metrics across all classes of the data set `summaryDataset`, and for each class in the data set `summaryClass`.

`[summaryDataset,summaryClass] = summarize(metrics,MetricName=metricName)` specifies the metrics category to return in the metrics summary.

Input Arguments

`metrics` — Instance segmentation performance metrics
`instanceSegmentationMetrics` object

instance segmentation performance metrics, specified as an `instanceSegmentationMetrics` object.

`metricName` — Metrics category to return
`"all"` (default) | `"AP"` | `"LAMR""AOS"`

Metrics category to return in the metrics summary, specified as one of these options.

- `"all"` – Returns the average precision and mean average precision metric values, aggregated over each overlap threshold, along with all additional metrics.
- `"AP"` – Returns only the average precision across all classes at each overlap threshold as a *numThresh*-by-1 numeric vector, and the mean average precision (mAP) computed across all overlap thresholds. *numThresh* is the number of overlap thresholds specified by the `OverlapThreshold` property of the `instanceSegmentationMetrics` object.
- `"LAMR"` – Returns only the log-average miss rate (LAMR) computed at each overlap threshold as a *numThresh*-by-1 numeric vector, and the mean of the log-average miss rate (mLAMR) computed across all overlap thresholds.
- `"AOS"` – Returns only the average orientation similarity (AOS) computed at each overlap threshold as a *numThresh*-by-1 array, and the mean of the average orientation similarity (mAOS) computed across all overlap thresholds.

  To specify `"AOS"` as the metric to return, your input data must contain rotated bounding boxes.
- `"AOS"` – Return only the average orientation similarity (AOS) computed at each overlap threshold as a *numThresh*-by-1 array, and the mean of the average orientation (mAOS) similarity computed across all overlap thresholds as a numeric scalar.

  To specify `"AOS"` as the metric to return, your input data must contain rotated bounding boxes.

Output Arguments

`summaryDataset` — Metrics summary over data set

table

Metric summary over the data set, returned as a 1-by-*numMetrics* table, where *numMetrics* is the number of specified metrics.

The columns of the table depend on the value of the `metricName` argument.

- `"all"` —

`NumObjects`	`mAPOverlapAvg`	`mAP`	`mLAMROverlapAvg`	`mLAMR`	`mAOSOverlapAvg`	`mAOS`
Number of object instances in the ground truth data, returned as a positive integer.	Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.	Mean log-average miss rate (mLAMR) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean log-average miss rate (mLAMR) averaged over all classes, computed at all overlap thresholds, specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.	Mean average orientation similarity (mAOS) averaged over all classes, and averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average orientation similarity (mAOS) averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

  To return the `"mAOSOverlapAvg"` and `"mAOS"` columns, your input data must contain rotated bounding boxes.

- `"AP"` —

NumObjects	mAPOverlapAvg	mAP
Number of object instances in the ground truth data, returned as a positive integer.	Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.

- `"LAMR"` —

mLAMROverlapAvg	mLAMR
Mean log-average miss rate (mLAMR) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a positive scalar.	Mean log-average miss rate (mLAMR) averaged over all classes, computed at all overlap thresholds, specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

- `"AOS"` —

mAOSOverlapAvg	mAOS
Mean average orientation similarity (mAOS) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average orientation similarity (mAOS) averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

`summaryClass` — Metrics summary per class
table

Metrics summary per class, returned as a *numClasses*-by-*numMetrics* table, where *numClasses* is the number of classes, and *numMetrics* is the number of specified metrics.

The columns of the table depend on the value of the `metricName` argument.

- `"all"` —

NumObjects	APOverlapAvg	AP	LAMROverlapAvg	LAMR	AOSOverlapAvg	AOS
Number of object instances in the ground truth data for each class, returned as a positive integer.	Average precision (AP) for each class averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average precision (AP) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.	Log-average miss rate (LAMR) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`.	Log-average miss rate (LAMR) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.	Average orientation similarity (AOS) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average orientation similarity (AOS) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

To return the `"AOSOverlapAvg"` and `"AOS"` columns, your input data must contain rotated bounding boxes.

- `"AP"` —

NumObjects	APOverlapAvg	AP
Number of object instances in the ground truth data for each class, returned as a positive integer.	Average precision (AP) for each class averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average precision (AP) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.

- `"LAMR"` —

LAMROverlapAvg	LAMR
Log-average miss rate (LAMR) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`.	Log-average miss rate (LAMR) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

- `"AOS"` —

AOSOverlapAvg	AOS
Average orientation similarity (AOS) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average orientation similarity (AOS) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

Version History

Introduced in R2024b

See Also

`instanceSegmentationMetrics` | `evaluateInstanceSegmentation`

Topics

“Perform Instance Segmentation Using SOLOv2”

averagePrecision

Evaluate average precision metric of object detection results

Syntax

```
ap = averagePrecision(metrics)
ap = averagePrecision(metrics,Name=Value)
```

Description

`ap = averagePrecision(metrics)` evaluates the average precision (AP) for all classes and overlap thresholds of `metrics`.

AP aggregates the precision across different recall levels, providing a single metric to assess the overall ability of an object detector to identify objects accurately while minimizing false detections.

`ap = averagePrecision(metrics,Name=Value)` specifies options for the average precision evaluation using one or more name-value arguments. For example, `ClassNames=["cars" "people"]` specifies to evaluate the average precision metric for the cars and people classes.

Examples

Evaluate Average Precision

Load a table containing image filenames and ground truth bounding box labels into the workspace. The first column contains the images, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label column from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load a pretrained YOLO v2 object detector, trained to detect vehicles, into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Compute Average Precision Metric

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This enables you to evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Compute the metrics for evaluating the performance of the object detector.

```
metrics = evaluateObjectDetection(results,blds);
```

Evaluate the average precision for all classes. The high AP across all classes indicates that the object detection model demonstrates near-perfect precision in identifying objects correctly, while also achieving comprehensive recall, missing very few actual objects across all evaluated classes.

```
ap = averagePrecision(metrics)
```

```
ap =
0.9910
```

Input Arguments

`metrics` — Object detection performance metrics

`objectDetectionMetrics` object

Object detection performance metrics, specified as an `objectDetectionMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `averagePrecision(metrics,ClassNames=[cars people])` specifies to evaluate the average precision metric for the cars and people classes.

`ClassNames` — Class names

string array | cell array of character vectors

Class names of detected objects, specified as an array of strings or a cell array of character vectors. By default, the `averagePrecision` function returns the average precision metrics for all classes specified by the `ClassNames` property of the `objectDetectionMetrics` object `metrics`.

`OverlapThreshold` — Overlap threshold

numeric scalar | numeric vector

Overlap threshold to use for evaluating the average precision, specified as a numeric scalar or numeric vector of box overlap threshold values. To evaluate multiple overlap thresholds, specify this argument as a numeric vector. By default, the `averagePrecision` object function returns the average precision metrics for all overlap thresholds specified by the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`.

Output Arguments

ap — Average precision

M-by-*N* matrix

Average precision (AP) for specified classes and overlap thresholds, specified as an *M*-by-*N* matrix. *M* is the number of classes in the `ClassNames` property and *N* is the number of specified overlap thresholds `OverlapThreshold`.

The AP metric evaluates object detection performance by quantifying the accuracy of the model in identifying objects across different confidence thresholds, enabling you to assess both the precision (correctness of detections) and recall (completeness of detections).

Version History

Introduced in R2024b

See Also

Apps

Object Detector Analyzer

Functions

`objectDetectionMetrics` | `evaluateObjectDetection` | `imageMetrics`

Topics

“Evaluate Object Detector Performance”
Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Choose an Object Detector”
“Get Started with Object Detection Using Deep Learning”

confusionMatrix

Compute confusion matrix of object detection results

Syntax

```
[confMat,confusionClassNames] = confusionMatrix(metrics)
[confMat,confusionClassNames] = confusionMatrix(metrics,Name=Value)
```

Description

`[confMat,confusionClassNames] = confusionMatrix(metrics)` computes the confusion matrix for all overlap thresholds in the `OverlapThreshold` property of `metrics`, and returns the confusion matrix and the class names. The function considers all detections to compute the confusion matrix.

`[confMat,confusionClassNames] = confusionMatrix(metrics,Name=Value)` specifies one or more name-value arguments. For example, `ScoreThreshold=0.3` specifies to disregard detections with a confidence score value below `0.3` when computing the confusion matrix.

Input Arguments

`metrics` — Object detection performance metrics

`objectDetectionMetrics` object

Object detection performance metrics, specified as an `objectDetectionMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `confusionMatrix(metrics,ScoreThreshold=0.3)` specifies detections with a confidence score value below `0.3` to be disregarded when computing the confusion matrix.

`ScoreThreshold` — Confidence score threshold values

`0` (default) | numeric scalar in range `[0, 1]` | numeric vector

Confidence score threshold values to use for computing the confusion matrix, specified as a numeric scalar in the range `[0, 1]` or a numeric vector of values in the range `[0, 1]`. The function filters out predictions with confidence scores less than the threshold value when evaluating the confusion matrix.

Increase this value to reduce the number of false positives, at the possible expense of missing some true positives.

`OverlapThreshold` — Overlap threshold values

numeric scalar | numeric vector

Overlap threshold values to use for computing the confusion matrix, specified as a numeric scalar or numeric vector. Each overlap threshold value must be an element of the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`. By default, the function returns the confusion matrix computed at all the overlap thresholds specified by the `OverlapThreshold` property.

`Normalize` — Confusion matrix normalization
`false` or `0` (default) | `true` or `1`

Confusion matrix normalization option, specified as a numeric or logical `0` (or `false`) or `1` (or `true`). If you specify `Normalize` as `true`, the function normalizes the elements of the confusion matrix in the `confMat` argument by the number of bounding boxes known to belong to each class. For each overlap threshold, each element (*i*, *j*) in the normalized confusion matrix is the count of ground truth bounding boxes that belong to class *i*, but are predicted to belong to class *j*, divided by the total number of bounding boxes predicted in class *j*.

For other normalization options, see “Tips” on page 2-1104.

Output Arguments

`confMat` — Confusion matrix
M-by-*N* cell array

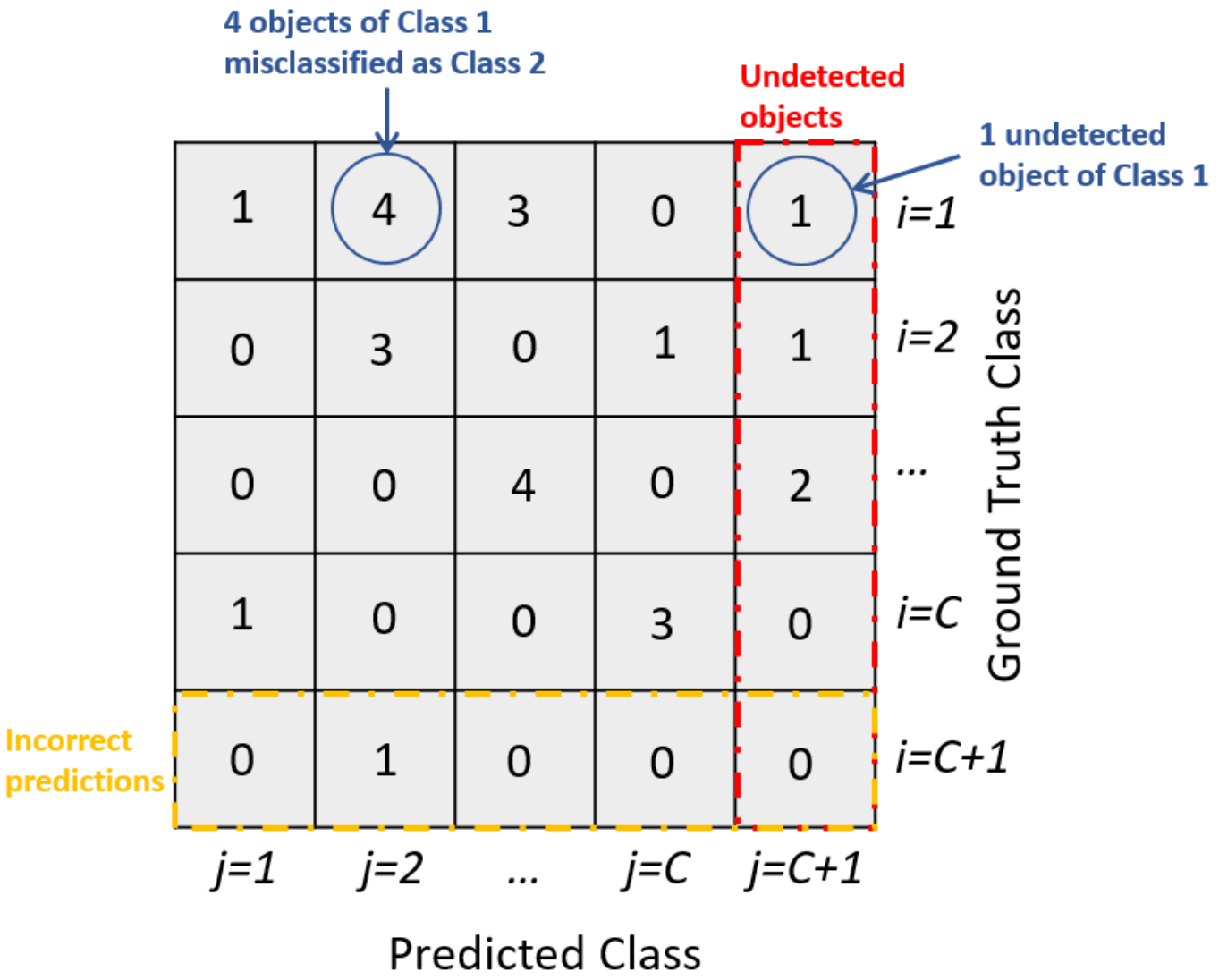
Confusion matrix, returned as an *M*-by-*N* cell array. *M* is the number of confidence score thresholds specified by the `ScoreThreshold` argument, and *N* is the number of overlap thresholds specified by the `OverlapThreshold` argument.

Each element of the `confMat` cell array is a square numeric matrix of size (*C*+1)-by-(*C*+1), where *C* is the number of classes. Each element (*i*, *j*) of the matrix is the count of *matched* bounding boxes predicted to belong to class *j*, but that have a ground truth annotation class of *i*.

Row and column *C*+1 in the confusion matrix correspond to these *unmatched* conditions:

- Column *C*+1 — Undetected objects, or *false negative* predictions. Each element in this column is the number of ground truth annotations of the corresponding class unmatched with any predicted bounding box.
- Row *C*+1 — Incorrect predictions, or *false positive* predictions. Each element in this row is the number of predicted bounding boxes of the corresponding class unmatched with any ground truth annotation.

This image shows the structure of a sample confusion matrix, displayed as a confusion chart. Each element of the matrix contains the number of predictions that fall into a matched or unmatched category. The sum of values in each class row is the total number of ground truth bounding boxes that belong to the corresponding class. The sum of values in each class column is the total number of predicted bounding boxes that belong to the corresponding class.



Confusion Matrix

`confusionClassNames` — Class names
string array

Class names corresponding to the returned confusion matrices, returned as a string array of size $C+1$. The elements of the string array correspond to the classes stored in the `ClassNames` property of the `objectDetectionMetrics` object `metrics` and an extra background class. The background class, `"background"`, corresponds to the row and column $C+1$ of the confusion matrix, which count the false positives and false negatives.

Tips

Normalize the confusion matrix output of the `confusionMatrix` function in these ways:

- Specify the `Normalize` name-value argument as `true` to divide each entry of the confusion matrix, `confMat`, by the sum of the corresponding column, so each value represents the proportion of predicted bounding boxes for a given class.
- For additional normalization options, use the `confusionchart` function. Pass the confusion matrix, `confMat`, to the `confusionchart` function, and specify the `Normalization` name-value argument. You can use this option to normalize the confusion matrix by row or by the total number of detected bounding boxes, in addition to column normalization.

Version History

Introduced in R2024b

See Also

Apps

Object Detector Analyzer

Functions

`objectDetectionMetrics` | `evaluateObjectDetection` | `imageMetrics`

Topics

Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Evaluate Object Detector Performance”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

precisionRecall

Get precision recall metrics of object detection results

Syntax

```
[precision,recall,scores] = precisionRecall(metrics)
[precision,recall,scores] = precisionRecall(metrics,Name=Value)
```

Description

`[precision,recall,scores] = precisionRecall(metrics)` gets the precision, recall, and prediction score values for all classes in the `ClassNames` property and all overlap thresholds in the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`.

`[precision,recall,scores] = precisionRecall(metrics,Name=Value)` specifies precision and recall evaluation options using one or more name-value arguments. For example, `ClassNames=["cars" "people"]` specifies to get the precision recall metrics for the cars and people classes.

Examples

Plot Precision-Recall Curve

Load a table containing image file names and ground truth bounding box labels into the workspace. The first column contains the image file names, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label column from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load a pretrained YOLO v2 object detector, trained to detect vehicles, into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Plot Precision-Recall Curve

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This enables you to evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Compute metrics for evaluating the performance of the object detector.

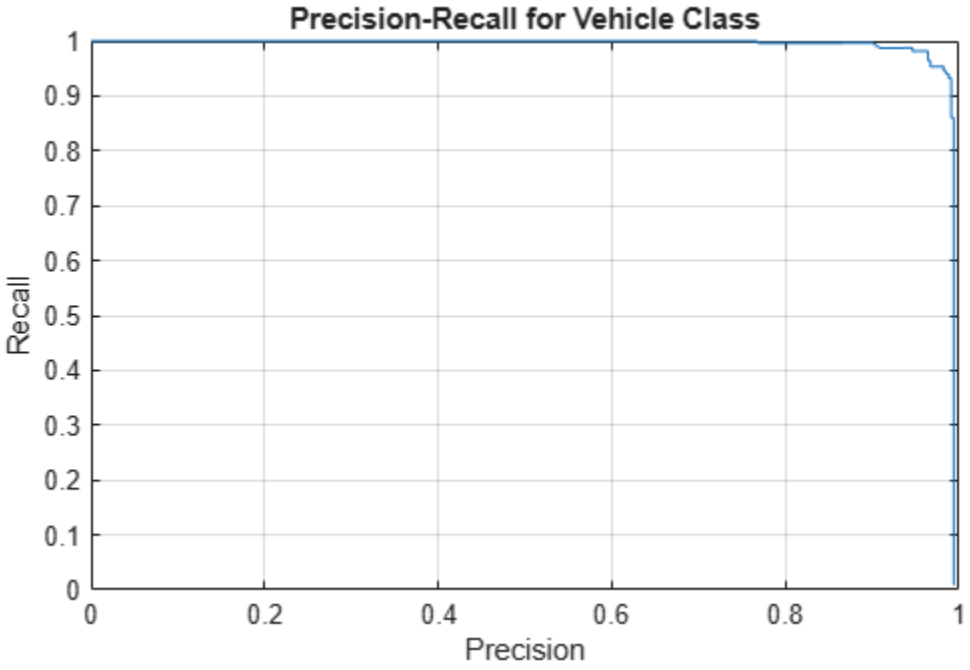
```
metrics = evaluateObjectDetection(results,blds);
```

Get the precision, recall, and confidence scores for the vehicle class using the `precisionRecall` object function.

```
[precision,recall,scores] = precisionRecall(metrics);
```

Plot the precision-recall curve.

```
figure
plot(recall{1},precision{1})
grid on
title("Precision-Recall for Vehicle Class")
xlabel("Precision")
ylabel("Recall")
```



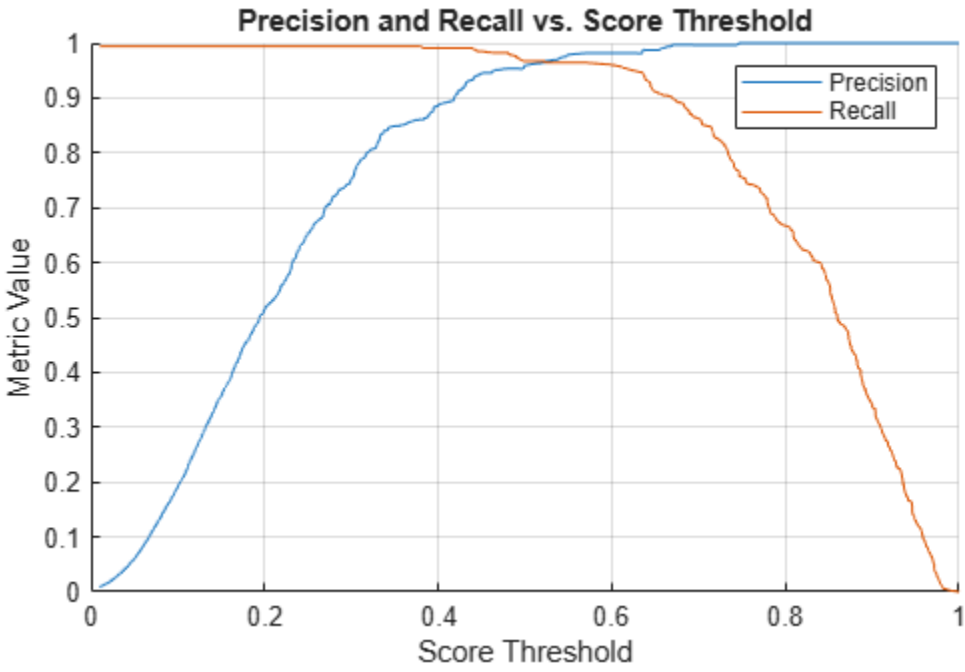
Display the confidence score thresholds and the corresponding values of precision and recall for each point on the precision-recall curve.

```
figure
hold on

plot(scores{1}, precision{1}, DisplayName="Precision");
```

```
plot(scores{1}, recall{1}, DisplayName="Recall");

xlabel("Score Threshold");
ylabel("Metric Value");
title("Precision and Recall vs. Score Threshold");
legend("show")
grid on
hold off
```



Input Arguments

`metrics` — Object detection performance metrics

`objectDetectionMetrics` object

Object detection performance metrics, specified as an `objectDetectionMetrics` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[precision,recall,scores]=precisionRecall(metrics,ClassNames=[cars people])` specifies the precision recall metric to be evaluated for the cars and people classes.

`ClassNames` — Class names

string array | cell array of character vectors

Class names of detected objects, specified as an array of strings or a cell array of character vectors. By default, the `precisionRecall` function returns the precision recall metrics for all classes specified by the `ClassNames` property of the `objectDetectionMetrics` object `metrics`.

`OverlapThresholds` — Overlap threshold

numeric scalar | numeric vector

Overlap threshold, or Intersection over Union (IoU) threshold, for which to get the precision recall metrics, specified as a numeric scalar or numeric vector of box overlap threshold values. By default, the `precisionRecall` object function returns the precision recall metrics for all overlap thresholds specified by the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`. To learn more about how to use overlap thresholds in object detection evaluation, see “Evaluate Detector Performance Using Precision Recall at Multiple Overlap Thresholds” on page 2-1109.

Output Arguments

`precision` — Precision

M-by-*N* cell array

Precision, returned as an *M*-by-*N* cell array. *M* is the number of classes in the `ClassNames` property, and *N* is the number of overlap thresholds in the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`. Each element of the cell array contains a (*numPredictions*+1)-element numeric vector of precision values for the bounding boxes, sorted in descending order of bounding box confidence scores. *numPredictions* is the number of predicted bounding boxes.

Precision is the ratio of the number of true positives (TP) and the total number of predicted positives. Larger precision scores indicate that more detected objects match ground truth objects. To learn more about the precision metric, see “Compute Precision and Recall” on page 2-1110.

`recall` — Recall

M-by-*N* cell array

Recall, returned as an *M*-by-*N* cell array. *M* is the number of classes in the `ClassNames` property, and *N* is the number of overlap thresholds in the `OverlapThreshold` property of the `objectDetectionMetrics` object `metrics`. Each element of the cell array contains a (*numPredictions*+1)-element numeric vector of recall values, sorted in descending order of bounding box confidence scores. *numPredictions* is the number of predicted bounding boxes.

Recall is the ratio of the number of true positives (*TP*) and the total number of ground truth. Larger recall scores indicate that more detected objects match ground truth objects. To learn more about the recall metric, see “Compute Precision and Recall” on page 2-1110.

`scores` — Confidence score

M-by-1 cell array

Confidence score for each detection, returned as an *M*-by-1 cell array. *M* is the number of classes in the `ClassNames` property. Each element of the array is a (*numPredictions*+1)-element numeric vector. *numPredictions* is the number of predicted objects.

More About

Evaluate Detector Performance Using Precision Recall at Multiple Overlap Thresholds

To evaluate how precisely a detector can localize bounding boxes, plot the precision recall curve for a range of overlap thresholds.

At higher overlap thresholds, the criteria for how well the bounding boxes of the detection and ground truth must overlap to be a true positive (TP) are stricter. Therefore, average precision is lower at higher overlap thresholds, which corresponds to a decrease in the area under the precision-recall curve, which plots the recall against precision. Plotting the precision-recall curves across different overlap thresholds can help you evaluate the detector in these ways.

- Evaluate how well the detector can generate detection at higher overlap thresholds.
- Identify specific overlap thresholds where a detector's performance significantly drops, highlighting areas for model improvement.
- Choose an optimal overlap threshold that balances the tradeoff between detecting as many objects as possible (high recall) and ensuring high accuracy in these detections (high precision).
- Use precision-recall curves at different overlap thresholds for a more granular comparison between object detection models than single metric evaluations. Nuances in performance can be obscured when considering only one overlap threshold or when relying solely on aggregated metrics such as mean average precision.

Algorithms

Compute Precision and Recall

Precision recall is a powerful tool you can use to evaluate object detector performance. The `precisionRecall` function computes the precision and recall in these steps.

1. Sorts the detections which satisfy the criteria specified by the `ClassNames` and `OverlapThresholds` name-value arguments in descending order based on their confidence scores. Higher confidence scores indicate that the model is more certain that the detection is correct.
2. Identifies true positives (TP) and false positives (FP) according to the specified overlap threshold criteria. A detection is considered a TP if its IoU is greater to or equal to the specified overlap threshold. A detection is considered an FP if it does not match any ground truth object or if it matches a ground truth object but the IoU is below the specified overlap threshold.
3. Starting with the detection with the highest confidence score, the `precisionRecall` function iterates through each detection and calculates the cumulative number of TPs and FPs at each step. This step-by-step process enables the calculation of precision and recall at every confidence level. At each score, the function calculates the precision and recall.

   Precision measures the accuracy of positive predictions, i.e., the proportion of detections that are correctly identified from all detections made.
   $\text{Precision} = TP / (TP + FP)$

   Recall measures the ability of the detector to detect all object instances, i.e. the proportion of actual positives that are correctly identified. False negatives (FNs) are ground truth objects that haven't been detected or have been detected with a confidence score below the current threshold being considered.
   $\text{Recall} = TP / (TP + FN)$
4. For each detection, the set of TPs, FPs, and FNs changes, which enables you to recalculate precision and recall at each confidence level.

To visualize the tradeoff between precision and recall for different confidence thresholds, plot the calculated precision and recall values. For an example, see “Plot Precision-Recall Curve” on page 2-1106.

Version History

Introduced in R2024b

See Also

Apps

Object Detector Analyzer

Functions

`objectDetectionMetrics` | `evaluateObjectDetection` | `imageMetrics`

Topics

Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Evaluate Object Detector Performance”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Get Started with Object Detection Using Deep Learning”

summarize

Summarize object detection performance metrics at data set and class level

Syntax

```
[summaryDataset,summaryClass] = summarize(metrics)
[summaryDataset,summaryClass] = summarize(metrics,MetricName=metricName)
```

Description

`[summaryDataset,summaryClass] = summarize(metrics)` summarizes the object detection evaluation metrics across all classes of the data set `summaryDataset`, and for each class in the data set `summaryClass`.

`[summaryDataset,summaryClass] = summarize(metrics,MetricName=metricName)` specifies the metrics category to return in the metrics summary.

Examples

Summarize Object Detection Metrics

Load a table containing image file names and ground truth bounding box labels into the workspace. The first column contains the image file names, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label column from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load a pretrained YOLO v2 object detector, trained to detect vehicles, into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This enables you to evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Compute metrics for evaluating the performance of the object detector.

```
metrics = evaluateObjectDetection(results,blds);
```

Summarize the object detection metrics for the vehicle class using the `summarize` object function.

```
[summaryDataset,summaryClass] = summarize(metrics);
summaryClass
```

```
summaryClass=1×3 table
               NumObjects    APOverlapAvg     AP0.5
               __________    ____________    _______

    vehicle       336          0.99096       0.99096
```

Input Arguments

`metrics` — Object detection performance metrics

`objectDetectionMetrics` object

Object detection performance metrics, specified as an `objectDetectionMetrics` object.

`metricName` — Metrics category to return

`"all"` (default) | `"AP"` | `"LAMR"` | `"AOS"`

Metrics category to return in the metrics summary, specified as one of these options.

- `"all"` – Returns the average precision and mean average precision metric values, aggregated over each overlap threshold, along with all additional metrics.
- `"AP"` – Returns only the average precision across all classes at each overlap threshold as a *numThresh*-by-1 numeric vector, and the mean average precision (mAP) computed across all overlap thresholds. *numThresh* is the number of overlap thresholds specified by the `OverlapThreshold` property of the `objectDetectionMetrics` object.
- `"LAMR"` – Returns only the log-average miss rate (LAMR) computed at each overlap threshold as a *numThresh*-by-1 numeric vector, and the mean of the log-average miss rate (mLAMR) computed across all overlap thresholds.
- `"AOS"` – Returns only the average orientation similarity (AOS) computed at each overlap threshold as a *numThresh*-by-1 array, and the mean of the average orientation similarity (mAOS) computed across all overlap thresholds.

  To specify `"AOS"` as the metric to return, your input data must contain rotated bounding boxes.

Output Arguments

`summaryDataset` — Metrics summary over data set

table

Metric summary over the data set, returned as a 1-by-*numMetrics* table, where *numMetrics* is the number of specified metrics.

The columns of the table depend on the value of the `metricName` argument.

- `"all"` —

NumObjects	mAPOverlapAvg	mAP	mLAMROverlapAvg	mLAMR	mAOSOverlapAvg	mAOS
Number of objects in the ground truth data, returned as a positive integer.	Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.	Mean log-average miss rate (mLAMR) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean log-average miss rate (mLAMR) averaged over all classes, computed at all overlap thresholds, specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.	Mean average orientation similarity (mAOS) averaged over all classes, and averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average orientation similarity (mAOS) averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

  To return the `"mAOSOverlapAvg"` and `"mAOS"` columns, your input data must contain rotated bounding boxes.

- `"AP"` —

NumObjects	mAPOverlapAvg	mAP
Number of objects in the ground truth data, returned as a positive integer.	Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.

- `"LAMR"` —

mLAMROverlapAvg	mLAMR
Mean log-average miss rate (mLAMR) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a positive scalar.	Mean log-average miss rate (mLAMR) averaged over all classes, computed at all overlap thresholds, specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

- `"AOS"` —

mAOSOverlapAvg	mAOS
Mean average orientation similarity (mAOS) averaged over all classes and all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Mean average orientation similarity (mAOS) averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

`summaryClass` — Metrics summary per class
table

Metrics summary per class, returned as a *numClasses*-by-*numMetrics* table, where *numClasses* is the number of classes, and *numMetrics* is the number of specified metrics.

The columns of the table depend on the value of the `metricName` argument.

- `"all"` —

NumObjects	APOverlapAvg	AP	LAMROverlapAvg	LAMR	AOSOverlapAvg	AOS
Number of objects in the ground truth data for each class, returned as a positive integer.	Average precision (AP) for each class averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average precision (AP) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.	Log-average miss rate (LAMR) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`.	Log-average miss rate (LAMR) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.	Average orientation similarity (AOS) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average orientation similarity (AOS) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

To return the `"AOSOverlapAvg"` and `"AOS"` columns, your input data must contain rotated bounding boxes.

- `"AP"` —

NumObjects	APOverlapAvg	AP
Number of objects in the ground truth data for each class, returned as a positive integer.	Average precision (AP) for each class averaged over all thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average precision (AP) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.

- `"LAMR"` —

LAMROverlapAvg	LAMR
Log-average miss rate (LAMR) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`.	Log-average miss rate (LAMR) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

- `"AOS"` —

AOSOverlapAvg	AOS
Average orientation similarity (AOS) for each class, averaged over all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a numeric scalar.	Average orientation similarity (AOS) for each class, computed at all overlap thresholds specified by the `OverlapThreshold` property of `metrics`, returned as a *numThresh*-by-1 numeric vector.

Version History

Introduced in R2024b

See Also

Apps

Object Detector Analyzer

Functions

`objectDetectionMetrics` | `evaluateObjectDetection`

Topics

“Evaluate Object Detector Performance”
“Multiclass Object Detection Using YOLO v2 Deep Learning”

metricsByArea

Evaluate detection performance across object size ranges

Syntax

```
areaMetrics = metricsByArea(metrics,objectAreaRanges)
areaMetrics = metricsByArea(metrics,objectAreaRanges,ClassNames)
```

Description

`areaMetrics = metricsByArea(metrics,objectAreaRanges)` evaluates object detection metrics for the object size range or ranges, specified as bounding box areas, `objectAreaRanges`. The function evaluates the specified bounding box size range within the existing object detection `metrics`.

`areaMetrics = metricsByArea(metrics,objectAreaRanges,ClassNames)` also specifies the names of the classes for which to evaluate area-based metrics.

Examples

Evaluate Detection Metrics Across Object Size Ranges

Load a table containing images and ground truth bounding box labels. The first column contains the images, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load a pretrained YOLO v2 object detector trained to detect vehicles into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Compute Object Detection Performance Metrics

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This helps you evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Compute metrics for evaluating the performance of an object detector using the `evaluateObjectDetection` function.

```
metrics = evaluateObjectDetection(results,blds);
```

Evaluate Metrics Across Object Size Ranges

Extract the anchor boxes from the detector, calculate their areas, and sort the areas.

```
areas = prod(detector.AnchorBoxes,2);
sortedAreas = sort(areas);
```

Define area range limits using the calculated areas. The upper limit for the last range is set to three times the size of the largest area, which is sufficient for the objects in this data set.

```
lowerLimit = sortedAreas;
upperLimit = [sortedAreas(2:end);3*sortedAreas(end)];
areaRanges = [lowerLimit upperLimit]
```

```
areaRanges = 4×2

          64         960
         960        1536
        1536        3456
        3456       10368
```

Evaluate detector performance across vehicle size ranges using the `metricsByArea` object function. Specify the object size ranges using the bounding box area ranges. The number of vehicles falling into each area range is computed in the `NumObjects` column.

```
areaMetrics = metricsByArea(metrics,areaRanges,ClassName="vehicle");
disp(areaMetrics(:,1:3))
```

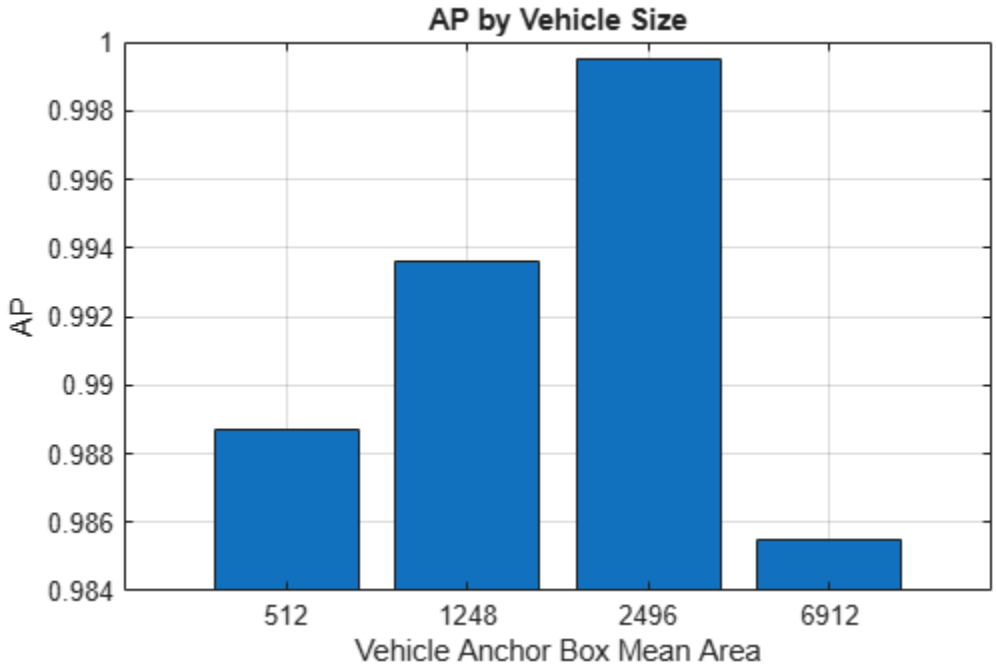
```
      AreaRange      NumObjects    APOverlapAvg
    _____________    __________    ____________

      64      960       119           0.9887
     960     1536        12          0.99359
    1536     3456        45          0.99952
    3456    10368       141          0.98551
```

Plot the average precision (AP) metric, averaged across all overlap thresholds, as a function of the mean object size per vehicle size range.

```
figure
bar(categorical(mean(areaRanges,2)),areaMetrics.APOverlapAvg)
title("AP by Vehicle Size")
xlabel("Vehicle Anchor Box Mean Area")
ylabel("AP")
```

```
ylim([0.984 1])
grid on
```



Input Arguments

`metrics` — Object detection metrics

M-by-4 numeric matrix | vector

Object detection metrics, specified as an `objectDetectionMetrics` object.

`objectAreaRanges` — Bounding box area ranges

M-by-2 numeric matrix | vector

Bounding box area ranges (object size ranges), specified as one of the following:

- An *M*-by-2 numeric matrix, with each row specifying the upper (exclusive) and lower (inclusive) limits of an object area range in evaluation. For example, if the input is $[x_1\ x_2;\ x_3\ x_4]$. The evaluated bounding box area ranges are $x_1 \leq$ area $< x_2$ and $x_3 \leq$ area $< x_4$.
- A vector with *M*+1 elements, specifying the edge values of *M* contiguous area ranges. Each range is inclusive in the lower limit, and exclusive in the upper limit. For example, if the input is $[x_1\ x_2\ x_3]$, the area ranges evaluated are $x_1 \leq$ area $< x_2$ and $x_2 \leq$ area $< x_3$.

`ClassNames` — Names of classes

`''` (default) | string scalar | vector

Names of the classes for which to evaluate area-based metrics. If multiple classes are specified, the function computes averaged metrics across the classes. By default, all classes will be considered and the returned metrics will be the average of all classes.

Output Arguments

`areaMetrics` — Metrics by area

table

Metrics by area, returned as a table with *M* rows. Each row contains metrics using objects within an area range (for a total of *M* area ranges). The columns are specified by these table variables.

- `AreaRange`: The upper and lower limits of each area range, returned as a two-element row vector.
- `NumObjects`: The number of objects falling within each area range, returned as a positive integer.
- `mAP`: Mean average precision (mAP), or average precision averaged over all classes, computed at all overlap thresholds specified by the `OverlapThreshold` property, returned as a *numThresh*-by-1 numeric vector. *numThresh* is the number of overlap thresholds.
- `mAPOverlapAvg`: Mean average precision (mAP) averaged over all thresholds specified by the `OverlapThreshold` property, returned as a numeric scalar.

Version History

Introduced in R2023b

See Also

Apps

Object Detector Analyzer

Functions

`objectDetectionMetrics` | `evaluateObjectDetection` | `imageMetrics`

Topics

Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Detect Small Objects Using Tiled Training of YOLOX Network”
“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Evaluate Object Detector Performance”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

semanticSegmentationMetrics

Semantic segmentation quality metrics

Description

A `semanticSegmentationMetrics` object encapsulates semantic segmentation quality metrics for a set of images.

Creation

Create a `semanticSegmentationMetrics` object using the `evaluateSemanticSegmentation` function.

Properties

`ConfusionMatrix` — Confusion matrix
table

This property is read-only.

Confusion matrix, specified as a table with C rows and columns, where C is the number of classes in the semantic segmentation. Each table element (i,j) is the count of pixels known to belong to class i but predicted to belong to class j.

`NormalizedConfusionMatrix` — Normalized confusion matrix
table

This property is read-only.

Normalized confusion matrix, specified as a table with C rows and columns, where C is the number of classes in the semantic segmentation. The `NormalizedConfusionMatrx` represents a confusion matrix normalized by the number of pixels known to belong to each class. Each table element (i,j) is the count of pixels known to belong to class i but predicted to belong to class j, divided by the total number of pixels predicted in class i. Elements are in the range [0, 1].

`DataSetMetrics` — Data set metrics
table

This property is read-only.

Semantic segmentation metrics aggregated over the data set, specified as a table with one row. `DataSetMetrics` has up to five columns, corresponding to the metrics that were specified by the `'Metrics'` name-value pair used with `evaluateSemanticSegmentation`:

- `GlobalAccuracy` — Ratio of correctly classified pixels to total pixels, regardless of class.
- `MeanAccuracy` — Ratio of correctly classified pixels in each class to total pixels, averaged over all classes. The value is equal to the mean of `ClassMetrics.Accuracy`.

- `MeanIoU` — Average intersection over union (IoU) of all classes. The value is equal to the mean of `ClassMetrics.IoU`.
- `WeightedIoU` — Average IoU of all classes, weighted by the number of pixels in the class.
- `MeanBFScore` — Average boundary F1 (BF) score of all images. The value is equal to the mean of `ImageMetrics.BFScore`. This metric is not available when you create a `semanticSegmentationMetrics` object by using a confusion matrix as the input to `evaluateSemanticSegmentation`.

Note A value of `NaN` in the dataset, class, or image metrics, indicates that one or more classes were missing during the computation of the metrics when using the `evaluateSemanticSegmentation` function. In this case, the software was unable to accurately compute the metrics.

The missing classes can be found by looking at the `ClassMetrics` property, which provides the metrics for each class. To more accurately evaluate your network, augment your ground truth with more data that includes the missing classes.

`ClassMetrics` — Class metrics

table

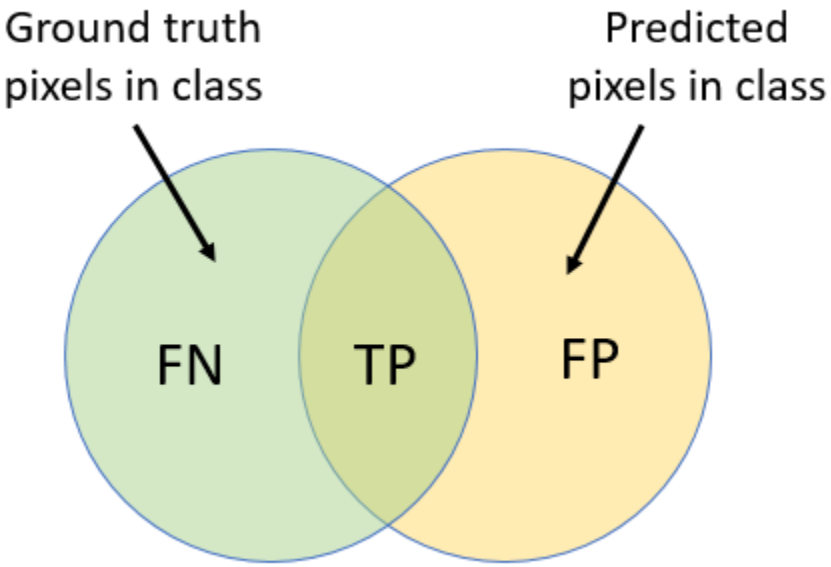
This property is read-only.

Semantic segmentation metrics for each class, specified as a table with *C* rows, where *C* is the number of classes in the semantic segmentation. `ClassMetrics` has up to three columns, corresponding to the metrics that were specified by the `'Metrics'` name-value pair used with `evaluateSemanticSegmentation`:

- `Accuracy` — Ratio of correctly classified pixels in each class to the total number of pixels belonging to that class according to the ground truth. Accuracy can be expressed as:
  `Accuracy` = (TP + TN) / (TP + TN + FP + FN)

	Positive	**Negative**
Positive	TP: True Positive	FN: False Negative
Negative	FP: False Positive	TN: True Negative

  TP: True positives and FN is the number of false negatives.
- `IoU` — Ratio of correctly classified pixels to the total number of pixels that are assigned that class by the ground truth and the predictor. IoU can be expressed as:
  `IoU` = TP / (TP + FP + FN)
  The image describes the true positives (TP), false positives (FP), and false negatives (FN).



- `MeanBFScore` — Boundary F1 score for each class, averaged over all images. This metric is not available when you create a `semanticSegmentationMetrics` object by using a confusion matrix as the input to `evaluateSemanticSegmentation`.

`ImageMetrics` — Image metrics
table

This property is read-only.

Semantic segmentation metrics for each image in the data set, specified as a table with *N* rows, where *N* is the number of images in the data set. `ImageMetrics` has up to five columns, corresponding to the metrics that were specified by the `'Metrics'` name-value pair used with `evaluateSemanticSegmentation`:

- `GlobalAccuracy` — Ratio of correctly classified pixels to total pixels, regardless of class.
- `MeanAccuracy` — Ratio of correctly classified pixels to total pixels, averaged over all classes in the image.
- `MeanIoU` — Average IoU of all classes in the image.
- `WeightedIoU` — Average IoU of all classes in the image, weighted by the number of pixels in each class.
- `MeanBFScore` — Average BF score of each class in the image. This metric is not available when you create a `semanticSegmentationMetrics` object by using a confusion matrix as the input to `evaluateSemanticSegmentation`.

Each image metric returns a vector, with one element for each image in the data set. The order of the rows matches the order of the images defined by the input `PixelLabelDatastore` objects representing the data set.

Examples

Evaluate Semantic Segmentation Results

The `triangleImages` data set has 100 test images with ground truth labels. Define the location of the data set.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
```

Define the location of the test images and ground truth labels.

```
testImagesDir = fullfile(dataSetDir,"testImages");
testLabelsDir = fullfile(dataSetDir,"testLabels");
```

Create an `imageDatastore` holding the test images.

```
imds = imageDatastore(testImagesDir);
```

Define the class names and their associated label IDs.

```
classNames = ["triangle" "background"];
labelIDs   = [255 0];
```

Create a `pixelLabelDatastore` holding the ground truth pixel labels for the test images.

```
pxdsTruth = pixelLabelDatastore(testLabelsDir,classNames,labelIDs);
```

Load a semantic segmentation network that has been trained on the training images of `triangleImages`.

```
net = load("triangleSegmentationNetwork");
net = net.net;
```

Run the network on the test images. Predicted labels are written to disk in a temporary directory and returned as a `pixelLabelDatastore`.

```
pxdsResults = semanticseg(imds,net,Classes=classNames,WriteLocation=tempdir);
```

```
Running semantic segmentation network
-------------------------------------
* Processed 100 images.
```

Evaluate the prediction results against the ground truth.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth);
```

```
Evaluating semantic segmentation results
----------------------------------------
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 100 images.
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

       0.99074           0.99183       0.91118      0.98299        0.80563
```

Display the properties of the `semanticSegmentationMetrics` object.

```
metrics
```

```
metrics =
  semanticSegmentationMetrics with properties:

             ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
            DataSetMetrics: [1×5 table]
              ClassMetrics: [2×3 table]
              ImageMetrics: [100×5 table]
```

Display the classification accuracy, the intersection over union, and the boundary F-1 score for each class. These values are stored in the `ClassMetrics` property.

```
metrics.ClassMetrics
```

```
ans=2×3 table
                 Accuracy      IoU      MeanBFScore
                 ________    _______    ___________

    triangle     0.99302     0.83206      0.67208
    background   0.99063      0.9903      0.93918
```

Display the normalized confusion matrix that is stored in the `NormalizedConfusionMatrix` property.

```
metrics.ConfusionMatrix
```

```
ans=2×2 table
                 triangle    background
                 ________    __________

    triangle        4697           33
    background       915        96755
```

Version History

Introduced in R2017b

See Also

`evaluateSemanticSegmentation` | `plotconfusion` | `jaccard` | `bfscore`

Topics

“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

mAPObjectDetectionMetric

Mean average precision (mAP) metric for object detection

Description

Use an `mAPObjectDetectionMetric` object to track the mean average precision (mAP) metric when you train an object detector.

You can specify metrics options for training by using these arguments of the `trainingOptions` function:

- `Metrics` — Specify which metrics to use during training.
- `Plots` — To plot metrics during training, specify as `"training-progress"`.
- `ValidationData` — Specify this argument to additionally plot and record the metric values for the validation data.
- `Verbose` — To output the metric values to the Command Window, specify as true.

Creation

Syntax

```
metric = mAPObjectDetectionMetric
metric = mAPObjectDetectionMetric(Name=Value)
```

Description

`metric = mAPObjectDetectionMetric` creates a default mean average precision (mAP) metric to use when training an object detector.

When specified to the `Metrics` argument of the `trainingOptions` function, the default `mAPObjectDetectionMetric` object is equivalent to specifying `Metrics` as `"mAP50"`.

`metric = mAPObjectDetectionMetric(Name=Value)` specifies the `Name` and `OverlapThreshold` properties, used for metric calculation, using name-value arguments.

For example, `Name="mAP50"` sets the mean average precision as the metric to calculate.

Properties

`Name` — Metric name
`"mAP50"` (default) | string scalar | character vector

Metric name, specified as a string scalar or character vector. The metric name appears in the training plot and the verbose output.

Data Types: `char` | `string`

`OverlapThreshold` — Overlap threshold
`0.5` (default) | numeric scalar

Overlap threshold, specified as a numeric scalar in the range `[0, 1]`, which `mAPObjectDetectionMetric` uses to compute the mAP metric. When the intersection over union (IoU) of the pixels in the ground truth bounding box and the predicted bounding box is equal to or greater than the overlap threshold, the `trainingOptions` function considers the detection to be a match to the ground truth. The IoU is the number of pixels in the intersection of the bounding boxes divided by the number of pixels in the union of the bounding boxes.

Version History

Introduced in R2024a

See Also

`trainingOptions` | `aucMetric` | `trainYOLOXObjectDetector` | `trainYOLOv4ObjectDetector` | `dlnetwork` | `trainnet`

Topics

“Get Started with Object Detection Using Deep Learning”
“Set Up Parameters and Train Convolutional Neural Network” (Deep Learning Toolbox)

vision.StandardDeviation

Namespace: `vision`

(Removed) Find standard deviation values in input or sequence of inputs

Note `vision.StandardDeviation` function has been removed. Use the `std` and `movstd` functions to find standard deviation values, instead.

Description

Find the standard deviation values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.StandardDeviation` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
stdDev = vision.StandardDeviation
stdDev = vision.StandardDeviation(Name,Value)
```

Description

`stdDev = vision.StandardDeviation` returns an object, `stdDev`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`stdDev = vision.StandardDeviation(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `stdDev = vision.Maximum('RunningStandardDeviation',false)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`ValueOutputPort` — Output Minimum value
`true` (default) | `false`

Output standard deviation value, specified as `true` or `false`. Set this property to `true` to output the Minimum value of the input. This property applies when you set the `RunningStandardDeviation` property to `false`.

`RunningStandardDeviation` — Calculate over single input or multiple inputs
`false` (default) | `true`

Calculate over single input or multiple inputs, specified as `true` or `false`. When you set this property to `true`, the object computes the standard deviation value over a sequence of inputs. When you set this property to `false`, the object computes the standard deviation value over the current input.

`ResetInputPort` — Additional input to enable resetting of running standard deviation
`false` (default) | `true`

Additional input to enable resetting of running standard deviation, specified as `true` or `false`. When you set this property to `true`, a reset input must be specified to reset the running standard deviation. This property applies only when you set the `RunningStandardDeviation` property to `true`.

`ResetCondition` — Condition that triggers resetting of running standard deviation
`'Non-zero'` (default) | `'Rising edge'` | `'Falling edge'` | `'Either edge'`

Condition that triggers resetting of running standard deviation, specified as `'Rising edge'`, `'Falling edge'`, `'Either edge'`, or `'Non-zero'`. This property applies only when you set the `ResetInputPort` property to `true`.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningStandardDeviation` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
`1` (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

`ROIProcessing` — Enable region-of-interest processing
`false` (default) | `true`

Enable region-of-interest processing, specified as `true` or `false`. This property applies when you set the `Dimension` property to `'All'` and the `RunningStandardDeviation` property to `false`.

`ROIForm` — Type of ROI
`'Rectangles'` (default) | `Lines'` | `'Label matrix''Binary mask'`

Type of ROI, specified as `'Rectangles'`,`'Lines'`, `'Label matrix'`, or `'Binary mask'`. This property applies only when you set the `ROIProcessing` property to `true`.

`ROIPortion` — ROI or perimeter calculation
`'Entire ROI'` (default) | `'ROI perimeter'`

ROI or perimeter calculation, specified as `'Entire ROI'` or `'ROI perimeter'`. This property applies only when you set the `ROIForm` property to `'Rectangles'`.

`ROIStatistics` — Calculate statistics for each ROI or one for all ROIs
`'Individual statistics for each ROI'` (default) | `'Single statistic for all ROIs'`

Calculate statistics for each ROI or one for all ROIs, specified as `'Individual statistics for each ROI'` or `'Single statistic for all ROIs'`. This property applies only when you set the `'ROIForm'` property to `'Rectangles'`, `'Lines'`, or `'Label matrix'`.

`ValidityOutputPort` — Flag to indicate if any part of ROI is outside input image
`false` (default) | `true`

Flag to indicate if any part of ROI is outside input image, specified as `true` or `false`.

This applies when you set the `ROIForm` property to `'Lines'` or `'Rectangles'`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `'Label matrix'`.

Usage

Syntax

```
[value,index] = stdDev(input)
index = stdDev(input)

[ ___ ] = stdDev(I,ROI)
[ ___ ,flag] = stdDev(I,ROI)

[ ___ ] = stdDev(I,label,labelNumbers)
[ ___ ,flag] = stdDev(I,label,labelNumbers)
```

Description

`[value,index] = stdDev(input)` returns the standard deviation value and index of the input.

`index = stdDev(input)` returns the one-based index of the standard deviation value when you set the `IndexOutputPort` property to `true` and the `ValueOutputPort` property to `false`. The `RunningStandardDeviation` property must be set to `false`.

`[ ___ ] = stdDev(I,ROI)` returns the standard deviation value in the input image within the given region of interest.

`[ ___ ,flag] = stdDev(I,ROI)`additionally returns a flag to indicate whether the given ROI is within the bounds of the image.

`[ ___ ] = stdDev(I,label,labelNumbers)` returns the standard deviation of the input image for a region the labels specified in the `labelNumbers` vector. The regions are defined and labeled in the `label` matrix.

`[ ___ ,flag] = stdDev(I,label,labelNumbers)` additionally returns a flag to indicate whether the input label numbers are valid.

Input Arguments

`input` — Input data
vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

`ROI` — Region of interest
four-element vector

Region of interest, specified as a four-element vector, [*x y width height*]. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Lines'`, `'Rectangles'`, or `'Binary Mask'`.

`label` — Label numbers
matrix

Label numbers, specified as a matrix. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

`labelNumbers` — Label numbers
vector

Label numbers, specified as a vector. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

Output Arguments

`value` — Minimum value
same as input

Minimum value, returned as the same data type as the input

`index` — Index to standard deviation value
one-based index

Index to standard deviation value, returned as a one-based index.

`flag` — Flag for valid data
`true` | `false`

Flag for valid data, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics

reset Reset internal states of System object

Examples

Determine Standard Deviation in Grayscale Image

1. Read a color image.

```
img = imread('peppers.png');
```

2. Convert the image to grayscale.

```
img = im2single(rgb2gray(img));
```

3. Find the standard deviation.

```
stdDev = vision.StandardDeviation;
stdDevValue = stdDev(img);
```

Version History

Introduced in R2012a

R2024b: `vision.StandardDeviation` function has been removed

Errors starting in R2024b

`vision.StandardDeviation` function has been removed. Use the `std` and `movstd` functions to find standard deviation values, instead.

R2024a: `vision.StandardDeviation` function will be removed

Warns starting in R2024a

`vision.StandardDeviation` function will be removed in a future release. Use the `std` and `movstd` functions to find standard deviation values, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`std` | `movstd`

stereoParameters

Object for storing stereo camera system parameters

Description

The `stereoParameters` object stores the intrinsic and extrinsic parameters of two cameras and their geometric relationship.

Creation

You can create a `stereoParameters` object using the `stereoParameters` function described here. You can also create a `stereoParameters` object by using the `estimateCameraParameters` with an *M*-by-2-by-*numImages*-by-2 array of input image points, where *M* is the number of keypoint coordinates in each pattern.

Syntax

```
stereoParams = stereoParameters(cameraParameters1,cameraParameters2,
poseCamera2)
stereoParams = stereoParameters(paramStruct)
```

Description

`stereoParams = stereoParameters(cameraParameters1,cameraParameters2, poseCamera2)` returns a stereo camera system parameters object using the camera parameters from two cameras and the pose of camera 2. `cameraParameters1` and `cameraParameters2` are `cameraParameters` or `cameraIntrinsics` objects that contain the intrinsics of camera 1 and camera 2, respectively. The three input arguments set the CameraParameters1, CameraParameters2, and PoseCamera2 properties, respectively.

`stereoParams = stereoParameters(paramStruct)` creates an identical `stereoParameters` object from an existing `stereoParameters` object with parameters stored in `paramStruct`.

Input Arguments

`paramStruct` — Stereo parameters
struct

Stereo parameters, specified as a stereo parameters struct. To get a `paramStruct` from an existing `stereoParameters` object, use the `toStruct` function.

Properties

Intrinsic and extrinsic parameters of the two cameras

`CameraParameters1` — Parameters of camera 1
`cameraParameters` object

Parameters of camera 1, specified as a `cameraParameters` object. The object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`CameraParameters2` — Parameters of camera 2

`cameraParameters` object

Parameters of camera 2, specified as a `cameraParameters` object. The object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

Geometric relationship between the two cameras

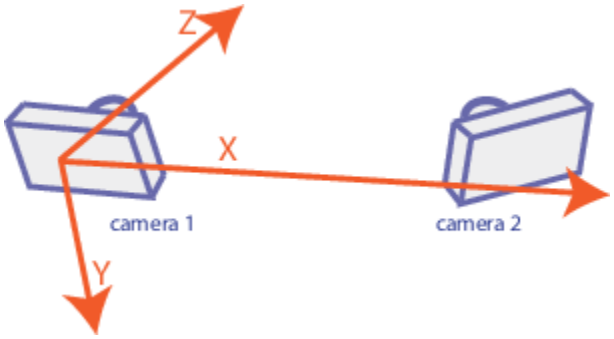
`PoseCamera2` — Relative pose of second camera

`rigidtform3d` object

Relative pose of the second camera, specified as a `rigidtform3d` object. The `R` and the `Translation` properties of the object represent the relative rotation and translation between camera 1 and camera 2, respectively. They convert camera 2 coordinates back to camera 1 coordinates using:

```
orientation1 = orientation2 * R
location1 = orientation2 * Translation + location2
```

`orientation1` and `location1` represent the absolute pose of camera 1, and `orientation2` and `location2` represent the absolute pose of camera 2.



You cannot set the value of `PoseCamera2` after you create the `stereoParameters` object.

`FundamentalMatrix` — Fundamental matrix

3-by-3 matrix

Fundamental matrix, stored as a 3-by-3 matrix. The fundamental matrix relates the two stereo cameras, such that the following equation must be true:

$$[P_1 \ 1] * FundamentalMatrix * \begin{bmatrix} P_2 \\ 1 \end{bmatrix} = 0$$

P_1, the point in image 1, corresponds to the point, P_2, in image 2. Both points are expressed in pixel coordinates.

`EssentialMatrix` — Essential matrix

3-by-3 matrix

Essential matrix, stored as a 3-by-3 matrix. The essential matrix relates the two stereo cameras, such that the following equation must be true:

$$[P_1\ 1] * EssentialMatrix * \begin{bmatrix} P_2 \\ 1 \end{bmatrix} = 0$$

P_1, the point in image 1, corresponds to P_2, the point in image 2. Both points are expressed in normalized image coordinates, where the origin is at the camera's optical center. The x and y pixel coordinates are normalized by the focal length f_x and f_y.

Accuracy of estimated parameters

`MeanReprojectionError` — Average Euclidean distance
number of pixels

Average Euclidean distance between reprojected points and detected points over all image pairs, specified in pixels.

Settings for camera parameter estimation

`NumPatterns` — Number of calibrated patterns
integer

Number of calibration patterns that estimate the extrinsics of the two cameras, stored as an integer.

`WorldPoints` — World coordinates
M-by-2 array

World coordinates of key points in the calibration pattern, specified as an *M*-by-2 array. *M* represents the number of key points in the pattern.

`WorldUnits` — World points units
`'mm'` (default) | character vector

World points units, specified as a character vector. The character vector describes the units of measure.

Object Functions

toStruct Convert a stereo parameters object into a struct

Examples

Perform Stereo Camera Calibration

Specify calibration images.

```
leftImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","left"));
rightImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = ...
  detectCheckerboardPoints(leftImages.Files,rightImages.Files);
```

Specify the world coordinates of the checkerboard keypoints. Square size is in millimeters.

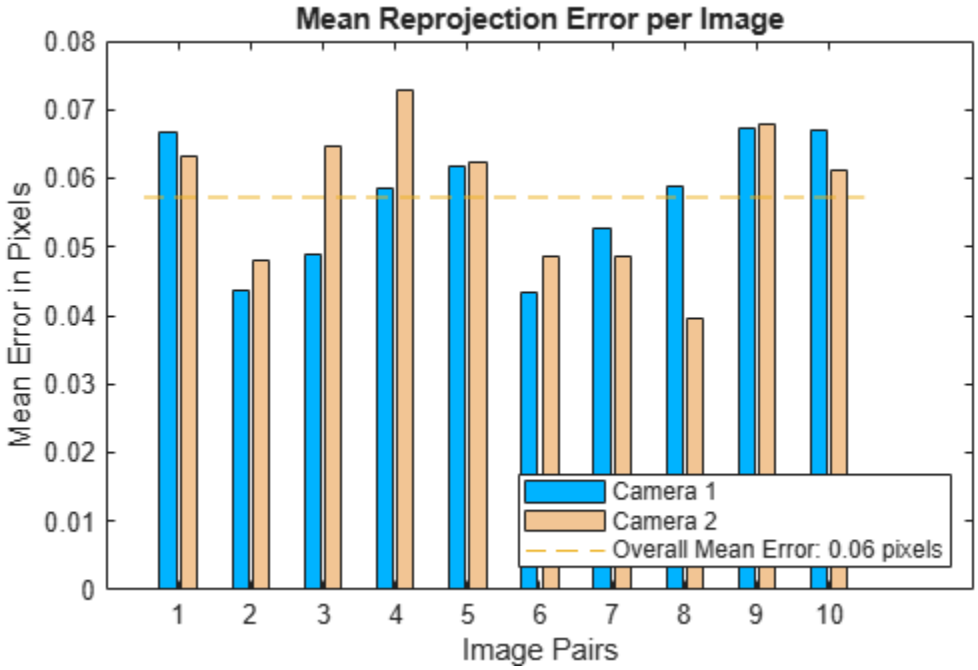
```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

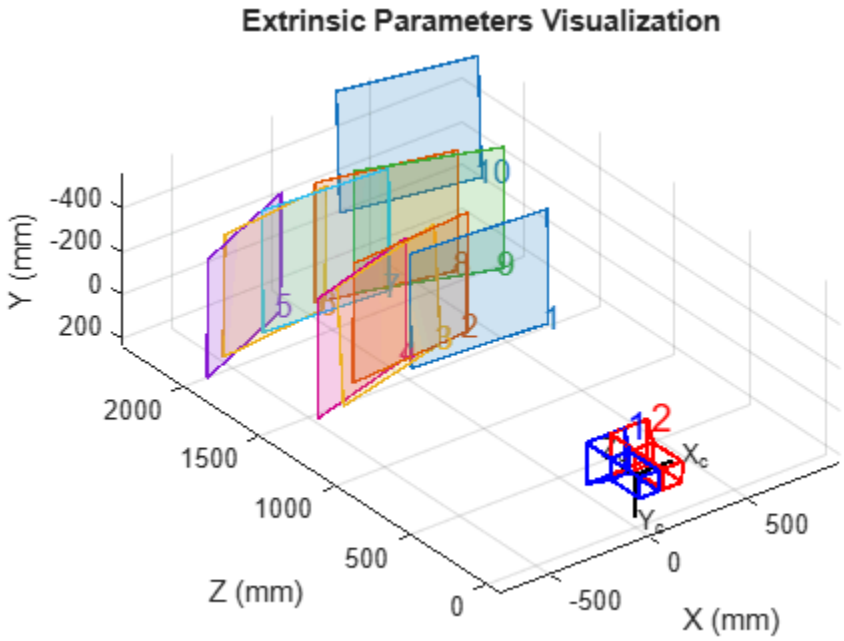
Visualize the calibration accuracy.

```
showReprojectionErrors(params)
```



Visualize camera extrinsics.

```
figure
showExtrinsics(params)
```



Convert to structure.

```
paramsStruct = toStruct(params)
```

```
paramsStruct = struct with fields:
       CameraParameters1: [1×1 struct]
       CameraParameters2: [1×1 struct]
       RotationOfCamera2: [3×3 double]
    TranslationOfCamera2: [-119.8720 -0.4005 -0.0258]
                 Version: [1×1 struct]
     RectificationParams: [1×1 struct]
```

Version History

Introduced in R2014a

R2022b: Supports premultiply matrix convention
Behavior changed in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, some input arguments and properties of `stereoParameters` object have changed to support the premultiply convention.

- You can now specify the `tformOfCamera2` argument as a `rigidtform3d` object, which uses the premultiply convention. Although you can still specify `tformOfCamera2` as a `rigid3d` object, this object uses the postmultiply convention and is not recommended.
- The new `PoseCamera2` property replaces the old `RotationOfCamera2` and `TranslationOfCamera2` properties. You can access the rotation matrix and translation vector by querying the `R` and `Translation` properties of the `rigidtform3d` object stored in the `PoseCamera2` property. The value of the `R` property is the transpose of `RotationOfCamera2`.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Zhang, Z. "A Flexible New Technique for Camera Calibration." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, no. 11 (November 2000): 1330–34. https://doi.org/10.1109/34.888718..

[2] Heikkila, J., and O. Silven. “A Four-Step Camera Calibration Procedure with Implicit Image Correction.” In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1106–12. San Juan, Puerto Rico: IEEE Comput. Soc, 1997. https://doi.org/10.1109/CVPR.1997.609468.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use the `toStruct` method to pass a `stereoParameters` object into generated code. See the “Code Generation for Depth Estimation from Stereo Video” example.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Objects

`stereoCalibrationErrors` | `intrinsicsEstimationErrors` | `extrinsicsEstimationErrors` | `cameraParameters` | `cameraIntrinsics`

Functions

`estimateCameraParameters` | `showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `undistortPoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `patternWorldPoints` | `reconstructScene` | `rectifyStereoImages` | `estimateFundamentalMatrix`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Code Generation for Depth Estimation from Stereo Video”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“Import Stereo Camera Parameters from ROS”

toStruct

Convert a stereo parameters object into a struct

Syntax

```
paramStruct = toStruct(stereoParams)
```

Description

`paramStruct = toStruct(stereoParams)` returns a struct containing the stereo parameters in the `stereoParams` input object. You can use the struct to create an identical `stereoParameters` object. Use the struct for C code generation. You can call `toStruct`, and then pass the resulting structure into the generated code, which recreates the `stereoParameters` object.

Examples

Perform Stereo Camera Calibration

Specify calibration images.

```
leftImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","left"));
rightImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = ...
  detectCheckerboardPoints(leftImages.Files,rightImages.Files);
```

Specify the world coordinates of the checkerboard keypoints. Square size is in millimeters.

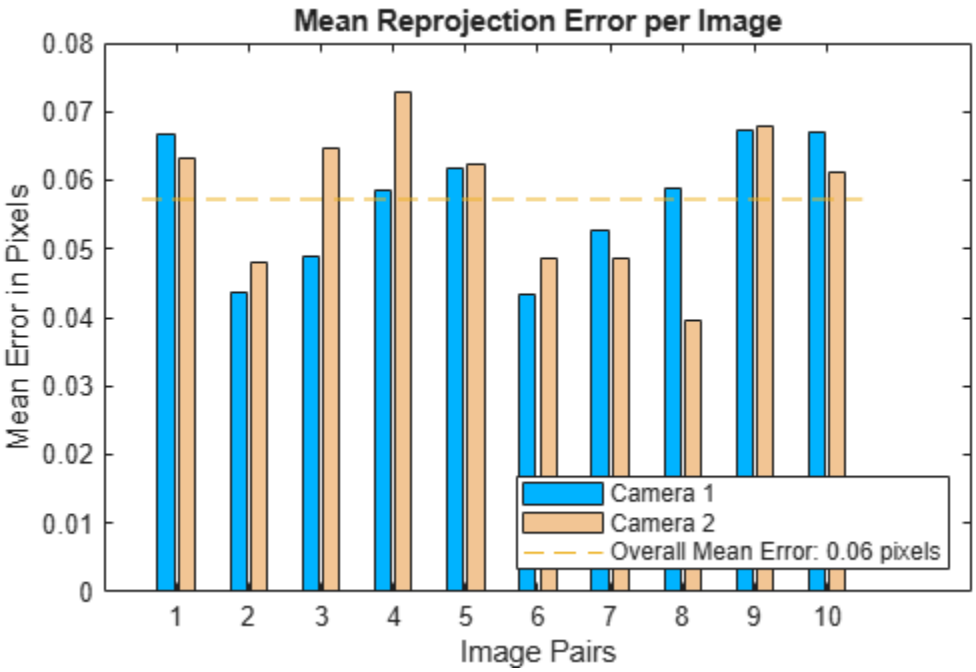
```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

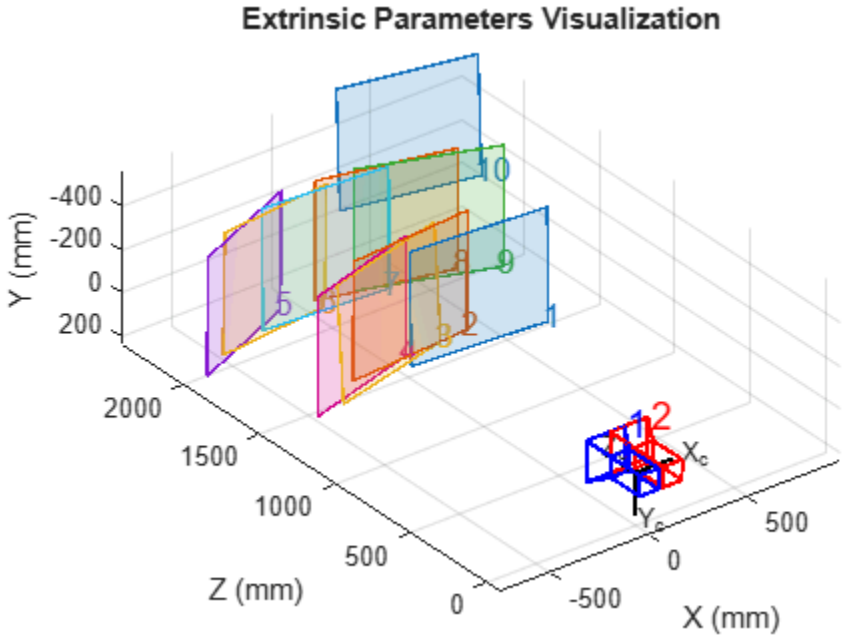
Visualize the calibration accuracy.

```
showReprojectionErrors(params)
```



Visualize camera extrinsics.

```
figure
showExtrinsics(params)
```



Convert to structure.

```
paramsStruct = toStruct(params)
```

```
paramsStruct = struct with fields:
       CameraParameters1: [1×1 struct]
       CameraParameters2: [1×1 struct]
       RotationOfCamera2: [3×3 double]
    TranslationOfCamera2: [-119.8720 -0.4005 -0.0258]
                 Version: [1×1 struct]
     RectificationParams: [1×1 struct]
```

Input Arguments

`stereoParams` — Stereo parameters

`stereoParameters` object

Stereo parameters, specified as a `stereoParameters` object. The object contains the parameters of the stereo camera system.

Output Arguments

`paramStruct` — Stereo parameters

struct

Stereo parameters, returned as a stereo parameters struct.

Version History

Introduced in R2015a

See Also

Objects

`stereoParameters`

Topics

“Code Generation for Depth Estimation from Stereo Video”

toStruct

Convert a camera parameters object into a struct

Syntax

```
paramStruct = toStruct(cameraParams)
```

Description

`paramStruct = toStruct(cameraParams)` returns a struct containing the camera parameters in the `cameraParams` input object. You can use the struct to create an identical `cameraParameters` object. Use the struct for C code generation. You can call `toStruct`, and then pass the resulting structure into the generated code, which re-creates the `cameraParameters` object.

Examples

Remove Distortion from an Image Using Camera Parameters Object

Use the camera calibration functions to remove distortion from an image. This example creates a `cameraParameters` object manually, but in practice, you would use the `estimateCameraParameters` or the **Camera Calibrator** app to derive the object.

Create a `cameraParameters` object manually.

```
k = [715.2699 0 565.6995; 0 711.5281 355.3466; 0 0 1];
radialDistortion = [-0.3361 0.0921];
cameraParams = cameraParameters("K",k,"RadialDistortion",radialDistortion)
```

```
cameraParams =
  cameraParameters with properties:

   Camera Intrinsics
                        Intrinsics: [0×0 cameraIntrinsics]

   Camera Extrinsics
                PatternExtrinsics: [0×0 rigidtform3d]

   Accuracy of Estimation
             MeanReprojectionError: NaN
                ReprojectionErrors: [0×2 double]
                 ReprojectedPoints: [0×2×0×0 double]

   Calibration Settings
                           NumViews: 0
                       PatternCount: 0
                  DetectedKeypoints: [0×2 double]
                        WorldPoints: [0×2 double]
                         WorldUnits: 'mm'
                       EstimateSkew: 0
    NumRadialDistortionCoefficients: 2
```

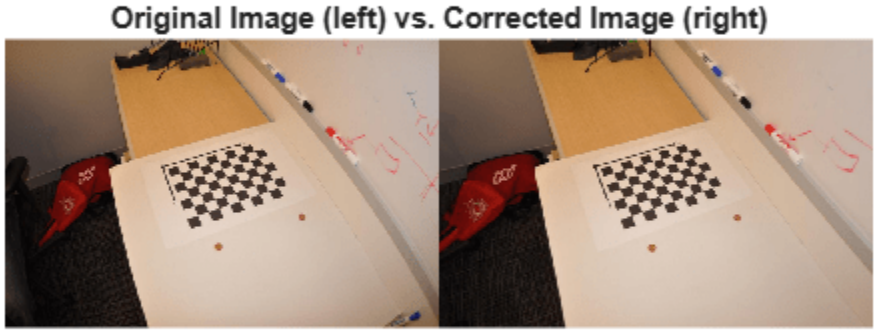
```
        EstimateTangentialDistortion: 0
```

Remove distortion from the images.

```
I = imread(fullfile(matlabroot,"toolbox","vision","visiondata","calibration","mono","image01.jpg
J = undistortImage(I,cameraParams);
```

Display the original and the undistorted images.

```
montage({I,J})
title("Original Image (left) vs. Corrected Image (right)")
```



Input Arguments

`cameraParams` — Camera parameters

`cameraParameters` object

Camera parameters, specified as a `cameraParameters` object. The object contains the parameters for the camera.

Output Arguments

`paramStruct` — Camera parameters

struct

Camera parameters, returned as a camera parameters struct.

Version History

Introduced in R2015a

See Also

Objects

`cameraParameters`

Topics

“Code Generation for Depth Estimation from Stereo Video”

vision.TemplateMatcher

Namespace: `vision`

Locate template in image

Description

To locate a template in an image.

1. Create the `vision.TemplateMatcher` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
tMatcher = vision.TemplateMatcher
tMatcher = vision.TemplateMatcher(Name,Value)
```

Description

`tMatcher = vision.TemplateMatcher` returns a template matcher object, `tMatcher`. This object performs template matching by shifting a template in single-pixel increments throughout the interior of an image.

`tMatcher = vision.TemplateMatcher(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `tMatcher = vision.TemplateMatcher('Metric','Sum of absolute differences')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Metric` — Metric used for template matching source
`'Sum of absolute differences'` (default) | `'Sum of squared differences'` | `'Maximum absolute difference'`

Metric used for template matching, specified as `'Sum of absolute differences'`, `'Sum of squared differences'`, or `'Maximum absolute difference'`.

`OutputValue` — Type of output
`'Best match location'` (default) | `'Metric matrix'`

Type of output, specified as `'Metric matrix'` or `'Best match location'`.

`SearchMethod` — Specify search criteria to find minimum difference between two inputs
`'Exhaustive'` (default) | `'Three-step'`

Specify search criteria to find minimum difference between two inputs, specified as `'Exhaustive'` or `'Three-step'`. If you set this property to `'Exhaustive'`, the object searches for the minimum difference pixel by pixel. If you set this property to `'Three-step'`, the object searches for the minimum difference using a steadily decreasing step size. The `'Three-step'` method is computationally less expensive than the `'Exhaustive'` method, but sometimes does not find the optimal solution. This property applies when you set the `OutputValue` property to `'Best match location'`.

`BestMatchNeighborhoodOutputPort` — Enable metric values output
`false` (default) | `true`

Enable metric values output, specified as `true` or `false`. This property applies when you set the `OutputValue` property to `'Best match location'`.

`NeighborhoodSize` — Size of the metric values
3 (default) | odd number

Size of the metric values, specified as an odd number. The size *N*, of the *N*-by-*N* matrix of metric values as an odd number. For example, if the matrix size is 3-by-3 set this property to 3. This property applies when you set the `OutputValue` property to `'Best match location'` and the `BestMatchNeighborhoodOutputPort` property to `true`.

`ROIInputPort` — Enable ROI specification through input
`false` (default) | `true`

Enable ROI specification through input, specified as `true` or `false`. Set this property to `true` to define the Region of Interest (ROI) over which to perform the template matching. If you set this property to `true`, the ROI must be specified. Otherwise the entire input image is used.

`ROIValidityOutputPort` — Enable output of a flag indicating if any part of ROI is outside input image
`false` (default) | `true`

Enable output of a flag indicating if any part of ROI is outside input image, specified as `true` or `false`. When you set this property to `true`, the object returns an ROI flag. The flag, when set to `false`, indicates a part of the ROI is outside of the input image. This property applies when you set the `ROIInputPort` property to `true`

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest'` | `'Round'` | `'Simplest'` | `'Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
location = tMatcher(I,T)

[location,numberOfValues,numValid] = tMatcher(I,T,ROI)
[location,numberOfValues,numValid,ROIvalid] = tMatcher(I,T,ROI)
[location,ROIvalid] = tMatcher(I,T,ROI)
```

Description

`location = tMatcher(I,T)` returns the [*x* *y*] location coordinates of the best template match relative to the top left corner of the image between the image matrix, `I`, and the template matrix, `T`. The object computes the location by shifting the template in single-pixel increments throughout the interior of the image.

`[location,numberOfValues,numValid] = tMatcher(I,T,ROI)`returns the location of the best template match `location`, the metric values around the best match `numberOfValues`, and a logical flag `numValid`. This applies when you set the `OutputValue` property to `'Best match location'` and the `BestMatchNeighborhoodOutputPort` property to `true`.

`[location,numberOfValues,numValid,ROIvalid] = tMatcher(I,T,ROI)` also returns a logical flag, `ROIvalid` to indicate whether the `ROI` is outside the bounds of the input image `I`. This applies when you set the `OutputValue` property to `'Best match location'`, and the `BestMatchNeighborhoodOutputPort`, `ROIInputPort`, and `ROIValidityOutputPort` properties to `true`.

`[location,ROIvalid] = tMatcher(I,T,ROI)` also returns a logical flag `ROIvalid` indicating if the specified `ROI` is outside the bounds of the input image `I`. This applies when you set the `OutputValue` property to `'Best match location'`, and both the `ROIInputPort` and `ROIValidityOutputPort` properties to `true`.

Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale image

Input image, specified as a 2-D grayscale image. Performance of the `TemplateMatcher` object can depend on the input type. For example, using datatype `double` inputs can result in faster performance.

`T` — Template

binary image | *M*-by-*N* 2-D grayscale image

Input template, specified as 2-D grayscale image.

`ROI` — Input ROI

four-element vector

Input ROI, specified as a four-element vector, [*x y width height*], where the first two elements represent the coordinates of the upper-left corner of the rectangular ROI.

Output Arguments

`location` — Location of best template match

vector

Location of best template match, returned as a vector in the format (*x*,*y*). The coordinates indicate the center of the template relative to the top-left corner of the image. For more details, see “Algorithms” on page 2-1152

`numberOfValues` — Metric value

matrix

Metric value matrix , returned as a matrix. A `false` value for `numValid` indicates that the neighborhood around the best match extended outside the borders of the metric value matrix `numberOfValues`.

`numValid` — Valid neighborhood

`true` | `false`

Valid neighborhood, returned as `true` or `false`. A `false` value for `numValid` indicates that the neighborhood around the best match extended outside the borders of the metric value matrix `numberOfValues`.

`ROIvalid` — Valid ROI

`true` | `false`

Valid ROI neighborhood, returned as `true` or `false`. A `false` value for `ROIvalid` indicates that the ROI is outside the bounds of the input image.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Video Stabilization

This example shows how to remove the effect of camera motion from a video stream.

Introduction

In this example we first define the target to track. In this case, it is the back of a car and the license plate. We also establish a dynamic search region, whose position is determined by the last known target location. We then search for the target only within this search region, which reduces the number of computations required to find the target. In each subsequent video frame, we determine how much the target has moved relative to the previous frame. We use this information to remove unwanted translational camera motions and generate a stabilized video.

Initialization

Create a System object™ to read video from a multimedia file. We set the output to be of intensity only video.

```
% Input video file which needs to be stabilized.
filename = "shaky_car.avi";

hVideoSource = VideoReader(filename);
```

Create a template matcher System object to compute the location of the best match of the target in the video frame. We use this location to find translation between successive video frames.

```
hTM = vision.TemplateMatcher("ROIInputPort", true, ...
                             "BestMatchNeighborhoodOutputPort", true);
```

Create a System object to display the original video and the stabilized video.

```
hVideoOut = vision.VideoPlayer("Name", "Video Stabilization");
hVideoOut.Position(1) = round(0.4*hVideoOut.Position(1));
hVideoOut.Position(2) = round(1.5*(hVideoOut.Position(2)));
hVideoOut.Position(3:4) = [650 350];
```

Here we initialize some variables used in the processing loop.

```
pos.template_orig = [109 100]; % [x y] upper left corner
pos.template_size = [22 18];   % [width height]
```

```
pos.search_border = [15 10];   % max horizontal and vertical displacement
pos.template_center = floor((pos.template_size-1)/2);
pos.template_center_pos = (pos.template_orig + pos.template_center - 1);
W = hVideoSource.Width; % Width in pixels
H = hVideoSource.Height; % Height in pixels
BorderCols = [1:pos.search_border(1)+4 W-pos.search_border(1)+4:W];
BorderRows = [1:pos.search_border(2)+4 H-pos.search_border(2)+4:H];
sz = [W, H];
TargetRowIndices = ...
  pos.template_orig(2)-1:pos.template_orig(2)+pos.template_size(2)-2;
TargetColIndices = ...
  pos.template_orig(1)-1:pos.template_orig(1)+pos.template_size(1)-2;
SearchRegion = pos.template_orig - pos.search_border - 1;
Offset = [0 0];
Target = zeros(18,22);
firstTime = true;
```

Stream Processing Loop

This is the main processing loop which uses the objects we instantiated above to stabilize the input video.

```
while hasFrame(hVideoSource)
    input = im2gray(im2double(readFrame(hVideoSource)));

    % Find location of Target in the input video frame
    if firstTime
      Idx = int32(pos.template_center_pos);
      MotionVector = [0 0];
      firstTime = false;
    else
      IdxPrev = Idx;

      ROI = [SearchRegion, pos.template_size+2*pos.search_border];
      Idx = hTM(input,Target,ROI);

      MotionVector = double(Idx-IdxPrev);
    end

    [Offset, SearchRegion] = updatesearch(sz, MotionVector, ...
        SearchRegion, Offset, pos);

    % Translate video frame to offset the camera motion
    Stabilized = imtranslate(input, Offset, "linear");

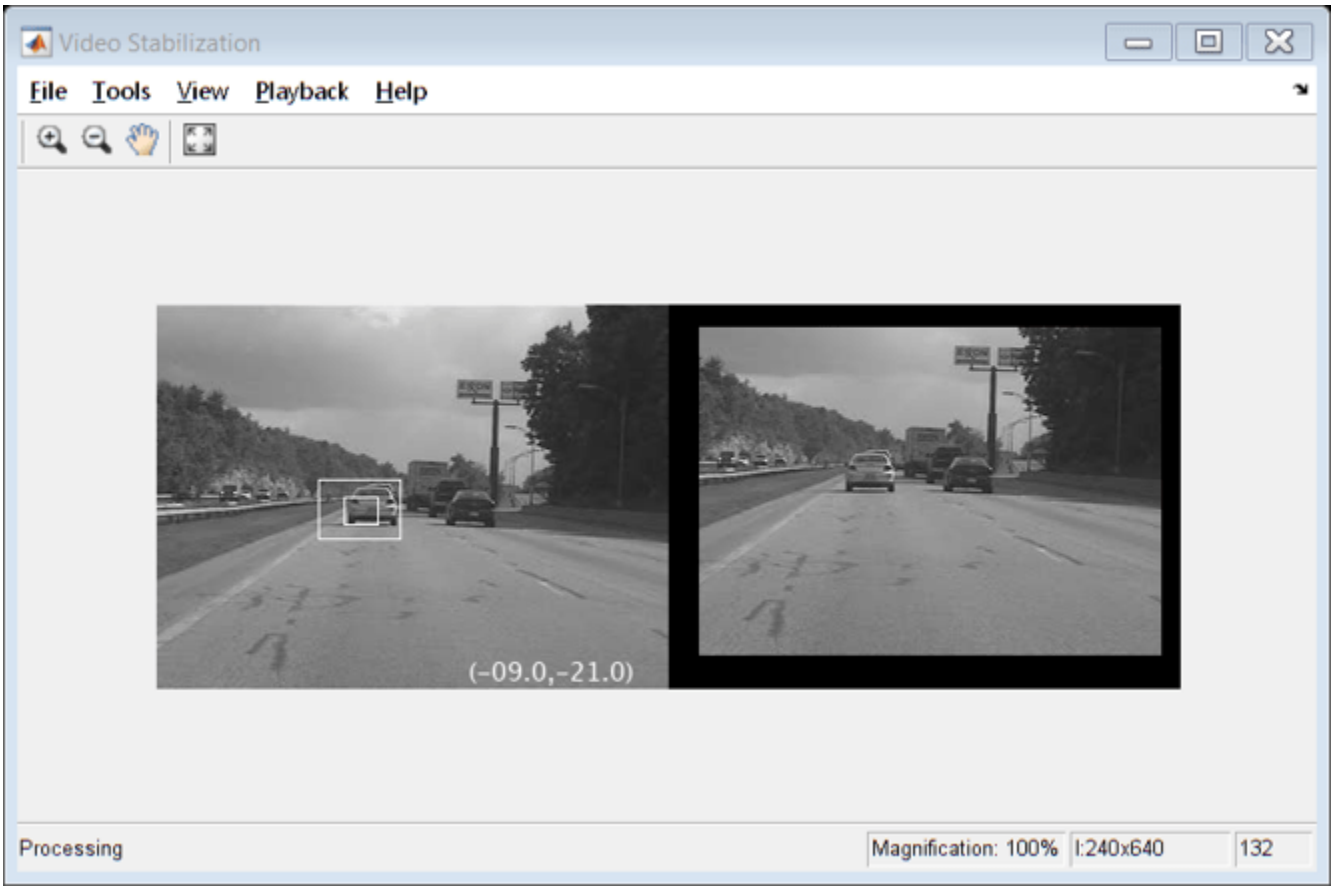
    Target = Stabilized(TargetRowIndices, TargetColIndices);

    % Add black border for display
    Stabilized(:, BorderCols) = 0;
    Stabilized(BorderRows, :) = 0;

    TargetRect = [pos.template_orig-Offset, pos.template_size];
    SearchRegionRect = [SearchRegion, pos.template_size + 2*pos.search_border];

    % Draw rectangles on input to show target and search region
    input = insertShape(input, "rectangle", [TargetRect; SearchRegionRect],...
                        "Color", "white");
    % Display the offset (displacement) values on the input image
```

```
    txt = sprintf("(%+05.1f,%+05.1f)", Offset);
    input = insertText(input(:,:,1),[191 215],txt,"FontSize",16, ...
                    "TextColor", "white", "BoxOpacity", 0);
    % Display video
    hVideoOut([input(:,:,1) Stabilized]);
end
```



Conclusion

Using the Computer Vision Toolbox™ functionality from MATLAB® command line it is easy to implement complex systems like video stabilization.

Appendix

The following helper function is used in this example.

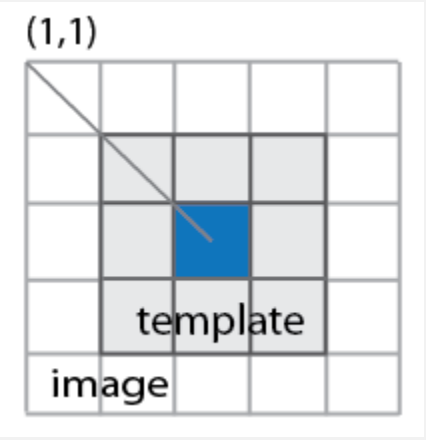
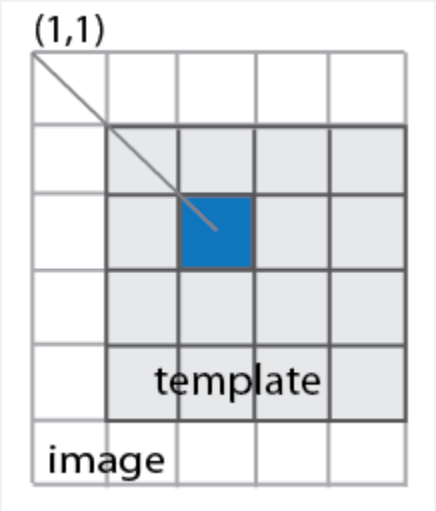
- updatesearch.m

Algorithms

Typical use of the template matcher involves finding a small region within a larger image. The region is specified by the template image which can be as large as the input image, but which is typically smaller than the input image.

The object outputs the best match coordinates, relative to the top-left corner of the image. The [*x* *y*] coordinates of the location correspond to the center of the template. When you use a template with

an odd number of pixels, the object uses the center of the template. When you use a template with an even number of pixels, the object uses the centered upper-left pixel for the location. The following table shows how the object outputs the location (`LOC`), of odd and even templates:

Odd number of pixels in template	Even number of pixels in template
	

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`opticalFlowHS` | `opticalFlowLKDoG` | `opticalFlowLK` | `opticalFlowFarneback` | `insertMarker`

vision.Variance

Namespace: `vision`

(Removed) Find variance values in input or sequence of inputs

Note `vision.Variance` function has been removed. Use the `var` and `movvar` functions to find variance values, instead.

Description

Find the variance values in an input or sequence of inputs.

To track a set of points:

1. Create the `vision.Variance` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
varObj = vision.Variance
varObj = vision.Variance(Name,Value)
```

Description

`varObj = vision.Variance` returns an object, `varObj`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`varObj = vision.Variance(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `varObj = vision.Variance('RunningVariance',false)`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`RunningVariance` — Calculate over single input or multiple inputs

`false` (default) | `true`

Calculate over single input or multiple inputs, specified as `true` or `false`. When you set this property to `true`, the object computes the variance value over a sequence of inputs. When you set this property to `false`, the object computes the variance value over the current input.

`ResetInputPort` — Additional input to enable resetting of running variance
`false` (default) | `true`

Additional input to enable resetting of running variance, specified as `true` or `false`. When you set this property to `true`, a reset input must be specified to reset the running variance. This property applies only when you set the `RunningVariance` property to `true`.

`ResetCondition` — Condition that triggers resetting of running variance
`'Non-zero'` (default) | `'Rising edge'` | `'Falling edge'` | `'Either edge'`

Condition that triggers resetting of running variance, specified as `'Rising edge'`, `'Falling edge'`, `'Either edge'`, or `'Non-zero'`. This property applies only when you set the `ResetInputPort` property to `true`.

`Dimension` — Dimension to operate along
`'Column'` (default) | `{'All'}` | `'Row'` | `'Custom'`

Dimension to operate along, specified as `{'All'}`, `'Row'`, `'Column'`, or `'Custom'`. This property applies only when you set the `RunningVariance` property to `false`.

`CustomDimension` — Numerical dimension to calculate over
1 (default) | numerical scalar

Numerical dimension to calculate over, specified as a numerical scalar. This property only applies when you set the `Dimension` property to `'Custom'`.

`ROIProcessing` — Enable region-of-interest processing
`false` (default) | `true`

Enable region-of-interest processing, specified as `true` or `false`. This property applies when you set the `Dimension` property to `'All'` and the `RunningVariance` property to `false`.

`ROIForm` — Type of ROI
`'Rectangles'` (default) | `Lines'` | `'Label matrix''Binary mask'`

Type of ROI, specified as `'Rectangles'`, `'Lines'`, `'Label matrix'`, or `'Binary mask'`. This property applies only when you set the `ROIProcessing` property to `true`.

`ROIStatistics` — Calculate statistics for each ROI or one for all ROIs
`'Individual statistics for each ROI'` (default) | `'Single statistic for all ROIs'`

Calculate statistics for each ROI or one for all ROIs, specified as `'Individual statistics for each ROI'` or `'Single statistic for all ROIs'`. This property applies only when you set the `'ROIForm'` property to `'Rectangles'`, `'Lines'`, or `'Label matrix'`.

`ValidityOutputPort` — Flag to indicate if any part of ROI is outside input image
`false` (default) | `true`

Flag to indicate if any part of ROI is outside input image, specified as `true` or `false`.

This applies when you set the `ROIForm` property to `'Lines'` or `'Rectangles'`.

Set this property to `true` to return the validity of the specified label numbers. This applies when you set the `ROIForm` property to `'Label matrix'`.

Fixed-Point Properties

`RoundingMethod` — Rounding method for fixed-point operations
`'Floor'` (default) | `'Ceiling'` | `'Convergent'` | `'Nearest''Round'` | `'Simplest''Zero'`

Rounding method for fixed-point operations, specified as `'Floor'`, `'Ceiling'`, `'Convergent'`, `'Nearest'` , `'Round'` , `'Simplest'` , or `'Zero'`.

`OverflowAction` — Action to take when integer input is out-of-range
`'Wrap'` (default) | `'Saturate'`

Action to take when integer input is out-of-range, specified as `'Wrap'` or `'Saturate'`.

`ProductDataType` — Product data type
`'Same as input'` (default) | `'Custom'`

Product data type, specified as `'Same as input'` or `'Custom'`.

`CustomProductDataType` — Product word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Product word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

`AccumulatorDataType` — Data type of accumulator
`'Same as product'` (default) | `'Same as input'` | `'Custom'`

Data type of accumulator, specified as `'Same as product'`, `'Same as input'`, or `'Custom'`.

`CustomAccumulatorDataType` — Accumulator word and fraction lengths
`numerictype(true,32,30)` (default) | scaled `numerictype` object

Accumulator word and fraction lengths, specified as a scaled `numerictype` object. This property applies only when you set the `AccumulatorDataType` property to `'Custom'`.

Usage

Syntax

```
[value,index] = varObj(input)
index = varObj(input)

[___] = varObj(I,ROI)
[___,flag] = varObj(I,ROI)

[___] = varObj(I,label,labelNumbers)
[___,flag] = varObj(I,label,labelNumbers)
```

Description

`[value,index] = varObj(input)` returns the variance value and index of the input.

`index = varObj(input)` returns the one-based index of the variance value when you set the `IndexOutputPort` property to `true` and the `ValueOutputPort` property to `false`. The `RunningVariance` property must be set to `false`.

`[ ___ ] = varObj(I,ROI)` returns the variance value in the input image within the given region of interest.

`[ ___ ,flag] = varObj(I,ROI)` additionally returns a flag to indicate whether the given ROI is within the bounds of the image.

`[ ___ ] = varObj(I,label,labelNumbers)` returns the variance of the input image for a region the labels specified in the `labelNumbers` vector. The regions are defined and labeled in the `label` matrix.

`[ ___ ,flag] = varObj(I,label,labelNumbers)` additionally returns a flag to indicate whether the input label numbers are valid.

Input Arguments

`input` — Input data
vector | matrix | multidimensional array

Input data, specified as a vector, matrix, or multidimensional array.

`ROI` — Region of interest
four-element vector

Region of interest, specified as a four-element vector, [*x y width height*]. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Lines'`, `'Rectangles'`, or `'Binary Mask'`.

`label` — Label numbers
matrix

Label numbers, specified as a matrix. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

`labelNumbers` — Label numbers
vector

Label numbers, specified as a vector. This option is available when you set the `ROIProcessing` property to `true` and the `ROIForm` property to `'Label matrix'`.

Output Arguments

`value` — Variance value
same as input

Variance value, returned as the same data type as the input

`index` — Index to variance value
one-based index

Index to variance value, returned as a one-based index.

`flag` — Flag for valid data
`true` | `false`

Flag for valid data, returned as `true` or `false`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Determine Variance in Grayscale Image

1. Read a color image.

```
img = imread('peppers.png');
```

2. Convert the image to grayscale.

```
img = im2single(rgb2gray(img));
```

3. Find the variance.

```
varObj = vision.Variance;
varValue = varObj(img);
```

Version History

Introduced in R2012a

R2024b: `vision.Variance` function has been removed

Errors starting in R2024b

`vision.Variance` function has been removed. Use the `var` and `movvar` functions to find variance values, instead.

R2024a: `vision.Variance` function will be removed

Warns starting in R2024a

`vision.Variance` function will be removed in a future release. Use the `var` and `movvar` functions to find variance values, instead.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`var` | `movvar`

vision.VideoPlayer

Namespace: `vision`

Play video or display image

Description

Play a video or display image sequences.

Note If you own the MATLAB Coder product, you can generate C or C++ code from MATLAB code in which an instance of this system object is created. When you do so, the scope system object is automatically declared as an extrinsic variable. In this manner, you are able to see the scope display in the same way that you would see a figure using the `plot` function, without directly generating code from it. For the full list of system objects supporting code generation, see Code Generation Support, Usage Notes, and Limitations.

Creation

Syntax

```
videoPlayer = vision.VideoPlayer
videoPlayer = vision.VideoPlayer( ___ ,Name,Value)
```

Description

`videoPlayer = vision.VideoPlayer` returns a video player object, `videoPlayer`, for displaying video frames.

`videoPlayer = vision.VideoPlayer( ___ ,Name,Value)` specifies properties using one or more name-value arguments in addition to the previous syntax. For example, `videoPlayer = vision.VideoPlayer('Name','Caption title')` sets the caption display on the video player window.

Properties

`Name` — Caption display on video player window

`Video` (default) | character vector

Caption display on video player window, specified as a character vector.

Tunable: Yes

`Position` — Size and position of the video player window in pixels

four-element vector

Size and position of the video player window in pixels, specified as a four-element vector, `[left bottom width height]`. The default size depends on your screen resolution. The window is positioned in the center of the screen, 400 pixels in width by 300 pixels in height.

Tunable: Yes

Usage

Syntax

```
videoPlayer(videoFrame)
```

Description

`videoPlayer(videoFrame)` displays one grayscale or truecolor RGB video frame,`videoFrame`, in the video player.

Input Arguments

`videoFrame` — Video frame

truecolor | 2-D grayscale image

Video frame, specified as a truecolor or 2-D grayscale image.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Play a Video File

Read video from a file and set up player object.

```
videoReader = VideoReader('viplanedeparture.mp4');
videoPlayer = vision.VideoPlayer;
```

Play video.

```
while hasFrame(videoReader)
   frame = readFrame(videoReader);
   videoPlayer(frame);
end
```



Close the video player.

```
release(videoPlayer);
```



Version History

Introduced in R2012a

See Also

`vision.DeployableVideoPlayer` | `VideoReader` | `vision.VideoFileWriter` | `imshow` | Video Viewer

Topics
“Face Detection and Tracking Using CAMShift”
“Face Detection and Tracking Using the KLT Algorithm”
“Face Detection and Tracking Using Live Video Acquisition”
“Video Display in a Custom User Interface”

fastRCNNObjectDetector

(Not recommended) Detect objects using Fast R-CNN deep learning detector

Note `fastRCNNObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Description

The `fastRCNNObjectDetector` object detects objects from an image, using a Fast R-CNN (regions with convolution neural networks) object detector. To detect objects in an image, pass the trained detector to the `detect` function. To classify image regions, pass the detector to the `classifyRegions` function.

When using the `detect` or `classifyRegions` functions with `fastRCNNObjectDetector`, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

Creation

Create a `fastRCNNObjectDetector` object by calling the `trainFastRCNNObjectDetector` function with training data (requires Deep Learning Toolbox).

```
detector = trainFastRCNNObjectDetector(trainingData,...)
```

Properties

`ModelName` — Name of classification model
character vector | string scalar

Name of the classification model, specified as a character vector or string scalar. By default, the name is set to the heading of the second column of the `trainingData` table specified in the `trainFastRCNNObjectDetector` function. You can modify this name after creating your `fastRCNNObjectDetector` object.

Example: `'stopSign'`

`Network` — Trained Fast R-CNN object detection network
DAGNetwork object

This property is read-only.

Trained Fast R-CNN detection network used within the Fast R-CNN detector, specified as a `DAGNetwork` object. This object stores the layers that define the convolutional neural network used within the Fast R-CNN detector. This network classifies region proposals produced by the `RegionProposalFcn` property.

`RegionProposalFcn` — Region proposal method
function handle

Region proposal method, specified as a function handle.

`ClassNames` — Object class names
cell array

This property is read-only.

Names of the object classes that the Fast R-CNN detector was trained to find, specified as a cell array. This property is set by the `trainingData` input argument for the `trainFastRCNNObjectDetector` function. Specify the class names as part of the `trainingData` table.

`MinObjectSize` — Minimum object size supported
[*height width*] vector

This property is read-only.

Minimum object size supported by the Fast R-CNN network, specified as a [*height width*] vector. The minimum size depends on the network architecture.

Object Functions

detect	(Not recommended) Detect objects using Fast R-CNN object detector
classifyRegions	(Not recommended) Classify objects in image regions using Fast R-CNN object detector

Examples

Detect Vehicles Using Faster R-CNN

Detect vehicles within an image by using a Faster R-CNN object detector.

Load a Faster R-CNN object detector pretrained to detect vehicles.

```
data = load('fasterRCNNVehicleTrainingData.mat', 'detector');
detector = data.detector;
```

Read in a test image.

```
I = imread('highway.png');
imshow(I)
```



Run the detector on the image and inspect the results. The labels come from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 2×4

   150    86    80    72
    91    89    67    48

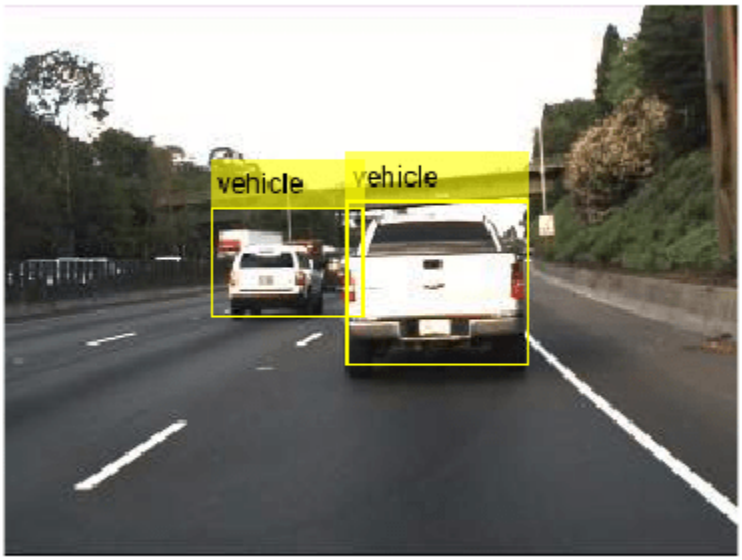

scores = 2×1 single column vector

    1.0000
    0.9001


labels = 2×1 categorical
     vehicle
     vehicle
```

The detector has high confidence in the detections. Annotate the image with the bounding boxes for the detections and the corresponding detection scores.

```
detectedI = insertObjectAnnotation(I,'Rectangle',bboxes,cellstr(labels));
figure
imshow(detectedI)
```



Version History

Introduced in R2017a

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

References

[1] Girshick, Ross, et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” *2014 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2014, pp. 580–87. *DOI.org (Crossref)*, https://doi.org/10.1109/CVPR.2014.81.

[2] Girshick, Ross. “Fast R-CNN.” *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015, pp. 1440–48. *DOI.org (Crossref)*, https://doi.org/10.1109/ICCV.2015.169.

[3] Zitnick, C. Lawrence, and Dollar, Piotr. “Edge boxes: Locating object proposals from edges.” *Computer Vision-ECCV 2014*, Springer International Publishing, 2014. 391-405. *DOI.org (Crossref)*, https://doi.org/10.1007/978-3-319-10602-1_26.

[4] Ren, Shaoqing, et al. “Faster R-CNN: Towards real-time object detection with region proposal networks.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 2016. 1137-1149. *DOI.org (Crossref)*, https://doi.org/10.1109/TPAMI.2016.2577031.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainFastRCNNObjectDetector` | `trainFasterRCNNObjectDetector` | `trainACFObjectDetector` | `selectStrongestBboxMulticlass`

Objects
`boxLabelDatastore`

Topics
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Datastores for Deep Learning” (Deep Learning Toolbox)

fasterRCNNObjectDetector

(Not recommended) Detect objects using Faster R-CNN deep learning detector

Note `fasterRCNNObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see "Version History".

Description

The `fasterRCNNObjectDetector` object detects objects from an image, using a Faster R-CNN (regions with convolution neural networks) object detector. To detect objects in an image, pass the trained detector to the `detect` function.

When using the `detect` function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see "GPU Computing Requirements" (Parallel Computing Toolbox).

Creation

Create a `fasterRCNNObjectDetector` object by calling the `trainFasterRCNNObjectDetector` function with training data (requires Deep Learning Toolbox).

```
detector = trainFasterRCNNObjectDetector(trainingData,...)
```

Properties

`ModelName` — Name of classification model
character vector | string scalar

This property is read-only.

Name of the classification model, specified as a character vector or string scalar. By default, the name is set to the heading of the second column of the `trainingData` table specified in the `trainFasterRCNNObjectDetector` function. You can modify this name after creating your `fasterRCNNObjectDetector` object.

`Network` — Trained Fast R-CNN object detection network
`DAGNetwork` object

This property is read-only.

Trained Fast R-CNN object detection network, specified as a `DAGNetwork` object. This object stores the layers that define the convolutional neural network used within the Faster R-CNN detector.

`AnchorBoxes` — Size of anchor boxes
M-by-2 matrix

This property is read-only.

Size of anchor boxes, specified as an *M*-by-2 matrix, where each row is in the format [*height width*]. This value is set during training.

`ClassNames` — Object class names
cell array

This property is read-only.

Names of the object classes that the Faster R-CNN detector was trained to find, specified as a cell array. This property is set by the `trainingData` input argument for the `trainFasterRCNNObjectDetector` function. Specify the class names as part of the `trainingData` table.

`MinObjectSize` — Minimum object size supported
[*height width*] vector

This property is read-only.

Minimum object size supported by the Faster R-CNN network, specified as a [*height width*] vector. The minimum size depends on the network architecture.

Object Functions

detect (Not recommended) Detect objects using Faster R-CNN object detector

Examples

Object Detection Using Faster R-CNN Deep Learning

This example shows how to train a Faster R-CNN (regions with convolutional neural networks) object detector.

Deep learning is a powerful machine learning technique that you can use to train robust object detectors. Several deep learning techniques for object detection exist, including Faster R-CNN and you only look once (YOLO) v2. This example trains a Faster R-CNN vehicle detector using the `trainFasterRCNNObjectDetector` function. For more information, see “Object Detection”.

Download Pretrained Detector

Download a pretrained detector to avoid having to wait for training to complete. If you want to train the detector, set the `doTraining` variable to true.

```
doTraining = false;
if ~doTraining && ~exist("fasterRCNNResNet50EndToEndVehicleExample.mat","file")
    disp("Downloading pretrained detector (118 MB)...");
    pretrainedURL = "https://www.mathworks.com/supportfiles/vision/data/fasterRCNNResNet50EndToEn
    websave("fasterRCNNResNet50EndToEndVehicleExample.mat",pretrainedURL);
end
```

```
Downloading pretrained detector (118 MB)...
```

Load Data Set

This example uses a small labeled dataset that contains 295 images. Many of these images come from the Caltech Cars 1999 and 2001 data sets, created by Pietro Perona and used with permission. Each image contains one or two labeled instances of a vehicle. A small dataset is useful for exploring the Faster R-CNN training procedure, but in practice, more labeled images are needed to train a robust detector. Unzip the vehicle images and load the vehicle ground truth data.

```
unzip vehicleDatasetImages.zip
data = load("vehicleDatasetGroundTruth.mat");
vehicleDataset = data.vehicleDataset;
```

The vehicle data is stored in a two-column table, where the first column contains the image file paths and the second column contains the vehicle bounding boxes.

Split the dataset into training, validation, and test sets. Select 60% of the data for training, 10% for validation, and the rest for testing the trained detector.

```
rng(0)
shuffledIndices = randperm(height(vehicleDataset));
idx = floor(0.6 * height(vehicleDataset));

trainingIdx = 1:idx;
trainingDataTbl = vehicleDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.1 * length(shuffledIndices) );
validationDataTbl = vehicleDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
testDataTbl = vehicleDataset(shuffledIndices(testIdx),:);
```

Use `imageDatastore` and `boxLabelDatastore` to create datastores for loading the image and label data during training and evaluation.

```
imdsTrain = imageDatastore(trainingDataTbl{:,"imageFilename"});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,"vehicle"));

imdsValidation = imageDatastore(validationDataTbl{:,"imageFilename"});
bldsValidation = boxLabelDatastore(validationDataTbl(:,"vehicle"));

imdsTest = imageDatastore(testDataTbl{:,"imageFilename"});
bldsTest = boxLabelDatastore(testDataTbl(:,"vehicle"));
```

Combine image and box label datastores.

```
trainingData = combine(imdsTrain,bldsTrain);
validationData = combine(imdsValidation,bldsValidation);
testData = combine(imdsTest,bldsTest);
```

Display one of the training images and box labels.

```
data = read(trainingData);
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,"rectangle",bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```



Create Faster R-CNN Detection Network

A Faster R-CNN object detection network is composed of a feature extraction network followed by two subnetworks. The feature extraction network is typically a pretrained CNN, such as ResNet-50 or Inception v3. The first subnetwork following the feature extraction network is a region proposal network (RPN) trained to generate object proposals - areas in the image where objects are likely to exist. The second subnetwork is trained to predict the actual class of each object proposal.

The feature extraction network is typically a pretrained CNN (for details, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox)). This example uses ResNet-50 for feature extraction. You can also use other pretrained networks such as MobileNet v2 or ResNet-18, depending on your application requirements.

Use `fasterRCNNLayers` to create a Faster R-CNN network automatically given a pretrained feature extraction network. `fasterRCNNLayers` requires you to specify several inputs that parameterize a Faster R-CNN network:

- Network input size
- Anchor boxes
- Feature extraction network

First, specify the network input size. When choosing the network input size, consider the minimum size required to run the network itself, the size of the training images, and the computational cost incurred by processing data at the selected size. When feasible, choose a network input size that is

close to the size of the training image and larger than the input size required for the network. To reduce the computational cost of running the example, specify a network input size of [224 224 3], which is the minimum size required to run the network.

```
inputSize = [224 224 3];
```

Note that the training images used in this example are bigger than 224-by-224 and vary in size, so you must resize the images in a preprocessing step prior to training.

Next, use `estimateAnchorBoxes` to estimate anchor boxes based on the size of objects in the training data. To account for the resizing of the images prior to training, resize the training data for estimating anchor boxes. Use `transform` to preprocess the training data, then define the number of anchor boxes and estimate the anchor boxes.

```
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 3;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)
```

```
anchorBoxes = 3×2

     38     29
    150    125
     80     77
```

For more information on choosing anchor boxes, see “Estimate Anchor Boxes from Training Data” (Computer Vision Toolbox™) and “Anchor Boxes for Object Detection”.

Now, use `resnet50` to load a pretrained ResNet-50 model.

```
featureExtractionNetwork = resnet50;
```

Select `"activation_40_relu"` as the feature extraction layer. This feature extraction layer outputs feature maps that are downsampled by a factor of 16. This amount of downsampling is a good trade-off between spatial resolution and the strength of the extracted features, as features extracted further down the network encode stronger image features at the cost of spatial resolution. Choosing the optimal feature extraction layer requires empirical analysis. You can use `analyzeNetwork` to find the names of other potential feature extraction layers within a network.

```
featureLayer = "activation_40_relu";
```

Define the number of classes to detect.

```
numClasses = width(vehicleDataset)-1;
```

Create the Faster R-CNN object detection network.

```
lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer
```

You can visualize the network using `analyzeNetwork` or Deep Network Designer from Deep Learning Toolbox™.

If more control is required over the Faster R-CNN network architecture, use Deep Network Designer to design the Faster R-CNN detection network manually. For more information, see “Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”.

Data Augmentation

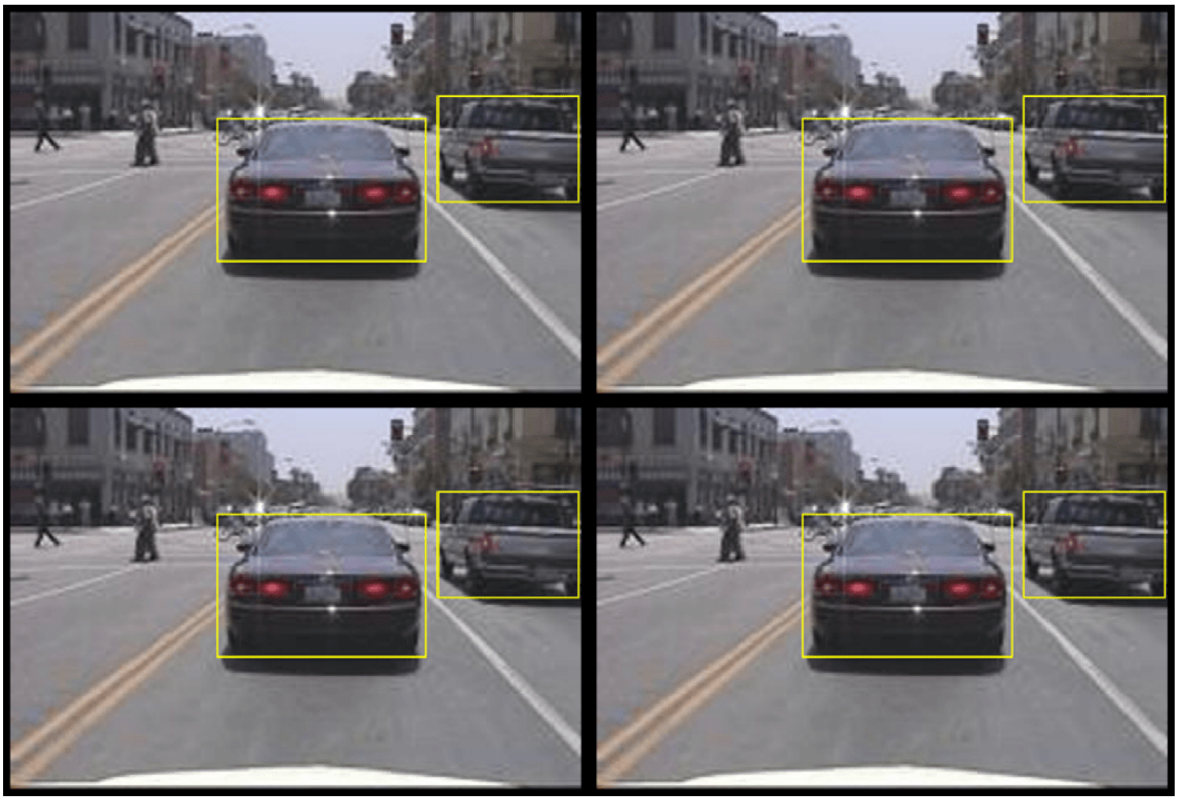
Data augmentation is used to improve network accuracy by randomly transforming the original data during training. By using data augmentation, you can add more variety to the training data without actually having to increase the number of labeled training samples.

Use `transform` to augment the training data by randomly flipping the image and associated box labels horizontally. Note that data augmentation is not applied to test and validation data. Ideally, test and validation data are representative of the original data and are left unmodified for unbiased evaluation.

```
augmentedTrainingData = transform(trainingData,@augmentData);
```

Read the same image multiple times and display the augmented training data.

```
augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1},"rectangle",data{2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData,BorderSize=10)
```



Preprocess Training Data

Preprocess the augmented training data, and the validation data to prepare for training.

```
trainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));
validationData = transform(validationData,@(data)preprocessData(data,inputSize));
```

Read the preprocessed data.

```
data = read(trainingData);
```

Display the image and box bounding boxes.

```
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,"rectangle",bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```



Train Faster R-CNN

Use `trainingOptions` to specify network training options. Set `"ValidationData"` to the preprocessed validation data. Set `"CheckpointPath"` to a temporary location. This enables the saving of partially trained detectors during the training process. If training is interrupted, such as by a power outage or system failure, you can resume training from the saved checkpoint.

```
options = trainingOptions("sgdm",...
    MaxEpochs=10,...
    MiniBatchSize=2,...
    InitialLearnRate=1e-3,...
    CheckpointPath=tempdir,...
    ValidationData=validationData);
```

Use `trainFasterRCNNObjectDetector` to train Faster R-CNN object detector if `doTraining` is true. Otherwise, load the pretrained network.

```
if doTraining
    % Train the Faster R-CNN detector.
    % * Adjust NegativeOverlapRange and PositiveOverlapRange to ensure
    %   that training samples tightly overlap with ground truth.
    [detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
        NegativeOverlapRange=[0 0.3], ...
        PositiveOverlapRange=[0.6 1]);
else
    % Load pretrained detector for the example.
    pretrained = load("fasterRCNNResNet50EndToEndVehicleExample.mat");
    detector = pretrained.detector;
end
```

This example was verified on an Nvidia(TM) Titan X GPU with 12 GB of memory. Training the network took approximately 20 minutes. The training time varies depending on the hardware you use.

As a quick check, run the detector on one test image. Make sure you resize the image to the same size as the training images.

```
I = imread(testDataTbl.imageFilename{3});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

Display the results.

```
I = insertObjectAnnotation(I,"rectangle",bboxes,scores);
figure
imshow(I)
```



Evaluate Detector Using Test Set

Evaluate the trained object detector on a large set of images to measure the performance. Computer Vision Toolbox™ provides an object detector evaluation function (`evaluateObjectDetection`) to measure common metrics such as average precision and log-average miss rate. For this example, use the average precision metric to evaluate performance. The average precision provides a single number that incorporates the ability of the detector to make correct classifications (precision) and the ability of the detector to find all relevant objects (recall).

Apply the same preprocessing transform to the test data as for the training data.

```
testData = transform(testData,@(data)preprocessData(data,inputSize));
```

Run the detector on all the test images. Set the detection threshold to a low value to detect as many objects as possible. This helps you evaluate the detector precision across the full range of recall values.

```
detectionResults = detect(detector,testData,...
    Threshold=0.2,...
    MiniBatchSize=4);
```

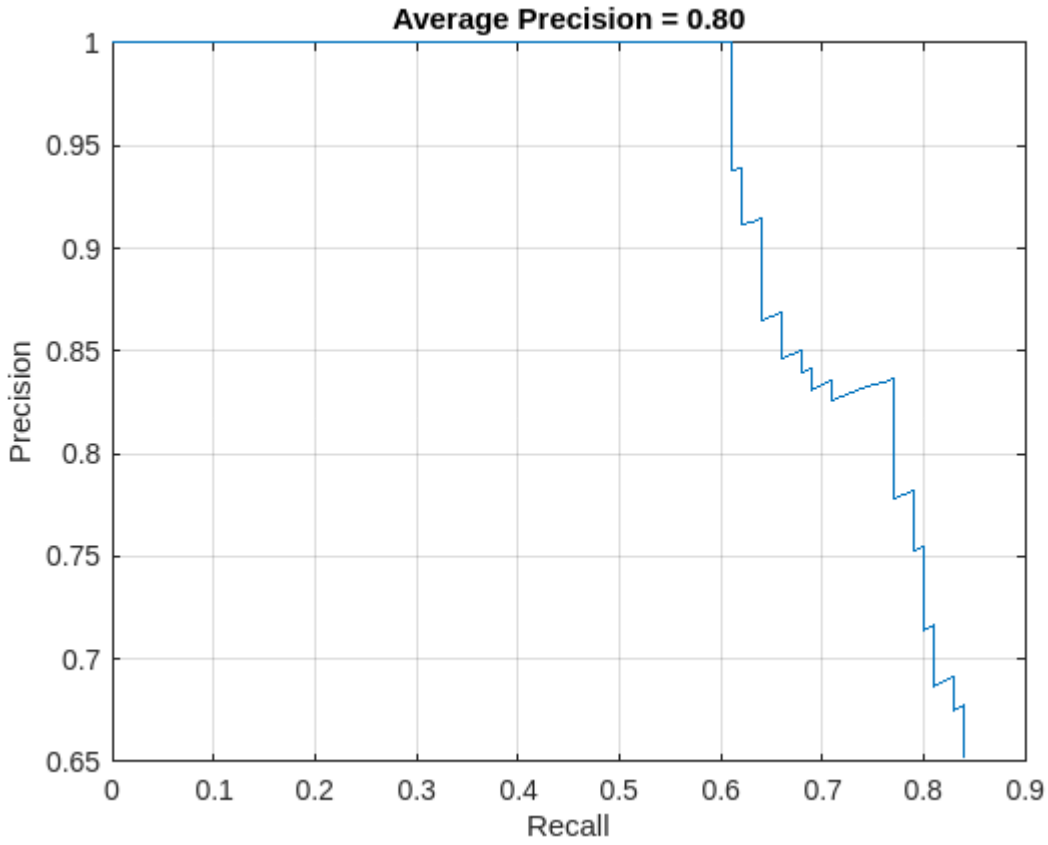
Evaluate the object detector using the average precision metric.

```
metrics = evaluateObjectDetection(detectionResults,testData);
[precision,recall] = precisionRecall(metrics,ClassName="vehicle");
AP = averagePrecision(metrics,ClassName="vehicle");
```

The precision-recall (PR) curve highlights how precise a detector is at varying levels of recall. The ideal precision is 1 at all recall levels. The use of more data can help improve the average precision but might require more training time. Plot the PR curve.

```
figure
plot(recall{:},precision{:})
xlabel("Recall")
ylabel("Precision")
```

```
grid on
title(sprintf("Average Precision = %.2f", AP))
```



Supporting Functions

```
function data = augmentData(data)
% Randomly flip images and bounding boxes horizontally.
tform = randomAffine2d("XReflection",true);
sz = size(data{1});
rout = affineOutputView(sz,tform);
data{1} = imwarp(data{1},tform,"OutputView",rout);

% Sanitize boxes, if needed. This helper function is attached as a
% supporting file. Open the example in MATLAB to open this function.
data{2} = helperSanitizeBoxes(data{2});

% Warp boxes.
data{2} = bboxwarp(data{2},tform,rout);
end

function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));
```

```
% Sanitize boxes, if needed. This helper function is attached as a
% supporting file. Open the example in MATLAB to open this function.
data{2} = helperSanitizeBoxes(data{2});

% Resize boxes.
data{2} = bboxresize(data{2},scale);
end
```

References

[1] Ren, S., K. He, R. Gershick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *IEEE Transactions of Pattern Analysis and Machine Intelligence*. Vol. 39, Issue 6, June 2017, pp. 1137-1149.

[2] Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH, June 2014, pp. 580-587.

[3] Girshick, R. "Fast R-CNN." *Proceedings of the 2015 IEEE International Conference on Computer Vision*. Santiago, Chile, Dec. 2015, pp. 1440-1448.

[4] Zitnick, C. L., and P. Dollar. "Edge Boxes: Locating Object Proposals from Edges." *European Conference on Computer Vision*. Zurich, Switzerland, Sept. 2014, pp. 391-405.

[5] Uijlings, J. R. R., K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. "Selective Search for Object Recognition." *International Journal of Computer Vision*. Vol. 104, Number 2, Sept. 2013, pp. 154-171.

Version History

Introduced in R2017a

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps

Image Labeler | Video Labeler

Functions

`trainFasterRCNNObjectDetector` | `trainACFObjectDetector` | `selectStrongestBboxMulticlass`

Topics

“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”

“Datastores for Deep Learning” (Deep Learning Toolbox)

maskrcnn

Detect objects using Mask R-CNN instance segmentation

Description

The `maskrcnn` object performs instance segmentation of objects in an image using a Mask R-CNN (regions with convolution neural networks) object detector. To detect objects in an image, pass the trained detector to the `segmentObjects` function.

Note This function requires the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation. You can install the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

Creation

Syntax

```
detector = maskrcnn("resnet50-coco")
detector = maskrcnn("resnet50-coco",classNames)
detector = maskrcnn("resnet50-coco",classNames,anchorBoxes)
detector = maskrcnn( ___ ,Name=Value)
```

Description

`detector = maskrcnn("resnet50-coco")` loads a pretrained Mask R-CNN object detector trained on the COCO data set with a ResNet-50 network as the feature extractor.

`detector = maskrcnn("resnet50-coco",classNames)` creates a pretrained Mask R-CNN object detector and configures it to perform transfer learning using a specified set of object classes. The `classNames` argument sets the `ClassNames` property. For optimal results, train the detector on new training images before performing detection.

`detector = maskrcnn("resnet50-coco",classNames,anchorBoxes)` creates a pretrained Mask R-CNN object detector and configures it to perform transfer learning using a specified set of object classes and anchor boxes. The `classNames` argument sets the `ClassNames` property. The `anchorBoxes` argument sets the `AnchorBoxes` property.

`detector = maskrcnn( ___ ,Name=Value)` uses name-value arguments to specify ROI pooling sizes or to set the `ModelName` or `InputSize` properties. Specify name-value arguments in addition to the input arguments from any of the previous syntaxes.

For example, `maskrcnn("resnet50-coco",classNames,anchorBoxes,PoolSize=[11 11])` specifies the ROI pooling size for the detection head as 11-by-11 pixels.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `maskrcnn("resnet50-coco",classNames,anchorBoxes,PoolSize=[11 11])` specifies the ROI pooling size for the detection head as 11-by-11 pixels.

`PoolSize` — ROI pooling size for detection head
`[14 14]` (default) | 1-by-2 vector

ROI pooling size for the detection head, specified as a 1-by-2 vector in the format [*height width*].

`MaskPoolSize` — ROI pooling size for mask segmentation head
`[14 14]` (default) | 1-by-2 vector

ROI pooling size for the mask segmentation head, specified as a 1-by-2 vector in the format [*height width*].

Properties

`ModelName` — Name of trained Mask R-CNN object detection network
`"maskrcnn"` | string scalar | character vector

Name of the trained Mask R-CNN object detection network, specified as a string scalar or character vector.

`AnchorBoxes` — Size of anchor boxes
M-by-2 matrix

Size of anchor boxes, specified as an *M*-by-2 matrix, where each row is in the format [*height width*]. The default value consists of 15 anchor boxes defined by the MS-COCO data set. When you specify the anchor boxes, the `maskrcnn` object reinitializes the final convolution layers in the region proposal subnetwork to the correct size based on the number of anchor boxes.

You cannot modify the value of this property after you create the object.

`ClassNames` — Object class names
cell array

Names of the object classes that the Mask R-CNN detector is trained to detect, specified as a cell array. The default value consists of the 80 object class names in the MS-COCO data set, such as "person", "bicycle", and "car". When you specify the class names, the `maskrcnn` object reinitializes the final convolution layers in the detection head and mask segmentation head to the correct size based on the number of classes.

You cannot modify the value of this property after you create the object.

`InputSize` — Image size
`[800 1200 3]` | 1-by-3 vector

Image size to use for detection, specified as a 1-by-3 vector of positive integers in the format [*height width* 3]. The detector resizes input images to this size while maintaining the aspect ratio. The default value is the network input size.

You cannot modify the value of this property after you create the object.

Object Functions

forward	Run forward pass on Mask R-CNN network
segmentObjects	Segment objects using Mask R-CNN instance segmentation

Examples

Segment Instances of Objects

Load a pretrained Mask R-CNN object detector.

```
detector = maskrcnn("resnet50-coco")
```

```
detector =
  maskrcnn with properties:

      ModelName: 'maskrcnn'
     ClassNames: {1×80 cell}
      InputSize: [800 1200 3]
    AnchorBoxes: [15×2 double]
```

Read a test image that includes objects that the network can detect, such as people.

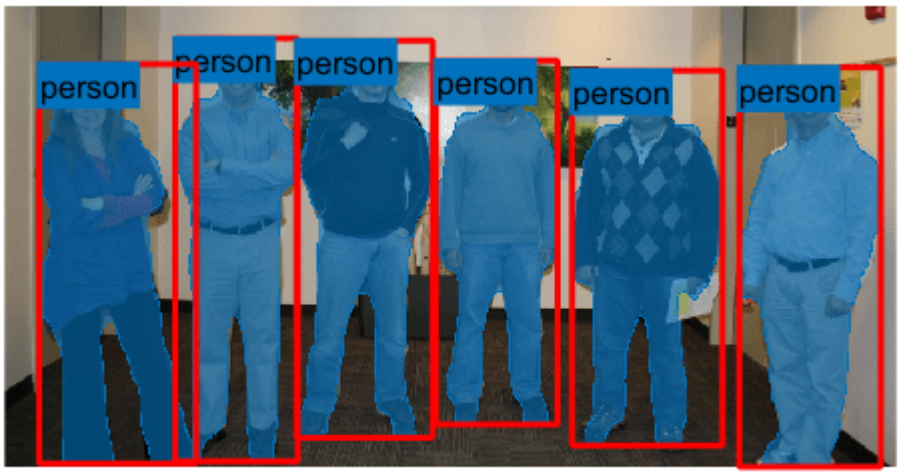
```
I = imread("visionteam.jpg");
```

Segment instances of objects using the Mask R-CNN object detector.

```
[masks,labels,scores,boxes] = segmentObjects(detector,I,Threshold=0.95);
```

Overlay the detected object masks in blue on the test image. Display the bounding boxes in red and the object labels.

```
overlayedImage = insertObjectMask(I,masks);
imshow(overlayedImage)
showShape("rectangle",boxes,Label=labels,LineColor=[1 0 0])
```



Version History

Introduced in R2021b

See Also

Apps

Image Labeler | Video Labeler

Functions

`trainMaskRCNN` | `evaluateInstanceSegmentation`

Topics

“Getting Started with Mask R-CNN for Instance Segmentation”

“Perform Instance Segmentation Using Mask R-CNN”

forward

Run forward pass on Mask R-CNN network

Syntax

```
[dlRPNScores,dlRPNReg,dlProposals,dlBoxScores,dlBoxReg,dlMasks] = forward(
detector,dlX)
[dlRPNScores,dlRPNReg,dlProposals,dlBoxScores,dlBoxReg,dlMasks,state] =
forward(detector,dlX)
```

Description

`[dlRPNScores,dlRPNReg,dlProposals,dlBoxScores,dlBoxReg,dlMasks] = forward( detector,dlX)` calculates features of the image `dlX` from the output layers of the Mask R-CNN object detector.

`[dlRPNScores,dlRPNReg,dlProposals,dlBoxScores,dlBoxReg,dlMasks,state] = forward(detector,dlX)` also returns the state information of the network. Use the state to update the network parameters.

Note This function requires the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation. You can install the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

Examples

Calculate Features of Color Image

Load a pretrained Mask R-CNN object detector.

```
detector = maskrcnn("resnet50-coco");
```

Read an image to use for training, and convert the image to a formatted `dlarray` object.

```
I = imread("visionteam.jpg");
dlX = dlarray(single(I),"SSCB");
```

Calculate features of the training image.

```
[dlRPNScores,dlRPNReg,dlProposals,dlBoxScores,dlBoxReg,dlMasks] = forward(detector,dlX);
```

Input Arguments

`detector` — Mask R-CNN object detector

`maskrcnn` object

Mask R-CNN object detector, specified as a `maskrcnn` object.

`dlX` — Training data
formatted `dlarray` object

Training data, specified as a formatted `dlarray` object containing real, nonsparse data. The dimension labels of the data must be `"SSCB"`.

Output Arguments

`dlRPNScores` — RPN classification output
`dlarray` object

RPN classification output after the softmax operation, returned as a `dlarray` object of size *h*-by-*w*-by-*numAnchors*-by-*B*. The feature map has spatial size *h*-by-*w*. *B* is the number of images in the batch.

`dlRPNReg` — RPN regression output
`dlarray` object

RPN regression output, returned as a `dlarray` object of size *h*-by-*w*-by-(4×*numAnchors*)-by-*B*. The feature map has spatial size *h*-by-*w*. *B* is the number of images in the batch.

`dlProposals` — Region proposals
`dlarray` object

Region proposals, returned as a `dlarray` object of size 5-by-*numProposals*-by-1-by-1. Each column of the proposals contains a box proposal in the format [*xStart, yStart, xEnd, yEnd, batchIdx*].

`dlBoxScores` — Detection network classification output
`dlarray` object

Detection network classification output after the softmax operation, returned as a `dlarray` object of size 1-by-1-by-(*numClasses*+1)-by-*numProposals*.

`dlBoxReg` — Detection network regression output
`dlarray` object

Detection network regression output, returned as a `dlarray` object of size 1-by-1-by-(4×*numClasses*)-by-*numProposals*.

`dlMasks` — Mask segmentation output
`dlarray` object

Mask segmentation output, returned as a `dlarray` object of size h_{mask}-by-w_{mask}-by-*numClasses*-by-*numProposals*. The mask segmentation output has spatial size h_{mask}-by-w_{mask}.

`state` — Updated network state
table

Updated network state, returned as a table. The network state is a table with three columns:

- `Layer` – Layer name, returned as a string scalar.
- `Parameter` – Parameter name, returned as a string scalar.

- `Value` – Value of parameter, returned as a numeric array object.

The network state contains information remembered by the network between iterations.

Version History

Introduced in R2021b

R2022a: Features returned in different format

Behavior changed in R2022a

The `forward` function now returns the feature maps for the six output layers in separate output arguments. Before, the function returned the feature maps as a 1-by-6 cell array in a single output argument.

See Also

`maskrcnn` | `segmentObjects`

segmentObjects

Segment objects using Mask R-CNN instance segmentation

Syntax

```
masks = segmentObjects(detector,I)
[masks,labels] = segmentObjects(detector,I)
[masks,labels,scores] = segmentObjects(detector,I)
[masks,labels,scores,bboxes] = segmentObjects(detector,I)

dsResults = segmentObjects(detector,imds)

[___] = segmentObjects(___,Name=Value)
```

Description

`masks = segmentObjects(detector,I)` detects object masks within a single image or an array of images, `I`, using a Mask R-CNN object detector.

`[masks,labels] = segmentObjects(detector,I)` also returns the labels assigned to the detected objects.

`[masks,labels,scores] = segmentObjects(detector,I)` also returns the detection score for each of the detected objects.

`[masks,labels,scores,bboxes] = segmentObjects(detector,I)` also returns the location of segmented object as bounding boxes, `bboxes`.

`dsResults = segmentObjects(detector,imds)` performs instance segmentation of images in a datastore using a Mask R-CNN object detector. The function returns a datastore with the instance segmentation results, including the instance masks, labels, detection scores, and bounding boxes.

`[___] = segmentObjects(___,Name=Value)` configures the segmentation using additional name-value arguments. For example, `segmentObjects(detector,I,Threshold=0.9)` specifies the detection threshold as `0.9`.

Note This function requires the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation. You can install the Computer Vision Toolbox Model for Mask R-CNN Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

Examples

Segment Instances of Objects

Load a pretrained Mask R-CNN object detector.

```
detector = maskrcnn("resnet50-coco")
```

```
detector =
  maskrcnn with properties:

      ModelName: 'maskrcnn'
     ClassNames: {1×80 cell}
      InputSize: [800 1200 3]
    AnchorBoxes: [15×2 double]
```

Read a test image that includes objects that the network can detect, such as people.

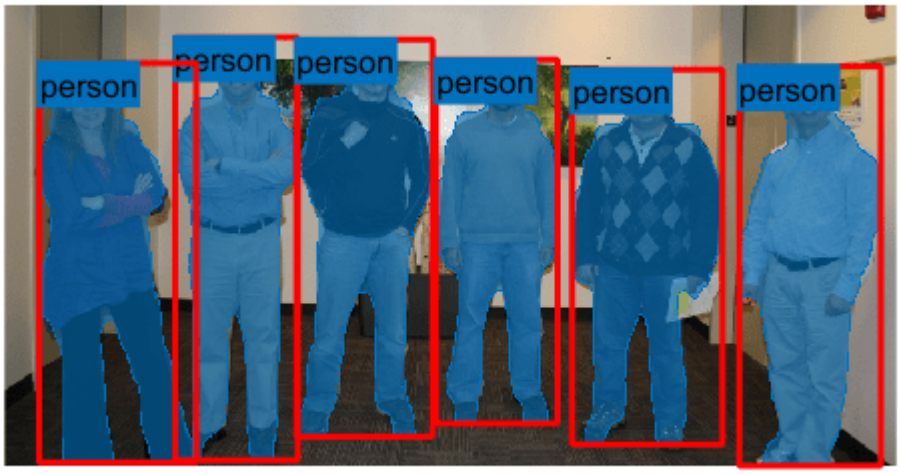
```
I = imread("visionteam.jpg");
```

Segment instances of objects using the Mask R-CNN object detector.

```
[masks,labels,scores,boxes] = segmentObjects(detector,I,Threshold=0.95);
```

Overlay the detected object masks in blue on the test image. Display the bounding boxes in red and the object labels.

```
overlayedImage = insertObjectMask(I,masks);
imshow(overlayedImage)
showShape("rectangle",boxes,Label=labels,LineColor=[1 0 0])
```



Segment Instances of Objects in Image Datastore

Load a pretrained Mask R-CNN object detector.

```
detector = maskrcnn("resnet50-coco");
```

Create a datastore of test images.

```
imageFiles = fullfile(toolboxdir("vision"),"visiondata","visionteam*.jpg");
dsTest = imageDatastore(imageFiles);
```

Segment instances of objects using the Mask R-CNN object detector.

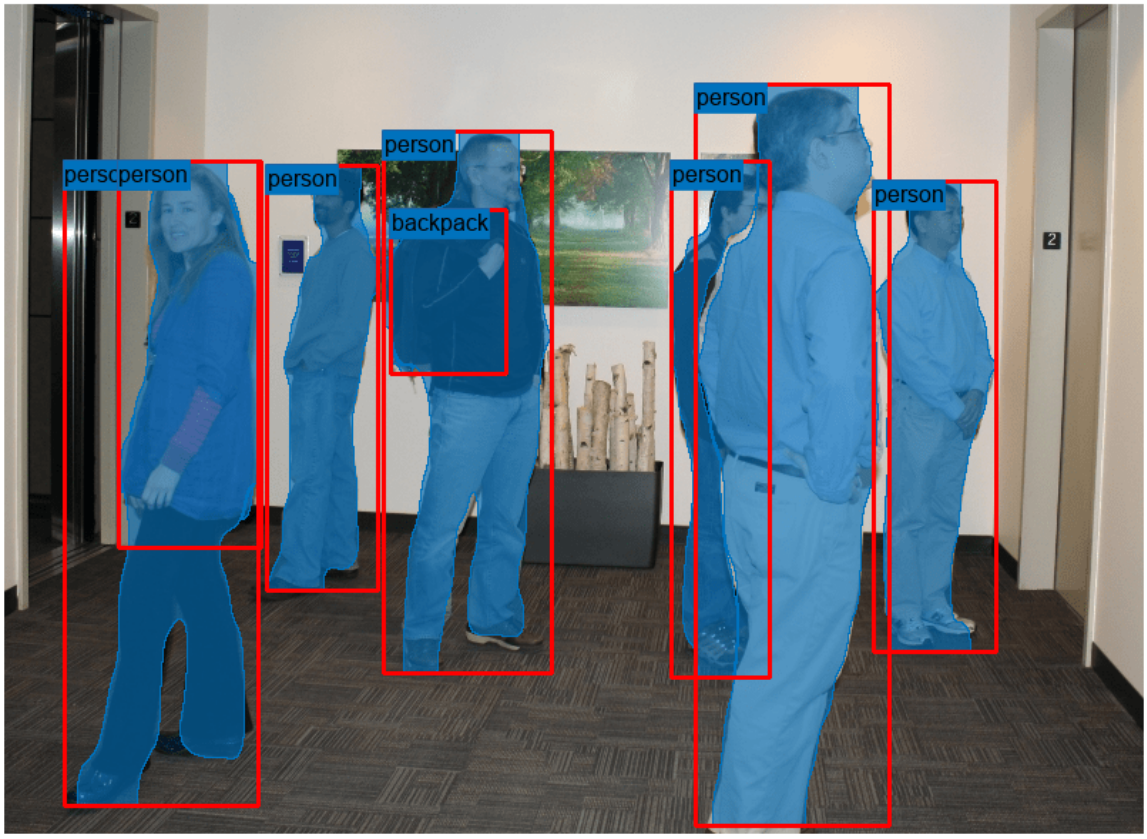
```
dsResults = segmentObjects(detector,dsTest);
```

```
Running Mask R-CNN network
-------------------------
* Processed 2 images.
```

For each test image, display the instance segmentation results. Overlay the detected object masks in blue on the test image. Display the bounding boxes in red and the object labels.

```
while(hasdata(dsResults))
    testImage = read(dsTest);
    results = read(dsResults);
    overlayedImage = insertObjectMask(testImage,results{1});
    figure
    imshow(overlayedImage)
    showShape("rectangle",results{4},Label=results{2},LineColor=[1 0 0])
end
```





Input Arguments

`detector` — Mask R-CNN object detector

`maskrcnn` object

Mask R-CNN object detector, specified as a `maskrcnn` object.

`I` — Image or batch of images

numeric matrix | numeric array

Image or batch of images to segment, specified as one of these values.

Image Type	Data Format
Single grayscale image	2-D matrix of size *H*-by-*W*
Single color image	3-D array of size *H*-by-*W*-by-3.
Batch of *B* grayscale or color images	4-D array of size *H*-by-*W*-by-*C*-by-*B*. The number of color channels *C* is 1 for grayscale images and 3 for color images.

The height *H* and width *W* of each image must be greater than or equal to the input height *h* and width *w* of the network.

`imds` — Datastore of images
datastore

Datastore of images, specified as a datastore such as an `imageDatastore` or a `CombinedDatastore`. If calling the datastore with the `read` function returns a cell array, then the image data must be in the first cell.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `segmentObjects(detector,I,Threshold=0.9)` specifies the detection threshold as `0.9`.

Options for All Image Formats

`Threshold` — Detection threshold
`0.7` (default) | numeric scalar

Detection threshold, specified as a numeric scalar in the range [0, 1]. The Mask R-CNN object detector does not return detections with scores less than the threshold value. Increase this value to reduce false positives.

`NumStrongestRegions` — Maximum number of strongest region proposals
`1000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals, specified as a positive integer. Reduce this value to speed up processing time at the cost of detection accuracy. To use all region proposals, specify this value as `Inf`.

`SelectStrongest` — Select strongest bounding box
`true` or `1` (default) | `false` or `0`

Select the strongest bounding box for each detected object, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Return the strongest bounding box per object. To select these boxes, the `segmentObjects` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.
- `false` — Return all detected bounding boxes. You can then create your own custom operation to eliminate overlapping bounding boxes.

`MinSize` — Minimum size of region
`[1 1]` (default) | two-element numeric vector

Minimum size of a region containing an object, in pixels, specified as a two-element numeric vector of the form [*height width*]. By default, `MinSize` is the smallest object that the trained detector can detect. Specify this argument to reduce the computation time.

`MaxSize` — Maximum size of region
two-element numeric vector

Maximum size of a region containing an object, in pixels, specified as a two-element numeric vector of the form [*height width*].

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `MaxSize` is set to the height and width of the input image, `I`.

`ExecutionEnvironment` — Hardware resource

`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource for processing images with a network, specified as `"auto"`, `"gpu"`, or `"cpu"`.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message.
`"cpu"`	Use the CPU.

Options for Datastore Inputs

`MiniBatchSize` — Number of observations in each batch

positive integer

Number of observations that are returned in each batch. The default value is equal to the `ReadSize` property of datastore `imds`.

You can specify this argument only when you specify a batch of images, `I`, or datastore of images, `imds`, as an input to the `segmentObjects` function.

`WriteLocation` — Location to place writable data

string scalar | character vector

Location to place writable data, specified as a string scalar or character vector. The specified folder must have write permissions. If the folder already exists, the `segmentObjects` function creates a new folder and adds a suffix to the folder name with the next available number. The default write location is `fullfile(pwd,"SegmentObjectResults")` where `pwd` is the current working directory.

You can specify this argument only when you specify a datastore of images, `imds`.

Data Types: `char` | `string`

`NamePrefix` — Prefix added to written filenames

`"segmentObj"` (default) | string scalar | character vector

Prefix added to written filenames, specified as a string scalar or character vector. The files are named *`NamePrefix_imageName.mat`*, where *`imageName`* is the name of the input image without its extension.

If `imds` contains identical file names in different folders, the function names the output files *`NamePrefix_folder_imageName.mat`*, where *`folder`* is the folder containing the files with the same name.

If `imds` contains identical file names in folders of the same name within different directory hierarchies, the function names the output files *`NamePrefix_folder_imageName_#.mat`*, where `#` is a distinct integer.

You can specify this argument only when you specify a datastore of images, `imds`.

Data Types: `char` | `string`

`Verbose` — Enable progress display to screen
`true` or `1` (default) | `false` or `0`

Enable progress display to screen, specified as a numeric or logical `1` (`true`) or `0` (`false`).

You can specify this argument only when you specify a datastore of images, `imds`.

Output Arguments

`masks` — Object masks
H-by-*W*-by-*M* logical array | *B*-by-1 cell array

Objects masks, returned as a logical array of size *H*-by-*W*-by-*M*. *H* and *W* are the height and width of the input image `I`. *M* is the number of objects detected in the image. Each of the *M* channels contains the mask for a single detected object.

When `I` represents a batch of *B* images, `masks` is returned as a *B*-by-1 cell array. Each element in the cell array indicates the masks for the corresponding input image in the batch.

`labels` — Object labels
M-by-1 categorical vector | *B*-by-1 cell array

Objects labels, returned as an *M*-by-1 categorical vector where *M* is the number of detected objects in image `I`.

When `I` represents a batch of *B* images, then `labels` is a *B*-by-1 cell array. Each element is an *M*-by-1 categorical vector with the labels of the objects in the corresponding image.

`scores` — Detection scores
M-by-1 vector | *B*-by-1 cell array

Detection confidence scores, returned as an *M*-by-1 numeric vector, where *M* is the number of detected objects in image `I`. A higher score indicates higher confidence in the detection.

When `I` represents a batch of *B* images, then `scores` is a *B*-by-1 cell array. Each element is an *M*-by-1 numeric vector with the labels of the objects in the corresponding image.

`bboxes` — Location of detected objects
M-by-4 matrix | *B*-by-1 cell array

Location of detected objects within the input image, returned as an *M*-by-4 matrix, where *M* is the number of detected objects in image `I`. Each row of `bboxes` contains a four-element vector of the

form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

When `I` represents a batch of *B* images, then `bboxes` is a *B*-by-1 cell array. Each element is an *M*-by-4 numeric matrix with the bounding boxes of the objects in the corresponding image.

`dsResults` — Predicted instance segmentation
`FileDatastore` object

Predicted instance segmentation results, returned as a `FileDatastore` object. The datastore is set up so that calling the datastore with the `read` and `readall` functions returns a cell array with four columns. This table describes the format of each column.

data	***boxes***	***labels***	***masks***
RGB image that serves as a network input, returned as an *H*-by-*W*-by-3 numeric array.	Bounding boxes, returned as *M*-by-4 matrices, where *M* is the number of objects within the image. Each bounding box has the format [*x y width height*], where [*x*, *y*] represent the top-left coordinates of the bounding box.	Object class names, returned as an *M*-by-1 categorical vector. All categorical data returned by the datastore must contain the same categories.	Binary masks, returned as a logical array of size *H*-by-*W*-by-*M*. Each mask is the segmentation of one instance in the image.

Version History

Introduced in R2021b

R2022b: Supports datastores

The `segmentObjects` function now supports specifying test images as a datastore. New name-value arguments enable more options to configure the segmentation of images in a datastore. When you specify test images as a datastore, the function returns all instance segmentation results as a file datastore.

See Also

`forward` | `evaluateInstanceSegmentation`

Topics
“Getting Started with Mask R-CNN for Instance Segmentation”
“Perform Instance Segmentation Using Mask R-CNN”

posemaskrcnn

Predict object pose using Pose Mask R-CNN pose estimation

Description

The `posemaskrcnn` object performs pose estimation of objects in an image using a pretrained Pose Mask R-CNN network, a region-based convolutional neural network designed for six degrees-of-freedom (6-DoF) pose estimation.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Pose Mask R-CNN 6-DoF Object Pose Estimation. You can install the Computer Vision Toolbox Model for Pose Mask R-CNN 6-DoF Object Pose Estimation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
net = posemaskrcnn(pretrainedNet)
net = posemaskrcnn(pretrainedNet,classNames)
net = posemaskrcnn(pretrainedNet,classNames,anchorBoxes)
net = posemaskrcnn( ___ ,Name=Value)
```

Description

`net = posemaskrcnn(pretrainedNet)` creates a pretrained Pose Mask R-CNN pose estimation network `net` by using the specified pretrained Pose Mask R-CNN deep learning network.

`net = posemaskrcnn(pretrainedNet,classNames)` creates a pretrained Pose Mask R-CNN pose estimation network, and configures it to perform transfer learning using a set of object classes specified by the `classNames` argument. For optimal results, train the network on new training data before performing pose estimation.

`net = posemaskrcnn(pretrainedNet,classNames,anchorBoxes)` creates a pretrained Pose Mask R-CNN network, and configures it to perform transfer learning using a set of object classes and anchor boxes specified by the `classNames` and `anchorBoxes` arguments, respectively. For optimal results, train the network on new training data before performing pose estimation.

`net = posemaskrcnn( ___ ,Name=Value)` specifies options using name-value arguments to specify ROI pooling sizes in addition to any combination of input arguments from previous syntaxes. You can also use name-value arguments to set the `ModelName` and `ImageInputSize` properties.

For example, `PoolSize=[11 11]` specifies the ROI pooling size for the detection head as 11-by-11 pixels.

Input Arguments

`pretrainedNet` — Pretrained Pose Mask R-CNN deep learning network

`"resnet50-pvc-parts"` | `"resnet50-coco"`

Pretrained Pose Mask R-CNN deep learning network, specified as one of these:

- `"resnet50-pvc-parts"` — A pretrained Pose Mask R-CNN deep learning network trained on images of PVC pipe connectors in various orientations. You can use the resulting pose estimation network to perform pose estimation on a custom bin-picking data set by using the `predictPose` object function.
- `"resnet50-coco"` — A pretrained Pose Mask R-CNN deep learning network which uses weights from a Mask R-CNN deep learning network with ResNet-50 as a backbone, trained on the COCO data set for instance segmentation. To train a pose estimation network created using this pretrained deep learning network, you must use the `trainPoseMaskRCNN` function in two stages: first with a `trainingMode` argument value of `"mask"`, and then with a `trainingMode` value of `"pose-and-mask"`.

Data Types: `char` | `string`

`classNames` — Names of object classes

vector of strings | cell array of character vectors | categorical vector

Names of the object classes that the Pose Mask R-CNN network is trained to estimate pose for, specified as a vector of strings, cell array of character vectors, or categorical vector. This argument sets the `ClassNames` property of the `posemaskrcnn` object.

Data Types: `string` | `cell` | `categorical`

`anchorBoxes` — Sizes of anchor boxes

M-by-2 matrix

Sizes of the anchor boxes, specified as an *M*-by-2 matrix, where each row is of the form [*height width*], in pixels. *M* is the number of anchor boxes. You can estimate the anchor boxes using the `estimateAnchorBoxes` function. This argument sets the `AnchorBoxes` property of the `posemaskrcnn` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `posemaskrcnn("resnet50-pvc-parts",classNames,anchorBoxes,PoolSize=[11 11])` specifies the ROI pooling size for the detection head as 11-by-11 pixels.

`PoolSize` — ROI pooling size for detection head

`[14 14]` (default) | 2-element row vector

ROI pooling size for the detection head, specified as a 2-element row vector of the form [*height width*]. Adjust the `PoolSize` value when your data set contains extreme variation in the scale of objects or images.

`MaskPoolSize` — ROI pooling size for mask segmentation head

`[14 14]` (default) | 2-element row vector

ROI pooling size for the mask segmentation head, specified as a 2-element row vector of the form [*height width*]. Adjust the `MaskPoolSize` value when your data contains extreme variation in the scale in the objects or images.

Properties

`ModelName` — Name of pretrained Pose Mask R-CNN pose estimation network

`"posemaskrcnn"` (default) | string scalar | character vector

This property is read-only.

Name of the pretrained Pose Mask R-CNN pose estimation network, specified as a string scalar or character vector.

To set the value of this property, you must specify it at object creation.

`AnchorBoxes` — Sizes of anchor boxes

M-by-2 matrix

This property is read-only.

Sizes of the anchor boxes, specified as an *M*-by-2 matrix where each row is of the form [*height width*], in pixels. The default value of this property depends on the specified value of `pretrainedNet`:

- `"resnet50-pvc-parts"` — An 8-by-2 matrix of anchor box values.
- `"resnet50-coco"` — A 15-by-2 matrix of anchor box values.

When you specify anchor boxes, the `posemaskrcnn` object reinitializes the final convolution layers in the region proposal subnetwork to the correct size based on the number of anchor boxes *M*.

The `anchorBoxes` argument sets this property.

`ClassNames` — Object class names

vector of strings | cell array of character vectors | categorical vector

This property is read-only.

Names of the object classes that the Pose Mask R-CNN network is trained to estimate pose for, specified as a vector of strings, cell array of character vectors, or categorical vector. The default value of this property depends on the specified value of `pretrainedNet`:

- `"resnet50-pvc-parts"` — Four object class names that correspond to the four object orientations in the PVC pipe data set.
- `"resnet50-coco"` — Eighty object class names that correspond to various objects detected in the COCO data set.

When you specify class names, the `posemaskrcnn` object reinitializes the final convolution layers in the detection head and mask segmentation head to the correct size based on the number of classes.

The `classNames` argument sets this property.

`ImageInputSize` — Image size

`[720 1280 3]` (default) | 3-element row vector

This property is read-only.

Image size to use for pose estimation, specified as a 1-by-3 vector of positive integers of the form [*height width channels*]. The values of *height* and *width* specify the image dimensions, in pixels, and *channels* specifies the number of color channels. The network resizes input images to this size while maintaining the aspect ratio. The default value is the network input size.

To set the value of this property, you must specify it at object creation.

`DepthImageInputSize` — Depth image size
`[720 1280 1]` (default) | 3-element row vector

This property is read-only.

Depth image size used for pose estimation, specified as a 3-element row vector of positive integers of the form [*height width* `1`].

Object Functions

predictPose Estimate object pose using Pose Mask R-CNN deep learning network

Examples

Create Pretrained Pose Mask R-CNN Network

Create a pretrained Pose Mask R-CNN pose estimation network that was trained on images of PVC pipe connectors in various orientations.

```
net = posemaskrcnn("resnet50-pvc-parts")
```

```
net =
  posemaskrcnn with properties:

              ModelName: 'posemaskrcnn'
             ClassNames: {4×1 cell}
         ImageInputSize: [720 1280 3]
    DepthImageInputSize: [720 1280 1]
            AnchorBoxes: [8×2 double]
```

Display the names of the object classes.

```
net.ClassNames
```

```
ans = 4×1 cell array
    "'I_shape'"
    "'X_shape'"
    "'L_shape'"
    "'T_shape'"
```

Version History

Introduced in R2024a

References

[1] Xiang, Yu, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes.” In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, 2018. doi:10.15607/RSS.2018.XIV.019.

[2] Jiang, Xiaoke, Donghai Li, Hao Chen, Ye Zheng, Rui Zhao, and Liwei Wu. “Uni6D: A Unified CNN Framework without Projection Breakdown for 6D Pose Estimation.” In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11164–74. New Orleans, LA, USA: IEEE, 2022. doi:10.1109/CVPR52688.2022.01089.

See Also

Functions

`trainPoseMaskRCNN` | `maskrcnn`

Topics

“Perform 6-DoF Pose Estimation for Bin Picking Using Deep Learning”
“Perform Instance Segmentation Using SOLOv2”
“Perform Instance Segmentation Using Mask R-CNN”

predictPose

Estimate object pose using Pose Mask R-CNN deep learning network

Syntax

```
poses = predictPose(net,I,depthImage,intrinsics)
[poses,labels,scores,bboxes] = predictPose( ___ )
[poses,labels,scores,bboxes,masks] = predictPose( ___ )

[ ___ ] = predictPose( ___ ,Name=Value)
```

Description

`poses = predictPose(net,I,depthImage,intrinsics)` returns 6-degrees-of-freedom (6-DoF) pose of objects within a single image or a batch of images `I` using a trained Pose Mask R-CNN network.

`[poses,labels,scores,bboxes] = predictPose( ___ )` also returns the labels assigned to the detected objects, the detection score for each detected object, and the bounding box location of each detected object, using the input arguments from the previous syntax.

`[poses,labels,scores,bboxes,masks] = predictPose( ___ )` performs instance segmentation of the objects, and returns the binary object masks, `masks`.

`[ ___ ] = predictPose( ___ ,Name=Value)` specifies options using additional name-value arguments. For example, `Threshold=0.7` specifies the detection threshold as `0.7`.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Pose Mask R-CNN 6-DoF Object Pose Estimation. You can install the Computer Vision Toolbox Model for Pose Mask R-CNN 6-DoF Object Pose Estimation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Input Arguments

`net` — Pose Mask R-CNN pose estimation network

`posemaskrcnn` object

Pose Mask R-CNN pose estimation network, specified as a `posemaskrcnn` object.

`I` — Image or batch of images

numeric matrix | numeric array

Image to segment, specified as a numeric matrix or numeric array. The size of this argument depends on the number of images specified and whether they are color or grayscale.

Image Type and Number	Data Format
Single grayscale image	2-D matrix of size *H*-by-*W*

Image Type and Number	Data Format
Single color image	3-D array of size *H*-by-*W*-by-3.
Batch of *B* grayscale or color images	4-D array of size *H*-by-*W*-by-*C*-by-*B*. The number of color channels *C* is 1 for grayscale images and 3 for color images.

The height *H* and width *W* of each image must be greater than or equal to the input height *h* and width *w* of the network.

Tip For best network performance, use input image data of the same size that the network has been trained on.

`depthImage` — Depth map for estimating 3-D pose

numeric matrix | numeric array

Depth map for estimating the 3-D pose, specified as a numeric matrix or array. The size of this argument depends on the number of depth maps specified.

Number of Depth Maps	Data Format
Single depth map	2-D numeric matrix of size *H*-by-*W*.
Batch of *B* depth maps	4-D array of size *H*-by-*W*-by-1-by-*B*.

H and *W* must be equal to the corresponding values of `I`, and the number of depth maps must match the number of images specified.

`intrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object | *B*-by-1 cell array of `cameraIntrinsics` objects

Camera intrinsic parameters, specified as a `cameraIntrinsics` object or a *B*-by-1 cell array of `cameraIntrinsics` objects. If you specify this value as a scalar, the function applies the same camera intrinsic parameters to every input image. If you specify this value as a cell array, the number of `cameraIntrinsics` objects must match the number of input images *B*.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `predictPose(net,I,depthImage,intrinsics,Threshold=0.7)` specifies the detection threshold as `0.7`.

Options for All Image Formats

`Threshold` — Object detection threshold

`0.5` (default) | numeric scalar

Object detection threshold, specified as a numeric scalar in the range [0, 1]. The Pose Mask R-CNN network does not use object detections with scores less than the threshold value for pose estimation. Increase this value to reduce false positives.

`NumStrongestRegions` — Maximum number of strongest region proposals
`1000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals, specified as a positive integer or `Inf`. Reduce this value to increase processing speed at the cost of detection accuracy. To use all region proposals, specify this value as `Inf`.

`SelectStrongestMulticlassThreshold` — Strongest bounding box threshold per class
`0.5` (default) | numeric scalar

Strongest bounding box threshold per class, specified as a positive numeric scalar in the range [0, 1]. The strongest bounding boxes per class are returned when their confidence scores are higher than this value. To select these boxes, `predictPose` uses the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes with the same class label based on their confidence scores.

Note `predictPose` first performs nonmaximal suppression using the `selectStrongestBboxMulticlass` function and the threshold value specified by `SelectStrongestMulticlassThreshold`. In the second step, `predictPose` performs nonmaximal suppression using the `selectStrongestBbox` function and the threshold value specified by `SelectStrongestThreshold`. You can manually determine the optimal threshold values for your application by using a validation data subset when training on a custom data set.

`SelectStrongestThreshold` — Strongest bounding box threshold per object
`0.3` (default) | numeric scalar

Strongest bounding box threshold per object, specified as a positive numeric scalar in the range [0, 1]. The strongest bounding boxes per object are returned when their confidence scores are higher than this value. To select these boxes, `predictPose` uses the `selectStrongestBbox` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores across all classes.

Note `predictPose` first performs nonmaximal suppression using the `selectStrongestBboxMulticlass` function and the threshold value specified by `SelectStrongestMulticlassThreshold`. In the second step, `predictPose` performs nonmaximal suppression using the `selectStrongestBbox` function and the threshold value specified by `SelectStrongestThreshold`. You can manually determine the optimal threshold values for your application by using a validation data subset when training on a custom data set.

`MinSize` — Minimum size of object-containing region
`[1 1]` (default) | two-element numeric vector

Minimum size of an object-containing region, in pixels, specified as a two-element numeric vector of the form [*height width*]. `MinSize` is the size of the smallest object that the trained detector can detect. Specify larger values for this argument to reduce computation time.

`MaxSize` — Maximum size of object-containing region
two-element numeric vector

Maximum size of an of object-containing region, in pixels, specified as a two-element numeric vector of the form [*height width*].

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `MaxSize` uses the height and width of the input image `I`.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource for processing images with the network, specified as `"auto"`, `"gpu"`, or `"cpu"`.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message.
`"cpu"`	Use the CPU.

Options for Batch Inputs

`MiniBatchSize` — Number of observations in each batch
1 (default) | positive integer

Number of observations returned in each batch, specified as a positive integer.

Output Arguments

`poses` — Object poses
M-by-1 vector of `rigidtform3d` objects | *B*-by-1 cell array

Object poses, returned as an *M*-by-1 vector of `rigidtform3d` objects or a *B*-by-1 cell array. The value of this output depends on the number of input images *B*.

Image Type	`poses` Value
Single image	*M*-by-1 vector of `rigidtform3d` objects. *M* is the number of objects detected in the image.
Batch of *B* images	*B*-by-1 cell array, where each cell contains an *M*-by-1 vector of `rigidtform3d` objects.

`labels` — Object labels
M-by-1 categorical vector | *B*-by-1 cell array

Object labels, returned as an *M*-by-1 categorical vector or a *B*-by-1 cell array. When `I` is a single image, `labels` is an *M*-by-1 categorical vector, where *M* is the number of detected objects in the image.

When `I` is a batch of *B* images, `labels` is a *B*-by-1 cell array in which each cell contains an *M*-by-1 categorical vector with the labels of the objects detected in the corresponding image.

`scores` — Detection confidence scores
M-by-1 numeric vector | *B*-by-1 cell array

Detection confidence scores, returned as an *M*-by-1 numeric vector, or a *B*-by-1 cell array. When `I` is a single image, `scores` is an *M*-by-1 numeric vector, where *M* is the number of detected objects in the image.

When `I` is a batch of *B* images, then `scores` is a *B*-by-1 cell array. Each element is an *M*-by-1 numeric vector with the scores of the objects detected in the corresponding image.

The score for each object is in the range [0, 1]. A higher score indicates higher confidence in the detection.

`bboxes` — Locations of detected objects
M-by-4 numeric matrix | *B*-by-1 cell array

Locations of detected objects within the input image, returned as an *M*-by-4 numeric matrix, or a *B*-by-1 cell array. When `I` is a single image, `bboxes` is an *M*-by-1 numeric matrix, where *M* is the number of detected objects in the image. Each row of the matrix is of the form [*x y width height*], where *x* and *y* specify the upper-left corner of the corresponding bounding box, and *width* and *height* specify its size in pixels.

When `I` is a batch of *B* images, `bboxes` is a *B*-by-1 cell array in which each cell contains an *M*-by-4 numeric matrix with the bounding boxes of the objects detected in the corresponding image.

`masks` — Object masks
H-by-*W*-by-*M* logical array | *B*-by-1 cell array

Object masks, returned as a *H*-by-*W*-by-*M* logical array or a *B*-by-1 cell array. When `I` is a single image, `masks` is an *H*-by-*W*-by-*M* logical array, where *H* and *W* are the height and width of the input image, respectively, and *M* is the number of detected objects in the image.

When `I` is a batch of *B* images, `masks` is a *B*-by-1 cell array in which each cell contains an *H*-by-*W*-by-*M* logical array with the masks for the corresponding image.

Tips

- To refine estimated object poses predicted by the network, use point cloud registration techniques to postprocess your results, especially for objects of a symmetric shape. To learn more about refining pose estimation results, see the “Perform 6-DoF Pose Estimation for Bin Picking Using Deep Learning” example.

Version History

Introduced in R2024a

See Also

`posemaskrcnn` | `trainPoseMaskRCNN`

Topics
“Perform 6-DoF Pose Estimation for Bin Picking Using Deep Learning”

reidentificationNetwork

Re-identification deep learning network for re-identifying and tracking objects

Description

The `reidentificationNetwork` object configures a re-identification (ReID) deep learning network to extract features and train with a new set of classes. You can use the configured ReID network to re-identify objects in a sequence of image frames.

Note This functionality requires Deep Learning Toolbox.

Creation

Syntax

```
reID = reidentificationNetwork(backbone,classNames)
reID = reidentificationNetwork( ___ ,Name=Value)
```

Description

`reID = reidentificationNetwork(backbone,classNames)` creates a ReID network and configures it to perform transfer learning using a set of object classes specified by the `classNames` argument. For optimal performance, train the network on new training data using the `trainReidentificationNetwork` function.

`reID = reidentificationNetwork( ___ ,Name=Value)` specifies options using name-value arguments in addition to the input arguments from the previous syntax. You can also use name-value arguments to set the `ModelName`, `InputSize`, `FeatureLength`, and `FeatureLayer` properties of the network.

For example, `InputSize=[460 460 3]` sets the size of image used for classification and re-identification.

Input Arguments

backbone — Backbone feature extraction network
`dlnetwork` object

Backbone feature extraction network with an image input layer, specified as a `dlnetwork` object. The network must have a layer that produces an array with only one non-singleton dimension. If you do not specify the `FeatureLayer` property `reidentificationNetwork` appends the classification head network to the last layer in the network that meets the singleton criteria.

classNames — Names of object classes
vector of strings | cell array of character vectors | categorical vector

Names of the object classes that the ReID network is trained to re-identify, specified as a vector of strings, cell array of character vectors, or categorical vector. This argument sets the `ClassNames` property of the `reidentificationNetwork` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `reidentificationNetwork(net,classNames,InputSize=[460 460 3])` sets the size of image used for classification and re-identification.

`ClassificationNetwork` — Classification layers

`dlnetwork` object

Classification layers in the final layers of the ReID network, specified as a `dlnetwork` object. The classification layers contain these components by default: a batch normalization layer, a ReLU layer, a dropout layer, and a custom layer that contains a fully connected layer corresponding to the total number of classes and a softmax layer. The specified network must contain a fully connected layer and be able to process features output by the feature extraction layer `FeatureLayer`. The ReID network uses the output of the `ClassificationNetwork` layers for classification training.

`NormalizationStatistics` — Normalization statistics

1-by-1 structure (default) | structure

Normalization statistics, specified as a structure with mean and standard deviation fields. The structure must contain the fields `Mean` and `StandardDeviation`, which specify the means and standard deviations per channel, respectively, as 1-by-*C* vectors. The input image layer `imageInputLayer` of the backbone network `network` specifies which normalization method to use.

When `InputSize` is different than the `imageInputLayer` of the backbone network, `NormalizationStatistics` is a structure with these fields, which contain values from the COCO data set:

Field	Description	Value
`Mean`	1-by-*C* vector of means per channel.	`[123.6750 116.2800 103.5300]`
`StandardDeviation`	1-by-*C* vector of standard deviations per channel.	`[58.3950 57.1200 57.3750]`

The number of channels *C* must match the number of channels in the image specified by `InputSize`.

Properties

`ModelName` — Name of ReID network

`"ReidentificationNetwork"` (default) | string scalar | character vector

This property is read-only.

Name of the ReID network, specified as a string scalar or character vector.

To set the value of this property, you must specify it at object creation.

`Network` — Configured ReID network

`dlnetwork` object

This property is read-only.

Configured ReID network, stored as a `dlnetwork` object.

`InputSize` — Input image size

1-by-3 vector

This property is read-only.

Input image size to use for classification and re-identification, specified as a 1-by-3 vector of positive integers [*height width channels*]. The values of *height* and *width* specify the image dimensions, in pixels, and *channels* specifies the number of color channels. The network resizes input images to this size. The default value is the network input size.

To set the value of this property, you must specify it at object creation.

`ClassNames` — Object class names

vector of strings | cell array of character vectors | categorical vector

This property is read-only.

Names of the object classes that the ReID network is trained to re-identify, specified as a cell array, categorical vector, or cell array of character vectors.

`FeatureLayer` — Feature extraction layer

character vector | string scalar

This property is read-only.

Feature extraction layer of the backbone ReID network, specified as a character vector or a string scalar. This layer returns the appearance feature vectors when you use the `extractReidentificationFeatures` function. The default feature extraction layer is a fully connected layer named `"fc_num_features"` with an output size specified by the `FeatureLength` property.

The output of the feature extraction layer must be a valid input to the specified classification layers `ClassificationNetwork`, and consist of an array with only one non-singleton dimension. If you specify a `FeatureLayer` value, `reidentificationNetwork` ignores the `FeatureLength` value.

To set the value of this property, you must specify it at object creation.

`FeatureLength` — Feature vector size

`256` (default) | positive integer

This property is read-only.

Feature vector size for ReID inference, specified as a positive integer. When you call the `extractReidentificationFeatures` object function, the output is an *M*-by-*B* matrix, where *M* is equal to `FeatureLength` and *B* is the batch size.

Object Functions

extractReidentificationFeatures Extract object re-identification (ReID) features from image

Version History

Introduced in R2024a

See Also

Apps

Video Labeler

Functions

`trainReidentificationNetwork` | `imagePretrainedNetwork` | `evaluateReidentificationNetwork`

Topics

“Reidentify People Throughout a Video Sequence Using ReID Network”
“Automate Ground Truth Labeling for Object Tracking and Re-Identification”
“Convert Ground Truth Labeling Data for Object Re-Identification”

extractReidentificationFeatures

Extract object re-identification (ReID) features from image

Syntax

```
features = extractReidentificationFeatures(reID,I)
[features,labels] = extractReidentificationFeatures(reID,ds)

[___] = extractReidentificationFeatures(___,Name=Value)
```

Description

`features = extractReidentificationFeatures(reID,I)` extracts object re-identification features from an image or a batch of images `I`.

Note This functionality requires Deep Learning Toolbox.

`[features,labels] = extractReidentificationFeatures(reID,ds)` extracts features and their respective labels from the images returned by the `read` function of the input datastore `ds`. Specify this syntax to obtain all the feature vectors from a datastore, or to evaluate ReID network performance using the `evaluateReidentificationNetwork` function.

`[___] = extractReidentificationFeatures(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MiniBatchSize=4` specifies the minimum batch size to use for inference as `4`.

Input Arguments

`reID` — Re-identification network

`reidentificationNetwork` object

Re-identification network, specified as a `reidentificationNetwork` object.

`I` — Images

numeric array

Images, specified as a size *H*-by-*W*-by-*C* or *H*-by-*W*-by-*C*-by-*B* numeric array. You must specify real and nonsparse grayscale or RGB images.

- *H* — Height of the input images.
- *W* — Width of the input images.
- *C* — Number of channels. The channel size of each image must be equal to the input channel size of the network. For example, for grayscale images, *C* must be `1`. For RGB color images, it must be `3`.
- *B* — Number of test images in the batch. The `extractReidentificationFeatures` function extracts features for each test image in the batch.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore of RGB or grayscale images
`ImageDatastore` object | datastore

Datastore of RGB or grayscale images, specified as an `imageDatastore` object with a populated `Labels` property, or a datastore whose `read` function returns a 2-by-*B* cell array, where *B* is the number of images in the datastore. Each row of the cell array is of the form {*Image Class*}.

Image	Class
RGB image, stored as an *H*-by-*W*-by-3 numeric array, or grayscale image, stored as an *H*-by-*W* matrix.	String or a categorical cell vector which contains the object class name for the corresponding input image in *Image*. All categorical data returned by the datastore must have the same categories.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MiniBatchSize=4` specifies the minimum batch size to use for inference as 4.

`MiniBatchSize` — Minimum batch size
1 (default) | positive integer

Minimum batch size, specified as a positive integer. Adjusting the `MiniBatchSize` value can help you process a large collection of images. The `extractReidentificationFeatures` function groups images into minibatches of the specified size and processes them together, which can improve computation efficiency at the cost of increased memory demand. Increase the minibatch size to decrease processing time. Decrease the minibatch size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the re-identification network feature extraction, specified as one of these values:

- `"auto"` — Use a GPU if Parallel Computing Toolbox is installed and a supported GPU device is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply compatible optimizations suitable for the input network and hardware resource.
- `"none"` — Disable all acceleration.

The `Acceleration` option `"auto"` can improve performance on subsequent calls with compatible parameters, at the expense of an increased initial run time. Use performance optimization when you plan to call the function multiple times using new input data.

`Verbose` — Visible progress display
`true` or `1` (default) | `false` or `0`

Visible progress display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Output Arguments

`features` — Features extracted from re-identification network
M-by-1 vector | *M*-by-*B* matrix | *M*-by-*N* matrix

Features extracted from the re-identification network, returned as one of these options:

- *M*-by-1 vector — `I` is a single image. *M* is the feature vector size `FeatureLength`.
- *M*-by-*B* matrix — `I` is a batch of images. *B* is the number of images in the batch.
- *M*-by-*N* matrix — `I` is a datastore, `ds`. *N* is the number of images in the datastore.

`labels` — Labels
1-by-*N* vector of strings

Labels corresponding to the object ID or class name of each feature vector in `features`, returned as a 1-by-*N* vector of strings. *N* is the number of images in the input datastore, `ds`.

Version History

Introduced in R2024a

See Also

`reidentificationNetwork` | `trainReidentificationNetwork` | `evaluateReidentificationNetwork` | `reidentificationMetrics` | `imageDatastore`

Topics
“Reidentify People Throughout a Video Sequence Using ReID Network”
“Automate Ground Truth Labeling for Object Tracking and Re-Identification”
“Convert Ground Truth Labeling Data for Object Re-Identification”
“Datastores for Deep Learning” (Deep Learning Toolbox)

reidentificationMetrics

Re-identification (ReID) quality metrics

Description

An `reidentificationMetrics` object stores ReID quality metrics, such as the cumulative matching characteristic (CMC) curve and mean average precision (mAP), for a set of images.

Creation

Create an `reidentificationMetrics` object by using the `evaluateReidentificationNetwork` function.

Properties

`DataSetMetrics` — Metrics aggregated over the data set
table

This property is read-only.

Metrics aggregated over the data set, stored as a table with one row. The `DatasetMetrics` property has columns corresponding to these evaluation metrics. These metrics are not in output table order.

- `AP` — Average precision, stored as a numeric scalar. The average precision metric represents the accuracy of image search results for a query image.
- `mAP` — Mean average precision (mAP), stored as a numeric scalar. The mAP is an average of the average precision `AP` in the same table across all corresponding queries.
- `CMC` — Cumulative matching characteristic, stored as a numeric scalar.

`ClassMetrics` — Metrics for each class
table

This property is read-only.

Metrics for each class, stored as a table with *C* rows, where *C* is the number of object classes specified to the `labels` argument of the `evaluateReidentificationNetwork` function. The `ClassMetrics` has columns corresponding to these evaluation metrics. These metrics are not in output table order.

- `AP` — Average precision, stored as a numeric scalar. The average precision metric represents the accuracy of image search results for a query image.
- `mAP` — Mean average precision (mAP), stored as a numeric scalar. The mAP is an average of the average precision `AP` in the same table across all corresponding queries.
- `CMC` — Cumulative matching characteristic, stored as a numeric scalar.

`ClassNames` — Object class names
cell array | categorical vector | cell array of character vectors

This property is read-only.

Names of the object classes that the ReID network is trained to re-identify, stored as a cell array, categorical vector, or cell array of character vectors.

Object Functions

plot Plot re-identification (ReID) metrics

Version History

Introduced in R2024a

See Also

`evaluateReidentificationNetwork` | `trainReidentificationNetwork` | `reidentificationNetwork` | `extractReidentificationFeatures`

Topics

“Reidentify People Throughout a Video Sequence Using ReID Network”

solov2

Segment objects using SOLOv2 instance segmentation network

Description

The `solov2` object performs instance segmentation of objects in an image using a Segmenting Objects by Locations version 2 (SOLOv2) instance segmentation network. To perform instance segmentation of objects in an image, pass the pretrained or trained network to the `segmentObjects` function.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation. You can install the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = solov2(detectorName)
detector = solov2(detectorName,classNames)
detector = solov2( ___ ,Property=Value)
```

Description

`detector = solov2(detectorName)` creates a pretrained SOLOv2 instance segmentation network `detectorName` by using the specified SOLOv2 deep learning network trained on the COCO data set.

`detector = solov2(detectorName,classNames)` creates a pretrained SOLOv2 instance segmentation network and configures it to perform transfer learning on a specified set of object classes. The `classNames` argument sets the `ClassNames` property. For optimal results, train the network on new training images before performing segmentation.

`detector = solov2( ___ ,Property=Value)` sets writeable properties `ModelName` and `InputSize` using name-value arguments, in addition to any combination of input arguments from previous syntaxes.

For example, `InputSize=[400 400 3]` specifies the input image size as 400-by-400 pixels with 3 channels.

Input Arguments

`detectorName` — Name of pretrained SOLOv2 deep learning network
`"resnet50-coco"` | `"light-resnet18-coco"`

Name of the pretrained SOLOv2 deep learning network, specified as one of these options:

- `"resnet50-coco"` — A pretrained SOLOv2 deep learning network that uses ResNet-50 as the base network, trained on the COCO data set.
- `"light-resnet18-coco"` — A pretrained SOLOv2 deep learning network that uses a version of ResNet-18 with a reduced number of features as a base network, trained on the COCO data set.

Data Types: `char` | `string`

`classNames` — Names of object classes
vector of strings | cell array of character vectors | categorical vector

Names of object classes for training the network, specified as a vector of strings, cell array of character vectors, or categorical vector. This argument sets the `classNames` property of the `solov2` object.

Data Types: `char` | `string` | `categorical`

Properties

`ModelName` — Name of trained SOLOv2 instance segmentation network
string scalar | character vector

Name of the trained SOLOv2 instance segmentation network, represented as string scalar or character vector. The default `ModelName` is empty unless specified by `detectorName`.

To set the value of this property, you must specify it at object creation.

`InputSize` — Image size
`[800 800 3]` (default) | 1-by-3 vector

Image size to use during training and inference, represented as a 1-by-3 vector of positive integers of the form [*height width channel*]. The network resizes input images to this size while maintaining the aspect ratio.

To set the value of this property, you must specify it at object creation.

`ClassNames` — Object class names
string array

This property is read-only.

Object class names, represented as a string array. The default value consists of the 80 object class names in the MS-COCO data set, such as `"person"`, `"bicycle"`, and `"car"`. When you specify the `ClassNames` argument, the `solov2` object reinitializes the final convolution layers in the detection head and mask segmentation head to the correct size based on the number of classes you specify.

`GridSizes` — Grid sizes
`[12 16 24 46 40]` (default) | 1-by-5 vector

This property is read-only.

Grid sizes used at the five levels of the feature pyramid network in the SOLOv2 network detection head, represented as a 1-by-5 vector. Each level of the feature pyramid corresponds to a different spatial resolution, with higher stages corresponding to lower resolution and larger receptive fields. The model uses multiple grid sizes in the detection head to predict object masks at different scales and resolutions, which helps to improve the accuracy of the segmentation results.

Object Functions

segmentObjects Segment objects using SOLOv2 instance segmentation

Examples

Segment Instances of Objects with SOLOv2

Load a pretrained SOLOv2 instance segmentation network.

```
model = solov2("resnet50-coco")
```

```
model =
  solov2 with properties:

     ModelName: "resnet50-coco"
    ClassNames: {1×80 cell}
     InputSize: [800 800 3]
     GridSizes: [12 16 24 36 40]
```

Read a test image that includes objects that the network is trained detect, such as people, into the workspace.

```
I = imread("visionteam.jpg");
```

Segment instances of objects in the image using the SOLOv2 instance segmentation model.

```
[masks,labels,scores] = segmentObjects(model,I,Threshold=0.6);
```

Overlay the detected object masks on the test image.

```
overlayedImage = insertObjectMask(I,masks);
figure
imshow(overlayedImage)
```



Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `solov2` object for code generation, use `coder.loadDeepLearningNetwork`.
- For code generation, the `solov2` object does not support `imageDatastore` object as input.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `solov2` object for code generation, use `coder.loadDeepLearningNetwork`.
- For code generation, the `solov2` object does not support `imageDatastore` object as input.

See Also

Apps

Image Labeler | Video Labeler

Functions
`trainSOLOV2` | `evaluateInstanceSegmentation`

Topics
“Get Started with SOLOv2 for Instance Segmentation”
“Perform Instance Segmentation Using SOLOv2”

segmentObjects

Segment objects using SOLOv2 instance segmentation

Syntax

```
masks = segmentObjects(detector,I)
[masks,labels] = segmentObjects(detector,I)
[masks,labels,scores] = segmentObjects(detector,I)

dsResults = segmentObjects(detector,imds)

[___] = segmentObjects(___,Name=Value)
```

Description

`masks = segmentObjects(detector,I)` segments objects within a single image or array of images `I` using SOLOv2 instance segmentation, and returns the predicted object masks for the input image or images.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation. You can install the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[masks,labels] = segmentObjects(detector,I)` also returns the labels assigned to the predicted object instance masks.

`[masks,labels,scores] = segmentObjects(detector,I)` also returns the prediction score for each predicted object instance mask.

`dsResults = segmentObjects(detector,imds)` segments objects within images in a datastore using SOLOv2 instance segmentation. The function returns a datastore with the instance segmentation results, including the instance masks, labels, prediction scores, and bounding boxes.

`[___] = segmentObjects(___,Name=Value)` specifies options using additional name-value arguments in addition to any combination of arguments from previous syntaxes.. For example, `Threshold=0.9` specifies the confidence threshold as `0.9`.

Examples

Segment Instances of Objects

Create a pretrained SOLOv2 instance segmentation network.

```
model = solov2("light-resnet18-coco");
```

Read a test image that includes objects that the network can detect, such as dogs, into the workspace.

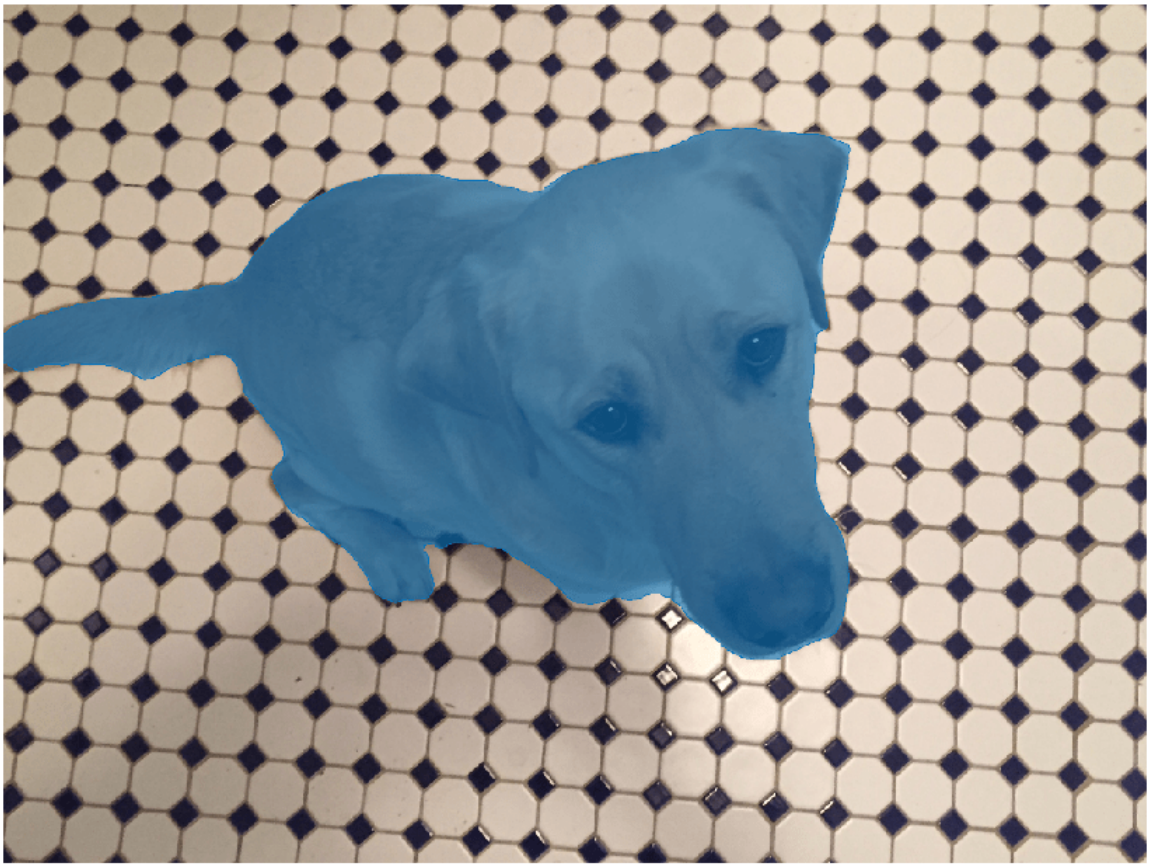
```
I = imread("kobi.png");
```

Segment instances of objects in the image using the SOLOv2 instance segmentation model.

```
[masks,labels,scores] = segmentObjects(model,I);
```

Display the instance segmentation results. Overlay the detected object instance mask on the test image.

```
overlayedImage = insertObjectMask(I,masks);
imshow(overlayedImage)
```



Segment Instances of Objects in Image Datastore with SOLOv2

Load a pretrained SOLOv2 instance segmentation network.

```
model = solov2("resnet50-coco");
```

Create a datastore of test images.

```
imageFiles = fullfile(toolboxdir("vision"),"visiondata","visionteam*.jpg");
dsTest = imageDatastore(imageFiles);
```

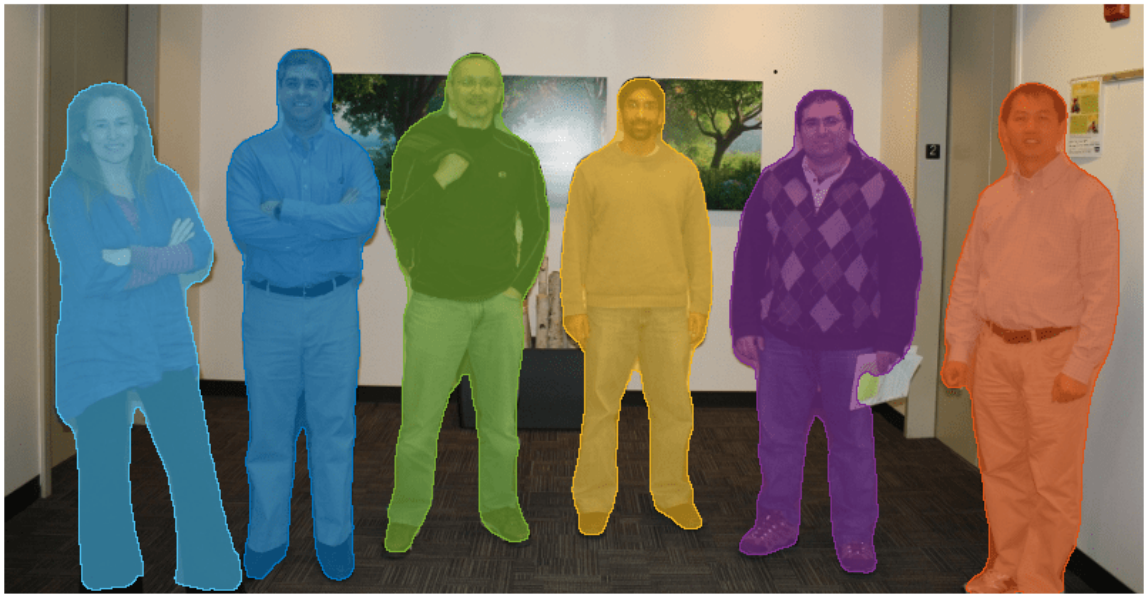
Segment instances of objects using the SOLOv2 instance segmentation model.

```
dsResults = segmentObjects(model,dsTest,Threshold=0.55);
```

```
Running SoloV2 network
-------------------------
* Processed 2 images.
```

For each test image, display the instance segmentation results. Overlay the detected object masks on the test image.

```
while(hasdata(dsResults))
    testImage = read(dsTest);
    results = read(dsResults);
    maskColors = lines(numel(results{2}));
    figure
    overlayedImage = insertObjectMask(testImage,results{1},Color=maskColors);
    imshow(overlayedImage)
end
```





Input Arguments

`detector` — SOLOv2 instance segmentation object

`solov2` object

SOLOv2 instance segmentation model, specified as a `solov2` object.

`I` — Image or batch of images

numeric matrix | numeric array

Image or batch of images on which to perform instance segmentation, specified as one of these values.

Image Type	Data Format
Single grayscale image	2-D matrix of size *H*-by-*W*
Single color image	3-D array of size *H*-by-*W*-by-3.
Batch of *B* grayscale or color images	4-D array of size *H*-by-*W*-by-*C*-by-*B*. The number of color channels *C* is 1 for grayscale images and 3 for color images.

The height *H* and width *W* of each image must be greater than or equal to the input height *h* and width *w* of the network.

`imds` — Datastore of images
datastore

Datastore of images, specified as a datastore such as an `ImageDatastore` or `CombinedDatastore` object. If calling the datastore with the `read` function returns a cell array, then the image data must be in the first cell.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `segmentObjects(detector,I,Threshold=0.9)` specifies the confidence threshold as `0.9`.

Options for All Image Formats

`Threshold` — Confidence threshold
`0.5` (default) | numeric scalar in range `[0, 1]`

Confidence threshold, specified as a numeric scalar in the range `[0, 1]`. The `segmentObjects` function filters out predictions with confidence scores less than the threshold value. Increase this value to reduce the number of false positives, at the possible expense of missing some true positives.

`MaskThreshold` — Mask probability threshold
`0.5` (default) | numeric scalar in range `[0, 1]`

Mask probability threshold, specified as a numeric scalar in the range `[0, 1]`. The mask probability threshold is the threshold value for the mask probabilities, determined by an output activation function, that separate object mask pixels from background pixels. If the threshold is too high, the function might incorrectly classify some foreground object pixels as background pixels, reducing the accuracy of the segmentation.

`SelectStrongest` — Select strongest mask prediction
`true` or `1` (default) | `false` or `0`

Select the strongest mask prediction for each segmented object instance using non-maximum suppression, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Return the strongest object mask prediction per object. The `segmentObjects` function selects these predictions by using non-maximum suppression to eliminate overlapping bounding boxes based on their confidence scores.
- `false` — Return all predictions. You can then create a custom operation to eliminate overlapping object masks.

`Acceleration` — Network acceleration type
`"auto"` (default) | `"mex"` | `"none"`

Network acceleration type to use for performance optimization, specified as one of these options:

- `"auto"` — Automatically select optimizations suitable for the input network and environment.
- `"mex"` — Compile and execute a MEX function. This option is available when using a GPU only. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Disable all acceleration.

Use network acceleration to improve performance when using the same instance segmentation network and segmentation parameters across multiple image inputs, at the expense of additional overhead on the initial function call, and a possible increase in memory usage.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to process images with the network, specified as one of the execution environment options in this table.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

`MaxNumKernels` — Maximum number of convolution kernels
`"auto"` (default) | positive integer in range `[1, 3096]`

Maximum number of convolution kernels, specified as a positive integer in the range [1, 3096]. This value sets the maximum number of convolution kernels, or filters, that the SOLOv2 network uses to perform a convolution operation for producing segmentation masks.

Specify the value of `MaxNumKernels` only when performing code generation. Otherwise, use the default value. The default value, `"auto"`, sets the maximum number of kernels depending on the content of the image, based on the number of kernels with an acceptable confidence threshold.

Tip Determine the optimal value of `MaxKernelSize` for your application by using the `evaluateInstanceSegmentation` function to evaluate the network performance at different `MaxKernelSize` values. Increase the value of `MaxKernelSize` to increase instance segmentation accuracy at the expense of slower inference speed.

Options for Datastore Inputs

`MiniBatchSize` — Number of observations returned in each batch

`1` (default) | positive integer

Number of observations returned in each batch, specified as a positive integer. If you set a higher `MiniBatchSize`, segmentation requires more memory, which can cause errors if your system does not have sufficient memory.

You can specify this argument only when you specify a batch of images, `I`, or datastore of images, `imds`, as an input to the `segmentObjects` function.

`WriteLocation` — Location to store writable data

string scalar | character vector

Location to store writable data, specified as a string scalar or character vector. The specified folder must have write permissions. If the folder already exists, the `segmentObjects` function creates a new folder and adds a suffix to the folder name with the next available number. The default write location is `fullfile(pwd,"SegmentObjectResults")`, where `pwd` is the current working directory.

You can specify this argument only when you specify a datastore of images, `imds`, as an input to the `segmentObjects` function.

Data Types: `char` | `string`

`NamePrefix` — Prefix added to written filenames

`"segmentObj"` (default) | string scalar | character vector

Prefix added to written filenames, specified as a string scalar or character vector. The function names the output files *`NamePrefix_imageName.mat`*, where *`imageName`* is the name of the input image without its file extension.

If `imds` contains identical file names in different folders, the function names the output files *`NamePrefix_folder_imageName.mat`*, where *`folder`* is the folder containing the files with the same name.

If `imds` contains identical file names in folders of the same name within different directory hierarchies, the function names the output files *`NamePrefix_folder_imageName_#.mat`*, where `#` is a distinct integer.

You can specify this argument only when you specify a datastore of images, `imds`.

Data Types: `char` | `string`

`Verbose` — Visible progress display

`true` or `1` (default) | `false` or `0`

Visible progress display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

You can specify this argument only when you specify a datastore of images, `imds`.

Output Arguments

`masks` — Object masks

H-by-*W*-by-*M* logical array | *B*-by-1 cell array

Object masks, returned as an *H*-by-*W*-by-*M* logical array for a single image or a *B*-by-1 cell array for a batch of *B* images. *H* and *W* are the height and width, respectively, of the input image `I`, and *M* is the number of objects masks predicted in the image. Each of the *M* channels contains the mask for a single predicted object instance.

For a batch of *B* images, each cell of the *B*-by-1 cell array contains an *H*-by-*W*-by-*M* array of object masks the corresponding image from the batch.

`labels` — Object labels
M-by-1 categorical vector | *B*-by-1 cell array

Objects labels, returned as an *M*-by-1 categorical vector for a single image or a *B*-by-1 cell array for a batch of *B* images. *M* is the number of predicted object instances in the input image `I`.

For a batch of *B* images, each cell of the *B*-by-1 cell array contains an *M*-by-1 categorical vector with the labels of the objects in the corresponding image from the batch.

`scores` — Prediction scores
M-by-1 vector | *B*-by-1 cell array

Prediction confidence scores, returned as an *M*-by-1 numeric vector for a single image or a *B*-by-1 cell array for a batch of *B* images. *M* is the number of predicted object instances in the input image `I`. A higher score indicates higher confidence in the object instance segmentation.

For a batch of *B* images, each cell of the *B*-by-1 cell array contains an *M*-by-1 numeric vector with the confidence scores for the object segmentation predictions in the corresponding image from the batch.

`dsResults` — Predicted instance segmentation results
`FileDatastore` object

Predicted instance segmentation results, returned as a `FileDatastore` object. The function organizes the datastore so that calling the `read` and `readall` functions on it returns a cell array with three columns. This table describes the format of each cell in each column.

`masks`	`labels`	`scores`
Binary masks, returned as a logical array of size *H*-by-*W*-by-*M*, where *M* is the number of predicted object instances in the corresponding image. Each mask is the segmentation of one object instance in the image.	Object class names, returned as an *M*-by-1 categorical vector, where *M* is the number of predicted object instances in the corresponding image. All categorical data returned by the datastore contains the same categories.	Prediction scores, returned as an *M*-by-1 numeric vector, where *M* is the number of predicted object instances in the corresponding image.

Version History

Introduced in R2023b

R2024b: Specify maximum number of convolution kernels to use for code generation

For code generation, the `segmentObjects` function now requires you to specify the maximum number of convolution kernels to use for mask prediction using the new `MaxNumKernels` name-value argument.

R2024b: Support for C and GPU code generation

`segmentObjects` now supports the generation of C code (requires MATLAB Coder) and optimized CUDA code (requires GPU Coder).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For code generation, the `segmentObjects` function does not support `WriteLocations`, `NamePrefix`, and `Verbose` name-value arguments.
- For code generation, the `MiniBatchSize` name-value argument must be a code generation constant (`coder.const()`).
- For code generation, the `MaxNumKernels` name-value argument must be specified as a non-default value.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- For code generation, the `segmentObjects` function does not support `WriteLocations`, `NamePrefix`, and `Verbose` name-value arguments.
- For code generation, the `MiniBatchSize` name-value argument must be a code generation constant (`coder.const()`).
- For code generation, the `MaxNumKernels` name-value argument must be specified as a non-default value.

See Also
`solov2` | `trainSOLOV2` | `evaluateInstanceSegmentation`

Topics
“Get Started with SOLOv2 for Instance Segmentation”
“Get Started with Instance Segmentation Using Deep Learning”
“Perform Instance Segmentation Using SOLOv2”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Datastores for Deep Learning” (Deep Learning Toolbox)

yolov2ObjectDetector

Detect objects using YOLO v2 object detector

Description

The `yolov2ObjectDetector` object creates a you only look once version 2 (YOLO v2) object detector for detecting objects in an image. Using this object, you can:

- Create a pretrained YOLO v2 object detector by using YOLO v2 deep learning networks trained on COCO data set.
- Create a custom YOLO v2 object detector by using a custom trained YOLO v2 deep learning network.

Creation

Syntax

```
detector = yolov2ObjectDetector(name)

detector = yolov2ObjectDetector(name,classes,aboxes)
detector = yolov2ObjectDetector(net,classes,aboxes)
detector = yolov2ObjectDetector(baseNet,classes,
aboxes,DetectionNetworkSource=layer)
detector = yolov2ObjectDetector( ___ ,Name=Value)
```

Description

Pretrained YOLO v2 Object Detector

`detector = yolov2ObjectDetector(name)` creates a pretrained YOLO v2 object detector by using YOLO v2 deep learning networks trained on a COCO data set.

Note To use pretrained YOLO v2 networks trained on the COCO data set, you must download and install the Computer Vision Toolbox Model for YOLO v2 Object Detection support package. You can download the Computer Vision Toolbox Model for YOLO v2 Object Detection from the Add-On Explorer. For more information, see “Get and Manage Add-Ons”.

Custom YOLO v2 Object Detector

`detector = yolov2ObjectDetector(name,classes,aboxes)` creates a pretrained YOLO v2 object detector and configures it to perform transfer learning.

For optimal results, you must train the detector on new training images before performing detection. Use the `trainYOLOv2ObjectDetector` function for training the detector.

`detector = yolov2ObjectDetector(net,classes,aboxes)` creates an object detector by using the YOLO v2 deep learning network `net`.

If `net` is a pretrained YOLO v2 deep learning network, then the function creates a pretrained YOLO v2 object detector. `classes` and `aboxes` specify the object classes and anchor boxes that were used to train the network.

If `net` is an untrained YOLO v2 deep learning network, then the function creates a YOLO v2 object detector to use for training and inference. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v2 network. You must train the detector on a training data set before performing object detection. For information about how to train a YOLO v2 object detector, see “Object Detection Using YOLO v2 Deep Learning”.

The input network can also be an imported network from ONNX™ (Open Neural Network Exchange). For more information, see “Import Pretrained ONNX YOLO v2 Object Detector”.

`detector = yolov2ObjectDetector(baseNet,classes, aboxes,DetectionNetworkSource=layer)` adds a detection head to the specified feature extraction layer `layer` in the base network, `baseNet`.

If `baseNet` is a pretrained deep learning network, the function creates a YOLO v2 object detector and configures it to perform transfer learning with the specified object classes and anchor boxes.

If `baseNet` is an untrained deep learning network, the function creates a YOLO v2 object detector and configures it for training. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v2 network.

You must train the detector on a training data set before performing object detection.

`detector = yolov2ObjectDetector( ___ ,Name=Value)` sets properties on page 2-1231 of the object detector by using name-value arguments. You can specify one or more of the `InputSize on page 2-0`, `LossFactors on page 2-0`, and `ModelName on page 2-0` properties, in addition to any combination of input arguments from previous syntaxes.

You can enable multiscale training by setting the `TrainingImageSize on page 2-0` property using a name-value argument.

When you specify the `layer` input argument, then you can improve detection accuracy for smaller objects by setting the `ReorganizeLayerSource on page 2-0` property using a name-value argument.

Example: `detector = yolov2ObjectDetector(net,classes,aboxes,InputSize=[224 224 3])` sets the size of the images used for training to `[224 224 3]`.

Input Arguments

`name` — Name of pretrained YOLO v2 deep learning network

`"darknet19-coco"` | `"tiny-yolov2-coco"`

Name of the pretrained YOLO v2 deep learning network, specified as one of these:

- `"darknet19-coco"` — A pretrained YOLO v2 deep learning network created using DarkNet-19 as the base network and trained on COCO data set.
- `"tiny-yolov2-coco"` — A pretrained YOLO v2 deep learning network created using a small base network and trained on COCO data set.

Data Types: `char` | `string`

`classes` — Names of object classes
string vector | cell array of character vectors | categorical vector

Names of object classes for training the detector, specified as a string vector, cell array of character vectors, or categorical vector. This argument sets the `ClassNames on page 2-0` property of the `yolov2ObjectDetector` object.

Data Types: `char` | `string` | `categorical`

`aboxes` — Anchor boxes
N-by-1 cell array

Anchor boxes for training the detector, specified as an *N*-by-1 cell array. *N* is the number of output layers in the YOLO v2 deep learning network. Each cell contains an *M*-by-2 matrix, where *M* is the number of anchor boxes in that layer. Each cell can contain a different number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of an anchor box in the form [*height width*].

The first element in the cell array specifies the anchor boxes to associate with the first output layer, the second element in the cell array specifies the anchor boxes to associate with the second output layer, and so on. For accurate detection results, specify large anchor boxes for the first output layer and small anchor boxes for the last output layer. That is, the anchor box sizes must decrease for each output layer in the order in which the layers appear in the YOLO v2 deep learning network.

This argument sets the `AnchorBoxes on page 2-0` property of the `yolov2ObjectDetector` object.

Data Types: `cell`

`net` — Custom trained YOLO v2 network
`dlnetwork` object

Custom trained YOLO v2 network, specified as a `dlnetwork` object. The `dlnetwork` must have an image input layer and a YOLO v2 transform layer.

`baseNet` — Base network
`dlnetwork` object

Base network for creating the YOLO v2 deep learning network, specified as a `dlnetwork` object. The network can be either an untrained or a pretrained deep learning network.

`layer` — Name of feature extraction layer
character vector | string scalar

Name of the feature extraction layer, specified as a character vector or string scalar. The features extracted from this layer are given as input to the YOLO v2 object detection subnetwork.

You can specify the feature extraction layer as any network layer except the fully connected layer. The feature extraction layer is typically one of the deeper layers in `network`. You can use the `analyzeNetwork` function to view the names of the layers in the input network.

Data Types: `char` | `string`

Properties

`Network` — YOLO v2 object detection network
`dlnetwork` object

This property is read-only.

YOLO v2 object detection network, specified as a `dlnetwork` object.

`InputSize` — Size of input image
two-element vector | three-element vector

Size of input image, specified as one of these values:

- Two-element vector of form [*H W*] - For a grayscale image of size *H*-by-*W*
- Three-element vector of form [*H W* 3] - For an RGB color image of size *H*-by-*W*

The default value is the size of the image input layer of the network.

`TrainingImageSize` — Set of image sizes used for training
M-by-2 matrix

Set of image sizes used for training, specified as an *M*-by-2 matrix, where each row is of the form [`height` `width`]. By default, `yolov2ObjectDetector` uses a single image size, equal to the height and width specified by the `InputSize on page 2-0` property.

Note

- The image sizes specified by `TrainingImageSize` must be greater than or equal to the `InputSize` property of the `imageInputLayer` within the network of the detector.
- The `height` and `width` values of `TrainingImageSize` must be divisible by the height and width of the `BlockSize` property of the `spaceToDepthLayer` object within the network of the detector.

You cannot change the value of this property after you create the object.

`AnchorBoxes` — Set of anchor boxes
N-by-2 matrix

This property is read-only.

Set of anchor boxes, specified as an *N*-by-2 matrix defining the width and the height of *N* anchor boxes. Each row in the *M*-by-2 matrix denotes the size of the anchor box in the form of [*height width*].

The size of each anchor box is determined based on the scale and aspect ratio of different object classes present in input training data. Also, the size of each anchor box must be smaller than or equal to the size of the input image. You can use the clustering approach for estimating anchor boxes from the training data. For more information, see “Estimate Anchor Boxes from Training Data”.

`ClassNames` — Names of object classes
cell array of character vectors

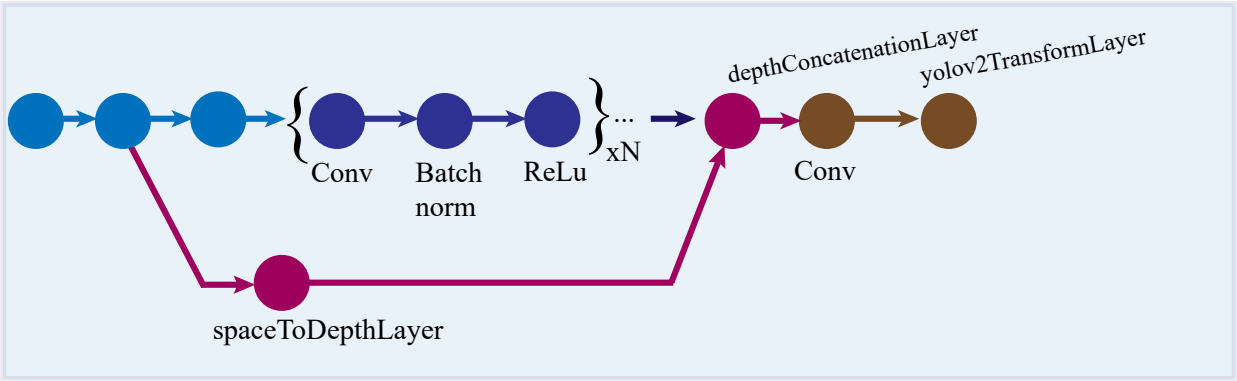
This property is read-only.

Names of object classes that the YOLO v2 object detector is trained to find, specified as a cell array of character vectors.

`ReorganizeLayerSource` — Name of layer providing input to reorganization layer
"" (default) | character vector | string scalar

Name of the layer providing input to the reorganization layer, specified as a character vector or string scalar. A reorganization layer improves detection accuracy for smaller objects by adding low-level image information. To include a reorganization layer in the network, you must also specify the `layer` argument. If you do not set the `ReorganizeLayerSource` property, then the detector does not include a reorganization layer in the network.

When you include a reorganization branch, the object adds a `spaceToDepthLayer` and `depthConcatenationLayer` to the network. The space-to-depth layer extracts low-level features from the base network. The depth concatenation layer combines the high-level features from the detection head with the low-level features. The figure shows a modified network that includes a reorganization branch.


depthConcatenationLayer
yolov2TransformLayer
Conv
Batch norm
ReLu
...
xN
Conv
spaceToDepthLayer

The input to the reorganization layer must be from any one of the network layers that lie above the feature extraction layer specified by `layer`. Typically, you attach the reorganization layer to a layer within the feature extraction network whose output feature map is larger than the feature extraction layer output. You can use the `analyzeNetwork` function to view the names of the layers in the input `network`.

`LossFactors` — Loss factors
`[5 1 1 1]` (default) | 1-by-4 numeric vector

This property is read-only.

Loss factors for each component of the YOLO v2 loss function, specified as a 1-by-4 numeric vector. For more information, see YOLO v2 Training Loss on page 3-1402.

`ModelName` — Name for object detector
character vector | string scalar

Name for object detector, specified as a character vector or string scalar. The default value of this property is set to the name of the pretrained YOLO v2 network specified at the input.

Object Functions

detect Detect objects using YOLO v2 object detector

Examples

Detect Objects Using Pretrained YOLO v2 Object Detector

To run this example, you must download and install the Computer Vision Toolbox Model™ for YOLO v2 Object Detection support package.

Specify the name of a pretrained YOLO v2 deep learning network.

```
name = "tiny-yolov2-coco";
```

Create a YOLO v2 object detector by using the pretrained YOLO v2 network.

```
detector = yolov2ObjectDetector(name);
```

Display and inspect the properties of the YOLO v2 object detector.

```
disp(detector)
```

```
  yolov2ObjectDetector with properties:

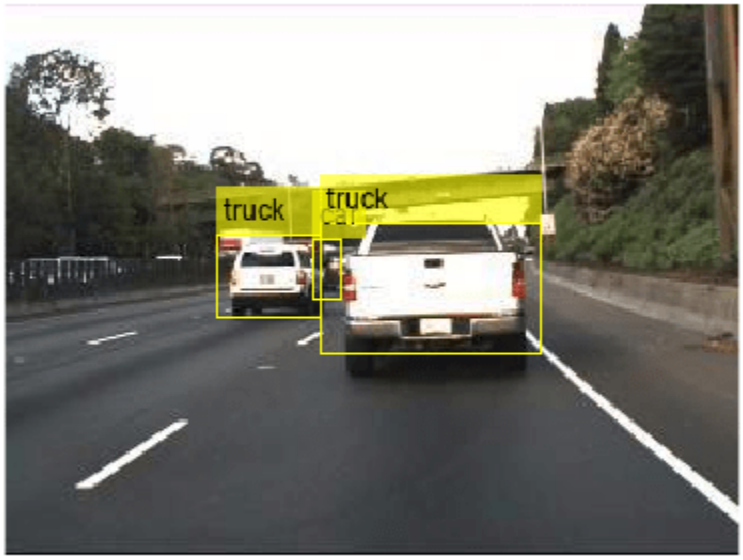
                Network: [1×1 dlnetwork]
              InputSize: [416 416 3]
      TrainingImageSize: [416 416]
            AnchorBoxes: [5×2 double]
             ClassNames: [80×1 categorical]
  ReorganizeLayerSource: ''
            LossFactors: [5 1 1 1]
              ModelName: 'tiny-yolov2-coco'
```

Detect objects in an unknown image by using the pretrained YOLO v2 object detector.

```
img = imread("highway.png");
[bboxes,scores,labels] = detect(detector,img);
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(img,"Rectangle",bboxes,labels);
imshow(detectedImg)
```



Create YOLO v2 Network with Reorganization Layer

Specify the class for the network to detect. The class must be one of the classes from the ImageNet database.

```
classes = "car";
```

Define anchor boxes.

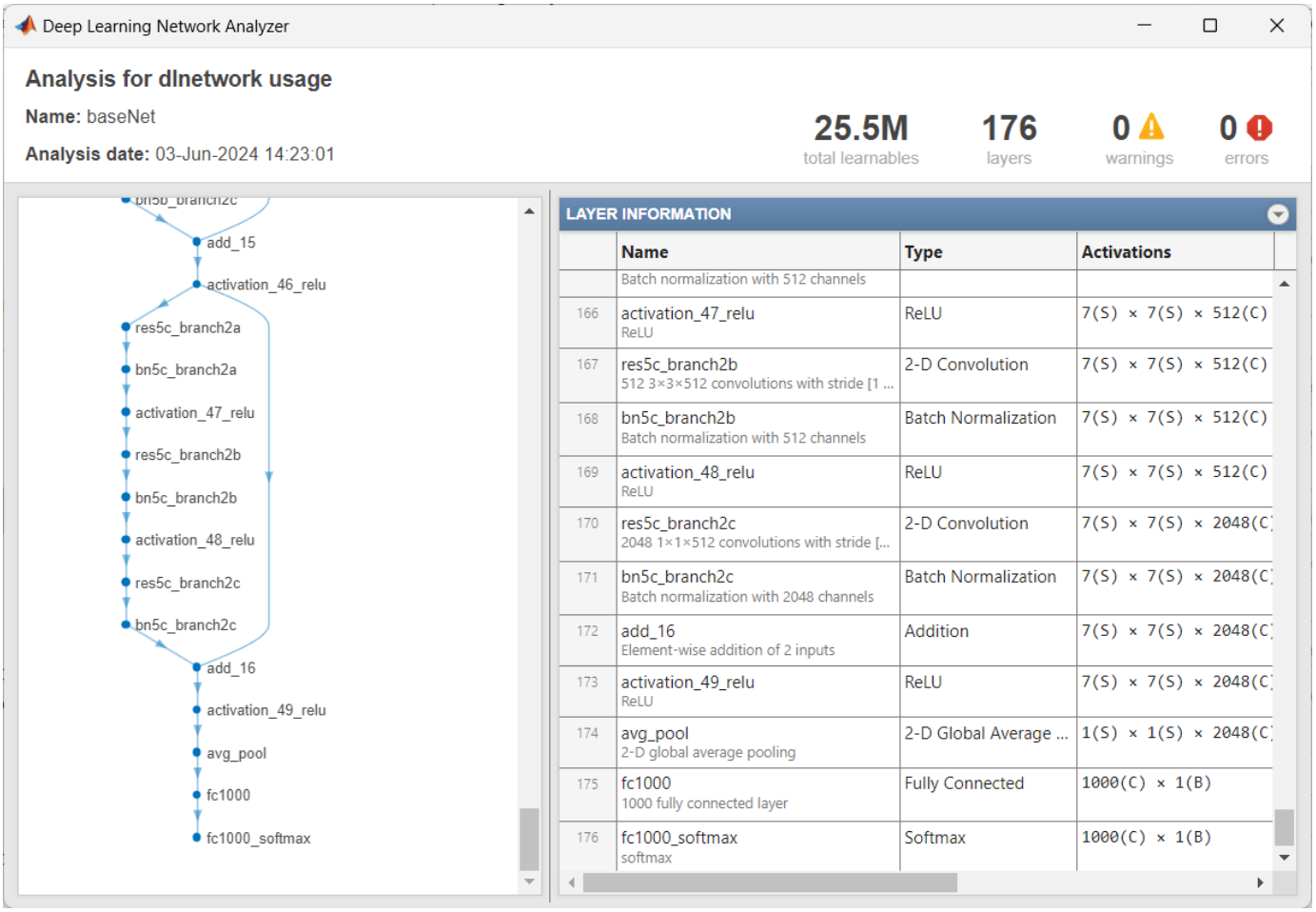
```
anchorBoxes = [1 1;4 6;5 3;9 6];
```

Specify the base network as the pretrained ResNet-50 network. To use this pretrained network, you need to install the *Deep Learning Toolbox Model for ResNet-50 Network* support package.

```
baseNet = imagePretrainedNetwork("resnet50");
```

Analyze the network architecture and view all of the network layers.

```
analyzeNetwork(baseNet)
```



Specify the network layer to be used for feature extraction. You can choose any layer except the fully connected layer.

```
featureSource = "activation_49_relu";
```

Specify the network layer to be used as input to the reorganization layer.

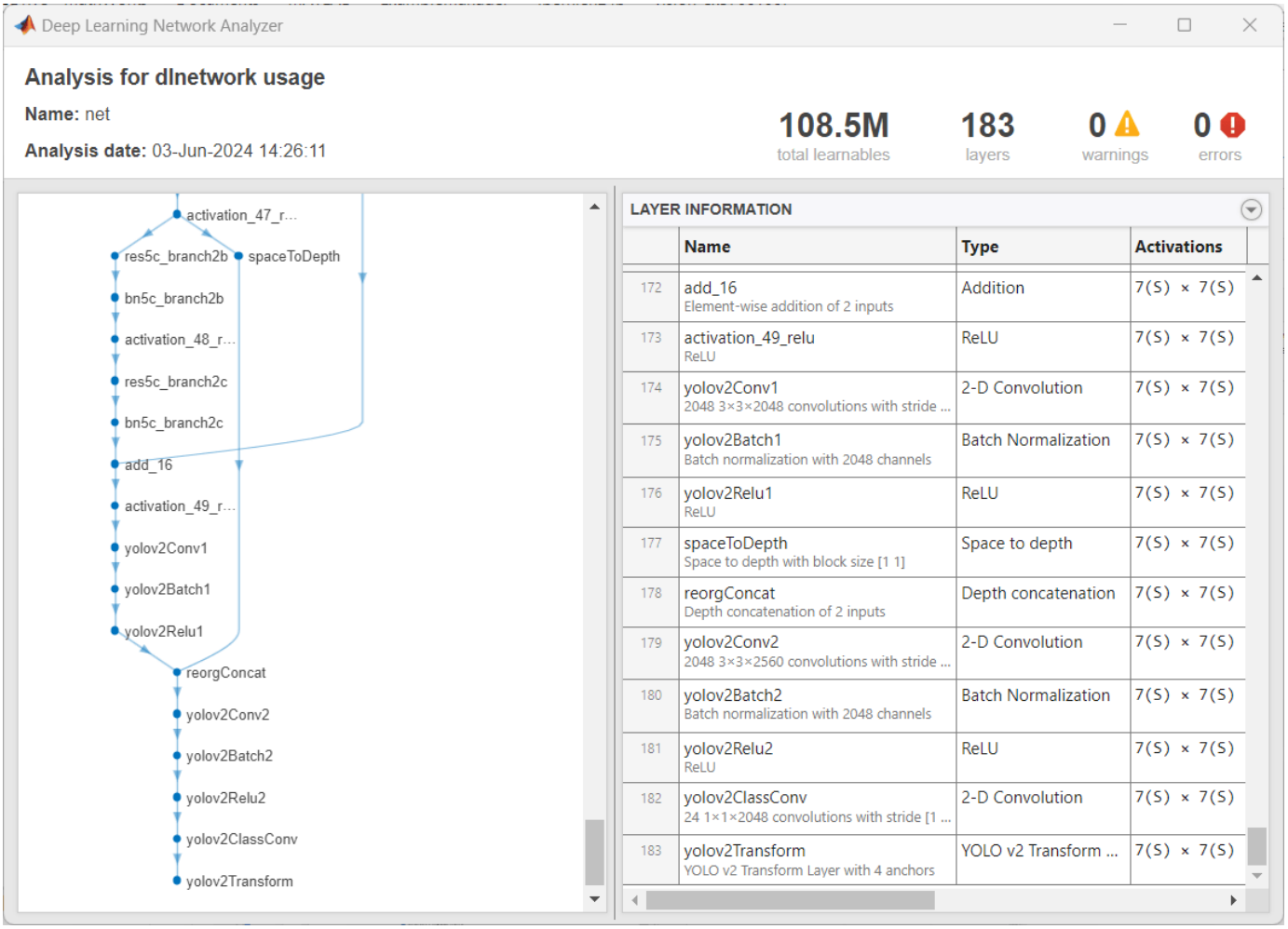
```
reorgSource = "activation_47_relu";
```

Create the YOLO v2 object detector. The detector creates and stores the YOLO v2 network as a `dlnetwork` object.

```
detector = yolov2ObjectDetector(baseNet,classes,anchorBoxes, ...
    DetectionNetworkSource=featureSource,ReorganizeLayerSource=reorgSource);
```

Analyze the YOLO v2 network architecture. The layers after the feature extraction layer are removed. The detection subnetwork, including the YOLO v2 transform layer, follow the feature extraction layer. The reorganization layer (a space to depth layer) and the depth concatenation layer are also added to the network.

```
net = detector.Network;
```



Version History

Introduced in R2019a

R2024b: New properties

Starting in R2024b, these new properties enable training custom YOLO v2 object detectors specified as `dlnetwork` objects:

- `InputSize` on page 2-0
- `ReorganizeLayerSource` on page 2-0
- `LossFactors` on page 2-0

R2024b: `DAGNetwork` objects are not recommended

Not recommended starting in R2024b

Starting in R2024b, `DAGNetwork` objects are not recommended. If you want to use a custom pretrained YOLO v2 network, specify `net` or `baseNet` as a `dlnetwork` object instead. You must also specify the class names and anchor boxes using new input arguments.

There are no plans to remove support for `DAGNetwork` objects. However, `dlnetwork` objects have these advantages and are recommended instead:

- `dlnetwork` objects are a unified data type that supports network building, prediction, built-in training, visualization, compression, verification, and custom training loops.
- `dlnetwork` objects support a wider range of network architectures that you can create or import from external platforms.
- Training and prediction with `dlnetwork` objects is typically faster than with `DAGNetwork` objects.

You can convert a trained `DAGNetwork` object to a `dlnetwork` object by using the `dag2dlnetwork` function.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Only the `detect` method of the `yolov2ObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, and `MaxSize` name-value arguments for `detect` are supported.
- To create a `yolov2ObjectDetector` object for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation,

- Only the `detect` method of the `yolov2ObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported.
- The height, width, channel, and batch size of the input image must be fixed size.
- The minimum batch size value passed to detect method must be fixed size.
- To create a `yolov2ObjectDetector` object for code generation, see “Load Pretrained Networks for Code Generation” (GPU Coder).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainYOLOv2ObjectDetector` | `estimateAnchorBoxes`

Objects
`yolov3ObjectDetector` | `yolov4ObjectDetector` | `yoloxObjectDetector` | `boxLabelDatastore`

Topics
“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using YOLO v2”
“Import Pretrained ONNX YOLO v2 Object Detector”
“Export YOLO v2 Object Detector to ONNX”
“Getting Started with YOLO v2”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

yolov3ObjectDetector

Detect objects using YOLO v3 object detector

Description

The `yolov3ObjectDetector` object creates a you only look once version 3 (YOLO v3) object detector for detecting objects in an image. Using this object, you can:

- Create a pretrained YOLO v3 object detector by using YOLO v3 deep learning networks trained on COCO dataset.
- Create a custom YOLO v3 object detector by using any pretrained or untrained YOLO v3 deep learning network.

Creation

Syntax

```
detector = yolov3ObjectDetector(name)

detector = yolov3ObjectDetector(name,classes,aboxes)
detector = yolov3ObjectDetector(net,classes,aboxes)
detector = yolov3ObjectDetector(baseNet,classes,
aboxes,DetectionNetworkSource=layer)
detector = yolov3ObjectDetector( ___ ,Name=Value)
```

Description

Pretrained YOLO v3 Object Detector

`detector = yolov3ObjectDetector(name)` creates a pretrained YOLO v3 object detector by using YOLO v3 deep learning networks trained on a COCO dataset.

Note To use the pretrained YOLO v3 deep learning networks trained on COCO dataset, you must install the Computer Vision Toolbox Model for YOLO v3 Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

Custom YOLO v3 Object Detector

`detector = yolov3ObjectDetector(name,classes,aboxes)` creates a pretrained YOLO v3 object detector and configures it to perform transfer learning using a specified set of object classes and anchor boxes. For optimal results, you must train the detector on new training images before performing detection.

`detector = yolov3ObjectDetector(net,classes,aboxes)` creates an object detector by using the deep learning network `net`.

If `net` is a pretrained YOLO v3 deep learning network, the function creates a pretrained YOLO v3 object detector. The `classes` and `aboxes` are values used for training the network.

If `net` is an untrained YOLO v3 deep learning network, the function creates a YOLO v3 object detector to use for training and inference. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v3 network.

You must train the detector on a training dataset before performing object detection. For information about how to train a YOLO v3 object detector, see **Preprocess Training Data** and **Train Model** sections in the “Object Detection Using YOLO v3 Deep Learning” example.

`detector = yolov3ObjectDetector(baseNet,classes, aboxes,DetectionNetworkSource=layer)` creates a YOLO v3 object detector by adding detection heads to a base network, `baseNet`.

The function adds detection heads to the specified feature extraction layers `layer` in the base network. To specify the names of the feature extraction layers, use the name-value argument `DetectionNetworkSource=layer`.

If `baseNet` is a pretrained deep learning network, the function creates a YOLO v3 object detector and configures it to perform transfer learning with the specified object classes and anchor boxes.

If `baseNet` is an untrained deep learning network, the function creates a YOLO v3 object detector and configures it for object detection. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v3 network.

You must train the detector on a training dataset before performing object detection.

`detector = yolov3ObjectDetector( ___ ,Name=Value)` specifies one or more options using name-value arguments in addition to any combination of input arguments from previous syntaxes. Use this syntax to:

- Modify the detection network sources in a YOLO v3 object detection network and train the network with different numbers of object classes, anchor boxes, or both. Specify the new detection network sources using the `DetectionNetworkSource=layer` name-value argument.
- Set the `InputSize on page 2-0` and `ModelName on page 2-0` properties of the object detector. For example, `InputSize=[224 224 3]` sets the size of the images used for training to `[224 224 3]`.

Input Arguments

`name` — Name of pretrained YOLO v3 deep learning network

`'darknet53-coco'` | `'tiny-yolov3-coco'`

Name of the pretrained YOLO v3 deep learning network, specified as one of these:

- `'darknet53-coco'` — A pretrained YOLO v3 deep learning network created using DarkNet-53 as the base network and trained on COCO dataset.
- `'tiny-yolov3-coco'` — A pretrained YOLO v3 deep learning network created using a small base network and trained on COCO dataset.

Data Types: `char` | `string`

`classes` — Names of object classes

string vector | cell array of character vectors | categorical vector

Names of object classes for training the detector, specified as a string vector, cell array of character vectors, or categorical vector. This argument sets the `ClassNames` property of the `yolov3ObjectDetector` object.

Data Types: `char` | `string` | `categorical`

aboxes — Anchor boxes
N-by-1 cell array

Anchor boxes for training the detector, specified as an *N*-by-1 cell array. *N* is the number of output layers in the YOLO v3 deep learning network. Each cell contains an *M*-by-2 matrix, where *M* is the number of anchor boxes in that layer. Each cell can contain a different number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of an anchor box in the form [*height width*].

The first element in the cell array specifies the anchor boxes to associate with the first output layer, the second element in the cell array specifies the anchor boxes to associate with the second output layer, and so on. For accurate detection results, specify large anchor boxes for the first output layer and small anchor boxes for the last output layer. That is, the anchor box sizes must decrease for each output layer in the order in which the layers appear in the YOLO v3 deep learning network.

This argument sets the `AnchorBoxes` property of the `yolov3ObjectDetector` object.

Data Types: `cell`

`net` — YOLO v3 deep learning network
`dlnetwork` object

YOLO v3 deep learning network, specified as a `dlnetwork` object. The input network can be either an untrained or a pretrained deep learning network.

`baseNet` — Base network
`dlnetwork` object

Base network for creating the YOLO v3 deep learning network, specified as a `dlnetwork` object. The network can be either an untrained or a pretrained deep learning network.

`layer` — Names of feature extraction layers
cell array of character vectors | string array

Names of the feature extraction layers in the base network, specified as a cell array of character vectors, or a string array. The function creates a YOLO v3 network by adding detection head layers to the output of the feature extraction layers in the base network.

In the pre-trained YOLO v3 base network, you must connect the detection head layers to feature extraction layers where the input image feature map changes spatial dimension by downsampling factors of 4, 8, 16, and 32.

- Use the `analyzeNetwork` function to display the pre-trained YOLO v3 network architecture to obtain the information about the spatial dimensions of the feature maps.
- Use this table to know about the default feature extraction layers and choose alternate feature extraction layers supported by the pre-trained YOLO v3 networks. Choose feature extraction layers using empirical evaluation based on the detection accuracy, training speed, and object size to detect.
    - For multi-scale feature extraction and to achieve higher detection accuracy, choose feature extraction layers with both higher and lower downsampling factors. Layers with lower

downsampling factor capture small objects and layers with higher downsampling factor ensure a thorough feature refinement.

- To achieve higher training and inference speeds, choose feature extraction layers with lower downsampling factor.
- The number of detection network sources specified must be equal to the number of anchor boxes specified at the input.

Backbone	Number of Detection Heads	Default Feature Extraction Layers		Examples of Alternate Feature Extraction Layers		Remarks
		Layer	Downsampling Factor	Layer	Downsampling Factor	
`tiny-yolov3-coco`	2	`leaky_relu_7` `leaky_relu_5`	32 16	`leaky_relu_6` `leaky_relu_3`	16 8	Select any two feature extraction layers with different downsampling factors preceding the `leaky_relu_7` layer to use as detection network sources. For example, specify the new detection network sources as `layer = {'leaky_relu_6','leaky_relu_3'}`

`darknet53-coco`	3	`add_23`
		`add_19`
		`add_11`
			32
			16
			8
				`add_22`
				`add_21`
				`add_20`
					32	Select any three feature extraction layers with different downsampling factors preceding the `add_23`

				`add_18`
				`add_17`
				`add_16`
				`add_15`
				`add_14`
				`add_13`
				`add_12`
					16	layer to use as detection network sources.
						
						For example, specify the new detection network sources as `layer = {'add_18','add_10','add_3'}`

				`add_10`
				`add_9`
				`add_8`
				`add_7`
				`add_6`
				`add_5`
				`add_4`
					8	

				`add_3` `add_2`	4	

Data Types: `char` | `string` | `cell`

Properties

`Network` — YOLO v3 deep learning network

`dlnetwork` object

This property is read-only.

YOLO v3 deep learning network to use for object detection, stored as a `dlnetwork` object.

`AnchorBoxes` — Set of anchor boxes

N-by-1 cell array

This property is read-only.

Set of anchor boxes, stored as a *N*-by-1 cell array. *N* is the number of output layers in the YOLO v3 deep learning network for which the anchor boxes are defined. Each element in the cell is a *M*-by-2 matrix. *M* denotes the number of anchor boxes. Each cell can contain a different number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of the anchor box in the form of [*height width*]. The first element in the cell array specifies the anchor boxes for the first output layer, the second element in the cell array specifies the anchor boxes for the second output layer, and so on.

You can set this property by using the input argument `aboxes`.

`ClassNames` — Names of object classes

categorical vector

This property is read-only.

Names of object classes to detect, stored as a categorical vector. You can set this property by using the input argument `classes`.

`InputSize` — Image size used for training

vector

This property is read-only.

Image size used for training, stored as a vector of form [*height width*] or [*height width channels*]. To set this property, specify it at object creation.

For example, `detector = yolov3ObjectDetector(net,classes,aboxes,InputSize=[224 224 3])`.

`ModelName` — Name for object detector

`' '` (default) | character vector

Name for the object detector, stored as a character vector. To set this property, specify it at object creation.

For example, `yolov3ObjectDetector(net,classes,aboxes,'ModelName','customDetector')` sets the name for the object detector to `'customDetector'`.

`PredictedBoxType` — Bounding box format for object detector

`"axis-aligned"` (default) | `"rotated"`

This property is read-only.

Bounding box format for an object detector, stored as `"axis-aligned"` or `"rotated"`. When the `PredictedBoxType` is `"axis-aligned"`, the object detector will train and perform inference on only axis-aligned bounding boxes. If it is set to `"rotated"`, the object detector will train and perform inference on only rotated bounding boxes. Set this property when you create the object.

Object Functions

detect	Detect objects using YOLO v3 object detector
preprocess	Preprocess training and test images
forward	Compute YOLO v3 deep learning network output for training
predict	Compute YOLO v3 deep learning network outputs for inference

Examples

Create Pretrained YOLO v3 Object Detector

Specify the name of a pretrained YOLO v3 deep learning network.

```
name = 'tiny-yolov3-coco';
```

Create YOLO v3 object detector by using the pretrained YOLO v3 network.

```
detector = yolov3ObjectDetector(name);
```

Display and inspect the properties of the YOLO v3 object detector.

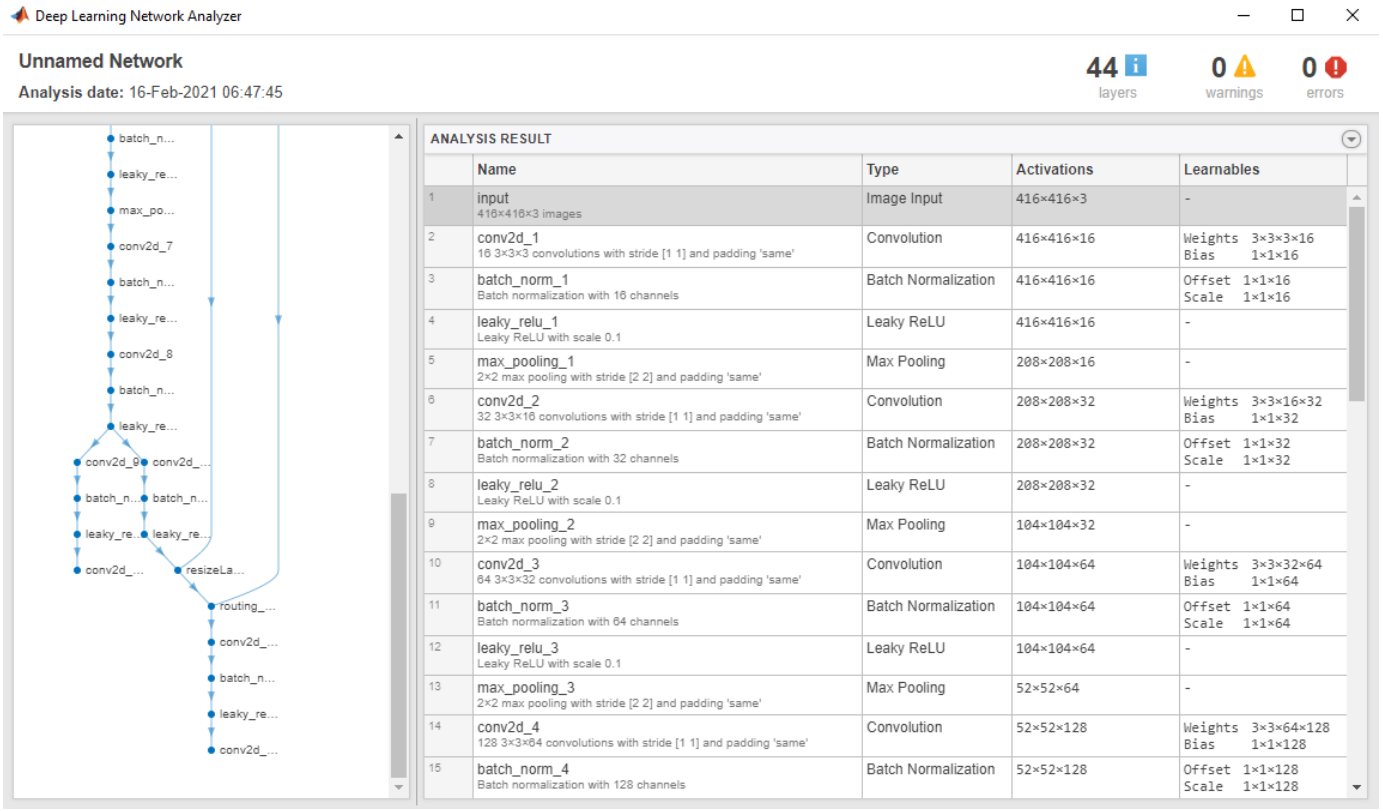
```
disp(detector)
```

```
  yolov3ObjectDetector with properties:

           Network: [1×1 dlnetwork]
       AnchorBoxes: {2×1 cell}
        ClassNames: [80×1 categorical]
         InputSize: [416 416 3]
    PredictedBoxType: 'axis-aligned'
         ModelName: 'tiny-yolov3-coco'
```

Use `analyzeNetwork` to display the YOLO v3 network architecture and get information about the network layers. The network has two detection heads attached to the feature extraction network.

```
analyzeNetwork(detector.Network)
```

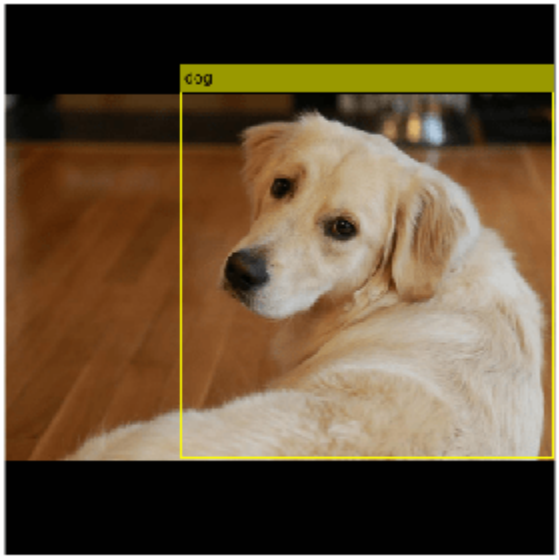


Detect objects in an unknown image by using the pretrained YOLO v3 object detector.

```
img = imread('sherlock.jpg');
img = preprocess(detector,img);
img = im2single(img);
[bboxes,scores,labels] = detect(detector,img,'DetectionPreprocessing','none');
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(img,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```



Create Custom YOLO v3 Object Detector

This example shows how to create a custom YOLO v3 object detector by using a pretrained SqueezeNet as the base network.

Load a pretrained SqueezeNet network. The SqueezeNet network is a convolutional neural network that you can use as the base network for creating a YOLO v3 object detector.

```
net = imagePretrainedNetwork("squeezenet");
```

Inspect the architecture of the base network by using `analyzeNetwork` (Deep Learning Toolbox) function.

```
analyzeNetwork(net)
```

Specify the anchor boxes and the classes to use to train the YOLO v3 network.

```
aboxes = {[150,127;97,90;68,67];[38,42;41,29;31,23]};
classes = ["Car","Truck"];
```

Select two feature extraction layers in the base network to serve as the source for detection subnetwork.

```
layer = ["fire9-concat","fire8-concat"];
```

Create a custom YOLO v3 object detector by adding detection heads to the feature extraction layers of the base network. Specify the model name, classes, and the anchor boxes.

```
detector = yolov3ObjectDetector(net,classes,aboxes, ...
    ModelName="Custom YOLO v3",DetectionNetworkSource=layer);
```

Inspect the architecture of the YOLO v3 deep learning network by using `analyzeNetwork` (Deep Learning Toolbox) function.

```
analyzeNetwork(detector.Network)
```

Inspect the properties of the YOLO v3 object detector. You can now train the YOLO v3 object detector on a custom training dataset and perform object detection.

```
disp(detector)
```

```
  yolov3ObjectDetector with properties:

            Network: [1×1 dlnetwork]
        AnchorBoxes: {2×1 cell}
         ClassNames: [2×1 categorical]
          InputSize: [227 227 3]
    PredictedBoxType: 'axis-aligned'
          ModelName: 'Custom YOLO v3'
```

For information about how to train a YOLO v3 object detector, see the “Object Detection Using YOLO v3 Deep Learning” example.

Version History

Introduced in R2021a

R2024b: Code generation support for rotated rectangle bounding boxes

Starting in R2024b, YOLO v3 object detector with `PredictedBoxType` property set as `"rotated"` supports both CPU and GPU code generation.

R2024a: `DAGNetwork` objects are not recommended

Not recommended starting in R2024a

Starting in R2024a, `DAGNetwork` objects are not recommended. If you specify a base network `baseNet`, use a `dlnetwork` object instead.

There are no plans to remove support for `DAGNetwork` objects.

R2024a: Rotated rectangle bounding box support

The deep learning YOLO v3 object detector now supports rotated rectangle bounding boxes.

R2024b: Modify detection network source

Starting in R2024b, you can modify the detection network sources of a YOLO v3 object detection network when you perform transfer learning. You can specify the new detection network sources using the name-value argument `DetectionNetworkSource=layer`.

- Specify new detection network sources for the `tiny-yolov3-coco` and `darknet53-coco` pretrained YOLO v3 object detectors by using the syntax

  ```
  detector = yolov3ObjectDetector(name,classes,aboxes,DetectionNetworkSource=layer);
  ```

- Specify new detection network sources for a YOLO v3 object detection network by using the syntax

  ```
  detector = yolov3ObjectDetector(net,classes,aboxes,DetectionNetworkSource=layer);
  ```

For an example, the below code modifies the detection network sources of the pretrained `tiny-yolov3-coco` network to `leaky_relu_2` and `leaky_relu_5`.

```
name = "tiny-yolov3-coco";
layer = ["leaky_relu_2","leaky_relu_5"];
detector = yolov3ObjectDetector(name,classes,aboxes,DetectionNetworkSource=layer);
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Only the `detect` method of the `yolov3ObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, and `MaxSize` name-value pairs for `detect` are supported.
- To create a `yolov3ObjectDetector` object for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Only the `detect` method of the `yolov3ObjectDetector` is supported for code generation.
- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1x4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, and `MaxSize` name-value pairs are supported.
- The height, width, and channel of the input image must be fixed size.
- To create a `yolov3ObjectDetector` object for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only the `detect` method of the `yolov3ObjectDetector` is supported for GPU Arrays based processing.
- GPU Arrays is not supported for rotated rectangle bounding box inputs.

See Also

`trainYOLOv3ObjectDetector` | `yolov4ObjectDetector` | `trainYOLOv4ObjectDetector`

Topics

“Object Detection Using YOLO v3 Deep Learning”
“Object Detection Using Custom Training Loop”
“Getting Started with YOLO v3”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

detect

Detect objects using YOLO v3 object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
[bboxes,scores,labels,info] = detect(detector,I)

detectionResults = detect(detector,ds)

[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using a you only look once version 3 (YOLO v3) object detector, `detector`. The input size of the image must be greater than or equal to the network input size of the pretrained detector. The locations of objects detected are returned as a set of bounding boxes.

Note To use the pretrained YOLO v3 object detection networks trained on COCO dataset, you must install the Computer Vision Toolbox Model for YOLO v3 Object Detection. You can download and install the Computer Vision Toolbox Model for YOLO v3 Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

`[bboxes,scores] = detect(detector,I)` also returns the class-specific confidence scores for each bounding box.

`[bboxes,scores,labels] = detect(detector,I)` returns a categorical array of labels assigned to the bounding boxes. The labels for object classes are defined during training.

`[bboxes,scores,labels,info] = detect(detector,I)` also returns information about the class probabilities and objectness scores for each detection.

`detectionResults = detect(detector,ds)` detects objects within all the images returned by the `read` function of the input datastore `ds`.

`[___] = detect(___,roi)` detects objects within the rectangular search region `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments, in addition to any combination of arguments from previous syntaxes.

Examples

Detect Objects Using YOLO v3 Object Detector

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Read a test image and preprocess the test image by using the `preprocess` function.

```
img = imread('sherlock.jpg');
img = preprocess(detector,img);
```

Detect objects in the test image.

```
[bboxes,scores,labels] = detect(detector,img);
```

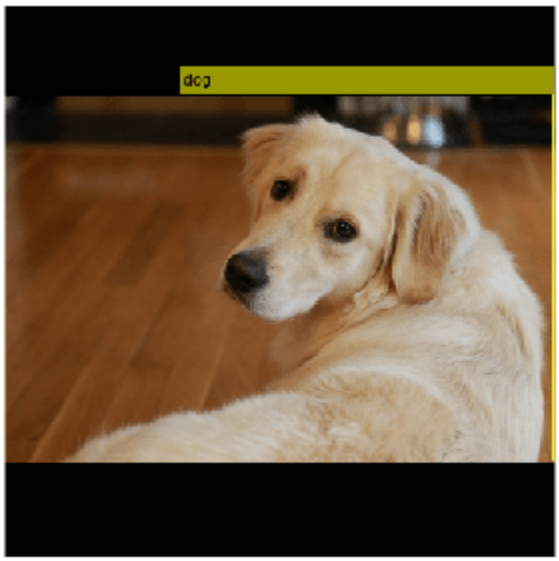
Display the detection results.

```
results = table(bboxes,labels,scores)
```

```
results=1×3 table
             bboxes              labels    scores
    _____________________________    ______    _______

    133     67    283    278      dog       0.51784
```

```
detectedImg = insertObjectAnnotation(img,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects from Images Stored in Image Datastore

Load a pretrained YOLOv3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Read the test data and store as an image datastore object.

```
location = fullfile(matlabroot,'toolbox','vision','visiondata','vehicles');
imds = imageDatastore(location);
```

Detect objects in the test dataset. Set the `Threshold` parameter value to 0.3 and `MiniBatchSize` parameter value to 32.

```
detectionResults = detect(detector,imds,'Threshold',0.3,'MiniBatchSize',32);
```

Read an image from the test dataset and extract the corresponding detection results.

```
num = 10;
I = readimage(imds,num);
bboxes = detectionResults.Boxes{num};
labels = detectionResults.Labels{num};
scores = detectionResults.Scores{num};
```

Perform non-maximal suppression to select strongest bounding boxes from the overlapping clusters. Set the `OverlapThreshold` parameter value to 0.2.

```
[bboxes,scores,labels] = selectStrongestBboxMulticlass(bboxes,scores,labels,'OverlapThreshold',0
```

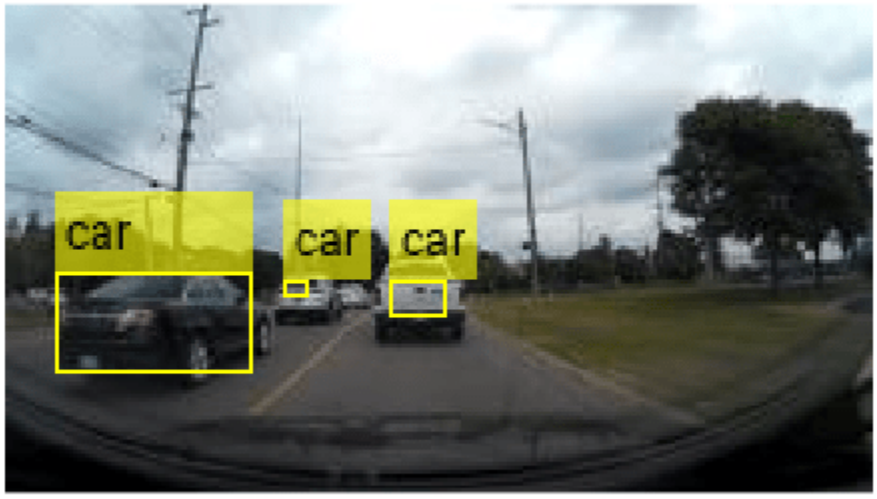
Display the detection results.

```
results = table(bboxes,labels,scores)
```

```
results=3×3 table
             bboxes             labels    scores
    ____________________________    ______    _______

     14     71     52     27      car       0.93352
     74     73      7      5      car       0.65369
    102     73     15     10      car       0.85313
```

```
detectedImg = insertObjectAnnotation(I,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects Within ROI

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Read a test image.

```
img = imread('highway.png');
```

Specify a region of interest (ROI) within the test image.

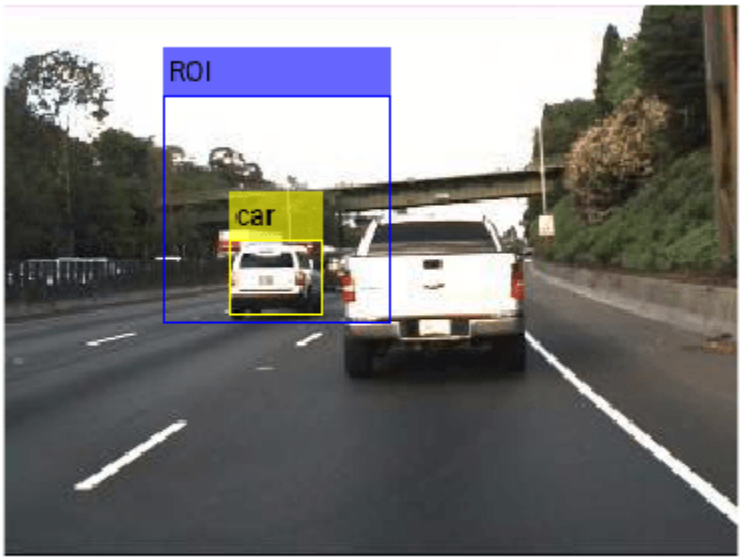
```
roiBox = [70 40 100 100];
```

Detect objects within the specified ROI.

```
[bboxes,scores,labels] = detect(detector,img,roiBox);
```

Display the ROI and the detection results.

```
img = insertObjectAnnotation(img,'Rectangle',roiBox,'ROI','AnnotationColor',"blue");
detectedImg = insertObjectAnnotation(img,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```



Input Arguments

`detector` — YOLO v3 object detector

`yolov3ObjectDetector` object

YOLO v3 object detector, specified as a `yolov3ObjectDetector` object.

`I` — Test images

numeric array

Test images, specified as a numeric array of size *H*-by-*W*-by*C* or *H*-by-*W*-by*C*-by-*T*. Images must be real, nonsparse, grayscale or RGB image.

- *H*: Height
- *W*: Width
- *C*: The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to `1`. For RGB color images, it must be equal to `3`.
- *T*: Number of test images in the array. The function computes the object detection results for each test image in the array.

The intensity range of the test image must be similar to the intensity range of the images used to train the detector. For example, if you train the detector on `uint8` images, rescale the test image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of the test image must be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the test image differs from the scale of the objects the detector was trained to identify.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Test images, specified as a `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing full filenames of the test images. The images in the datastore must be grayscale, or RGB images.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as an [*x y width height*] vector. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.25)`

`Threshold` — Detection threshold
`0.5` (default) | scalar in the range [0, 1]

Detection threshold, specified as a scalar in the range [0, 1]. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Returns the strongest bounding box per object. The method calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `selectStrongestBboxMulticlass` function is called as follows

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.5);
  ```

- `false` — Return all the detected bounding boxes. You can then write your own custom method to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of the form [*height width*]

Minimum region size, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest region containing the object.

By default, `MinSize` is 1-by-1.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of the form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest region containing the object.

By default, `MaxSize` is the height and width of the input image, `I`. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test image.

`MiniBatchSize` — Minimum batch size
128 (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of image. Images are grouped into minibatches and processed as a batch, which can improve computational efficiency at the cost of increased memory demand. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified one of the following:

- `"auto"` — Automatically apply a number of optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available when using a GPU only. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Disable all acceleration.

The default option is `"auto"`. If `"auto"` is specified, MATLAB applies a number of compatible optimizations. If you use the `"auto"` option, MATLAB does not ever generate a MEX function.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is only available for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is only available when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

`DetectionPreprocessing` — Option to preprocess test images
`"auto"` (default) | `"none"`

Option to preprocess the test images before performing object detection, specified as one of these values:

- `"auto"` — To preprocess the test image before performing object detection. The `detect` function calls the `preprocess` function that perform these operations:
  - Rescales the intensity values of the training images to the range [0, 1].
  - Resizes the training images to one of the nearest network input sizes and updates the bounding box coordinate values for accurate training. The function preserves the original aspect ratio of the training data.
- `"none"` — To perform object detection without preprocessing the test image. If you choose this option, the datatype of the test image must be either `single` or `double`.

Data Types: `char` | `string`

Output Arguments

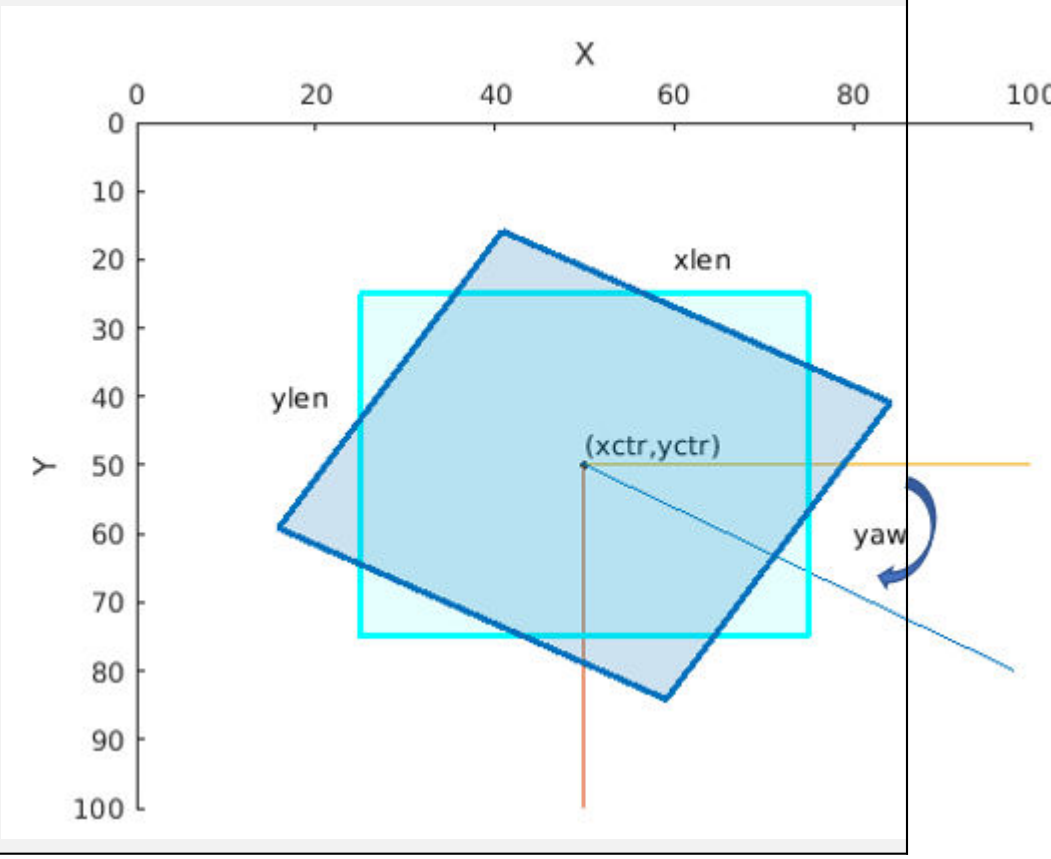
`bboxes` — Location of objects detected
M-by-4 matrix | *M*-by-5 matrix | *T*-by-1 cell array

Location of objects detected within the input image or images, returned as a

- *M*-by-4 matrix or an *M*-by-5 matrix if the input is a single test image.
- *T*-by-1 cell array if the input is an array of test images. *T* is the number of test images in the array. *M* is the number of bounding boxes in an image

The table describes the format of bounding boxes.

Bounding Box	Description
`rectangle`	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
`rotated-rectangle`	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

`scores` — Detection scores

M-by-1 numeric vector | *T*-by-1 cell array

Detection confidence scores for each bounding box, returned as one of these options:

- *M*-by-1 numeric vector — The input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — The input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

A higher score indicates higher confidence in the detection. The confidence score for each detection is a product of the corresponding objectness score and maximum class probability. The objectness score is the probability that the object in the bounding box belongs to a class in the image. The maximum class probability is the largest probability that a detected object in the bounding box belongs to a particular class.

`labels` — Labels for bounding boxes
M-by-1 categorical vector | *T*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector if the input is a single test image.
- *T*-by-1 cell array if the input is an array of test images. *T* is the number of test images in the array. Each cell in the array contains a *M*-by-1 categorical vector containing the names of the object classes.

M is the number of bounding boxes detected in an image.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column can contain rectangles or rotated rectangle bounding boxes of the form :

- rectangle — *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row specifies a rectangle as a 4-element vector of the form [*x*,*y*,*width*,*height*], where (*x*,*y*) specifies the upper-left corner location and (*width*, *height*) specifies the size in pixels
- rotated rectangle — *M*-by-5 matrices of *M* bounding boxes for the objects found in the image. Each row specifies a rotated rectangle as a 5-element vector of the form [*xctr*,*yctr*,*width*, *height*,*yaw*], where (*xctr*,*yctr*) specifies the center, (*width*,*height*) specifies the size, and *yaw* specifies the rotated angle.

`info` — Class probabilities and objectness scores
structure array

Class probabilities and objectness scores of the detections, returned as a structure array with these fields.

- `ClassProbabilities` — Class probabilities for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the class probabilities as an *M*-by-*N* numeric matrix. *M* is the number of bounding boxes and *N* is the number of classes. Each class probability is a numeric scalar, indicating the probability that the detected object in the bounding box belongs to a class in the image.
- `ObjectnessScores` — Objectness scores for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the objectness score for each bounding box as an *M*-by-1 numeric vector. *M* is the number of bounding boxes. Each objectness score is a numeric scalar, indicating the probability that the bounding box contains an object belonging to one of the classes in the image.

Version History

Introduced in R2021a

R2024b: Code generation support for rotated rectangle bounding boxes

Starting in R2024b, `bboxes` output argument returned as rotated rectangle bounding box predictions, supports both CPU and GPU code generation.

R2024a: Rotated rectangle bounding box support

The output argument `bboxes` now support rotated rectangle bounding box predictions.

R2024a: Option to return class probabilities and objectness scores

Specify the `info` output argument to return information about the class probability and objectness score for each detection.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MiniBatchSize`, and `MaxSize` name-value arguments are supported.
- The `MiniBatchSize` name-value argument is not supported for code generation when the image input `I` is an *H*-by-*W*-by-*C*-by-*B* numeric array of images. All images will processed in one batch.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, and `MaxSize` name-value arguments are supported.
- The height, width, and channel of the input image must be fixed size.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- GPU Arrays is not supported for rotated rectangle bounding box.

See Also

`predict` | `forward` | `yolov3ObjectDetector` | `imageDatastore` | `CombinedDatastore` | `TransformedDatastore` | `evaluateObjectDetection`

Topics
“Object Detection Using YOLO v3 Deep Learning”
“Getting Started with YOLO v3”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Datastores for Deep Learning” (Deep Learning Toolbox)

preprocess

Preprocess training and test images

Syntax

```
outputData = preprocess(detector,trainingData)
outputImg = preprocess(detector,img)
[___,scaleInfo] = preprocess(___)
```

Description

`outputData = preprocess(detector,trainingData)` preprocesses the training data `trainingData` before using it to train the YOLO v3 object detector. The training images and the corresponding bounding boxes are stored in the `trainingData`. The `preprocess` function performs these operations:

- Rescales the intensity values of the training images to the range [0, 1].
- Resizes the training images to one of the nearest network input sizes and updates the bounding box coordinate values for accurate training. The function preserves the original aspect ratio of the training data.

`outputImg = preprocess(detector,img)` preprocesses the test images `img` for object detection using a YOLO v3 object detector. The `preprocess` function performs these operations:

- Rescales the intensity values of the test images to the range [0, 1].
- Resizes the test images to one of the nearest network input sizes and preserves the original aspect ratio of each test image.

`[___,scaleInfo] = preprocess(___)` returns information on the scale factor applied for image resizing, in addition to any combination of arguments from previous syntaxes.

Examples

Preprocess Training Data Stored in Cell Array

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Load the training dataset into the workspace. The training data is a cell array that contains three training images and the corresponding bounding box values and class labels.

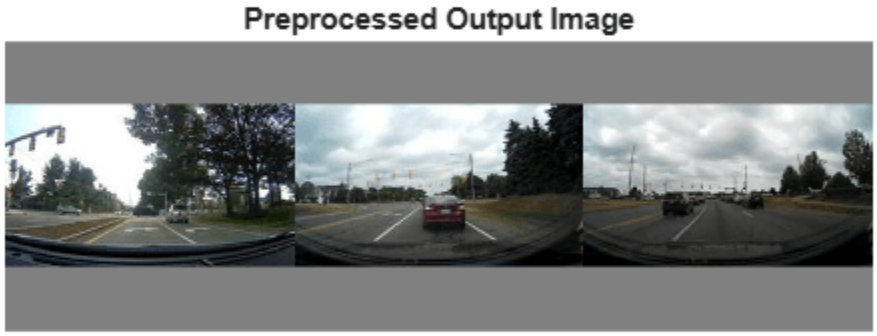
```
load('trainingData.mat','trainingData');
```

Resize the training images to the nearest network input size and rescale the intensity values by using the `preprocess` function.

```
[outputData,info] = preprocess(detector,trainingData);
```

Display the output images and the scale information used for resizing the images.

```
figure
montage({outputData{1,1},outputData{2,1},outputData{3,1}},Size=[1 3])
title("Preprocessed Output Image")
```




```
ScaleX = [info{1,1}.ScaleX;info{2,1}.ScaleX;info{3,1}.ScaleX];
ScaleY = [info{1,1}.ScaleY;info{2,1}.ScaleY;info{3,1}.ScaleY];
table(ScaleX,ScaleY)
```

```
ans=3×2 table
     ScaleX       ScaleY
    _________    _________

    0.0072115    0.0072115
    0.0072115    0.0072115
    0.0072115    0.0072115
```

Display the input and the preprocessed image size and bounding box values.

```
bboxIn = cell2table(trainingData,'VariableNames',{'Images','Bounding Boxes','Labels'})
```

```
bboxIn=3×3 table
         Images                Bounding Boxes             Labels
    ___________________    ____________________    ___________

    {224×399×3 uint8}    220    136     35     28    {'vehicle'}
    {224×399×3 uint8}    175    126     61     45    {'vehicle'}
    {224×399×3 uint8}    108    120     45     33    {'vehicle'}
```

```
bboxOut = cell2table(outputData,'VariableNames',{'Images','Bounding Boxes','Labels'})
```

```
bboxOut=3×3 table
          Images                Bounding Boxes           Labels
```

```
    ____________________    ____________________________    ___________

    {416×416×3 single}    229    232     36     29    {'vehicle'}
    {416×416×3 single}    182    222     64     46    {'vehicle'}
    {416×416×3 single}    112    215     47     35    {'vehicle'}
```

Preprocess Test Image

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Read a test image.

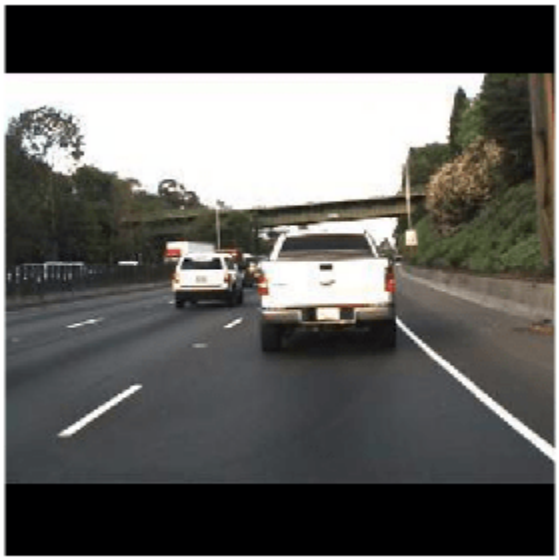
```
I = imread('highway.png');
```

Resize the test image to the network input size and rescale the intensity values by using the `preprocess` function.

```
[outputImg,scaleInfo] = preprocess(detector,I);
```

Display the output image and the scale information used for resizing the image.

```
figure
imshow(outputImg)
```



```
disp(scaleInfo)
```

```
                  ScaleX: 0.7692
                  ScaleY: 0.5769
    PreprocessedImageSize: [416 416 3]
```

Input Arguments

`detector` — YOLO v3 object detector

`yolov3ObjectDetector` object

YOLO v3 object detector, specified as a `yolov3ObjectDetector` object.

`trainingData` — Training data for YOLO v3 object detector

N-by-3 cell array

Training data for YOLO v3 object detector, specified as a *N*-by-3 cell array that contains the images, bounding boxes, and the class labels. Each row is of the form [*images bounding boxes labels*]. *N* is the number of output layers in the network. The bounding boxes must be stored as a *K*-by-4 matrix of the form [*x y width height*]. *K* is the number of object classes.

`img` — Test images

numeric array

Test images, specified as a numeric array of size *M*-by-*N*-by-*C* or *M*-by-*N*-by-*C*-by-*T*. *M* is the number of rows, *N* is the number of columns, and *C* is the number of color channels. The value of *C* is `1` for grayscale images and `3` for RGB color images. *T* is the number of test images in the array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`outputData` — Preprocessed training data

N-by-3 cell array

Preprocessed training data, returned as a *N*-by-3 cell array.

Data Types: `cell`

`outputImg` — Preprocessed test images

numeric array

Preprocessed test images, returned as a numeric array of size *P*-by-*Q*-by-*C* or *P*-by-*Q*-by-*C*-by-*T*. *P* and *Q* are the number of rows and columns, respectively, in the preprocessed image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`scaleInfo` — Information about scale factor for image resizing

structure

Information about the scale factor for resizing the input images, returned as a structure with fields `PreprocessedImageSize`, `ScaleX`, and `ScaleY`.

- `PreprocessedImageSize` — Size of the output resized image.
- `ScaleX` — Scale factor for resizing an image in the *X*-direction (along the rows).
- `ScaleY` — Scale factor for resizing an image in the *Y*-direction (along the columns).

Data Types: `struct`

Version History

Introduced in R2021a

See Also

`predict` | `forward` | `detect` | `yolov3ObjectDetector`

Topics

“Object Detection Using YOLO v3 Deep Learning”
“Getting Started with YOLO v3”

predict

Compute YOLO v3 deep learning network outputs for inference

Syntax

```
output = predict(detector,dlX)
```

Description

`output = predict(detector,dlX)` computes the YOLO v3 deep learning network outputs during inference, given the detector and the test data. Use this function to get predictions from the output layers of the YOLO v3 deep learning network during inference.

Note To run this function, you will require the Deep Learning Toolbox.

Examples

Compute YOLO v3 Network Predictions

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Load the test image for prediction.

```
I = imread('highway.png');
```

Preprocess the test image and convert the preprocessed image to a formatted dlarray object.

```
[Ip,info] = preprocess(detector,I);
Ip = im2single(Ip);
dlX = dlarray(Ip,'SSCB');
```

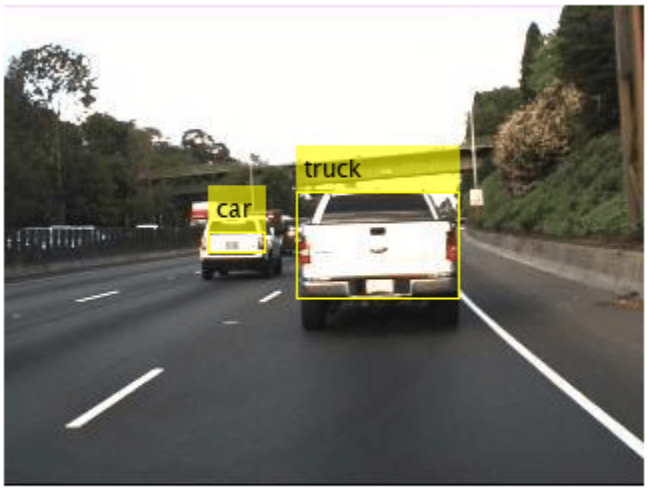
Compute predictions for the test image. The `predict` function returns the predictions for the feature maps from the output layers of the YOLO v3 deep learning network. The first column contains the confidence scores. Columns 2 to 5 contain the bounding box locations computed relative to the grid cell coordinates. The sixth column contains the class probabilities for each class used during training. The seventh and the eighth column contains the prior width and prior height of bounding boxes as computed by the network, respectively.

```
output = predict(detector,dlX)
```

```
output=2×8 cell array
    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    ←
    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    ←
```

You can then get the final detections by using the predictions for features with maximum objectness scores. The objectness score is the product of confidence score and class probability. To compute the

exact bounding box location, you must map the predicted bounding box values to box coordinates. Alternatively, you can use the `detect` function to directly get the detection results. The `detect` function internally calls the `predict` function to compute the feature maps.



Input Arguments

`detector` — YOLO v3 object detector

`yolov3ObjectDetector` object

YOLO v3 object detector, specified as a `yolov3ObjectDetector` object.

`dlX` — Test data

formatted `dlarray`

Test data, specified as a formatted `dlarray` object. The test data can contain one or more test images.

Output Arguments

`output` — Output predictions

N-by-8 cell array of formatted `dlarray`

Output predictions, returned as an *N*-by-8 cell array of formatted `dlarray` objects. *N* is the number of output layers in the YOLO v3 deep learning network. Each row in the cell array is of form [*conf* b_x b_y b_w b_h *prob* t_w t_h]. The function returns predictions as a formatted `dlarray` value..

Predictions	Description
conf	Confidence scores for each bounding box.

b_x	*X*-coordinate of the center of the predicted bounding box relative to the location of the grid cell.
b_y	*Y*-coordinate of the center of the predicted bounding box relative to the location of the grid cell.
b_w	Width of the predicted bounding box relative to the location of the grid cell.
b_h	Height of the predicted bounding boxes relative to the location of the grid cell.
prob	Class probabilities predicted for each feature in the output feature map.
t_w	Prior width of the bounding boxes as computed by the network.
t_h	Prior height of the bounding boxes as computed by the network.

Version History

Introduced in R2021a

See Also

`forward` | `yolov3ObjectDetector` | `preprocess` | `detect`

Topics

“Object Detection Using YOLO v3 Deep Learning”
“Getting Started with YOLO v3”

forward

Compute YOLO v3 deep learning network output for training

Syntax

```
features = forward(detector,dlX)
[features,activations] = forward(detector,dlX)
[features,activations,state] = forward(detector,dlX)
```

Description

`features = forward(detector,dlX)` computes the output features of the network during training given the input data `dlX`.

`[features,activations] = forward(detector,dlX)` also computes the activations of the network that you can use for modeling the gradient loss.

`[features,activations,state] = forward(detector,dlX)` also returns the updated network state.

Note To run this function, you will require the Deep Learning Toolbox.

Examples

Compute YOLO v3 Network Output During Forward Pass

Load a pretrained YOLO v3 object detector.

```
detector = yolov3ObjectDetector('tiny-yolov3-coco');
```

Read an image to use for training.

```
I = imread('highway.png');
```

Preprocess the training data and convert the preprocessed training data to a formatted `dlarray` object.

```
[Ip,info] = preprocess(detector,I);
Ip = im2single(Ip);
dlX = dlarray(Ip,'SSCB');
```

Compute the network outputs obtained during training. The `forward` function returns the activations from the output layers of the YOLO v3 deep learning network. The first column contains the confidence scores. Columns 2 to 5 contain the bounding box locations computed relative to the grid cell coordinates. The sixth column contains the class probabilities for each class used during training. The seventh and the eighth column contains the prior width and prior height of bounding boxes as computed by the network, respectively. The output features computed during the forward pass are used to model the gradient losses for the network.

```
[output,activations,state] = forward(detector,dlX)

output=2×8 cell array
    {13×13×3 single}    {13×13×3 single}    {13×13×3 single}    {13×13×3 single}    {13×13×3 sing
    {26×26×3 single}    {26×26×3 single}    {26×26×3 single}    {26×26×3 single}    {26×26×3 sing


activations=2×8 cell array
    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    {13×13×3×1 dlarray}    
    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    {26×26×3×1 dlarray}    


state=22×3 table
        Layer             Parameter              Value
    ______________    _________________    ________________

    "batch_norm_1"    "TrainedMean"        {1×1×16   dlarray}
    "batch_norm_1"    "TrainedVariance"    {1×1×16   dlarray}
    "batch_norm_2"    "TrainedMean"        {1×1×32   dlarray}
    "batch_norm_2"    "TrainedVariance"    {1×1×32   dlarray}
    "batch_norm_3"    "TrainedMean"        {1×1×64   dlarray}
    "batch_norm_3"    "TrainedVariance"    {1×1×64   dlarray}
    "batch_norm_4"    "TrainedMean"        {1×1×128  dlarray}
    "batch_norm_4"    "TrainedVariance"    {1×1×128  dlarray}
    "batch_norm_5"    "TrainedMean"        {1×1×256  dlarray}
    "batch_norm_5"    "TrainedVariance"    {1×1×256  dlarray}
    "batch_norm_6"    "TrainedMean"        {1×1×512  dlarray}
    "batch_norm_6"    "TrainedVariance"    {1×1×512  dlarray}
    "batch_norm_7"    "TrainedMean"        {1×1×1024 dlarray}
    "batch_norm_7"    "TrainedVariance"    {1×1×1024 dlarray}
    "batch_norm_8"    "TrainedMean"        {1×1×256  dlarray}
    "batch_norm_8"    "TrainedVariance"    {1×1×256  dlarray}
      ⋮
```

Input Arguments

`detector` — YOLO v3 object detector

`yolov3ObjectDetector` object

YOLO v3 object detector, specified as a `yolov3ObjectDetector` object.

`dlX` — Training data

formatted `dlarray`

Training data, specified as a formatted `dlarray` object.

Output Arguments

`features` — Output features in box coordinates

N-by-8 cell array

Output features in box coordinates, returned as an *N*-by-8 cell array. *N* is the number of output layers in the YOLO v3 deep learning network.

`activations` — Activations of network
N-by-8 cell array of formatted `dlarray`

Activations of the network, returned as an *N*-by-8 cell array of formatted `dlarray` objects. *N* is the number of output layers in the YOLO v3 deep learning network.

Each row in the cell array is of form [*conf* b_x b_y b_w b_h *prob* t_w t_h]. The function returns each activation as a formatted `dlarray` value.

Activations	Description
conf	Estimated confidence scores for each bounding box.
b_x	Estimated *X*-coordinate value for the center of the bounding box relative to the location of the grid cell.
b_y	Estimated *Y*-coordinate value for the center of the bounding box relative to the location of the grid cell.
b_w	Estimated width of the bounding box relative to the location of the grid cell.
b_h	Estimated height of the bounding box relative to the location of the grid cell.
prob	Class probabilities estimated for each feature in the output feature map.
t_w	Prior width of the bounding box as estimated by the network.
t_h	Prior height of the bounding box as estimated by the network.

`state` — Updated network state
table

Updated network state, returned as a table. The network state is a table with three columns:

- `Layer` – Layer name, returned as a string scalar.
- `Parameter` – Parameter name, returned as a string scalar.
- `Value` – Value of parameter, returned as a numeric array object.

The network state contains information remembered by the network between iterations.

Version History

Introduced in R2021a

See Also

`yolov3ObjectDetector` | `preprocess` | `detect`

Topics
“Object Detection Using YOLO v3 Deep Learning”

“Getting Started with YOLO v3”

yolov4ObjectDetector

Detect objects using YOLO v4 object detector

Description

The `yolov4ObjectDetector` object creates a you only look once version 4 (YOLO v4) one-stage object detector for detecting objects in an image. Using this object, you can:

- Create a pretrained YOLO v4 object detector by using YOLO v4 deep learning networks trained on COCO dataset.
- Create a custom YOLO v4 object detector by using any pretrained or untrained YOLO v4 deep learning network.

Creation

Syntax

```
detector = yolov4ObjectDetector(name)

detector = yolov4ObjectDetector(name,classes,aboxes)
detector = yolov4ObjectDetector(net,classes,aboxes)
detector = yolov4ObjectDetector(baseNet,classes,
aboxes,DetectionNetworkSource=layer)
detector = yolov4ObjectDetector( ___ ,Name=Value)
```

Description

Pretrained YOLO v4 Object Detector

`detector = yolov4ObjectDetector(name)` creates a pretrained YOLO v4 object detector by using YOLO v4 deep learning networks trained on a COCO dataset.

Custom YOLO v4 Object Detector

`detector = yolov4ObjectDetector(name,classes,aboxes)` creates a pretrained YOLO v4 object detector and configures it to perform transfer learning using a specified set of object classes and anchor boxes. For optimal results, you must train the detector on new training images before performing detection. Use the `trainYOLOv4ObjectDetector` function for training the detector.

`detector = yolov4ObjectDetector(net,classes,aboxes)` creates an object detector by using the deep learning network `net`.

If `net` is a pretrained YOLO v4 deep learning network, the function creates a pretrained YOLO v4 object detector. The `classes` and `aboxes` are values used for training the network.

If `net` is an untrained YOLO v4 deep learning network, the function creates a YOLO v4 object detector to use for training and inference. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v4 network.

Use the `trainYOLOv4ObjectDetector` function to train the network before performing object detection.

`detector = yolov4ObjectDetector(baseNet,classes, aboxes,DetectionNetworkSource=layer)` creates a YOLO v4 object detector by adding detection heads to a base network, `baseNet`.

The function adds detection heads to the specified feature extraction layers `layer` in the base network. To specify the names of the feature extraction layers, use the name-value argument `DetectionNetworkSource=layer`.

If `baseNet` is a pretrained deep learning network, the function creates a YOLO v4 object detector and configures it to perform transfer learning with the specified object classes and anchor boxes.

If `baseNet` is an untrained deep learning network, the function creates a YOLO v4 object detector and configures it for object detection. `classes` and `aboxes` specify the object classes and the anchor boxes, respectively, for training the YOLO v4 network.

You must train the detector on a training dataset before performing object detection. Use the `trainYOLOv4ObjectDetector` function for training the detector.

`detector = yolov4ObjectDetector( ___ ,Name=Value)` specifies one or more options using name-value arguments in addition to any combination of input arguments from previous syntaxes. Use this syntax to:

- Modify the detection network sources in a YOLO v4 object detection network and train the network with different numbers of object classes, anchor boxes, or both. Specify the new detection network sources using the `DetectionNetworkSource=layer` name-value argument.
- Set the `InputSize on page 2-0` and `ModelName on page 2-0` properties of the object detector. For example, `InputSize=[224 224 3]` sets the size of the images used for training to `[224 224 3]`.

Note To use the pretrained YOLO v4 object detection networks trained on COCO dataset, you must install the Computer Vision Toolbox Model for YOLO v4 Object Detection. You can download and install the Computer Vision Toolbox Model for YOLO v4 Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

Input Arguments

`name` — Name of pretrained YOLO v4 deep learning network

`"csp-darknet53-coco"` | `"tiny-yolov4-coco"`

Name of the pretrained YOLO v4 deep learning network, specified as one of these:

- `"csp-darknet53-coco"` — A pretrained YOLO v4 deep learning network created using CSP-DarkNet-53 as the base network and trained on COCO dataset.
- `"tiny-yolov4-coco"` — A pretrained YOLO v4 deep learning network created using a small base network and trained on COCO dataset.

Data Types: `char` | `string`

`classes` — Names of object classes
string vector | cell array of character vectors | categorical vector

Names of object classes for training the detector, specified as a string vector, cell array of character vectors, or categorical vector. This argument sets the `ClassNames` property of the `yolov4ObjectDetector` object.

Data Types: `char` | `string` | `categorical`

`aboxes` — Anchor boxes
cell array

Anchor boxes for training the detector, specified as an *N*-by-1 column vector cell array or a 1-by-*N* row vector cell array. *N* is the number of output layers in the YOLO v4 deep learning network. Each cell contains an *M*-by-2 matrix, where *M* is the number of anchor boxes in that layer. Each cell can contain a different number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of an anchor box in the form [*height width*].

The first element in the cell array specifies the anchor boxes to associate with the first output layer, the second element in the cell array specifies the anchor boxes to associate with the second output layer, and so on. For accurate detection results, specify large anchor boxes for the first output layer and small anchor boxes for the last output layer. That is, the anchor box sizes must decrease for each output layer in the order in which the layers appear in the YOLO v4 deep learning network.

This argument sets the `AnchorBoxes` property of the `yolov4ObjectDetector` object.

Data Types: `cell`

`net` — YOLO v4 deep learning network
`dlnetwork` object

YOLO v4 deep learning network, specified as a `dlnetwork` object. The input network can be either an untrained or a pretrained deep learning network. The input network must have an image input layer.

`baseNet` — Base network
`dlnetwork` object

Base network for creating the YOLO v4 deep learning network, specified as a `dlnetwork` object. The network can be either an untrained or a pretrained deep learning network. The input network must have an image input layer.

`layer` — Names of feature extraction layers
cell array of character vectors | string array

Names of the feature extraction layers in the base network, specified as a cell array of character vectors, or a string array. The function creates a YOLO v4 network by adding detection head layers to the output of the feature extraction layers in the base network.

In the pre-trained yolo v4 base network, you must connect the detection head layers to feature extraction layers where the input image feature map changes spatial dimension by downsampling factors of 4, 8, 16, and 32.

- Use the `analyzeNetwork` function to display the pre-trained YOLO v4 network architecture to obtain the information about the spatial dimensions of the feature maps.

- Use this table to know about the default feature extraction layers and choose alternate feature extraction layers supported by the pre-trained YOLO v4 networks. Choose feature extraction layers using empirical evaluation based on the detection accuracy, training speed, and object size to detect.
  - To achieve higher detection accuracy, choose feature extraction layers with both higher and lower downsampling factors. Layers with lower downsampling factor capture small objects and layers with higher downsampling factor ensure a thorough feature refinement.
  - To achieve higher training and inference speeds, choose feature extraction layers with lower downsampling factor.
  - The number of detection network sources specified must be equal to the number of anchor boxes specified at the input.

Backbone	**Number of Detection Heads**	**Default Feature Extraction Layers**		**Examples of Alternate Feature Extraction Layers**		**Remarks**
		Layer	**Downsampling Factor**	**Layer**	**Downsampling Factor**	
`tiny-yolov4-coco`	2	`leaky_28` `leaky_25`	32 16	`leaky_17` `leaky_9`	8 4	Select any two feature extraction layers with different downsampling factors preceding the `leaky_28` layer to use as detection network sources. For example, specify the new detection network sources as `layer = {'leaky_17','leaky_9'}`

`csp-darknet53-coco`	3	`mish_106`
		`mish_87`
		`mish_56`
			32
			16
			8
				`mish_88`
				`mish_102`
					32	Select any three feature extraction layers with different

				`mish_68` `mish_57`	16	

				`mish_37` `mish_26`	8	downsampling factors preceding the `mish_106` layer to use as detection network sources. For example, specify the new detection network sources as `layer = {'mish_57','mish_26','mish_13'}`

				`mish_25`
				`mish_18`
				`mish_13`
					4	

Data Types: `char` | `string` | `cell`

Properties

`Network` — YOLO v4 deep learning network

`dlnetwork` object

YOLO v4 deep learning network to use for object detection, stored as a `dlnetwork` object.

`ClassNames` — Names of object classes

categorical vector

Names of object classes to detect, stored as a categorical vector. You can set this property by using the input argument `classes`.

`AnchorBoxes` — Set of anchor boxes

N-by-1 cell array

This property is read-only.

Set of anchor boxes, stored as a *N*-by-1 cell array. *N* is the number of output layers in the YOLO v4 deep learning network for which the anchor boxes are defined. Each element in the cell is a *M*-by-2 matrix. *M* denotes the number of anchor boxes. Each cell can contain a different number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of the anchor box in the form of [*height width*]. The first element in the cell array specifies the anchor boxes for the first output layer, the second element in the cell array specifies the anchor boxes for the second output layer, and so on.

You can set this property by using the input argument `aboxes`.

`InputSize` — Image size used for training

vector

This property is read-only.

Image size used for training, stored as a vector of form [*height width*] or [*height width channels*]. To set this property, specify it at object creation. The size of the training images must be a multiple of 32.

For example, `detector = yolov4ObjectDetector(net,classes,aboxes,InputSize=[224 224 3])`.

`ModelName` — Name for object detector
`' '` (default) | character vector | string scalar

Name for the object detector, stored as a character vector or string scalar. To set this property, specify it at object creation.

For example, `yolov4ObjectDetector(net,classes,aboxes,ModelName="customDetector")` sets the name for the object detector to `"customDetector"`.

`PredictedBoxType` — Bounding box format for object detector
`"axis-aligned"` (default) | `"rotated"`

This property is read-only.

Bounding box format for an object detector, stored as `"axis-aligned"` or `"rotated"`. When the `PredictedBoxType` is `"axis-aligned"`, the object detector will train and perform inference on only axis-aligned bounding boxes. If it is set to `"rotated"`, the object detector will train and perform inference on only rotated bounding boxes. Set this property when you create the object.

Object Functions

detect Detect objects using YOLO v4 object detector

Examples

Create Pretrained YOLO v4 Object Detector

Specify the name of a pretrained YOLO v4 deep learning network.

```
name = "tiny-yolov4-coco";
```

Create YOLO v4 object detector by using the pretrained YOLO v4 network.

```
detector = yolov4ObjectDetector(name);
```

Display and inspect the properties of the YOLO v4 object detector.

```
disp(detector)
```

```
  yolov4ObjectDetector with properties:

       Network: [1×1 dlnetwork]
    AnchorBoxes: {2×1 cell}
     ClassNames: {80×1 cell}
      InputSize: [416 416 3]
      ModelName: 'tiny-yolov4-coco'
```

Use `analyzeNetwork` to display the YOLO v4 network architecture and get information about the network layers.

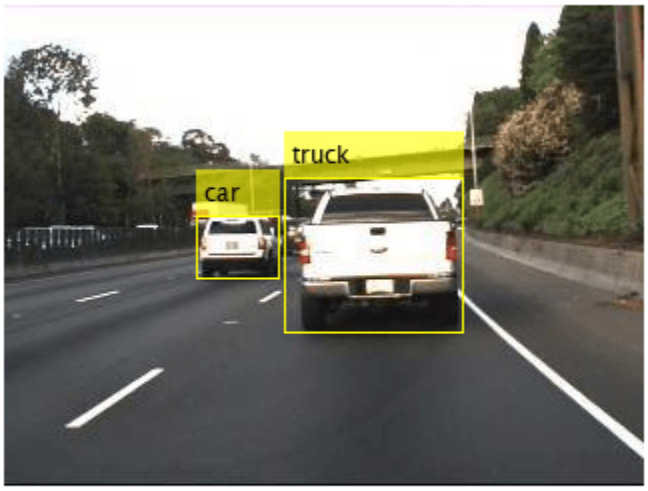
```
analyzeNetwork(detector.Network)
```

Detect objects in an unknown image by using the pretrained YOLO v4 object detector.

```
img = imread("highway.png");
[bboxes,scores,labels] = detect(detector,img);
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(img,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Create Custom YOLO v4 Object Detector

This example shows how to create a YOLO v4 object detection network based on a pretrained ResNet-50 convolutional neural network.

Load a pretrained deep learning network to use as the base network. This example uses ResNet-50 pretrained network as the base network. For information about other available pretrained networks, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

```
basenet = imagePretrainedNetwork("resnet50");
```

Use `analyzeNetwork` to display the architecture of the base network.

```
analyzeNetwork(basenet)
```

The first layer in the base network is the image input layer. Inspect the property of the image input layer in the base network.

```
basenet.Layers(1)
```

```
ans =
  ImageInputLayer with properties:

                      Name: 'input_1'
                 InputSize: [224 224 3]
        SplitComplexInputs: 0

   Hyperparameters
          DataAugmentation: 'none'
             Normalization: 'zerocenter'
    NormalizationDimension: 'auto'
                      Mean: [224×224×3 single]
```

To create a YOLO v4 deep learning network you must set the `Normalization` property of the `ImageInputLayer` in the base network to "`none`". Define an image input layer with the `Normalization` property set as "`none`" and other property values the same as those of the base network.

```
imageSize = basenet.Layers(1).InputSize;
layerName = basenet.Layers(1).Name;
newInputLayer = imageInputLayer(imageSize,Normalization="none",Name=layerName);
```

Replace the image input layer in the base network with the new input layer.

```
dlnet = replaceLayer(basenet,layerName,newInputLayer);
```

Specify the names of the feature extraction layers in the base network to use as the detection heads.

```
featureExtractionLayers = ["activation_22_relu","activation_40_relu"];
```

Specify the class names and anchor boxes to use for training the YOLO v4 deep learning network created using ResNet-50 as the base network.

```
classes = ["car","person"];
anchorBoxes = {[122,177;223,84;80,94];...
               [111,38;33,47;37,18]};
```

Create a YOLO v4 object detector by using the specified base network and the detection heads.

```
detector = yolov4ObjectDetector(dlnet,classes,anchorBoxes, ...
    DetectionNetworkSource=featureExtractionLayers);
```

Display and inspect the properties of the YOLO v4 object detector.

```
disp(detector)
```

```
  yolov4ObjectDetector with properties:

            Network: [1×1 dlnetwork]
        AnchorBoxes: {2×1 cell}
         ClassNames: {2×1 cell}
          InputSize: [224 224 3]
```

```
    PredictedBoxType: 'axis-aligned'
           ModelName: ''
```

Use `analyzeNetwork` to display the YOLO v4 network architecture and get information about the network layers.

```
analyzeNetwork(detector.Network)
```

Detect People Using YOLO v4 Object Detector

This example shows how to detect people using a pretrained YOLO v4 object detector.

Load the pretrained YOLO v4 detector that can detect 80 common object classes.

```
detector = yolov4ObjectDetector();
```

Read image to process.

```
I = imread("visionteam.jpg");
```

Run object detector.

```
[bboxes, scores, labels] = detect(detector, I, Threshold=0.4);
```

Select detections for the person class.

```
personBoxes = bboxes(labels=="person", :);
```

Display results.

```
detectedImg = insertObjectAnnotation(I, "Rectangle", personBoxes, "person");
figure
imshow(detectedImg)
```



Version History

Introduced in R2022a

R2024b: Modify detection network source

Starting in R2024b, you can modify the detection network sources of a YOLO v4 object detection network when you perform transfer learning. You can specify the new detection network sources using the name-value argument `DetectionNetworkSource=layer`.

- Specify new detection network sources for the `tiny-yolov4-coco` and `csp-darknet53-coco` pretrained YOLO v4 object detectors by using the syntax

  ```
  detector = yolov4ObjectDetector(name,classes,aboxes,DetectionNetworkSource=layer);
  ```

- Specify new detection network sources for a YOLO v4 object detection network by using the syntax

  ```
  detector = yolov4ObjectDetector(net,classes,aboxes,DetectionNetworkSource=layer);
  ```

For an example, the below code modifies the detection network sources of the pretrained `tiny-yolov4-coco` network to `leaky_23` and `leaky_15`.

```
name = "tiny-yolov4-coco";
layer = ["leaky_23","leaky_15"];
detector = yolov4ObjectDetector(name,classes,aboxes,DetectionNetworkSource=layer);
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value pairs for `detect` are supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value pairs for `detect` are supported.

For information about how to create a `yolov4ObjectDetector` object for code generation, see "Load Pretrained Networks for Code Generation" (MATLAB Coder).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- GPU Arrays is not supported for rotated rectangle bounding box inputs.

See Also

`detect` | `trainYOLOv4ObjectDetector` | `yoloxObjectDetector`

Topics

“Object Detection Using YOLO v4 Deep Learning”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Getting Started with YOLO v4”
“Get Started with Object Detection Using Deep Learning”
“Getting Started with YOLOX for Object Detection”
“Choose an Object Detector”

detect

Detect objects using YOLO v4 object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
[bboxes,scores,labels,info] = detect(detector,I)

detectionResults = detect(detector,ds)

[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using a you only look once version 4 (YOLO v4) object detector, `detector`. The `detect` function automatically resizes and rescales the input image to match that of the images used for training the detector. The locations of objects detected in the input image are returned as a set of bounding boxes.

Note To use the pretrained YOLO v4 object detection networks trained on COCO dataset, you must install the Computer Vision Toolbox Model for YOLO v4 Object Detection. You can download and install the Computer Vision Toolbox Model for YOLO v4 Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

`[bboxes,scores] = detect(detector,I)` also returns the class-specific confidence scores for each bounding box.

`[bboxes,scores,labels] = detect(detector,I)` returns a categorical array of labels assigned to the bounding boxes. The labels for object classes are defined during training.

`[bboxes,scores,labels,info] = detect(detector,I)` also returns information about the class probabilities and objectness scores for each detection.

`detectionResults = detect(detector,ds)` detects objects within all the images returned by the `read` function of the input datastore `ds`.

`[___] = detect(___,roi)` detects objects within the rectangular search region `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments, in addition to any combination of arguments from previous syntaxes..

Examples

Detect Objects Using Pretrained YOLO v4 Object Detector

Specify the name of a pretrained YOLO v4 deep learning network.

```
name = 'tiny-yolov4-coco';
```

Create YOLO v4 object detector by using the pretrained YOLO v4 network.

```
detector = yolov4ObjectDetector(name);
```

Detect objects in an unknown image by using the pretrained YOLO v4 object detector.

```
img = imread('sherlock.jpg');
img = im2single(imresize(img,0.5));
[bboxes,scores,labels] = detect(detector,img,Threshold=0.4)
```

```
bboxes = 1×4 single row vector

    80.9433    31.6083    398.4628    288.3917

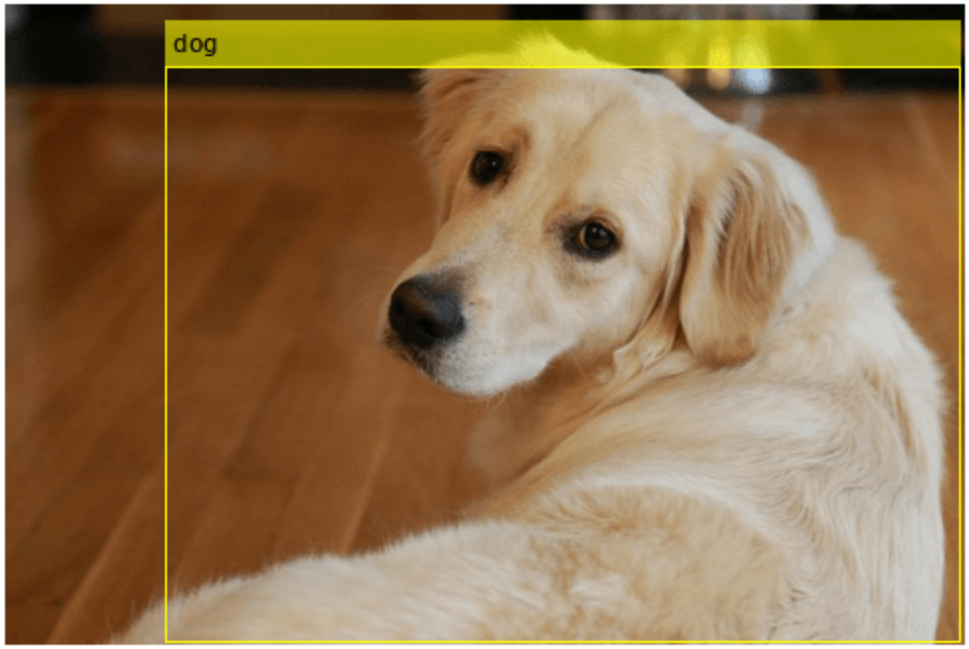

scores = single

0.4281

labels = categorical
     dog
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(img,'Rectangle',bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects in Image Datastore by Using YOLO v4 Detector

Load a pretrained YOLO v4 object detector.

```
detector = yolov4ObjectDetector("csp-darknet53-coco");
```

Read the test data and store as an image datastore object.

```
location = fullfile(matlabroot,'toolbox','vision','visiondata','vehicles');
imds = imageDatastore(location);
```

Detect objects in the test dataset. Set the `Threshold` parameter value to 0.4 and `MiniBatchSize` parameter value to 32.

```
detectionResults = detect(detector,imds,Threshold=0.4,MiniBatchSize=32);
```

Read an image from the test dataset and extract the corresponding detection results.

```
num = 20;
I = readimage(imds,num);
bboxes = detectionResults.Boxes{num};
labels = detectionResults.Labels{num};
scores = detectionResults.Scores{num};
```

Perform non-maximal suppression to select strongest bounding boxes from the overlapping clusters. Set the `OverlapThreshold` parameter value to 0.5.

```
[bboxes,scores,labels] = selectStrongestBboxMulticlass(bboxes,...
                                scores,labels,OverlapThreshold=0.5);
```

Display the detection results.

```
results = table(bboxes,labels,scores)
```

```
results=2×3 table
    17.8178,69.9661,23.4595,11.3807    car    0.9027
    75.2063,66.0107,26.1342,23.5411    car    0.5830
```

```
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects Within ROI by Using YOLO v4 Detector

Load a pretrained YOLO v4 object detector.

```
detector = yolov4ObjectDetector("csp-darknet53-coco");
```

Read a test image.

```
img = imread("stopsign.jpg");
```

Specify a region of interest (ROI) within the test image.

```
roiBox = [250 60 500 300];
```

Detect objects within the specified ROI.

```
[bboxes,scores,labels] = detect(detector,img,roiBox);
```

Display the ROI and the detection results.

```
img = insertObjectAnnotation(img,"Rectangle",roiBox,"ROI",AnnotationColor="blue");
detectedImg = insertObjectAnnotation(img,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Input Arguments

`detector` — YOLO v4 object detector

`yolov4ObjectDetector` object

YOLO v4 object detector, specified as a `yolov4ObjectDetector` object.

`I` — Test images

numeric array

Test images, specified as a numeric array of size *H*-by-*W*-by*C* or *H*-by-*W*-by*C*-by-*T*. Images must be real, nonsparse, grayscale or RGB image.

- *H*: Height
- *W*: Width
- *C*: The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to 1. For RGB color images, it must be equal to 3.
- *T*: Number of test images in the array. The function computes the object detection results for each test image in the array.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Test images, specified as a `ImageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing full filenames of the test images. The images in the datastore must be grayscale, or RGB images.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as an [*x y width height*] vector. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.25)`

`Threshold` — Detection threshold
`0.5` (default) | scalar in the range [0, 1]

Detection threshold, specified as a scalar in the range [0, 1]. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Returns the strongest bounding box per object. The method calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `selectStrongestBboxMulticlass` function is called as follows

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.5);
  ```

- `false` — Return all the detected bounding boxes. You can then write your own custom method to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of the form [*height width*]

Minimum region size, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest region containing the object.

By default, `MinSize` is 1-by-1.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of the form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest region containing the object.

By default, `MaxSize` is set to the height and width of the input image, `I`. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test image.

`MiniBatchSize` — Minimum batch size
128 (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of image. Images are grouped into minibatches and processed as a batch, which can improve computational efficiency at the cost of increased memory demand. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified one of the following:

- `"auto"` — Automatically apply a number of optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available when using a GPU only. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Disable all acceleration.

The default option is `"auto"`. If `"auto"` is specified, MATLAB applies a number of compatible optimizations. If you use the `"auto"` option, MATLAB does not ever generate a MEX function.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is only available for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is only available when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

bboxes — Location of objects detected

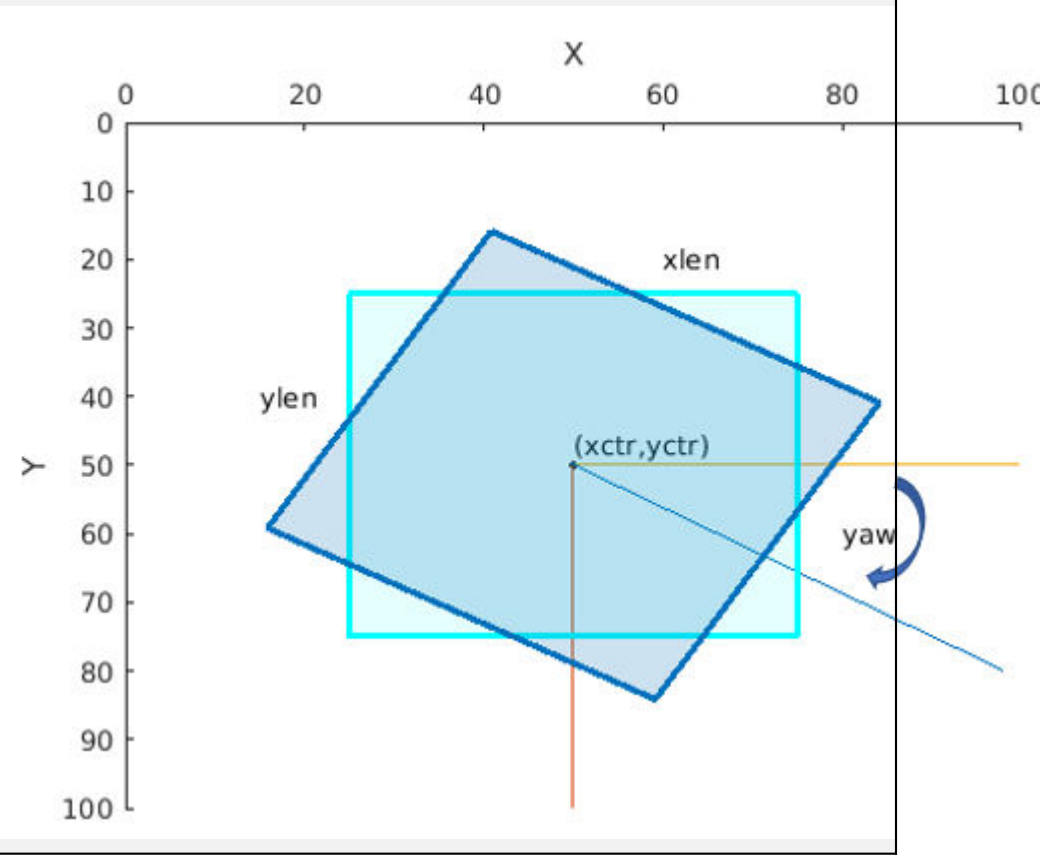
M-by-4 matrix | *M*-by-5 matrix | *T*-by-1 cell array

Location of objects detected within the input image or images, returned as a

- *M*-by-4 matrix or an *M*-by-5 matrix if the input is a single test image.
- *T*-by-1 cell array if the input is an array of test images. *T* is the number of test images in the array. *M* is the number of bounding boxes in an image

The table describes the format of bounding boxes.

Bounding Box	Description
`rectangle`	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
`rotated-rectangle`	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

`scores` — Detection scores

M-by-1 numeric vector | *T*-by-1 cell array

Detection confidence scores for each bounding box, returned as one of these options:

- *M*-by-1 numeric vector — The input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — The input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

A higher score indicates higher confidence in the detection. The confidence score for each detection is a product of the corresponding objectness score and maximum class probability. The objectness score is the probability that the object in the bounding box belongs to a class in the image. The maximum class probability is the largest probability that a detected object in the bounding box belongs to a particular class.

`labels` — Labels for bounding boxes
M-by-1 categorical vector | *T*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector if the input is a single test image.
- *T*-by-1 cell array if the input is an array of test images. *T* is the number of test images in the array. Each cell in the array contains a *M*-by-1 categorical vector containing the names of the object classes.

M is the number of bounding boxes detected in an image.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column can contain rectangles or rotated rectangle bounding boxes of the form :

- rectangle — *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row specifies a rectangle as a 4-element vector of the form [*x*,*y*,*width*,*height*], where (*x*,*y*) specifies the upper-left corner location and (*width*, *height*) specifies the size in pixels
- rotated rectangle — *M*-by-5 matrices of *M* bounding boxes for the objects found in the image. Each row specifies a rotated rectangle as a 5-element vector of the form [*xctr*,*yctr*,*width*, *height*,*yaw*], where (*xctr*,*yctr*) specifies the center, (*width*,*height*) specifies the size, and *yaw* specifies the rotated angle.

`info` — Class probabilities and objectness scores
structure array

Class probabilities and objectness scores of the detections, returned as a structure array with these fields.

- `ClassProbabilities` — Class probabilities for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the class probabilities as an *M*-by-*N* numeric matrix. *M* is the number of bounding boxes and *N* is the number of classes. Each class probability is a numeric scalar, indicating the probability that the detected object in the bounding box belongs to a class in the image.
- `ObjectnessScores` — Objectness scores for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the objectness score for each bounding box as an *M*-by-1 numeric vector. *M* is the number of bounding boxes. Each objectness score is a numeric scalar, indicating the probability that the bounding box contains an object belonging to one of the classes in the image.

Version History

Introduced in R2022a

R2024a: Option to return class probabilities and objectness scores

Specify the `info` output argument to return information about the class probability and objectness score for each detection.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported.
- The `MiniBatchSize` name-value argument is not supported for code generation when the image input `I` is an *H*-by-*W*-by-*C*-by-*B* numeric array of images. All images will processed in one batch.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments for are supported.

For information about how to create a `yolov4ObjectDetector` object for code generation, see "Load Pretrained Networks for Code Generation" (MATLAB Coder).

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- GPU Arrays is not supported for rotated rectangle bounding box inputs.

See Also

`forward` | `imageDatastore` | `CombinedDatastore` | `TransformedDatastore` | `yolov4ObjectDetector` | `trainYOLOv4ObjectDetector` | `evaluateObjectDetection`

Topics

"Object Detection Using YOLO v4 Deep Learning"
"Object Detection in Large Satellite Imagery Using Deep Learning"
"Getting Started with YOLO v4"
"Get Started with Object Detection Using Deep Learning"
"Choose an Object Detector"
"Datastores for Deep Learning" (Deep Learning Toolbox)

yoloxObjectDetector

Detect objects using YOLOX object detector

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `yoloxObjectDetector` object creates a You Only Look Once X (YOLOX) object detector for detecting objects in an image of arbitrary size. YOLOX is a type of YOLO detector that offers one-stage architecture and real-time performance without the need for predefined anchor boxes. Using this object, you can:

- Create a pretrained YOLOX object detector by using YOLOX deep learning networks trained on the COCO data set.
- Create a custom YOLOX object detector by using a pretrained or untrained YOLOX deep learning network.

Note This functionality requires Deep Learning Toolbox.

Creation

Syntax

```
detector = yoloxObjectDetector
detector = yoloxObjectDetector(name)
detector = yoloxObjectDetector(name,classes)
detector = yoloxObjectDetector( ___ ,Name=Value)
```

Description

`detector = yoloxObjectDetector` creates a YOLOX object detector trained to detect 80 object classes from the COCO data set using a CSP-Darknet-53 network.

`detector = yoloxObjectDetector(name)` creates a pretrained YOLOX object detector by using YOLOX deep learning networks trained on the COCO data set.

`detector = yoloxObjectDetector(name,classes)` creates a pretrained YOLOX object detector and configures it to perform transfer learning using a specified set of object classes. For optimal results, you must train the detector on new training images before performing detection.

Use the `trainYOLOXObjectDetector` function to train the detector before performing object detection.

`detector = yoloxObjectDetector( ___ ,Name=Value)` sets the `ModelName` and `InputSize` properties of the object, and specifies the `NormalizationStatistics` of the detector, using name-value arguments in addition to any combination of input arguments from previous syntaxes. For

example, `ModelName="customDetector"` sets the name of the object detector to `"customDetector"`.

Input Arguments

name — Name of pretrained YOLOX deep learning network

`"nano-coco"` | `"tiny-coco"` | `"small-coco"` | `"medium-coco"` | `"large-coco"`

Name of the pretrained YOLOX deep learning network, created using a CSP-DarkNet-53 as the base network and trained on the COCO data set [1], specified as one of these values.

- `"nano-coco"` — Pretrained YOLOX-nano deep learning network with fewest filters and fewest convolutional layers. Use YOLOX-nano when you have few available computational resources and require the fastest performance speed and real-time inference.
- `"tiny-coco"` — Pretrained YOLOX-tiny deep learning network. Use YOLOX-tiny for slightly less constrained real-time applications with an improvement in accuracy from YOLOX-nano.
- `"small-coco"` — Pretrained YOLOX-small deep learning network. Use YOLOX-small for a higher level of accuracy than YOLOX-tiny when moderate computational resources are available.
- `"medium-coco"` — Pretrained YOLOX-medium deep learning. Use YOLOX-medium for a higher level of accuracy than YOLOX-small when some computational resources (e.g. GPU hardware) are available.
- `"large-coco"` — Pretrained YOLOX-large deep learning network with the largest number of filters and convolutional layers. Use the YOLOX-large network for the highest level accuracy of object detection at the expense of computational cost and speed.

Tip To improve accuracy and performance of complex detection tasks, increase the pretrained model size at the expense of decreased inference speed, larger computation time, and greater memory requirements. Use YOLOX-nano for real-time inference at the expense of detection accuracy. Use YOLOX-tiny and YOLOX-small for near-real time applications due to fast inference. Use YOLOX-medium and YOLOX-large when sufficient computational resources are available for the largest accuracy, at the expense of lower computation speeds.

Data Types: `char` | `string`

classes — Names of object classes

vector of strings | cell array of character vectors | categorical vector

Names of object classes for training the detector, specified as a vector of strings, cell array of character vectors, or categorical vector. This argument sets the `ClassNames` property of the `yoloxObjectDetector` object.

Data Types: `char` | `string` | `categorical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `yoloxObjectDetector(detector,classes,ModelName="customDetector")` sets the name for the object detector to `"customDetector"`.

`ModelName` — Name for object detector

"" (default) | character vector | string scalar

Name for the object detector, specified as a character vector or string scalar.

`InputSize` — Image size used for training

row vector

Image size used for training the object detector, specified as a row vector `[H W C]`, where *H*, *W*, and *C* correspond to the height, width, and number of channels, respectively. The default is the network input size.

`NormalizationStatistics` — *Z*-score normalization statistics

1-by-1 structure (default) | structure

Z-score normalization statistics, specified as a structure with these fields, which contain the values from the COCO data set:

Field	Description	Default Value
`Mean`	1-by-*C* vector of means per channel.	`[123.6750 116.2800 103.5300]`
`StandardDeviation`	1-by-*C* vector of standard deviations per channel.	`[58.3950 57.1200 57.3750]`

The number of channels *C* must match the number of channels in the image specified by `InputSize`.

Properties

`ModelName` — Name for object detector

"" (default) | character vector | string scalar

This property is read-only.

Name for the object detector, stored as a character vector or string scalar. To set this property, specify it at object creation.

`ClassNames` — Names of object classes

categorical vector

This property is read-only.

Names of object classes to detect, stored as a categorical vector. To set this property, use the `classes` argument at object creation.

`InputSize` — Image size used for training

row vector of integers

This property is read-only.

Image size used for training and inference, stored as a row vector of integers, of the form [*height width channels*]. To set this property, specify it at object creation.

Object Functions

detect Detect objects using YOLOX object detector

Examples

Create Pretrained YOLOX Object Detector

Specify the name of a pretrained YOLOX deep learning network.

```
networkName = "small-coco"
```

```
networkName =
"small-coco"
```

Create a YOLOX object detector by using the pretrained YOLOX network.

```
detector = yoloxObjectDetector(networkName);
```

Display and inspect the properties of the YOLOX object detector.

```
disp(detector)
```

```
  yoloxObjectDetector with properties:

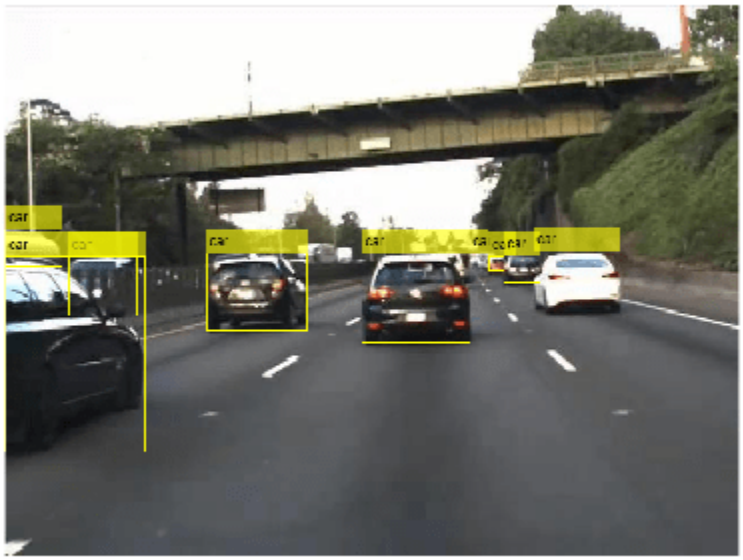
                ClassNames: {80×1 cell}
                 InputSize: [640 640 3]
    NormalizationStatistics: [1×1 struct]
                 ModelName: 'small-coco'
```

Detect objects in a chosen image.

```
I = imread("carsonroad.png");
[bboxes,scores,labels] = detect(detector,I);
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Version History

Introduced in R2023b

R2024b: Use `coder.loadDeepLearningNetwork` for code generation

Starting in R2024b, you can use the `coder.loadDeepLearningNetwork` function to load a `yoloxObjectDetector` object for CPU and GPU code generation.

R2024b: Specify base network as pretrained YOLOX-nano, YOLOX-medium, or YOLOX-large

The `yoloxObjectDetector` object enables you to specify a YOLOX-nano, YOLOX-medium, or YOLOX-large deep learning network as the base network of the pretrained detector using the `name` input argument.

References

[1] Ge, Zheng, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. "YOLOX: Exceeding YOLO Series in 2021." *arXiv*, August 5, 2021. http://arxiv.org/abs/2107.08430.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `yoloxObjectDetector` object for CPU code generation, you can use either`vision.loadYOLOXObjectDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- The `AutoResize` argument to the `detect` method must be a code generation constant (`coder.const()`).
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value pairs for `detect` are supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `yoloxObjectDetector` object for GPU code generation, you can use either`vision.loadYOLOXObjectDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (GPU Coder).
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- The `AutoResize` argument to the `detect` method must be a code generation constant (`coder.const()`).
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize`, and `AutoResize` name-value pairs for `detect` are supported.

See Also
`trainYOLOXObjectDetector` | `detect` | `evaluateObjectDetection`

Topics
“Getting Started with YOLOX for Object Detection”
“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Get Started with Object Detection Using Deep Learning”
“Get Started with Image Preprocessing and Augmentation for Deep Learning”
“Choose an Object Detector”

hrnetObjectKeypointDetector

Create object keypoint detector using HRNet deep learning network

Description

The `hrnetObjectKeypointDetector` object creates a high resolution deep learning network (HRNet) object keypoint detector for detecting object keypoints in an image. Using this object, you can:

- Create a pretrained HRNet object keypoint detector by using HRNet deep learning networks trained on the COCO keypoint detection data set.
- Create a custom HRNet object keypoint detector by using a pretrained HRNet deep learning network.

Creation

Syntax

```
detector = hrnetObjectKeypointDetector
detector = hrnetObjectKeypointDetector(name)

detector = hrnetObjectKeypointDetector(name,keypointClassNames)
```

Description

Pretrained HRNet Object Keypoint Detector

`detector = hrnetObjectKeypointDetector` creates a default pretrained HRNet object keypoint detector. To create the default HRNet object keypoint detector, MATLAB uses an HRNet deep learning network, HRNet-W32, trained on the COCO keypoint detection data set. In an HRNet-W32 network, the last three stages of the high-resolution subnetworks have 32 convolved feature maps. For more information about HRNet architecture, see “Getting Started with HRNet”.

`detector = hrnetObjectKeypointDetector(name)` creates a pretrained HRNet object keypoint detector by using the specified HRNet deep learning network, which has been trained on the COCO keypoint detection data set.

Custom HRNet Object Keypoint Detector

`detector = hrnetObjectKeypointDetector(name,keypointClassNames)` creates a pretrained HRNet object detector, and configures it to perform transfer learning using a specified set of keypoint classes. For optimal results, you must train the detector on new training images before performing detection.

`detector = hrnetObjectKeypointDetector(name,keypointClassNames,Name=Value)` sets the “ModelName” on page 2-0 , “InputSize” on page 2-0 , “KeypointConnections” on page 2-0 and “Threshold” on page 2-0 properties of the object keypoint detector using one or more name-value arguments.

Example:
`hrnetObjectKeypointDetector(name,keypointClassNames,ModelName="customDetector")` sets the name for the object detector to `"customDetector"`.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Object Keypoint Detection. You can download and install the Computer Vision Toolbox Model for Object Keypoint Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Input Arguments

`name` — Name of pretrained HRNet deep learning network
`"human-full-body-w32"` | `"human-full-body-w48"`

Name of the pretrained HRNet deep learning network, specified as one of these options:

- `"human-full-body-w32"` — A pretrained HRNet deep learning network created using HRNet-W32 as the base network, and trained on the COCO keypoint detection data set.
- `"human-full-body-w48"` — A pretrained HRNet deep learning network created using HRNet-W48 as the base network, and trained on the COCO keypoint detection data set.

In HRNet-W32 and HRNet-W48 networks, the last three stages of the high-resolution subnetworks have 32 and 48 convolved feature maps, respectively.

Data Types: `char` | `string`

`keypointClassNames` — Names of object keypoint classes
vector of strings | cell array of character vectors | categorical vector

Names of object keypoint classes for training the detector, specified as a vector of strings, cell array of character vectors, or categorical vector. This argument sets the `KeyPointClasses` property of the `hrnetObjectKeypointDetector` object.

Data Types: `cell` | `string` | `categorical`

Properties

`Network` — HRNet deep learning network
`dlnetwork` object

This property is read-only.

HRNet deep learning network to use for object keypoint detection, stored as a `dlnetwork` object.

`ModelName` — Name of object keypoint detector
`'human-full-body-w32'` (default) | character vector | string scalar

This property is read-only.

Name for the object keypoint detector, specified as a character vector or string scalar. To set this property, specify it at object creation. You can set this property only when you also specify the `keypointClassNames` input argument.

Example: `hrnetObjectKeypointDetector(name,keypointClassNames,ModelName="customDetector")` sets the name for the object detector to `"customDetector"`.

`InputSize` — Image size used for training
row vector of integers

This property is read-only.

Image size used for training the object detector, specified as a row vector of integers of the form `[H W C]`, where *H*, *W*, and *C* are the height, width, and number of channels of the image, respectively. The default value is the network input size. To set this property, specify it at object creation. You can set this property only when also specify the `keypointClassNames` input argument.

Example: `detector = hrnetObjectKeypointDetector(name,keypointClassNames,InputSize=[384 288 3])` specifies the image size of the training images as 384-by-288 pixels across 3 channels.

`KeyPointClasses` — Names of object keypoint classes
categorical vector

This property is read-only.

Names of object keypoint classes to detect, specified as a categorical vector.

You can set this property only when you specify the `keypointClassNames` input argument. Otherwise, the object uses the default classes for the specified HRNet deep learning network.

`KeypointConnections` — Keypoint class indices of keypoint connections
L-by-2 matrix

This property is read-only.

Keypoint class indices of the keypoint connections, specified as an *L*-by-2 matrix. Each row denotes the keypoint class indices that connect two keypoints, where *L* is the number of keypoint connections. To set this property, specify it at object creation. You can set this property only when you specify the `keypointClassNames` input argument.

Example: `hrnetObjectKeypointDetector(name,keypointClassNames,KeypointConnections=[2 4; 3 5])` establishes two connections: one between a set of keypoints with keypoint class indices `2` and `4`, and the other between a set of keypoints with the keypoint classes indices `3` and `5`.

`Threshold` — Threshold value for valid keypoints
`0.2` (default) | scalar in the range [0, 1]

Threshold value for valid keypoints, specified as a scalar in the range `[0, 1]`. For a threshold value of 0.4, the object treats keypoints with a confidence score greater than or equal to 0.4 as valid keypoints. To set this property, specify it at object creation. You can set this property only when you specify the `keypointClassNames` input argument.

You can also specify a different threshold when calling the `detect` object function. This lets you adjust threshold for individual detection calls without creating a new `hrnetObjectKeypointDetector` object.

Object Functions

detect	Detect object keypoints using HRNet object keypoint detector
visibleKeypoints	Extract visible keypoints from keypoints detected by HRNet object keypoint detector

Examples

Create Pretrained HRNet Object Keypoint Detector

Specify the name of a pretrained HRNet object keypoint detector.

```
name = "human-full-body-w32";
```

Create an HRNet object keypoint detector by using the pretrained HRNet deep learning network.

```
keypointDetector = hrnetObjectKeypointDetector(name);
```

Display and inspect the properties of the HRNet object keypoint detector.

```
disp(keypointDetector)
```

```
  hrnetObjectKeypointDetector with properties:

             ModelName: 'human-full-body-w32'
               Network: [1×1 dlnetwork]
       KeyPointClasses: [17×1 categorical]
             InputSize: [384 288 3]
   KeypointConnections: [17×2 double]
             Threshold: 0.2000
```

```
analyzeNetwork(keypointDetector.Network)
```

Read a test image into the workspace.

```
I = imread("visionteam.jpg");
```

Use a YOLOv4 object detector to detect bounding boxes around humans in the image.

```
detector = yolov4ObjectDetector("csp-darknet53-coco");
[bboxes,scores,labels] = detect(detector,I,Threshold=0.4);
```

Select the bounding box detections for the `person` class.

```
personIndices = labels == "person";
personBoxes = bboxes(personIndices,:);
```

Detect the object keypoints of the people in the test image by using the pretrained HRNet object keypoint detector.

```
[keypoints,keypointScores,valid] = detect(keypointDetector,I,personBoxes);
```

Insert the detected keypoints into the test image and display the results.

```
detectedKeypoints = insertObjectKeypoints(I,keypoints,KeypointColor="yellow",KeypointSize=2);
imshow(detectedKeypoints)
```



Version History

Introduced in R2023b

R2025a: Adjust number of bounding boxes during code generation

Starting from R2025a, you can vary the number of bounding boxes specified as input to the `detect` function during code generation. This means that the limitations on bounding boxes from previous releases have been removed. You can now assign bounding boxes dynamically during runtime.

R2024b: Adjust bounding box values during code generation

Prior to R2024b, the `bboxes` argument to the `detect` function must be a code generation constant (`coder.const()`) and a `M`-by-4 matrix. The number of bounding boxes `M` and the values of the bounding boxes must be fixed for code generation.

Starting from R2024b, you can dynamically adjust the bounding box values during runtime. However, the number of bounding boxes `M` must be a code generation constant.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The `detect` function does not support `ImageDatastore` objects as input for code generation.
- Code generation does not support the `MiniBatchSize`, `ExecutionEnvironment`, and `Acceleration` name-value arguments for `detect`.

- To prepare an `hrnetObjectKeypointDetector` object for code generation, use `loadHRNETObjectKeypointDetector`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

- The `detect` function does not support `ImageDatastore` object as input for code generation.
- Code generation does not support the `MiniBatchSize`, `ExecutionEnvironment`, and `Acceleration` name-value arguments for `detect`.
- To prepare an `hrnetObjectKeypointDetector` object for code generation, use `loadHRNETObjectKeypointDetector`.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

Functions
`trainHRNetObjectKeypointDetector`

Objects
`yolov4ObjectDetector`

Topics
“Multi-Object Tracking and Human Pose Estimation”
“Hand Pose Estimation Using HRNet Deep Learning”
“Getting Started with HRNet”

detect

Detect object keypoints using HRNet object keypoint detector

Syntax

```
keypoints = detect(detector,I,bboxes)
[keypoints,scores] = detect(detector,I,bboxes)
[keypoints,scores,valid] = detect(detector,I,bboxes)

[keypoints,scores,valid] = detect(detector,ds)

detectionResults = detect( ___ ,Name=Value)
```

Description

`keypoints = detect(detector,I,bboxes)` detects object keypoints within an RGB image, `I`, using an HRNet object keypoint detector, `detector`. The `detect` function automatically resizes and rescales the input to match the size of the images used to train the detector. The function returns the locations of keypoints detected in each input image as a set of object keypoints.

`[keypoints,scores] = detect(detector,I,bboxes)` returns the detection confidence score for each keypoint. The confidence score determines the location accuracy of the detected keypoint. The value of keypoint confidence score is in the range of [0, 1]. A higher score indicates greater confidence in the detection.

`[keypoints,scores,valid] = detect(detector,I,bboxes)` returns the validity for each detected object keypoint.

`[keypoints,scores,valid] = detect(detector,ds)` detects object keypoints within all images returned by the `read` function of the input datastore `ds` and returns the detection results as a table.

`detectionResults = detect( ___ ,Name=Value)` specifies options using one or more name-value arguments.

Examples

Detect Objects Keypoints Using Pretrained HRNet

Read a sample image into the workspace.

```
I = imread("visionteam.jpg");
```

Crop and display one person from the image.

```
personBox = [262.5 19.51 127.98 376.98];
[personImg] = imcrop(I,personBox);
figure
imshow(personImg)
```



Specify a bounding box for the person in the image. You can use the bounding box region as input to the object keypoint detector to detect the person.

```
bbox = [3.87,21.845,118.97,345.91];
```

Load a pretrained HRNet object keypoint detector.

```
keypointDetector = hrnetObjectKeypointDetector("human-full-body-w32");
```

Detect the keypoints of the person in the image by using the pretrained HRNet object keypoint detector.

```
[keypoints,keypointScores,valid] = detect(keypointDetector,personImg,bbox)
```

```
keypoints = 17×2 single matrix

   72.8781   40.7078
   77.1927   36.2037
   63.9647   36.2037
   86.2008   45.2118
   55.2407   40.7078
  100.2811   85.7481
   33.9517   76.7401
  114.5508  135.2925
   17.5455  135.2925
  102.2699  180.3329
   43.3386  103.7643
   89.3260  193.8450
   45.1380  189.3410
   90.9360  270.4136
   38.0240  270.4136
      ⋮
```

```
keypointScores = 17×1 single column vector

    0.9616
    0.9819
    0.9797
    0.9824
    0.9627
    0.9540
    0.8812
    0.9696
    0.9909
    0.9675
    0.9327
    0.8247
    0.8356
    0.9527
    0.9111
      ⋮


valid = 17×1 logical array

   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
      ⋮
```

Insert the person keypoints into the image and display the results.

```
detectedKeypoints = insertObjectKeypoints(personImg,keypoints,KeypointColor="yellow",KeypointSize
imshow(detectedKeypoints)
```



Input Arguments

`detector` — HRNet object keypoint detector

`hrnetObjectKeypointDetector` object

HRNet object keypoint detector, specified as an `hrnetObjectKeypointDetector` object.

`I` — Test images

numeric array

Test images, specified as a numeric array of size *H*-by-*W*-by-*C*-by-*T*. Images must be real, nonsparse, and RGB.

- *H*— Height of the image, in pixels.
- *W*— Width of the image, in pixels.
- *C*— The channel size in each image must be equal to the input channel size. Because this function requires RGB images, the channel size must be `3`.
- *T*— Number of test images in the array. The function computes the object keypoint detection results for each test image in the array.

The intensity range of the test image must be similar to the intensity range of the images used to train the detector. For example, if you train the detector on `uint8` images, rescale the test image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of the test image must be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting object keypoints because the scale of the objects in the test image differs from the scale of the objects the detector is trained to identify.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

bboxes — Locations of objects detected in the image
M-by-4 matrix | *T*-by-1 cell array

Locations of the objects detected in the input image or images, specified as one of these options:

- *M*-by-4 matrix— The input is a single test image. *M* is the number of object bounding boxes in the image.
- *T*-by-1 cell array— The input is an array of test images. *T* is the number of test images in the array. Each cell contains an *M*-by-4 matrix of object bounding boxes for the corresponding image, where *M* is the number of bounding boxes in the image.

The table describes the possible formats of bounding boxes.

Bounding Box	Description
`rectangle`	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

ds — Test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Test images, specified as an `ImageDatastore`, `CombinedDatastore`, or `TransformedDatastore` object containing the full filenames of the test images. The images in the datastore must be RGB images.

When using the read function the datastore `ds` must return a table or a cell array with these columns:

`image data`	`bboxes`	`box labels`
T-by-1 cell array, in which each cell contains an input image. The images in the datastore must be RGB images.	*T*-by-1 cell array, in which each cell contains an *M*-by-4 matrix of 2-D bounding boxes specifying object locations within the corresponding input image. Each row of the matrix specifies the location of a bounding box in the format [*x y w h*].	*T*-by-1 cell array, in which each cell contains an *M*-by-1 categorical vector of object class names. All the categorical data returned by the datastore must use the same categories.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,MiniBatchSize=16)` detects object keypoints in each image using 16-element batches of bounding boxes.

`MiniBatchSize` — Minimum batch size
32 (default) | positive integer

Minimum batch size, specified as a positive integer. Use the `MiniBatchSize` argument when processing an image with a large number of objects. When you specify this argument, the function groups the bounding boxes surrounding the objects into batches of the specified size and processes them together to improve computational efficiency. Increase the minimum batch size to decrease processing time. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply a number of optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available only when using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"none"` — Disable all acceleration.

The default option is `"auto"`. If you specify `"auto"`, MATLAB applies a number of compatible optimizations. The `"auto"` option, MATLAB never generates a MEX function.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is available only for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other type of datastore supports the `"mex"` option.

The `"mex"` option is available only when you use a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

`Threshold` — Threshold for valid keypoints
`0.2` (default) | scalar in the range [0, 1]

Threshold for valid keypoints, specified as a scalar in the range `[0, 1]`. For a threshold value of 0.4, the detector treats keypoints with a confidence score greater than or equal to 0.4 as valid.

When specified as a name-value argument, the threshold applies only to the current function call. It overrides the value stored in the `hrnetObjectKeypointDetector` object, but does not modify the object's persistent `Threshold` property.

A low threshold treats more detected keypoints as valid, including those with lower confidence scores. This increases sensitivity and helps retain keypoints in challenging conditions such as small objects, occlusion, or low-quality images. However, it may also admit more false positives and noisier detections.

A high threshold restricts validity to only the most confident keypoints, improving robustness and reducing false positives. While this results in cleaner detections, it may discard true keypoints that have lower confidence due to image quality, scale, or pose complexity.

Output Arguments

`keypoints` — Locations of object keypoints detected in input image
17-by-2-by-*M* array | *T*-by-1 cell array

Locations of object keypoints detected in the input image or images, returned as one of these options:

- 17-by-2-by-*M* array — The input is a single test image. Each row in the array is of the form [*x* *y*] where *x* and *y* specify the location of a detected keypoint in an object. *M* is the number of objects in the image.
- *T*-by-1 cell array — The input is an array of test images. *T* is the number of test images in the array. Each cell in the cell array contains a 17-by-2-by-*M* array specifying the keypoint detections for the *M* objects in the image.

Each object has 17 detected keypoints.

Data Types: `double`

`scores` — Detection confidence scores for object keypoints
17-by-*M* matrix | *T*-by-1 cell array

Detection confidence scores for object keypoints, returned as one of these options:

- 17-by-*M* matrix — The input is a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The input is an array of test images. *T* is the number of test images in the array. Each cell in the array contains a 17-by-*M* matrix indicating the detection scores for the keypoints of each of the *M* objects in an image.

Each object in an image has 17 has keypoint confidence scores. A higher score indicates higher confidence in the detection.

Data Types: `double`

`valid` — Validity of detected object keypoints
17-by-*M* matrix | *T*-by-1 cell array

Validity of the detected object keypoints, returned as one of these options:

- 17-by-*M* logical matrix — The input is a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The input is an array of test images. *T* is the number of test images in the array. Each cell in the array contains a 17-by-*M* logical matrix indicating the keypoint validity values for the keypoints of each of the *M* objects in the corresponding image.

Each object in an image has 17 keypoint validity values. A value of `1` (`true`) indicates a valid keypoint and `0` (`false`) indicates an invalid keypoint.

The value is set to `1`, when the confidence score of a keypoint is greater than the value specified by the `Threshold` name-value argument. Otherwise, the value is set to 0.

Data Types: `logical`

`detectionResults` — Detection results
two-column table

Detection results, returned as a two-column table with variable names `Keypoints` and `Scores`. Each element of the `Keypoints` column contains a 17-by-2-by-*M* array, for *M* objects in the image. Each row of the array contains a keypoint location for the corresponding object in the format [*x y*]. Each element of the `Scores` column contains 17-by-1-by-*M* array, where *M* is the number of objects in the image. Each row of the array contains the keypoint detection confidence score for one of the 17 keypoints in the corresponding object in that image.

Version History

Introduced in R2023b

R2026a: Specify keypoint detection threshold per call using Name-value argument.

Starting from R2026a, you can specify threshold for detecting keypoints directly in the `detect` function by using the `Threshold` name-value argument. This argument overrides the `Threshold` property of the `hrnetObjectKeypointDetector` object for the current call. It does not modify the stored property.

R2025a: Adjust number of bounding boxes during code generation

Starting from R2025a, you can vary the number of bounding boxes specified as input to the `detect` function during code generation. This means that the limitations on bounding boxes from previous releases have been removed. You can now assign bounding boxes dynamically during runtime.

R2024b: Adjust bounding box values during code generation

Prior to R2024b, the `bboxes` argument to the `detect` function must be a code generation constant (`coder.const()`) and a `M`-by-4 matrix. The number of bounding boxes `M` and the values of the bounding boxes must be fixed for code generation.

Starting from R2024b, you can dynamically adjust the bounding box values during runtime. However, the number of bounding boxes `M` must be a code generation constant.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The `detect` function does not support `ImageDatastore` object as input for code generation.
- Code generation does not support the `MiniBatchSize`, `ExecutionEnvironment`, and `Acceleration` name-value arguments for `detect`.
- To prepare an `hrnetObjectKeypointDetector` object for code generation, use `loadHRNETObjectKeypointDetector`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

- The `detect` function does not support `ImageDatastore` object as input for code generation.
- Code generation does not support the `MiniBatchSize`, `ExecutionEnvironment`, and `Acceleration` name-value arguments for `detect`.
- To prepare an `hrnetObjectKeypointDetector` object for code generation, use `loadHRNETObjectKeypointDetector`.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`hrnetObjectKeypointDetector` | `visibleKeypoints` | `ImageDatastore` | `CombinedDatastore` | `TransformedDatastore`

Topics

“Getting Started with HRNet”

visibleKeypoints

Extract visible keypoints from keypoints detected by HRNet object keypoint detector

Syntax

```
visiblePoints = visibleKeypoints(detector,keypoints,scores,valid)
[visiblePoints,visibleScores,visibleLabels] = visibleKeypoints(detector,
keypoints,scores,valid)
```

Description

`visiblePoints = visibleKeypoints(detector,keypoints,scores,valid)` extracts visible object keypoint locations from the object keypoint locations `keypoints` detected by the HRNet object keypoint detector `detector`. The function obtains the locations of visible object keypoints by using the keypoint validity values `valid`, which are based on the keypoint detection confidence scores `scores`.

`[visiblePoints,visibleScores,visibleLabels] = visibleKeypoints(detector, keypoints,scores,valid)` returns the detection confidence scores for the visible keypoints `visibleScores`, and the labels assigned to the visible keypoints `visibleLabels`.

Examples

Detect Visible Keypoints in Objects

Read a sample image into the workspace.

```
I = imread("visionteam.jpg");
```

Crop and display one person from the image.

```
personBox = [18.51 42.51 137.21 370.98];
[personImg] = imcrop(I,personBox);
figure
imshow(personImg)
```



Specify a bounding box for the person in the image. You can use the bounding box region as input to the object keypoint detector to detect the person.

```
bbox = [3.87 21.845 118.97 345.91];
```

Create an HRNet object keypoint detector by using a pretrained HRNet deep learning network.

```
keypointDetector = hrnetObjectKeypointDetector("human-full-body-w32");
```

Detect the object keypoints of the person in the test image by using the pretrained HRNet object keypoint detector.

```
[keypoints,keypointScores,valid] = detect(keypointDetector,personImg,bbox);
```

Detect keypoints with keypoints confidence scores greater than `0.9`.

```
isValid = keypointScores > 0.9;
```

Extract visible keypoints from keypoints detected by HRNet object keypoint detector.

```
[visiblePoints,visibleScores,visibleLabels] = visibleKeypoints(keypointDetector,keypoints,keypoi
```

Insert and display visible keypoints.

```
detectedKeypoints = insertObjectKeypoints(personImg,visiblePoints{1},KeypointColor="yellow",Keypo
figure
imshow(detectedKeypoints);
```



Display all the visible keypoint levels.

```
visibleLabels{1}
```

```
ans = 10×1 categorical array
     "Nose"
     "Left-eye"
     "Right-eye"
     "Left-ear"
     "Right-ear"
     "Left-shoulder"
     "Right-shoulder"
     "Left-elbow"
     "Right-elbow"
     "Left-wrist"
```

Input Arguments

`detector` — HRNet object keypoint detector

`hrnetObjectKeypointDetector` object

HRNet object keypoint detector, specified as an `hrnetObjectKeypointDetector` object.

`keypoints` — Locations of object keypoints detected in input image

17-by-2-by-*M* array | *T*-by-1 cell array

Locations of object keypoints detected in the input image or images, returned as one of these options:

- 17-by-2-by-*M* array — The keypoints come from a single test image. Each row in the array is of the form [*x* *y*] where *x* and *y* specify the location of a detected keypoint in an object. *M* is the number of objects in the image.
- *T*-by-1 cell array — The keypoints come from an array of test images. *T* is the number of test images in the array. Each cell in the cell array contains a 17-by-2-by-*M* array specifying the keypoint detections for the *M* objects in the image.

Each object has 17 detected keypoints.

Data Types: `double`

`scores` — Detection confidence scores for object keypoints

17-by-*M* matrix | *T*-by-1 cell array

Detection confidence scores for object keypoints, returned as one of these options:

- 17-by-*M* matrix — The keypoint scores come from a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The keypoint scores come from an array of test images. *T* is the number of test images in the array. Each cell in the array contains a 17-by-*M* matrix indicating the detection scores for the keypoints of each of the *M* objects in an image.

Each object in an image has 17 has keypoint confidence scores. A higher score indicates higher confidence in the detection.

Data Types: `double`

`valid` — Validity of detected object keypoints

17-by-*M* matrix | *T*-by-1 cell array

Validity of the detected object keypoints, returned as one of these options:

- 17-by-*M* logical matrix — The keypoint validity values come from a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The keypoint validity values come from an array of test images. *T* is the number of test images in the array. Each cell in the array contains a 17-by-*M* logical matrix indicating the keypoint validity values for the keypoints of each of the *M* objects in the corresponding image.

Each object in an image has 17 keypoint validity values. A value of `1` (`true`) indicates a valid keypoint and `0` (`false`) indicates an invalid keypoint.

Data Types: `logical`

Output Arguments

`visiblePoints` — Locations of visible keypoints

17-by-2-by-*M* array | *T*-by-1 cell array

Location of visible keypoints detected in the input image or images, returned as one of these options:

- 17-by-2-by-*M* array — The visible keypoints come from a single test image. Each row in the array is of the form [*x* *y*] where *x* and *y* specify the location of a detected visible keypoint in an object. *M* is the number of objects in the image.
- *T*-by-1 cell array — The visible keypoints come from an array of test images. *T* is the number of test images in the array. Each cell in the cell array has an *N*-by-2 matrix specifying the valid keypoint locations of the form [*x*, *y*]. *N* is the number of visible keypoints for each object in the image, as determined by the `scores` input argument. *N* is less than or equal to 17. *M* is the number of objects in the image.

`visibleScores` — Detection confidence scores of visible keypoints
17-by-*M* matrix | *T*-by-1 cell array

Detection confidence scores of visible keypoints, returned as one of these options:

- 17-by-2-by-*M* array — The detection scores of visible keypoints come from a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The detection scores of visible keypoints come from an array of test images. *T* is the number of test images in the array. Each cell in the cell array has an *N*-by-1 matrix specifying the visible keypoint scores. *N* is the number of visible keypoints for each object in the image, as determined by the `scores` input argument. *N* is less than or equal to 17. *M* is the number of objects in the image.

`visibleLabels` — Validity labels of visible keypoints
17-by-*M* matrix | *T*-by-1 cell array

Validity labels of visible keypoints, returned as one of these options:

- 17-by-1-by-*M* array — The validity labels of visible keypoints come from a single test image. *M* is the number of objects in the image.
- *T*-by-1 cell array — The validity labels of visible keypoints come from an array of test images. *T* is the number of test images in the array. Each cell in the cell array has an *N*-by-1 matrix specifying the valid keypoint labels. *N* is the number of visible keypoints for each object in the image, as determined by the `scores` input argument. *N* is less than or equal to 17. *M* is the number of objects in the image.

Version History
Introduced in R2023b

See Also
`hrnetObjectKeypointDetector` | `detect`

Topics
“Getting Started with HRNet”

vision.ChromaResampler

Namespace: `vision`

Downsample or upsample chrominance components of images

Description

To downsample or upsample chrominance components of images:

1. Create the `vision.ChromaResampler` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
resampler = vision.ChromaResampler
gammaCorr = vision.ChromaResampler(Name,Value)
```

Description

`resampler = vision.ChromaResampler` returns a chroma resampling object, `Cresampler`, that downsamples or upsamples chroma components of a YCbCr signal to reduce the bandwidth and storage requirements.

`gammaCorr = vision.ChromaResampler(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `resampler = vision.ChromaResampler('InterpolationFilter','Linear')`

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Resampling` — Resampling format

`'4:4:4 to 4:2:2'` (default) | resample format

Resampling format, specified as one of the following:

```
'4:4:4 to 4:2:2'
'4:4:4 to 4:2:0 (MPEG1)'
```

```
'4:4:4 to 4:2:0 (MPEG2)'
'4:4:4 to 4:1:1'
'4:2:2 to 4:2:0 (MPEG1)'
'4:2:2 to 4:2:0 (MPEG2)'
```

To upsample the chrominance components of images, set this property to one of the following:

```
'4:2:2 to 4:4:4'
'4:2:0 (MPEG1) to 4:4:4'
'4:2:0 (MPEG2) to 4:4:4'
'4:1:1 to 4:4:4'
'4:2:0 (MPEG1) to 4:2:2'
'4:2:0 (MPEG2) to 4:2:2'
```

`InterpolationFilter` — Method used to approximate missing values
`Pixel replication` (default) | `Linear`

Method used to approximate missing values, specified as `Pixel replication` or `Linear`. The default is `Linear`. When you set this property to `Linear`, the object uses linear interpolation to calculate the missing values. When you set this property to `Pixel replication`, the object replicates the chrominance values of the neighboring pixels to create the upsampled image. This property applies when you upsample the chrominance values.

`AntialiasingFilterSource` — Lowpass filter used to prevent aliasing
`Auto` (default) | `Property` | `None`

Lowpass filter used to prevent aliasing, specified as `Auto`, `Property` , or `None`. When you set this property to `Auto`, the object uses a built-in lowpass filter. When you set this property to `Property`, the coefficients of the filters are specified by the `HorizontalFilterCoefficients` and `VerticalFilterCoefficients` properties. When you set this property to `None`, the object does not filter the input signal. This property applies when you downsample the chrominance values.

`HorizontalFilterCoefficients` — Horizontal filter coefficients
`[0.2 0.6 0.2]` (default) | three-element vector

Horizontal filter coefficients, specified as a three-element vector. This property applies when you set the `Resampling` property to one of `[4:4:4 to 4:2:2]` | `[4:4:4 to 4:2:0 (MPEG1)]` | `[4:4:4 to 4:2:0 (MPEG2)]` | `[4:4:4 to 4:1:1]` and the `AntialiasingFilterSource` property to `Property`.

`VerticalFilterCoefficients` — Specify the filter coefficients to apply to the input signal
`[0.5 0.5]` (default) | two-element vector

Specify the filter coefficients to apply to the input signal, specified as a two-element vector. This property applies when you set the `Resampling` property to one of `[4:4:4 to 4:2:0 (MPEG1)]` | `[4:4:4 to 4:2:0 (MPEG2)]` | `[4:2:2 to 4:2:0 (MPEG1)]` | `[4:2:2 to 4:2:0 (MPEG2)]` and the `AntialiasingFilterSource` property to `Property`.

`TransposedInput` — Input is row-major format
`false` (default) | `true`

Input is row-major format , specified as `true` or `false`. Set this property to `true` when the input contains data elements from the first row first, then data elements from the second row second, and so on through the last row. Otherwise, the object assumes that the input data is stored in column-major format.

Usage

Syntax

```
[Cb1,Cr1] = resampler(Cb,Cr)
```

Description

`[Cb1,Cr1] = resampler(Cb,Cr)` resamples the input chrominance components Cb and Cr and returns Cb1 and Cr1, as the resampled outputs.

Input Arguments

`Cb1` — Chrominance component
matrix

Chrominance component of an image, specified as a matrix.

`Cr2` — Chrominance component
matrix

Chrominance component of an image, specified as a matrix.

Output Arguments

`Cb1` — Resampled chrominance component
same as input (default)

Chrominance component of an image, returned as a matrix.

`Cr1` — Resampled chrominance component
same as input (default)

Chrominance component of an image, returned as a matrix.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Resample Chrominance Components of Image

Create a resampler object.

```
resampler = vision.ChromaResampler;
```

Read an RGB image and convert it to YCbCr.

```
imageRGB = imread('peppers.png');
imageYCbCr = rgb2ycbcr(imageRGB);
```

Resample the Cb and Cr chrominance components.

```
[Cb,Cr] = resampler(imageYCbCr(:,:,2), imageYCbCr(:,:,3));
```

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`rgb2ycbcr`

groundTruthDataSource

Object for storing ground truth data sources

Description

The `groundTruthDataSource` object defines the source of ground truth data. Use this object to specify a data source for the `groundTruth` object. To label the data source, load the `groundTruthDataSource` object into a labeling app.

- The Image Labeler supports data sources for collections of images.
- The Video Labeler supports data sources for videos and image sequences. This app also supports custom data sources.

Creation

Syntax

```
gtSource = groundTruthDataSource(imds)

gtSource = groundTruthDataSource(imageFiles)

gtSource = groundTruthDataSource(videoName)

gtSource = groundTruthDataSource(imageSeqFolder)
gtSource = groundTruthDataSource(imageSeqFolder,timestamps)

gtSource = groundTruthDataSource(sourceName,readerFcn,timestamps)
```

Description

Image Datastore Source

`gtSource = groundTruthDataSource(imds)` returns a ground truth data source object for an `imageDatastore` specified by `imds`.

Collection of Images Source

`gtSource = groundTruthDataSource(imageFiles)` returns a ground truth data source object for a collection of images specified by `imageFiles`. Images must be in a file format readable by `imread`.

Video File Source

`gtSource = groundTruthDataSource(videoName)` returns a ground truth data source object for a video file specified by `videoName`. Videos must be in a file format readable by `VideoReader`.

Video as a Sequence of Images Source

`gtSource = groundTruthDataSource(imageSeqFolder)` returns a ground truth data source object for an image sequence located in the folder specified by `imageSeqFolder`.

`gtSource = groundTruthDataSource(imageSeqFolder,timestamps)` returns a ground truth data source object for an image sequence with a corresponding timestamp for each image contained in the specified folder. `timestamps` sets the `TimeStamps` property.

Custom Data Source

`gtSource = groundTruthDataSource(sourceName,readerFcn,timestamps)` returns a ground truth data source object by using the custom reader function handle, `readerFcn`. `sourceName` sets the `Source` property and `timestamps` set the `TimeStamps` property. The custom reader function loads an image from `sourceName` that corresponds to the current timestamp specified in the `duration` vector `timestamps`.

Input Arguments

`imds` — Image datastore

`imagedatastore` object

Image datastore, specified as an `imageDatastore` object.

`imageFiles` — Image file names

string array | cell array of character vectors

Image file names, specified as a string array or a cell array of character vectors. Images must be in a file format readable by `imread`. For a list of the supported image file formats, see `imformats`.

`videoName` — Name of video file

string scalar | character vector

Name of video file, specified as a string scalar or character vector. Videos must be in a file format readable by `VideoReader`. For a list of the supported video file formats, see `VideoReader.getFileFormats`. If your video format is not supported, specify a custom reader function, `readerFcn`.

`imageSeqFolder` — Image sequence folder

string scalar | character vector

Image sequence folder, specified as a string scalar or a character vector. The image files name extensions must be supported by `imformats`. If your video format is not supported, specify a custom reader function, `readerFcn`.

The images are loaded in the order returned by the `dir` command.

`readerFcn` — Custom reader function

function handle

Custom reader function, specified as a function handle. The custom reader function must load an image from a source at a specified timestamp by using this syntax:

```
outputImage = readerFcn(sourceName,currentTimestamp)
```

- `readerFcn` is the name of your custom reader function.
- `sourceName` is the name of the data source.
- `currentTimestamp` is the current timestamp, as specified by the input vector `timestamp`.

The `outputImage` returned by the custom function must be a grayscale or RGB image in any format supported by `imshow`. For more information, see “Use Custom Image Source Reader for Labeling”.

Properties

`TimeStamps` — Timestamps of video or image sequence

`duration` vector

This property is read-only.

Timestamps of video or image sequence, specified as a `duration` vector.

- For a video file, `Timestamps` is automatically populated with the timestamps that are present for the video frames.
- For an image sequence or custom reader, `Timestamps` is populated with the values in the input `duration` vector `timestamps`.
- For an image collection, the `TimeStamps` property remains empty.

.

`Source` — Source of ground truth data

character vector | cell array of character vectors

This property is read-only.

Source of ground truth data, specified as a character vector or cell array of character vectors. The source name can refer to image file names, a video file name, image sequence file names, or custom data source names.

Examples

Create Data Source from Image Datastore

Load image collection file names.

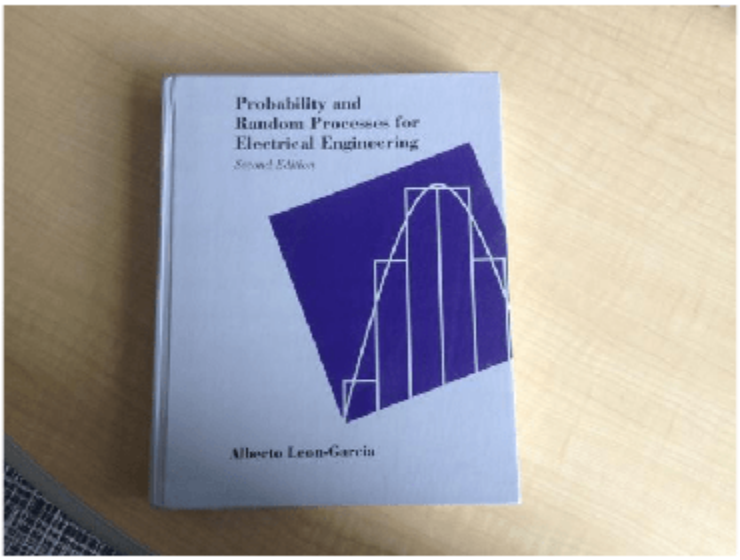
```
imageDir = fullfile(matlabroot,'toolbox','vision','visiondata','bookCovers');
imds = imageDatastore(imageDir);
```

Create a data source from an image datastore.

```
dataSource = groundTruthDataSource(imds);
```

Read and display an image from the datastore.

```
I = read(dataSource.Source);
figure,imshow(I)
```



Create a Ground Truth Data Source from a Video File

Use the `groundTruthDataSource` object to create a data source.

Read a video file and create a data source.

```
videoName = 'vipunmarkedroad.avi';
dataSource = groundTruthDataSource(videoName)
```

```
dataSource =
groundTruthDataSource for a video file with properties

        Source: ...tlab\toolbox\vision\visiondata\vipunmarkedroad.avi
    TimeStamps: [84×1 duration]
```

Create a VideoReader to read the video frames.

```
reader = VideoReader(videoName);
```

Read the 5th frame in the video and display

```
timeStamp = seconds(dataSource.TimeStamps(5));
reader.CurrentTime = timeStamp;
I = readFrame(reader);

figure
imshow(I)
```



Create Data Source from Image Sequence

Create a ground truth data source from an image sequence stored in a specified folder.

Specify the folder containing a sequence of images.

```
imageDir = fullfile(matlabroot,'toolbox','vision',...
        'visiondata','building');
```

Create a data source for the images that are in the `imageDir` folder.

```
    dataSource = groundTruthDataSource(imageDir)
```

```
dataSource =
groundTruthDataSource for a video as an image sequence with properties

                     Source: {
                             'B:\runnable\matlab\toolbox\vision\visiondata\building\building1.JP
                             'B:\runnable\matlab\toolbox\vision\visiondata\building\building2.JP
                             'B:\runnable\matlab\toolbox\vision\visiondata\building\building3.JP
                              ... and 2 more
                             }
                 TimeStamps: [5×1 duration]
```

Read the 5th frame in the sequence.

```
    I = imread(dataSource.Source{5});
    figure
    imshow(I)
```



Create Data Source Using Custom Reader

Create a ground truth data source by using a custom reader function.

Specify an image folder containing a sequence of road images.

```
imgFolder = fullfile(matlabroot,'toolbox','vision','visiondata','building');
```

Use an image datastore as the custom data source.

```
imgDataStore = imageDatastore(imgFolder);
```

Write a reader function, `readerFcn`, to read images from the datastore. The first input argument, `sourceName`, is not used. The second input argument, `currentTimestamp`, is the current timestamp. The function converts `currentTimestamp` from a `duration` scalar to a 1-based index suitable for reading images from the datastore.

```
readerFcn = @(~,idx)readimage(imgDataStore,seconds(idx));
```

Create a data source for the images in the image folder by using the custom reader function.

```
dataSource = groundTruthDataSource(imgFolder,readerFcn,1:5)
```

```
dataSource =
groundTruthDataSource for a custom data source with properties

        Source: B:\runnable\matlab\toolbox\vision\visiondata\building
    TimeStamps: [5×1 duration]
```

Read the fifth frame in the sequence.

```
I = readerFcn(imgFolder,seconds(5));
figure
imshow(I)
```



Tips

- `groundTruth` objects for video-based `groundTruthDataSource` objects rely on the video reading capabilities of your operating system. A `groundTruth` object created using a video data source remains consistent only for the same platform that was used to create it. To create a platform-specific `groundTruth` object, convert the video into a sequence of images.

Version History

Introduced in R2017a

See Also

Apps

Ground Truth Labeler | Image Labeler | Video Labeler

Functions

`groundTruth` | `labelType` | `objectDetectorTrainingData` | `imageDatastore` | `duration`

Topics

“Get Started with the Image Labeler”
“Get Started with the Video Labeler”
“Elements of Ground Truth Objects”
“Use Custom Image Source Reader for Labeling”
“Create Custom Automation Algorithm for Labeling”

“Training Data for Object Detection and Semantic Segmentation”

groundTruth

Ground truth label data

Description

The `groundTruth` object contains information about the data source, label definitions, and marked label annotations for a set of ground truth labels. You can export or import a `groundTruth` object from the Image Labeler and Video Labeler apps. For a summary about the data stored in the `groundTruth` object, see “Elements of Ground Truth Objects”.

- To create training data for an object detector from arrays of `groundTruth` objects, use the `objectDetectorTrainingData` function.
- To create training data for a semantic segmentation network from arrays of `groundTruth` objects, use the `pixelLabelTrainingData` function.

Creation

To export a `groundTruth` object from a labeling app, on the app toolstrip, select **Export Labels** > **To Workspace**. The app exports the object to the MATLAB workspace. To create a `groundTruth` object programmatically, use the `groundTruth` function (described here).

Syntax

```
gTruth = groundTruth(dataSource,labelDefs,labelData)
```

Description

`gTruth = groundTruth(dataSource,labelDefs,labelData)` returns an object containing ground truth labels that can be imported into the Image Labeler and Video Labeler apps.

- `dataSource` specifies the source of the ground truth data and sets the `DataSource` property.
- `labelDefs` specifies the label, sublabel, and attribute definitions of the ground truth data and sets the `LabelDefinitions` property.
- `labelData` specifies the identifying information, position, and timestamps for marked labels and sets the `LabelData` property.

Properties

`DataSource` — Source of ground truth data

`groundTruthDataSource` object

Source of ground truth data, specified as a `groundTruthDataSource` object. The object contains information that describes the video, image sequence, or custom data source from which ground truth data was labeled.

To access images from the original data source, use `VideoReader` or `imageDatastore`. You can also use a custom read function. For more details, see “Use Custom Image Source Reader for Labeling”.

`LabelDefinitions` — Label definitions

table

This property is read-only.

Label definitions, specified as a table. To create this table, use one of these options.

- In one of the labeling apps, create label definitions, and then export them as part of a `groundTruth` object.
- Use a `labelDefinitionCreator` object to generate a label definitions table. If you save this table to a MAT-file, you can then load the label definitions into a labeling app session by selecting **Load** > **Label Definitions** from the app toolstrip.
- Create the label definitions table at the MATLAB command line.

This table describes the required and optional columns of the table specified in the `LabelDefinitions` property.

Column	Description	Required or Optional
`Name`	Strings or character vectors specifying the name of each label definition.	Required
`Type`	`labelType` enumerations that specify the type of each label definition, such as `Rectangle` or `Scene`.	Required

Column	Description	Required or Optional
`LabelColor`	1-by-3 row vectors of RGB triplets that specify the colors of the label definitions. Values are in the range [0, 1]. The color yellow (RGB triplet [1 1 0]) is reserved for the color of selected labels in the labeling apps.	Optional When you define labels in a labeling app, you must specify a color. Therefore, an exported label definitions table always includes this column. When you create label definitions using the `labelDefinitionCreator` object without specifying colors, the returned label definitions table includes this column, but all column values are empty.

Column	Description	Required or Optional
`PixelLabelID`	Scalars, column vectors, or *M*-by-3 matrices of integer-valued label IDs. `PixelLabelID` specifies the pixel label values used to represent a label definition. Pixel label ID values must be between 0 and 255.	Optional When you define pixel labels in a labeling app or the `labelDefinitionCreator` object, the generated label definitions table includes this column. When creating a label definitions table at the MATLAB command line, if you set `Type` to `labelType.PixelLabel` for any label, then this column is required.

Column	Description	Required or Optional
`Group`	Strings or character vectors specifying the group to which each label definition belongs.	Optional If you create the label definitions table at the MATLAB command line, you do not need to include a `Group` column. If you export label definitions from a labeling app or create them using a `labelDefinitionCreator` object, the label definitions table includes this column, even if you did not specify groups. The app assigns each label definition a `Group` value of `'None'`.

Column	Description	Required or Optional
`Description`	Strings or character vectors that describe each label definition.	Optional If you create the label definitions table at the MATLAB command line, you do not need to include a `Description` column. If you export label definitions from a labeling app or create them using a `labelDefinitionCreator` object, the label definitions table includes this column, even if you did not specify descriptions. The `Description` for these label definitions is an empty character vector.
`Hierarchy`	Structures containing sublabel and attribute data for each label definition. For an example of the `Hierarchy` format, see “Get Started with the Image Labeler” or “Get Started with the Video Labeler”.	Optional In labeling apps, when you define sublabels or attributes, the exported `groundTruth` object includes this column.

For example, consider a table with label definitions named `Sky`, `Vegetation`, `Lanes`, `StopSign`, and `Vehicle`, and that was exported from the Video Labeler app.

```
>> gTruth.LabelDefinitions

ans =

  5×6 table

       Name          Type            LabelColor     PixelLabelID    Group      Description
    ____________    _____________    ____________    ____________    _______    ____________

    {'Sky'       }    PixelLabel       {1×3 double}    {[       1]}    {'None'}    {0×0 char}
    {'Vegetation'}    PixelLabel       {1×3 double}    {[       2]}    {'None'}    {0×0 char}
    {'Lanes'     }    Line             {1×3 double}    {0×0 double}    {'None'}    {0×0 char}
    {'StopSign'  }    Rectangle        {1×3 double}    {0×0 double}    {'None'}    {0×0 char}
    {'Vehicle'   }    ProjectedCuboid  {1×3 double}    {0×0 double}    {'None'}    {0×0 char}
```

- The label definitions include pixel labels, so the table includes a `PixelLabelID` column.
- Two of the labels contain attributes, so the app created a `Hierarchy` column that applies across all label definitions.
- The label definitions do not have assigned groups, so the `Group` column is `'None'` for all label definitions.

Note Labeler apps can only load pixel data that have pixel IDs between 1 and 255.

`LabelData` — Label data for each ROI and scene label

table | timetable

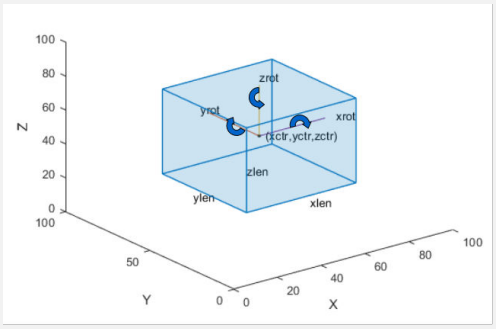
This property is read-only.

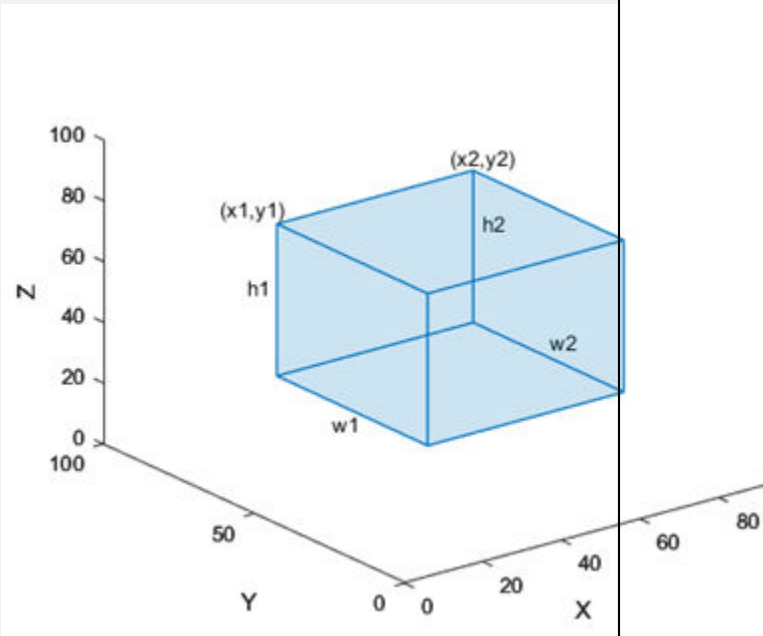
Label data for each ROI and scene label, specified as a `table` for image collections or a `timetable` for videos or image sequences. Each column of `LabelData` holds labels for a single label definition and corresponds to the `Name` value for each row in `LabelDefinitions`. These `LabelData` describes the elements of the table. The label categories are specified as `labelType` enumerations.

Alternatively, for ROI label data that is grouped by label type, a single column labeled `ROILabelData`, can be used and specified as a structure containing at least one label type, `RectangleData`, `PolygonData`, `LineData`, or `ProjCuboidData`.

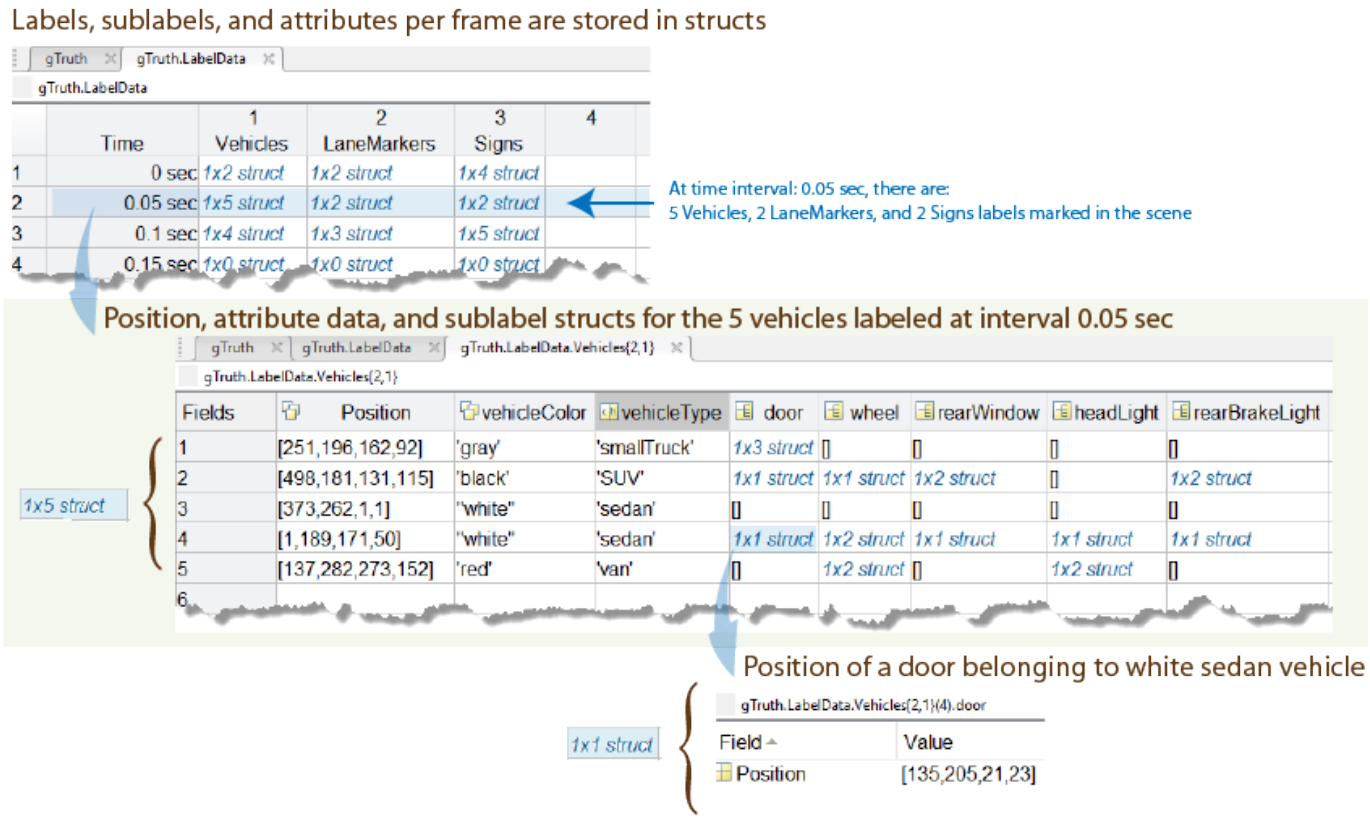
The storage format for each label type is described in the table.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.Rectangle`	`M`-by-4 numeric matrix of the form `[x, y, w, h]`, where: • `M` is the number of labels in the frame. • `x` and `y` specify the upper-left corner of the rectangle. • `w` specifies the width of the rectangle, which is its length along the *x*-axis. • `h` specifies the height of the rectangle, which is its length along the *y*-axis.
`labelType.RotatedRectangle`	For one or more rotated rectangles, specify in spatial coordinates as an *M*-by-5 numeric matrix, where each row specifies a rotated rectangle of the form [*xctr yctr w h yaw*]. • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the rectangle.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.Cuboid`	M-by-9 numeric matrix with rows of the form `[xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot]`, where: • `M` is the number of labels in the frame. • `xctr`, `yctr`, and `zctr` specify the center of the cuboid. • `xlen`, `ylen`, and `zlen` specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation has been applied. • `xrot`, `yrot`, and `zrot` specify the rotation angles for the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively. These angles are clockwise-positive when looking in the forward direction of their corresponding axes. The figure shows how these values determine the position of a cuboid. 

Label Type	Storage Format for Labels at Each Timestamp
`labelType.ProjectedCuboid`	`M`-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid. 
`labelType.Line`	`M`-by-1 vector of cell arrays, where `M` is the number of labels in the frame. Each cell array contains an `N`-by-2 numeric matrix of the form `[x1 y1; x2 y2; ... ; xN yN]` for `N` points in the polyline.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.PixelLabel`	Label data for all pixel label definitions is stored in a single *M*-by-1 `PixelLabelData` column for *M* images or frames. Each element contains a filename for a pixel label image. A pixel label image describes the label or labels contained in the corresponding image. The labels can be described as a 1- or 3- channel label matrix. To use `PixelLabelData` with any of the labeler apps, you must use a single-channel label matrix, where the values are of type `uint8`. You can convert a 3-channel pixel label data matrix to a single-channel label matrix programmatically to use with the labeler apps.
`labelType.Polygon`	`M`-by-1 vector of cell arrays, where `M` is the number of labels. Each cell array contains an `N`-by-2 numeric matrix of the form `[x1 y1; x2 y2; ... ; xN yN]` for `N` points in the polygon.
`labelType.Custom`	Labels are stored exactly as they are specified in the timetable. If you import a `groundTruthMultisignal` object containing custom label data into the Ground Truth Labeler app, this data is not imported into the app. Use custom data when gathering label data for training and combining it with data labeled in the app.
`labelType.Scene`	Stored as a `logical`.



To add ground truth data that is not in one of the native supported ROIs or a `Scene` label category to a `groundTruth` object, you must provide a label definition with a `labelType` that is `Custom`. The custom data is not visible when you load it into the labeling app.

Object Functions

selectLabelsByGroup	Select ground truth labels by label group
selectLabelsByType	Select ground truth labels by label type
selectLabelsByName	Select ground truth labels by label name
changeFilePaths	Change file paths in ground truth data
gatherLabelData	Gather label data from ground truth
writeFrames	Write ground truth video frames to disk
merge	Merge two or more ground truth objects

Examples

Create Ground Truth for Stop Signs and Cars

Create a data source from a collection of images.

```
data = load('stopSignsAndCars.mat');
imageFilenames = data.stopSignsAndCars.imageFilename(1:2)
```

```
imageFilenames = 2×1 cell
    {'stopSignImages/image001.jpg'}
```

```
    {'stopSignImages/image002.jpg'}
```

```
imageFilenames = fullfile(toolboxdir('vision'),'visiondata',imageFilenames);
dataSource = groundTruthDataSource(imageFilenames);
```

Define labels used to specify the ground truth. Use `labelDefinitionCreator` to create the label definitions table.

```
ldc = labelDefinitionCreator();
addLabel(ldc,'stopSign',labelType.Rectangle);
addLabel(ldc,'carRear',labelType.Rectangle);
labelDefs = create(ldc)
```

```
labelDefs=2×5 table
        Name          Type       LabelColor     Group      Description
    ____________    _________    __________    ________    ___________

    {'stopSign'}    Rectangle    {0×0 char}    {'None'}       {' '}
    {'carRear' }    Rectangle    {0×0 char}    {'None'}       {' '}
```

Initialize label data for rectangle ROIs.

```
stopSignTruth = {[856   318    39    41];[445   523    52    54]};
carRearTruth = {[398   378   315   210];[332   633   691   287]};
```

Construct a table of label data.

```
labelNames = {'stopSign';'carRear'};
labelData = table(stopSignTruth,carRearTruth,'VariableNames',labelNames)
```

```
labelData=2×2 table
        stopSign              carRear
    __________________    ____________________

    {[856 318 39 41]}    {[398 378 315 210]}
    {[445 523 52 54]}    {[332 633 691 287]}
```

Create a ground truth object.

```
gTruth = groundTruth(dataSource,labelDefs,labelData)
```

```
gTruth =
  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [2×5 table]
           LabelData: [2×2 table]
```

Create Ground Truth Data of Road Lanes

Create a `groundTruth` object to store data representing marked road lanes.

Create a data source from an image.

```
dataSource = groundTruthDataSource({'stopSignTest.jpg'});
```

Define labels used to specify ground truth. Use `labelDefinitionCreator` to create label definitions table.

```
ldc = labelDefinitionCreator();
addLabel(ldc,'Lane',labelType.Line);
labelDefs = create(ldc);
```

Assign two lane markers in the image.

```
laneMarkerTruth = {[257 254;311 180] [327 183;338 205;374 250]};
```

Construct a table of label data.

```
labelNames = {'Lane'};
labelData = table(laneMarkerTruth,'VariableNames',labelNames)
```

```
labelData=table
                Lane
    ________________________________

    {2×2 double}    {3×2 double}
```

Create a groundTruth object.

```
gTruth = groundTruth(dataSource,labelDefs,labelData)
```

```
gTruth =
  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [1×5 table]
           LabelData: [1×1 table]
```

Create Ground Truth Data for Pixel Labels

Create a `groundTruth` object to store data representing parts of a scene.

Create a data source.

```
dataSource = groundTruthDataSource({'visionteam.jpg'});
```

Use `labelDefinitionCreator` to create the label definitions table. Define labels, `'Person'` and `'Background'`. Assign their corresponding label type as `PixelLabel`.

```
ldc =labelDefinitionCreator();
addLabel(ldc,'Person',labelType.PixelLabel);
addLabel(ldc,'Background',labelType.PixelLabel);
labelDefs = create(ldc)
```

```
labelDefs=2×6 table
         Name            Type       LabelColor    PixelLabelID     Group      Description
    _____________    ___________    __________    _____________    ________    ___________
```

```
    {'Person'    }    PixelLabel    {0×0 char}       {[1]}        {'None'}       {' '}
    {'Background'}    PixelLabel    {0×0 char}       {[2]}        {'None'}       {' '}
```

Specify the location of the pixel label data for the image.

```
dataFile = {'visionteamPixelLabels.png'}
```

```
dataFile = 1×1 cell array
    {'visionteamPixelLabels.png'}
```

Construct a table of label data for the pixel label data.

```
labelData = table(dataFile,'VariableNames',{'PixelLabelData'})
```

```
labelData=table
           PixelLabelData
    ______________________________

    {'visionteamPixelLabels.png'}
```

Create a `groundTruth` object.

```
gTruth = groundTruth(dataSource,labelDefs,labelData)
```

```
gTruth =
  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [2×6 table]
           LabelData: [1×1 table]
```

Tips

- `groundTruth` objects for video-based data sources rely on the video reading capabilities of your operating system. A `groundTruth` object created using a video data source remains consistent only for the same platform that was used to create it. To create a platform-independent `groundTruth` object, convert the video into a sequence of images and include the associated timestamps with the image sequence.

Version History

Introduced in R2017a

See Also

Apps

Image Labeler | Video Labeler

Functions
`pixelLabelTrainingData` | `objectDetectorTrainingData` | `pixelLabelDatastore` | `groundTruthToOpenLabel`

Objects
`labelDefinitionCreator` | `groundTruthDataSource` | `labelType` | `attributeType`

Topics
“Elements of Ground Truth Objects”
“Get Started with the Image Labeler”
“Get Started with the Video Labeler”
“Use Custom Image Source Reader for Labeling”
“Share and Store Labeled Ground Truth Data”
“How Labeler Apps Store Exported Pixel Labels”

writeFrames

Write ground truth video frames to disk

Syntax

```
files = writeFrames(gTruth,location)
files = writeFrames(gTruth,location,timestamps)
files = writeFrames( ___ ,Name=Value)
```

Description

`files = writeFrames(gTruth,location)` writes all video frames from one or more ground truth objects `gTruth` to the folder specified by `location`. The function returns the paths to the output video frames `files`.

`files = writeFrames(gTruth,location,timestamps)` writes video frames with the specified timestamps to disk.

`files = writeFrames( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of input arguments from previous syntaxes. For example, `ImageFormat="jpg"` sets the returned video frames to the JPEG format.

Examples

Write Video Frames from Ground Truth Object

Load a ground truth object that contains rectangle labels for a video into the workspace.

```
data = load("peopleWalkingGroundtruth.mat");
gTruth = data.peopleWalking;
```

Gather rectangle labels from the video.

```
labelTypes = labelType.Rectangle;
[labelData,timestamps] = gatherLabelData(gTruth,labelTypes);
```

Write the image frames associated with the gathered label data to a temporary folder location. Use the timestamps returned by the `gatherLabelData` function to indicate which video frames to write.

```
outputFolder = fullfile(tempdir,"videoFrames");
fileNames = writeFrames(gTruth,outputFolder,timestamps);
```

```
Write images extracted for training to folder:
    C:\Users\ncorriel\AppData\Local\Temp\videoFrames

Writing 600 images extracted from atrium.mp4...Completed.
```

Create an image datastore of the written video image frames, and a box label datastore of the corresponding rectangle label data.

```
imds = imageDatastore(fileNames{1});
blds = boxLabelDatastore(labelData{1});
```

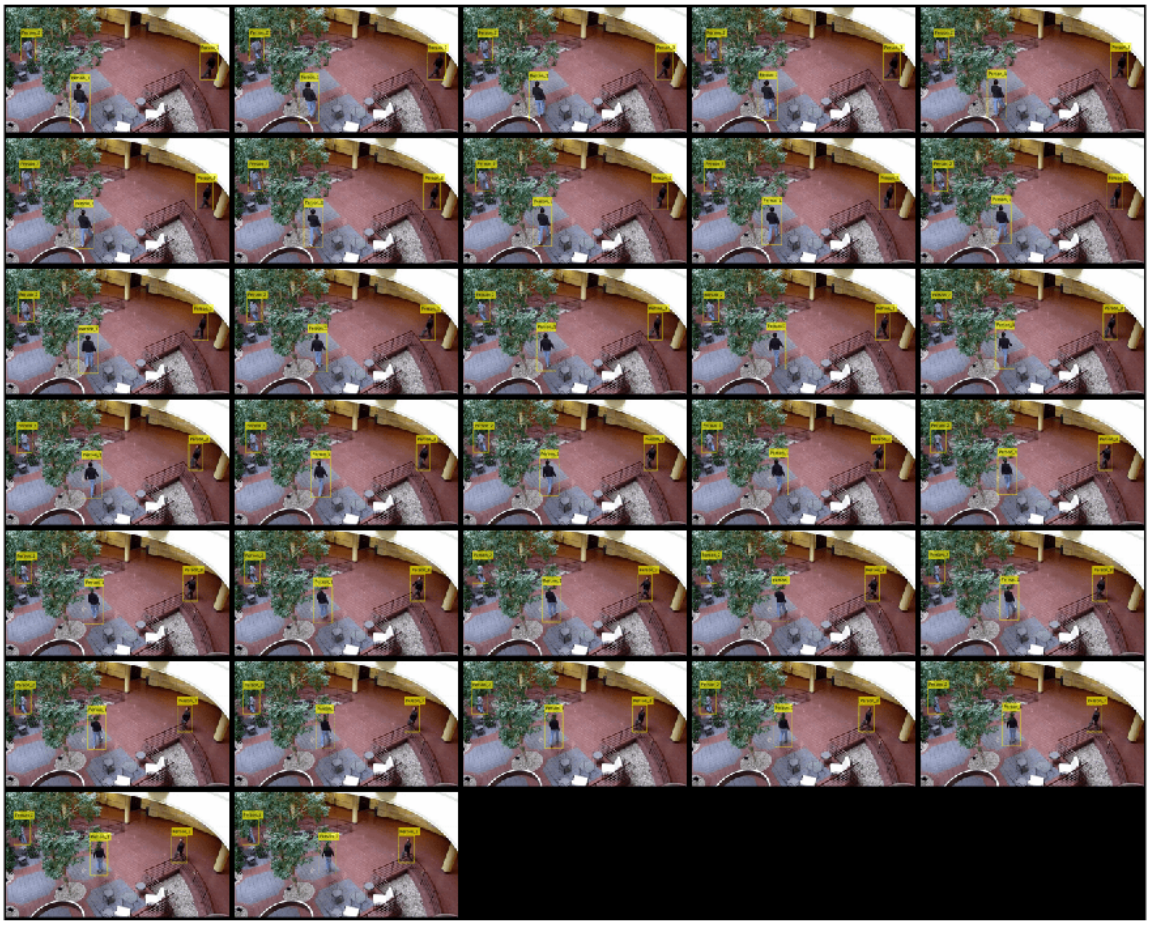
Combine the datastores using the `combine` function.

```
ds = combine(imds,blds);
```

Visualize the written video image frames using the `montage` function.

```
frameData = {};
while hasdata(ds)
    data = read(ds);
    % Only visualize frames with box data.
    if ~isempty(data{2})
        frameData{end + 1} = insertObjectAnnotation(data{1},"Rectangle",data{2},data{3});
    end
end

figure
montage(frameData,BorderSize=3)
```



Input Arguments

`gTruth` — Ground truth labels
`groundTruth` object | array of `groundTruth` objects

Ground truth labels, specified as a `groundTruth` object or as an array of `groundTruth` objects.

`location` — Destination folder
string scalar | character vector

Destination folder to which to write the video frames, specified as a string scalar or character vector.

`timestamps` — Timestamps
cell array of `duration` vectors

Timestamps, specified as an *M*-by-1 cell array of `duration` vectors, where *M* is the number of `groundTruth` objects in `gTruth`. To use this argument, you must specify an input ground truth object that contains duration-based data. If the input ground truth object does not have duration-based data, the function ignores the timestamps argument and writes all image data from the ground truth object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `writeFrames(gTruth,location,ImageFormat="jpg")` sets the returned video frames to the JPEG format.

`ImageFormat` — Image format
character vector | string scalar

Image format, specified as a character vector or string scalar. This value specifies the image format for the returned video frames `files`. You must specify a file format supported by `imwrite`.

`NamePrefix` — Prefix for output filenames
string scalar | character vector

Prefix for output filenames, specified as a string scalar or character vector. The output image files use this naming convention::

`NamePrefix`*_source_number_image_number*`.ImageFormat`, where,

- `NamePrefix` is the filename prefix, specified by the `NamePrefix` name-value argument.
- *source_number* corresponds to the ground truth object. For example, if the input `gTruth` is an array of three ground truth objects, the function outputs files with three different *source_number* values, `1`, `2`, and `3`, corresponding to the index of each ground truth object in the array.
- *image_number* is the image or video frame number.

The default value `NamePrefix` depends on the source of the data.

- Video or custom data source — `NamePrefix` uses the name of the data source, `sourceName`.

- Image datastore — `NamePrefix` uses the value `"datastore"`.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects contain datastores and use the default `read` functions.

`UseParallel` — Run parallel computations
`false` or `0` (default) | `true` or `1`

Run parallel computations , specified as a numeric or logical `1` (`true`) or `0` (`false`).

To run in parallel, set `UseParallel` to `true`, or enable parallel computation by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

`Verbose` — Flag to display training progress
`true` or `1` (default) | `false` or `0`

Flag to display training progress at the MATLAB command line, specified as a numeric or logical `1` (`true`) or `0` (`false`). This argument applies only for `groundTruth` objects created using a video file or a custom data source.

Output Arguments

`files` — File paths of written frames
M-by-1 cell array

File paths of written frames, returned as an *M*-by-1 cell array. *M* is the number of ground truth objects. Each cell contains an *N*-element vector of strings, where *N* is the number of image frames written to the specified folder. Each string specifies the full path of an output file. The output files use this naming convention:

`NamePrefix`*_source_number_image_number*.`ImageFormat`, where,

- `NamePrefix` is the filename prefix, specified by the `NamePrefix` name-value argument.
- *source_number* corresponds to the ground truth object. For example, if the input `gTruth` is an array of three ground truth objects, the function outputs files with three different *source_number* values, `1`, `2`, and `3`, corresponding to the index of each ground truth object in the array.
- *image_number* is the image or video frame number.

Version History

Introduced in R2025a

See Also

Functions
`groundTruth` | `gatherLabelData` | `objectDetectorTrainingData` | `pixelLabelTrainingData`

Topics
“Reidentify People Throughout a Video Sequence Using ReID Network”
“Convert Ground Truth Labeling Data for Object Re-Identification”

merge

Merge two or more ground truth objects

Syntax

```
gTruth = merge(gTruth1,gTruth2, ... gTruthN)
gTruth = merge( ___ ,OutputFolder=name)
```

Description

`gTruth = merge(gTruth1,gTruth2, ... gTruthN)` merges two or more ground truth objects into a new ground truth object `gTruth`. The resulting merged ground truth object contains all the data sources, label definitions, and label data from the input ground truth objects.

`gTruth = merge( ___ ,OutputFolder=name)` specifies the folder into which to save merged pixel label data in addition to the input arguments from the previous syntax. `name` must be the absolute or relative path to the save folder, specified as a character vector or string scalar. By default, the function writes merged pixel label data to a folder named `"mergedPixelLabelData"` within the current working directory. If this folder does not exist, the function creates it. If the folder already exists, the function adds a unique numeric suffix to the default name, for example `"mergedPixelLabelData_1"`, and creates a new folder.

Examples

Merge Two Ground Truth Objects

Load two separate ground truth objects into the workspace from a previously saved MAT file.

```
load("groundTruthData.mat","gTruth1","gTruth2");
```

Merge the two objects into one ground truth object. Use the logical output hasModifiedLabels, to specify any modified label data.

```
[gTruth] = merge(gTruth1,gTruth2);
```

Review merged ground truth in the Image Labeler app.

```
imageLabeler(gTruth)
```

Input Arguments

`gTruth1,gTruth2, ... gTruthN` — Ground truth data
`groundTruth` objects

Ground truth data, specified as two or more `groundTruth` objects. If any of the ground truth objects specifies a `groundTruthDataSource` or an `imageDatastore` with a custom `read` function, then all of the input ground truth objects must use the same custom `read` function.

Output Arguments

gTruth — Merged ground truth object
groundTruth object | array of groundTruth objects

Merged ground truth data, returned as a groundTruth object or an array of groundTruth objects. The function returns an array of groundTruth objects when you merge groundTruth objects that correspond to multiple video or image sequence sources. The object merges the label definitions, so that the output groundTruth contains the same label.

The merged labels obtain their description and color values from the first (of the input order) ground truth object that defines a non-empty value. The function automatically selects a new color value when required.

The method that the function uses to merge label data depends on the label type:

- Region-of-interest (ROI) — Merged by using concatenation. ROI labels include labelType.Rectangle, labelType.RotatedRectangle, labelType.Line, labelType.Point, labelType.ProjectedCuboid, labelType.Cuboid, and labelType.Polygon.

  The function merges ROIs that do not have attributes or sublabels, and contain the same position value, into one ROI. The function does not preserve the stacking order of ROI labels when merging ROI label data from ground truth objects that reference the same image.
- Scene — Merged by using majority consensus. In the case of a tie, the output uses the scene label data from the first input ground truth object.
- Pixel — Merged by using a per-pixel majority consensus. In the case of a tie, the output uses the pixel label value from the first input ground truth. This merge ignores unlabeled pixels. The function saves the merged pixel label data as PNG-format files, by using the naming convention:

  "Label_<*sourceIndex*>_<*sourceName*>.png", where <*sourceIndex*> is an index into the output ground truth object DataSource property, and <*sourceName*> is the filename of the data source.

  The object merges pixel label data to separate subfolders when the input groundTruth objects reference:

  - More than one unique video or image sequence source.
  - Image file sources and video or image sequence sources.

  The subfolders are named gTruth_<idx> and contains pixel label data that correspond to the gTruth(idx) output. These subfolders are located in the folder specified by OutputFolder.
- Custom — The merge function does not support custom labels.

Version History

Introduced in R2023b

See Also

groundTruth | boxLabelDatastore

vision.labeler.AutomationAlgorithm Class

Namespace: `vision.labeler`

Interface for algorithm automation in ground truth labeling

Description

The `vision.labeler.AutomationAlgorithm` class specifies the interface for defining custom automation algorithms to run in the Image Labeler, Video Labeler, and Ground Truth Labeler apps. Classes that inherit from the `AutomationAlgorithm` interface can be used with the automation workflow of the labeling apps to generate ground truth labels.

You can also use a custom function that creates an automation algorithm. The function, which you can specify in the labeling apps, enables you to adjust the automation parameters. For more details, see Create Automation Algorithm Function for Labeling.

The `vision.labeler.AutomationAlgorithm` class is a `handle` class.

Class Attributes

`Abstract`	`true`

For information on class attributes, see “Class Attributes”.

Properties

The `AutomationAlgorithm` class predefines this set of properties.

`GroundTruth` — Ground truth data

`groundTruth` object | `groundTruthMultisignal` object

Ground truth data, specified as a `groundTruth` or `groundTruthMultisignal` object. This property holds all the labels in the labeling app prior to automation.

- For automation algorithms used in the Image Labeler or Video Labeler app, this property must be a `groundTruth` object.
- For automation algorithms used in the Ground Truth Labeler app, this property must be a `groundTruthMultisignal` object.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`SelectedLabelDefinitions` — Selected label definitions

structure

Label definitions selected for automation in the labeling app, specified as a structure. The labeling apps support selection of only one labeling definition per automation session. In the labeling apps, the selected label definition is highlighted in yellow in either the **ROI Labels** or **Scene Labels** pane on the left.

The structure contains these fields.

Field	Description
`Type`	`labelType` enumeration that contains the type of label definition. Valid label types are: • `labelType.Rectangle` • `labelType.Polygon` • `labelType.Cuboid` (for point clouds) • `labelType.ProjectedCuboid` (for image and video data) • `labelType.Line` • `labelType.PixelLabel` • `labelType.Scene` `labelType.Custom` is not supported.
`Name`	Character vector that contains the name of the label definition.
`Attributes` (optional)	Structure array that contains one structure for each attribute in the label definition. If the label definition does not contain attributes, then this property does not include the `Attributes` field. The first field of each attribute structure in this structure array contains the attribute name. The second field contains a structure of values that are associated with that name. If you are defining a `List` attribute, you must also define the list of values for that attribute. Values for `Numeric Value`, `String`, or `Logical` attributes are optional. Descriptions for the attributes are optional for all cases.
`PixelLabelID` (optional)	Positive integer that contains the pixel label ID for the label definition. This `PixelLabelID` field applies only for label definitions of type `PixelLabel`.

To view a sample `SelectedLabelDefinitions` structure that contains an attribute, enter this code at the MATLAB command prompt.

```
selectedLabelDefs.Type = labelType.Rectangle;
selectedLabelDefs.Name = 'Car';
selectedLabelDefs.Attributes = struct('distance', ...
    struct('DefaultValue',0,'Description','Sensor distance'))
```

To view a sample `SelectedLabelDefinitions` structure that contains pixel label definitions, enter this code at the MATLAB command prompt.

```
selectedLabelDefs.Type = labelType.PixelLabel;
selectedLabelDefs.Name = 'Car';
selectedLabelDefs.Attributes = struct('distance', ...
    struct('DefaultValue',0,'Description','Sensor distance'))
```

Attributes:

```
GetAccess                                       public
SetAccess                                       private
```

`ValidLabelDefinitions` — Valid label definitions
structure array

Valid label definitions that the algorithm can automate, specified as a structure array. Each structure in the array corresponds to a valid label definition. To determine which label definitions are valid, the app uses the `checkLabelDefinition` method. This table describes the fields for each valid label definition structure.

Field	Description
`Type`	`labelType` enumeration that contains the type of label definition. Valid label types are: • `labelType.Rectangle` • `labelType.Polygon` • `labelType.Cuboid` (for point clouds) • `labelType.ProjectedCuboid` (for image and video data) • `labelType.Line` • `labelType.PixelLabel` • `labelType.Scene` `labelType.Custom` is not supported.
`Name`	Character vector that contains the name of the label definition.
`Attributes` (optional)	Structure array that contains one structure for each attribute in the label definition. If the label definition does not contain attributes, then this property does not include the `Attributes` field. The first field of each attribute structure in this structure array contains the attribute name. The second field contains a structure of values that are associated with that name. If you are defining a `List` attribute, you must also define the list of values for that attribute. Values for `Numeric Value`, `String`, or `Logical` attributes are optional. Descriptions for the attributes are optional for all cases.
`PixelLabelID` (optional)	Positive integer that contains the pixel label ID for the label definition. This `PixelLabelID` field applies only for label definitions of type `PixelLabel`.

To view a sample `ValidLabelDefinitions` structure that contains an attribute, enter this code at the MATLAB command prompt.

```
validLabelDefs(1).Type = labelType.Rectangle;
validLabelDefs(1).Name = 'Car';
validLabelDefs(2).Type = labelType.Line;
validLabelDefs(2).Name = 'LaneMarker';
validLabelDefs(3).Type = labelType.Scene
validLabelDefs(3).Name = 'Sunny';
```

To view a sample `ValidLabelDefinitions` structure that contains pixel label definitions, enter this code at the MATLAB command prompt.

```
validLabelDefs(1).Type = labelType.PixelLabel;
validLabelDefs(1).Name = 'Road';
validLabelDefs(1).PixelLabelID  = 1;
validLabelDefs(2).Type = labelType.PixelLabel;
validLabelDefs(2).Name = 'Sky';
validLabelDefs(2).PixelLabelID = 2
```

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`CurrentIndex` — Index of current frame

integer

Index of current frame, specified as an integer. The object updates `CurrentIndex` while the algorithm runs. Use this index value to access the current frame ground truth data from the `GroundTruth` property.

Clients of the `AutomationAlgorithm` class are required to define this set of properties. These properties set up the name, description, and user instructions for your automated algorithm.

`Name` — Automation algorithm name

character vector

Automation algorithm name, specified as a character vector.

Attributes:

`GetAccess`	`public`
`Abstract`	`true`
`Constant`	`true`
`NonCopyable`	`true`

`Description` — Automation algorithm description

character vector

Algorithm description, specified as a character vector.

Attributes:

`GetAccess`	`public`
`Abstract`	`true`
`Constant`	`true`
`NonCopyable`	`true`

`UserDirections` — Algorithm directions displayed in app

cell array

Algorithm directions displayed in app, specified as a cell array. `UserDirections` are specified as a `cellstr`, with each string representing a separate direction. Use the `checkSetup` method to verify that the directions have been adhered to.

Attributes:

`GetAccess`	`public`
`Abstract`	`true`
`Constant`	`true`
`NonCopyable`	`true`

Methods

Public Methods

Clients of an `AutomationAlgorithm` implement these user-defined functions to define execution of the algorithm. For more information on creating your own automation algorithm, see “Create Custom Automation Algorithm for Labeling”.

checkLabelDefinition	Validate label definition
checkSignalType	Validate signal type
supportsMultisignalAutomation	Set multisignal algorithm automation flag
checkSetup	Set up validation (optional)
initialize	Initialize state for algorithm execution (optional)
run	Run label automation on every frame in interval
terminate	Terminate automated algorithm (optional)
settingsDialog	Display algorithm settings (optional)

Version History

Introduced in R2017a

See Also

Apps

Image Labeler | Ground Truth Labeler | Video Labeler

Functions

`groundTruth` | `groundTruthMultisignal` | `labelType` | `vision.labeler.mixin.Temporal` | `vision.labeler.mixin.BlockedImageAutomation`

Topics

“Create Custom Automation Algorithm for Labeling”
Create Automation Algorithm Function for Labeling
“Temporal Automation Algorithms”
“Blocked Image Automation Algorithms”
“Automate Ground Truth Labeling of Lane Boundaries” (Automated Driving Toolbox)
“Automate Ground Truth Labeling for Semantic Segmentation” (Automated Driving Toolbox)
“Automate Attributes of Labeled Objects” (Automated Driving Toolbox)

checkLabelDefinition

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Validate label definition

Syntax

```
isValid = checkLabelDefinition(algObj,labelDef)
```

Description

In the labeling apps, the `checkLabelDefinition` method checks whether each label defined in the **ROI Labels** and **Scene Labels** panes is valid. The method restricts an automation algorithm to use only relevant labels. For example, a label definition of type `Rectangle` cannot be used to mark a lane boundary.

Clients of `AutomationAlgorithm` must implement this method.

`isValid = checkLabelDefinition(algObj,labelDef)` returns `true` for valid label definitions and `false` for invalid definitions for the automation algorithm provided by `algObj`. `labelDef` is a structure containing all the label definitions in the **ROI Labels** and **Scene Labels** panes. Definitions that return `false` are disabled during automation.

Examples

Restrict Automation to Rectangular ROI Labels

This implementation of the `checkLabelDefinition` method designates `Rectangle` labels as valid and all other labels as invalid.

```
function isValid = checkLabelDefinition(algObj,labelDef)
    isValid = (labelDef.Type == labelType.Rectangle);
end
```

Restrict Automation to ROI Labels of Any Type

This implementation of the `checkLabelDefinition` method designates ROI labels such as `Rectangle` and `Line` as valid and all other labels as invalid.

```
function isValid = checkLabelDefinition(algObj,labelDef)
    isValid = isROI(labelDef.Type);
end
```

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

`labelDef` — Label definition
structure

Label definition, specified as a structure containing `Type` and `Name` fields.

Field	Description
`Type`	`labelType` enumeration that contains the type of label definition. Valid label types are: • `labelType.Rectangle` • `labelType.Cuboid` (for point clouds) • `labelType.ProjectedCuboid` (for image and video data) • `labelType.Line` • `labelType.PixelLabel` • `labelType.Scene` `labelType.Custom` is not supported.
`Name`	Character vector that contains the name of the label definition.

To view a sample `labelDef` structure that contains a rectangle label definition, enter this code at the MATLAB command prompt.

```
labelDef(1).Type = labelType.Rectangle;
labelDef(1).Name = 'Car';
```

Output Arguments

`isValid` — True of false result of label definition validity check
`1` | `0`

True or false result of the label definition validity check, returned as a `1` or `0` of data type `logical`.

Tips

- To access the selected label definitions, use the SelectedLabelDefinitions property of the automation algorithm. In the labeling apps, the selected label definitions are highlighted in yellow in the **ROI Labels** and **Scene Labels** panes on the left.

Version History

Introduced in R2017a

See Also

`labelType` | `vision.labeler.AutomationAlgorithm` | `checkSignalType` | `checkSetup`

checkSetup

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Set up validation (optional)

Syntax

```
isReady = checkSetup(algObj)
isReady = checkSetup(algObj,labelsToAutomate)
```

Description

In the labeling apps, the `checkSetup` method checks the validity of the setup when you click **Run** in an automation session. If `checkSetup` returns `true`, then the setup is valid and the app proceeds to run the automation algorithm by using the `initialize`, `run`, and `terminate` methods.

Clients of `AutomationAlgorithm` can optionally implement this method.

`isReady = checkSetup(algObj)` returns `true` if you completed set up correctly and the automation algorithm `algObj` can begin execution. Otherwise, `checkSetup` returns `false`.

`isReady = checkSetup(algObj,labelsToAutomate)` additionally provides a table, `labelsToAutomate`, that contains labels selected for the automation algorithm to use for labeling. This syntax does not support pixel label automation. In addition, this syntax is available only for time-dependent (temporal) automation algorithms. The Ground Truth Labeler and Video Labeler apps support temporal algorithms but the Image Labeler does not. For more information on these types of algorithms, see “Temporal Automation Algorithms”.

Examples

Check Setup for ROI Labels

This implementation of the `checkSetup` method checks the setup for a temporal automation algorithm. This method determines that an automation algorithm is ready to run if at least one ROI label exists.

```
function isReady = checkSetup(algObj,labelsToAutomate)

    notEmpty = ~isempty(labelsToAutomate);
    hasROILabels = any(labelsToAutomate.Type == labelType.Rectangle);
    isReady = (notEmpty && hasROILabels)

end
```

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

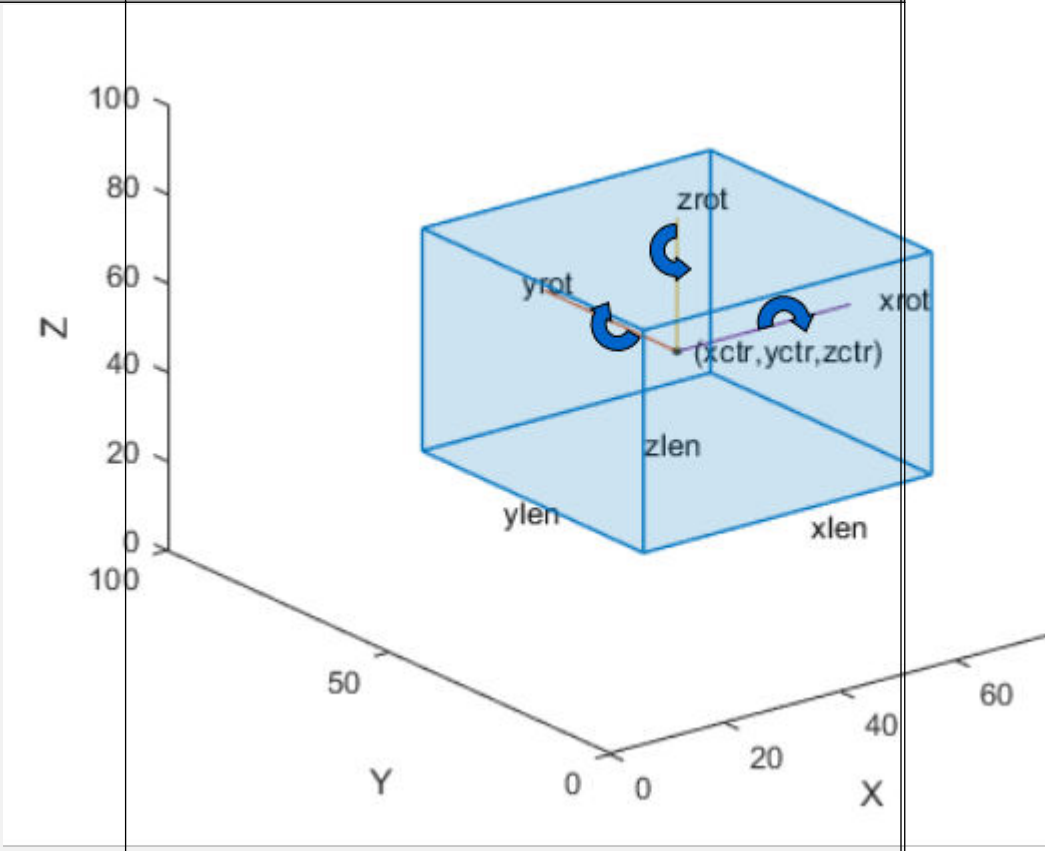
Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

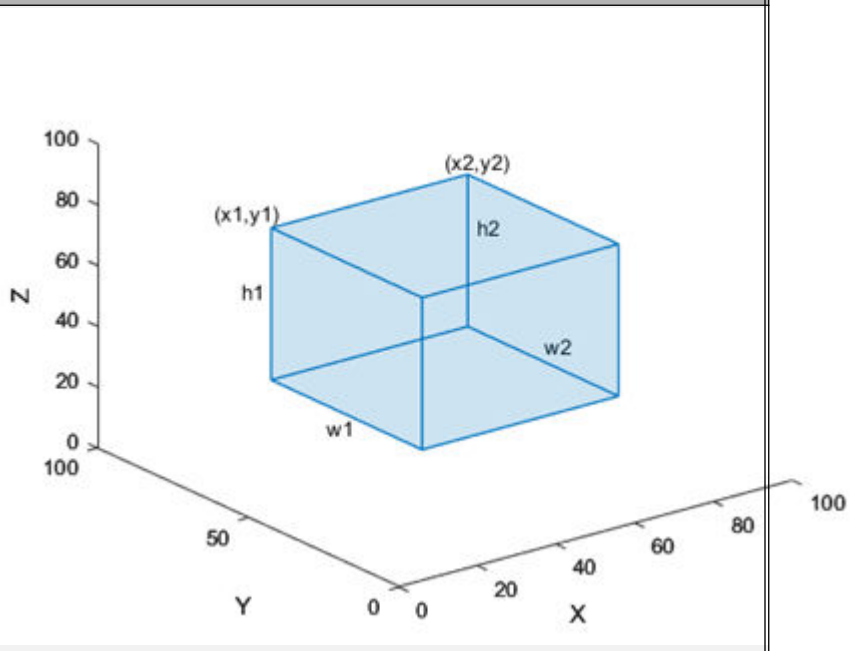
`labelsToAutomate` — Labels selected for automation
table

Labels selected for automation, specified as a table with these columns.

Column Name	Description
`Type`	`labelType` enumeration that contains the type of the label. Valid label types are: • `labelType.Rectangle` • `labelType.Cuboid` (Ground Truth Labeler app only) • `labelType.ProjectedCuboid` • `labelType.Line` • `labelType.Scene` `labelType.PixelLabel` and `labelType.Custom` are not supported.
`Name`	Character vector that contains the name of the label.
`Time`	Scalar of type `double` that specifies the time, in seconds, when the label was marked.

<table>
<tr><th>Column Name</th><th>Description</th></tr>
<tr><td>Position</td><td>Location of the label in the frame. The format of this vector depends on the label type.
<table>
<tr><th>Label Type</th><th>Position Format</th></tr>
<tr><td>Rectangle — Rectangular region of interest (ROI) labels</td><td>M-by-4 numeric vector of the form [x, y, w, h], where:
M is the number of labels in the frame.
x and y specify the upper-left corner of the rectangle.
w specifies the width of the rectangle, which is the length of the rectangle along the x-axis.
h specifies the height of the rectangle, which is the length of the rectangle along the y-axis.</td></tr>
<tr><td>Cuboid — Cuboid ROI labels</td><td>M-by-9 numeric vector of the form [xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot], where:
M is the number of labels in the frame.
xctr, yctr, and zctr specify the center of the cuboid.
xlen, ylen, and zlen specify the length of the cuboid along the x-axis, y-axis, and z-axis, respectively.
xrot, yrot, and zrot specify the rotation angles for the cuboid along the x-axis, y-axis, and z-axis, respectively. These angles are clockwise-positive when looking in the forward direction of their corresponding axes.
This figure shows how these values specify the position of a cuboid.</td></tr>
</table>
</td></tr>
</table>

Column Name	Description	
	Label Type	**Position Format**
		
	`ProjectedCuboid` — Cuboid ROI labels	`M`-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid.

Column Name	Description	
	Label Type	**Position Format**
		
	`Line` — Polyline ROI labels	`M`-by-1 vector of cell arrays, where `M` is the number of labels in the frame. Each cell array contains an `N`-by-2 numeric matrix of the form `[x1 y1; x2 y2; ... ; xN yN]` for `N` points in the polyline.
	`Scene` — Scene labels	Logical value of `1` if the label is present in the frame and `0` otherwise.

Each row of the table corresponds to a label selected for automation. This `labelsToAutomate` table contains a rectangle label, a line label with five points, and a cuboid label.

```
    Type          Name           Time        Position
_________    _____________    _________    _____________

Rectangle    'Car'            0.033333     [1x4 double]
Line         'LaneMarker'     0.066667     [5x2 double]
Cuboid       'Truck'          0.099999     [1x9 double]
```

Output Arguments

`isReady` — True or false result of setup check

`1 | 0`

True or false result of the setup check, returned as a `1` or `0` of data type `logical`.

Version History

Introduced in R2017a

See Also

`labelType` | `vision.labeler.AutomationAlgorithm` | `checkLabelDefinition` | `checkSignalType`

checkSignalType

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Validate signal type

Syntax

```
isValid = checkSignalType(signalType)
```

Description

In the labeling apps, the `checkSignalType` method validates whether each signal selected for automation supports the signal type relevant to the automation algorithm.

- Algorithms that automate the labeling of image collections, videos, or image sequences support signals of type `Image` only.
- Algorithms that automate the labeling of lidar point clouds support signals of type `PointCloud` only.

The implementation of this method depends on which labeling app the automation algorithm is being used with.

Labeling App of Automation Algorithm	`checkSignalType` Implementation
Image Labeler Video Labeler	You do not need to implement this method. By default, this method validates that the signal being automated is of type `Image`, which is the only signal type that these apps support.
Ground Truth Labeler	Update this method to validate whether the automation algorithm supports `Image` signals, `PointCloud` signals, or both types of signals.
Lidar Labeler	You do not need to implement this method. If you do implement this method, update it to validate that the signal is of type `PointCloud`, which is the only signal type that this app supports.

`isValid = checkSignalType(signalType)` returns logical `1` (true) when the specified signal type is valid. In the Ground Truth Labeling app, if you select an automation algorithm, select signals that are of an invalid type, and then click **Automate**, the app displays an error.

Examples

Restrict Automation to Image Signals

Implement the `checkSignalType` method to designate `Image` signals as valid and all other signals as invalid.

```
function isValid = checkSignalType(signalType)
    isValid = (signalType == vision.labeler.loading.SignalType.Image);
end
```

Restrict Automation to Point Cloud Signals

Implement the `checkSignalType` method to designate `PointCloud` signals as valid and all other signals as invalid.

```
function isValid = checkSignalType(signalType)
    isValid = (signalType == vision.labeler.loading.SignalType.PointCloud);
end
```

Enable Automation for Image and Point Cloud Signals

Implement the `checkSignalType` method to designate `Image` and `PointCloud` signals as valid.

```
function isValid = checkSignalType(signalType)
    isValid = any(signalType == vision.labeler.loading.SignalType.Image) && ...
    any(signalType == vision.labeler.loading.SignalType.PointCloud);
end
```

Input Arguments

`signalType` — Signal type

`vision.labeler.loading.SignalType` enumeration

Signal type, specified as a `vision.labeler.loading.SignalType` enumeration.

Example: `vision.labeler.loading.SignalType.Image`

Example: `vision.labeler.loading.SignalType.PointCloud`

Output Arguments

`isValid` — Result of signal type validity check

`1 | 0`

Result of the signal type validity check, returned as logical `1` (true) or logical `0` (false).

Attributes

`Static`	`true`

To learn about attributes of methods, see Method Attributes.

Version History

Introduced in R2020a

See Also

`vision.labeler.loading.SignalType` | `checkLabelDefinition` | `vision.labeler.AutomationAlgorithm` | `checkSetup`

initialize

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Initialize state for algorithm execution (optional)

Syntax

```
initialize(algObj,frame)
initialize(algObj,frame,labelsToAutomate)
```

Description

The `initialize` method initializes the state of the automation algorithm before the automation algorithm runs.

Clients of `AutomationAlgorithm` can optionally implement this method.

`initialize(algObj,frame)` initializes the state of the `algObj` automation algorithm using the first frame in the time range of the data being labeled.

Clients of `AutomationAlgorithm` must implement this user-defined method.

`initialize(algObj,frame,labelsToAutomate)` additionally provides a table, `labelsToAutomate`, that contains labels selected for the automation algorithm to use for labeling. This syntax does not support pixel label automation. In addition, this syntax is available only for time-dependent (temporal) automation algorithms. The Ground Truth Labeler (requires Automated Driving Toolbox™) and Video Labeler apps support temporal algorithms but the Image Labeler does not. For more information on these types of algorithms, see “Temporal Automation Algorithms”.

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

`frame` — Frame corresponding to start of time range

numeric matrix | `pointCloud` object

Frame corresponding to the start of time range, specified as a numeric matrix for `Image` signals or a `pointCloud` object for `PointCloud` signals.

`labelsToAutomate` — Labels selected for automation

table

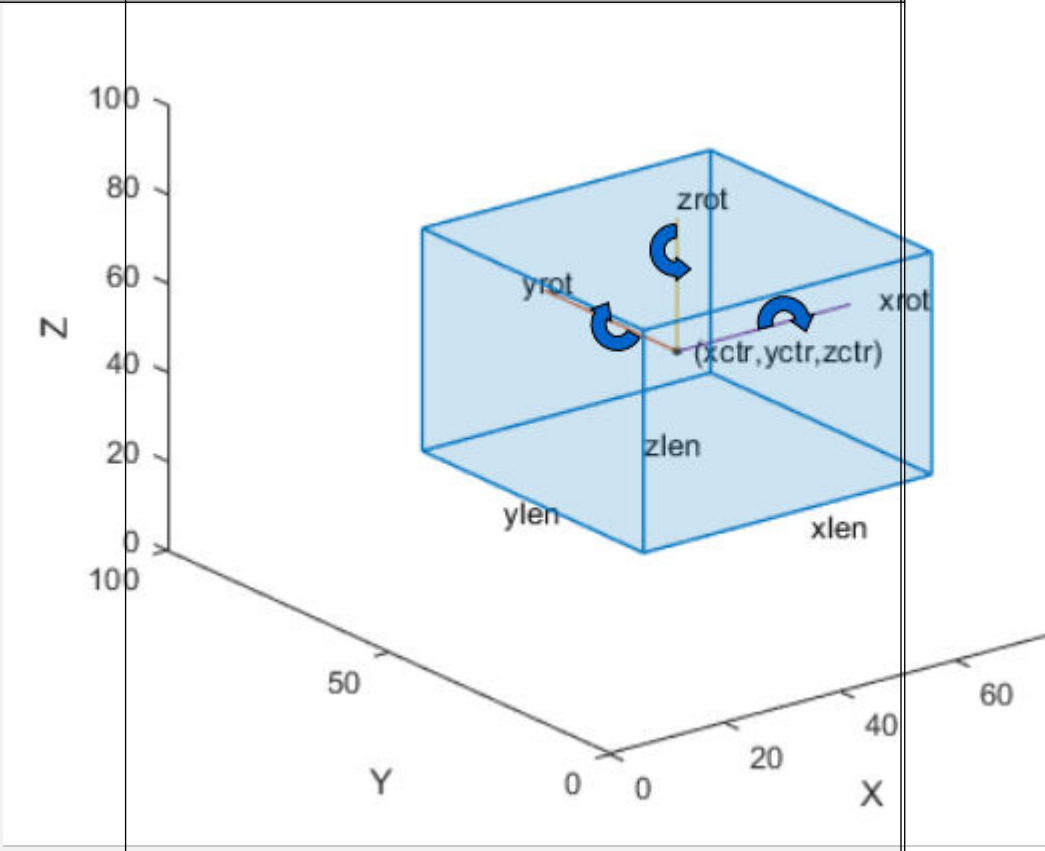
Labels selected for automation, specified as a table with these columns.

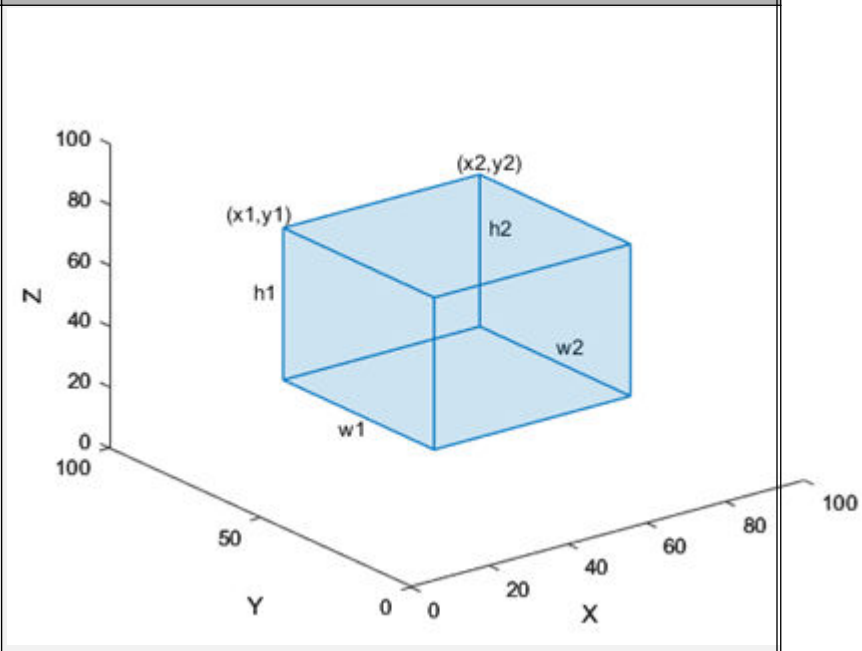
Column Name	Description
`Type`	`labelType` enumeration that contains the type of the label. Valid label types are: • `labelType.Rectangle` • `labelType.Cuboid` (Ground Truth Labeler app only) • `labelType.ProjectedCuboid` • `labelType.Line` • `labelType.Scene` `labelType.PixelLabel` and `labelType.Custom` are not supported.
`Name`	Character vector that contains the name of the label.
`Time`	Scalar of type `double` that specifies the time, in seconds, when the label was marked.

<table>
<tr><th>Column Name</th><th>Description</th></tr>
<tr><td><code>Position</code></td><td>Location of the label in the frame. The format of this vector depends on the label type.

<table>
<tr><th>Label Type</th><th>Position Format</th></tr>
<tr><td><code>Rectangle</code> — Rectangular region of interest (ROI) labels</td><td><code>M</code>-by-4 numeric vector of the form <code>[x, y, w, h]</code>, where:
• <code>M</code> is the number of labels in the frame.
• <code>x</code> and <code>y</code> specify the upper-left corner of the rectangle.
• <code>w</code> specifies the width of the rectangle, which is the length of the rectangle along the x-axis.
• <code>h</code> specifies the height of the rectangle, which is the length of the rectangle along the y-axis.</td></tr>
<tr><td><code>Cuboid</code> — Cuboid ROI labels</td><td><code>M</code>-by-9 numeric vector of the form <code>[xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot]</code>, where:
• <code>M</code> is the number of labels in the frame.
• <code>xctr</code>, <code>yctr</code>, and <code>zctr</code> specify the center of the cuboid.
• <code>xlen</code>, <code>ylen</code>, and <code>zlen</code> specify the length of the cuboid along the x-axis, y-axis, and z-axis, respectively.
• <code>xrot</code>, <code>yrot</code>, and <code>zrot</code> specify the rotation angles for the cuboid along the x-axis, y-axis, and z-axis, respectively. These angles are clockwise-positive when looking in the forward direction of their corresponding axes.

This figure shows how these values specify the position of a cuboid.</td></tr>
</table>
</td></tr>
</table>

Column Name	Description	
	Label Type	**Position Format**
		
	`ProjectedCuboid` — Cuboid ROI labels	`M`-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid.

<table>
<tr><th>Column Name</th><th colspan="2">Description</th></tr>
<tr><td></td><td>Label Type</td><td>Position Format</td></tr>
<tr><td></td><td></td><td>
</td></tr>
<tr><td></td><td><code>Line</code> — Polyline ROI labels</td><td><code>M</code>-by-1 vector of cell arrays, where <code>M</code> is the number of labels in the frame. Each cell array contains an <code>N</code>-by-2 numeric matrix of the form <code>[x1 y1; x2 y2; ... ; xN yN]</code> for <code>N</code> points in the polyline.</td></tr>
<tr><td></td><td><code>Scene</code> — Scene labels</td><td>Logical value of <code>1</code> if the label is present in the frame and <code>0</code> otherwise.</td></tr>
</table>

Each row of the table corresponds to a label selected for automation. This `labelsToAutomate` table contains a rectangle label, a line label with five points, and a cuboid label.

```
    Type          Name           Time        Position
    _________     ____________   _________   _____________

    Rectangle     'Car'          0.033333    [1x4 double]
    Line          'LaneMarker'   0.066667    [5x2 double]
    Cuboid        'Truck'        0.099999    [1x9 double]
```

Version History

Introduced in R2017a

See Also

`checkSetup` | `run` | `terminate` | `labelType` | `vision.labeler.AutomationAlgorithm`

run

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Run label automation on every frame in interval

Syntax

```
autoLabels = run(algObj,frame)
```

Description

The `run` method computes the automated labels for a single frame by executing the automation algorithm. During automation, the labeling apps run this method in a loop to compute the automated labels for each frame in the selection being automated. A single frame can contain multiple signals such as images and point clouds.

Clients of `AutomationAlgorithm` must implement this method.

`autoLabels = run(algObj,frame)` processes a single frame, `frame`, using the `algObj` automation algorithm, and returns the automated labels, `autoLabels`.

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

`frame` — Frame

numeric matrix | `pointCloud` object | *M* X *1* cell array

Frame whose labels are being computed, specified as a numeric image matrix, a `pointCloud` object or an *M* X *1* cell array of multiple image matrices and `pointCloud` objects. To specify a cell array of multiple signals, `algObj` must support multisignal automation.

Output Arguments

`autoLabels` — Labels produced by automation

categorical matrix | structure array | table

Labels produced by the automation algorithm, returned as a categorical matrix, structure array, or table.

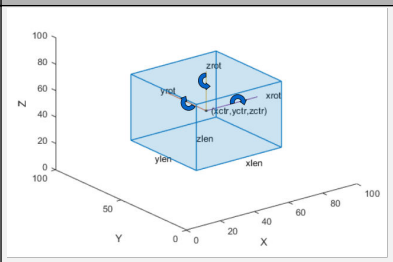
For algorithms that automate pixel labeling, implement the `run` method to return `autoLabels` as a `categorical` label matrix, where each category represents a pixel label. For more information, see “How Labeler Apps Store Exported Pixel Labels”.

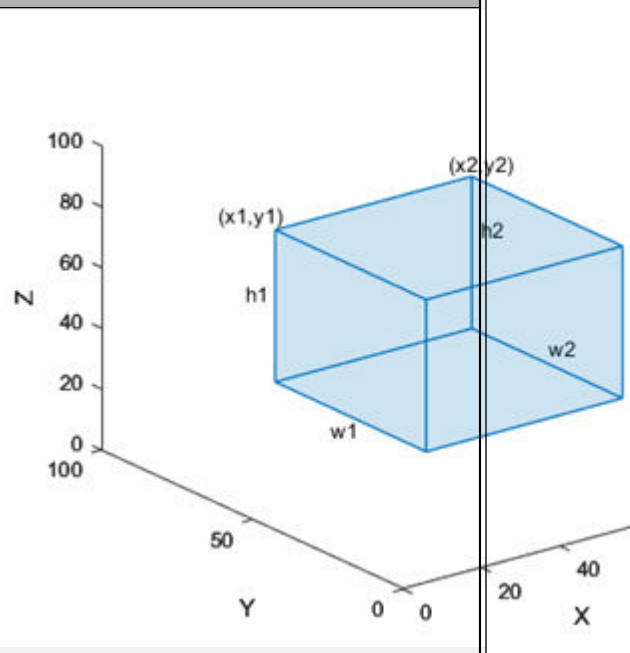
For algorithms that automate nonpixel labels, implement the `run` method to return a structure array. Each structure in the array contains the labels of a specific name and type. The method combines labels of the same name and type into a single structure in the array.

This table describes the columns of the `autoLabels` table or fields of each `autoLabels` structure.

Field Name	Description
`Type`	`labelType` enumeration that contains the type of the label. Valid label types are: • `labelType.Rectangle` • `labelType.Cuboid` (Ground Truth Labeler app only) • `labelType.ProjectedCuboid` • `labelType.Polygon` • `labelType.Line` • `labelType.Scene`
`Name`	Character vector, string scalar, or categorical scalar containing the name of the label.

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td><code>Position</code></td><td>Position of labels of the specified <code>Name</code> and <code>Type</code>. The format of <code>Position</code> depends on the label type.
<table>
<tr><th>Label Type</th><th>Position Format</th></tr>
<tr><td><code>Rectangle</code> — Rectangular region of interest (ROI) labels</td><td><code>M</code>-by-4 numeric matrix of the form <code>[x, y, w, h]</code>, where:
<code>M</code> is the number of labels in the frame.
<code>x</code> and <code>y</code> specify the upper-left corner of the rectangle.
<code>w</code> specifies the width of the rectangle, which is its length along the x-axis.
<code>h</code> specifies the height of the rectangle, which is its length along the y-axis.</td></tr>
<tr><td><code>Cuboid</code> — Cuboid ROI labels</td><td><code>M</code>-by-9 numeric matrix with rows of the form <code>[xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot]</code>, where:
<code>M</code> is the number of labels in the frame.
<code>xctr</code>, <code>yctr</code>, and <code>zctr</code> specify the center of the cuboid.
<code>xlen</code>, <code>ylen</code>, and <code>zlen</code> specify the length of the cuboid along the x-axis, y-axis, and z-axis, respectively, before rotation has been applied.
<code>xrot</code>, <code>yrot</code>, and <code>zrot</code> specify the rotation angles for the cuboid along the x-axis, y-axis, and z-axis, respectively. These angles are clockwise-positive when looking in the forward direction of their corresponding axes.
The figure shows how these values determine the position of a cuboid.</td></tr>
</table>
</td></tr>
</table>

Field Name	Description	
	Label Type	**Position Format**
		
	`ProjectedCuboid` — Cuboid ROI labels	`M`-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid.

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Label Type</th><th>Position Format</th></tr>
<tr><td></td><td>
</td></tr>
<tr><td><code>Line</code> — Polyline ROI labels</td><td><code>M</code>-by-1 vector of cell arrays, where <code>M</code> is the number of labels in the frame. Each cell array contains an <code>N</code>-by-2 numeric matrix of the form <code>[x1 y1; x2 y2; ... ; xN yN]</code> for <code>N</code> points in the polyline.</td></tr>
<tr><td><code>Polygon</code></td><td><code>M</code>-by-1 vector of cell arrays, where <code>M</code> is the number of labels. Each cell array contains an <code>N</code>-by-2 numeric matrix of the form <code>[x1 y1; x2 y2; ... ; xN yN]</code> for <code>N</code> points in the polygon.</td></tr>
<tr><td><code>Scene</code> — Scene labels</td><td>Logical value of <code>1</code> if the algorithm determines that the label is present in the frame and <code>0</code> otherwise.</td></tr>
</table>
</td></tr>
<tr><td><code>Attributes</code> (optional)</td><td>Structure array that contains one structure for each attribute in the label. If the label definition does not contain attributes, then the <code>autoLabels</code> output does not include this field.

For each structure in the <code>Attributes</code> structure array, the name of that structure is the name of the corresponding attribute. The value of the structure is the value of the corresponding attribute.</td></tr>
</table>

To view a sample `autoLabels` structure array, enter this code at the MATLAB command prompt.

```
autoLabels(1).Name      = "Car";
autoLabels(1).Type      = labelType.Rectangle;
autoLabels(1).Position  = [20 20 50 50];
```

```
autoLabels(2).Name      = "Truck";
autoLabels(2).Type      = labelType.Rectangle;
autoLabels(2).Position  = [70 70 50 50; 100 100 25 25];
```

Alternatively, for nonpixel labels, you can use the `run` method to return `autoLabels` as a table. The table rows are equivalent to the structures in a structure array. The table columns are equivalent to the structure fields. This table is equivalent to the sample `autoLabels` structure array previously specified.

```
      Name          Type         Position
    ____________    _________    ____________

    "Car"           Rectangle    [1x4 double]
    "Truck"         Rectangle    [2x4 double]
```

If you want to return no labels for all label definitions, specify an empty `autoLabels` structure. If you do not want to return any labels for a specific label definition, specify an empty `Position` field for the corresponding label definition in the `autoLabels` structure.

Version History

Introduced in R2017a

See Also

Objects

`labelType` | `vision.labeler.AutomationAlgorithm`

Functions

`checkSetup` | `initialize` | `terminate`

settingsDialog

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Display algorithm settings (optional)

Syntax

```
settingsDialog(algObj)
```

Description

The `settingsDialog` method runs when the user clicks Settings in the labeling app. Use this method to provide a dialog figure with controls for user settings required for the algorithm. Use a modal dialog, created using functions like `dialog`, `inputdlg` or `listdlg`.

Clients of `AutomationAlgorithm` can optionally implement this method.

`settingsDialog(algObj)` displays automated algorithm settings in a dialog.

Input Arguments

`algObj` — Automation algorithm
`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

Version History

Introduced in R2017a

supportsMultisignalAutomation

Namespace: `vision.labeler`

Set multisignal algorithm automation flag

Syntax

```
success = supportsMultisignalAutomation(algObj)
```

Description

`success = supportsMultisignalAutomation(algObj)` indicates whether the automation algorithm `algObj` supports the automation of multiple signals in a single automation session. Implement this method in automation algorithms developed for the Ground Truth Labeler app, which supports the labeling and automation of multiple signals. If the algorithm supports multisignal automation, then this method returns `success` as `true`.

In automation algorithms developed for the Image Labeler, Video Labeler, and Lidar Labeler apps, which support the labeling and automation of only one signal at a time, you can either delete this method or leave it unchanged. The default implementation of this method indicates that the automation algorithm does not support multisignal automation (`success = false`).

Examples

Indicate Algorithm Supports Multisignal Automation

Implement the `supportsMultisignalAutomation` method to indicate that the automation algorithm supports multisignal automation. This method is static and does not use the input automation algorithm, `algObj`. Therefore, you can specify the input argument as unused by using the tilde (~) operator.

```
function success = supportsMultisignalAutomation(~)
      success = true;
end
```

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

Version History

Introduced in R2021a

See Also

Apps
Ground Truth Labeler

Objects
`vision.labeler.AutomationAlgorithm`

terminate

Class: `vision.labeler.AutomationAlgorithm`
Namespace: `vision.labeler`

Terminate automated algorithm (optional)

Syntax

```
terminate(algObj)
```

Description

The `terminate` method cleans up the state of the automation algorithm after `run` processes the last frame in the specified interval or when you stop the automation algorithm.

Clients of `AutomationAlgorithm` can optionally implement this method.

`terminate(algObj)` cleans up the state of the automation algorithm.

Input Arguments

`algObj` — Automation algorithm
`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

Version History

Introduced in R2017a

See Also

`checkSetup` | `initialize` | `run` | `vision.labeler.AutomationAlgorithm`

vision.labeler.mixin.Temporal Class

Namespace: `vision.labeler.mixin`

Mixin interface for adding temporal context to automation algorithms

Description

The `vision.labeler.mixin.Temporal` class provides an interface for attaching temporal properties to an automation algorithm. You can add this class only to automation algorithms used by the Ground Truth Labeler or Video Labeler app.

The `vision.labeler.mixin.Temporal` class is a `handle` class.

Class Attributes

`Abstract`	`true`

For information on class attributes, see “Class Attributes”.

Properties

`StartTime` — Timestamp of first frame

scalar

Timestamp of the first frame of the algorithm interval, specified as a scalar.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`Dependent`	`true`
`NonCopyable`	`true`

`CurrentTime` — Timestamp of current executing frame

scalar

Timestamp of the current executing frame, specified as a scalar. This value updates during the execution of the algorithm.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`

`EndTime` — Timestamp of last frame

scalar

Timestamp of the last frame of the algorithm interval, specified as a scalar.

Attributes:

GetAccess	public
SetAccess	private
Dependent	true
NonCopyable	true

`StartFrameIndex` — Index of first frame

integer

Index of the first frame of the algorithm interval, specified as an integer.

Attributes:

GetAccess	public
SetAccess	private
Dependent	true
NonCopyable	true

`EndFrameIndex` — Index of last frame

integer

Index of the last frame of the algorithm interval, specified as an integer.

Attributes:

GetAccess	public
SetAccess	private
Dependent	true
NonCopyable	true

`AutomationDirection` — Direction in which to run automated algorithm

`'Forward'` | `'Reverse'`

Direction in which to run the automated algorithm, specified as `'Forward'` or `'Reverse'`.

Attributes:

GetAccess	public
SetAccess	private

Methods

Public Methods

supportsReverseAutomation Set reverse algorithm automation flag

Version History

Introduced in R2017b

See Also

Apps

Ground Truth Labeler | Video Labeler

Objects
`vision.labeler.AutomationAlgorithm`

Topics
“Create Custom Automation Algorithm for Labeling”
“Temporal Automation Algorithms”

supportsReverseAutomation

Set reverse algorithm automation flag

Syntax

```
flag = supportsReverseAutomation(algObj)
```

Description

`flag = supportsReverseAutomation(algObj)` indicates whether the temporal automation algorithm, `algObj`, supports automation in the reverse direction. A `true` value enables the Ground Truth Labeler or Video Labeler to open the algorithm in reverse mode.

Examples

Set Algorithm Automation Direction Flag

```
function flag = supportsReverseAutomation(algObj)
      flag = true;
end
```

Input Arguments

`algObj` — Temporal automation algorithm

object

Temporal automation algorithm, specified as a `vision.labeler.mixin.Temporal` object.

Output Arguments

`flag` — Reverse automation indicator

`true` | `false`

Reverse automation indicator, returned as `true` or `false`.

Version History

Introduced in R2017b

See Also

Apps

Ground Truth Labeler | Video Labeler

Objects

`vision.labeler.mixin.Temporal`

vision.labeler.mixin.BlockedImageAutomation

Mixin interface for performing blocked image automation

Description

The `BlockedImageAutomation` mixin class provides an interface for attaching blocked image properties to an automation algorithm. You can add this class only to automation algorithms used by the Image Labeler app.

Creation

Description

To use the `vision.labeler.mixin.BlockedImageAutomation` object, open the Image Labeler app, load a blocked image, and create at least one label. Then, click **Select Algorithm** in the **Automate Labeling** section of the app toolstrip. Select **Add Blocked Image Algorithm** and then choose **Create New Blocked Image Algorithm**. A class template opens in the editor that inherits from both the `vision.labeler.AutomationAlgorithm` class and the `vision.labeler.mixin.BlockedImageAutomation` mixin class. Edit this template to create your blocked image automation algorithm.

Properties

The `BlockedImageAutomation` mixin class predefines this set of properties.

`BatchSize` — Number of blocks supplied to the `apply` function at a time
numeric scalar

Number of blocks supplied to the `apply` function at a time, specified as an integer valued vector of length equal to the `NumDimensions` property of the blocked image.

`BorderSize` — Size of additional border elements
1-by-*D* numeric vector

Size of additional border elements to add around a block in each dimension, specified as a 1-by-*D* numeric vector, where *D* is equal to the `NumDimensions` property of the first `blockedImage` object.

`InclusionThreshold` — Ratio that determines whether a block should be considered for processing
double

Ratio that determines whether a block should be considered for processing, specified as a value in the range [0 1]. The value specifies the minimum percentage of nonzero pixels in the block. A value of 0 means that only one pixel in the block must be true/nonzero for it to be included in processing. A value of 1 requires that all pixels in the block be true/nonzero for it to be included in processing.

`PadMethod` — Method used to obtain padding pixels
scalar `string` | `char` vector

Method used to obtain padding pixels to honor the `BorderSize` or the `PadPartialBlocks` parameters, specified as scalar `string` or `char` vector

`PadPartialBlocks` — Pad partial blocks that might exist on the edges
`true` | `false`

Pad partial blocks that might exist on the edges, specified as a logical scalar, `true` or `false`.

`Resume` — Continue processing from where a previous run stopped
`true` | `false`

Continue processing from where a previous run stopped, specified as a logical scalar, `true` or `false`.

Object Functions

blockedImageAutomationAlgorithm Implements the blocked image automation algorithm

Version History

Introduced in R2021a

See Also

Apps
Image Labeler

Objects
`vision.labeler.AutomationAlgorithm`

Topics
“Create Custom Automation Algorithm for Labeling”
“Temporal Automation Algorithms”

blockedImageAutomationAlgorithm

Implements the blocked image automation algorithm

Syntax

```
res = blockedImageAutomationAlgorithm(algObj,bstruct)
```

Description

`res = blockedImageAutomationAlgorithm(algObj,bstruct)` computes labels based on the algorithm you specify in this method. The Image Labeler app invokes this method on each image you choose for blocked image automation in the app. `algObj` is a `vision.labeler.AutomationAlgorithm` object. `bstruct` is a structure. The format of the output, `res`, depends on the type of automation algorithm specified.

Examples

Sample Blocked Image Algorithm Automation Function

- Here is an example of a blocked image algorithm automation function.

```
function res = blockedImageAutomationAlgorithm(algObj,bstruct)
   % Detect people using aggregate channel features
   detector = peopleDetectorACF('inria-100x41');
   [bboxes, scores] = detect(detector, bstruct.Data,...
            'SelectStrongest', false);

   % Apply non-maximum suppression to select the strongest bounding boxes.
   [selectedBboxes, selectedScores] = selectStrongestBbox(bboxes, scores,...
            'RatioType', 'Min',...
            'OverlapThreshold', 0.65);

   % Consider only detections that meet specified score threshold
   selectedBboxes = selectedBboxes(selectedScores > 0, :);
   selectedBboxes(:,1) = selectedBboxes(:,1)+bstruct.Start(2);
   selectedBboxes(:,2) = selectedBboxes(:,2)+bstruct.Start(1);

   if algObj.SelectedLabelDefinitions.Type == "Rectangle"
      % Add the selected label at the bounding box position(s)
      res = struct(...
          'Type', labelType.Rectangle,...
          'Name', algObj.SelectedLabelDefinitions.Name,...
          'Position', selectedBboxes);
   end
end
```

Input Arguments

`algObj` — Automation algorithm

`vision.labeler.AutomationAlgorithm` object

Automation algorithm, specified as a `vision.labeler.AutomationAlgorithm` object.

`bstruct` — Structure of data passed to function
scalar `struct`

Structure of data passed to the blocked image `apply` object function, specified as a scalar `struct` with these fields.

Field	Description
`Data`	A block of array data from the blocked image.
`Start`	Array subscripts of the first element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`End`	Array subscripts of the last element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`Blocksub`	The block subscripts of the current block
`BorderSize`	The value of the `BorderSize` parameter
`BlockSize`	The value of the `BlockSize` parameter. Note: size(data) can be less than this value for border blocks `PadPartialValue` is false.
`BatchSize`	The value of the `BatchSize` parameter

Output Arguments

`res` — Result of processing
scalar `struct`

Result of processing, returned as a scalar `struct`. The format of the structure depends on the type of automation algorithm.

For automation algorithms without pixel labels, the `struct` has fields described in this table.

Field	Description
`Type`	A `labelType` enumeration that defines the type of label. Type can have values: `Rectangle`, `Line`, `Projected cuboid`, `Cuboid`, or `Scene`.
`Name`	A character vector specifying a label name that returns `true` for `checkLabelDefinition`. Only existing label names previously defined in the Image Labeler app can be used.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td><code>Position</code></td><td>Positions of the labels. The type of label determines the format of the position data.

<table><tr><th>Label Type</th><th>Description</th></tr><tr><td>Rectangle</td><td>P-by-1 cell array specifying P Rectangles, each containing a 1-by-4 vector specifying position of the bounding box locations as <code>[x y w h]</code> or multiple Rectangle ROIs specified as an M-by-4 matrix.</td></tr><tr><td>Line</td><td>P-by-1 cell array specifying P polylines each containing N-by-2 vector specifying N points along each polyline as: <code>[x1,y1; x2,y2;...xN,yN]</code></td></tr><tr><td>Polygons</td><td>P-by-1 cell array specifying P polygons each containing N-by-2 vector specifying N points along each polygon as: <code>[x1,y1; x2,y2;...xN,yN]</code></td></tr><tr><td>Projected Cuboid</td><td>P-by-1 cell array specifying P projected cuboids each containing 1-by-8 vector specifying position of primary and secondary faces as: <code>[x1 y1 w1 h1 x2 y2 w2 h2]</code> or multiple projected cuboid ROIs can be specified as an M-by-8 matrix.</td></tr></table></td></tr>
<tr><td><code>Attributes</code></td><td>An array of <code>structs</code> representing the attributes contained by the automated labels. Each attribute is specified as a field of the <code>struct</code> with the name of the field representing the name of the attribute and the value of the field representing the value of the attribute. This is an optional field, only present if the labels defined have attributes.</td></tr>
</table>

For automation algorithms with pixel labels, `res` is a `categorical` label matrix, where each category represents a pixel label.

Tips

- For automation algorithms without pixel labels, the `Position` field in `res` must be in a world coordinate system. This can be achieved by adding the *X* and *Y* indices in the `bstruct.Start` field to the output of the automation algorithm. To get the correct *X* coordinate, add `bstruct.Start(2)` to get the position of the automation output in world coordinates. To get the correct *Y* coordinate, add `bstruct.Start(1)` to get the position of the automation output in world coordinates.

Version History

Introduced in R2021a

See Also

Apps
Image Labeler

Objects
`vision.labeler.mixin.Temporal`

labelType

Label type enumerations for labeling

Description

The `labelType` enumerations enable you to specify the types of labels used in these labeling apps: Image Labeler, Video Labeler, Lidar Labeler, and Ground Truth Labeler. When creating label definitions by using a `labelDefinitionCreator` or `labelDefinitionCreatorMultisignal` object, use these enumerations to create label definitions of specific types. When selecting labels from a `groundTruth` or `groundTruthMultisignal` or `groundTruthLidar` (Lidar Toolbox) object, use these enumerations to select labels of a specific type.

Creation

Syntax

```
labelType("Rectangle")
labelType("RotatedRectangle")
labelType("Line")
labelType("PixelLabel")
labelType("Polygon")
labelType("Cuboid")
labelType("ProjectedCuboid")
labelType("Scene")
labelType('Custom')
```

Description

`labelType("Rectangle")` creates an axis-aligned rectangular region of interest (ROI) label type for labeling image and video data. You can also use the programmatic format, `labelType.Rectangle`.

`labelType("RotatedRectangle")` creates a rotated rectangular region of interest (ROI) label type for labeling image and video data. You can also use the programmatic format, `labelType.RotatedRectangle`.

`labelType("Line")` creates a polyline ROI label type for labeling image and video data. You can also use the programmatic format, `labelType.Line`.

`labelType("PixelLabel")` creates a pixel ROI label type for labeling image and video data. You can also use the programmatic format, `labelType.PixelLabel`.

`labelType("Polygon")` creates a polygon ROI label type for labeling image and video data. You can also use the programmatic format, `labelType.Polygon`.

`labelType("Cuboid")` creates a cuboid ROI label type for labeling lidar point cloud data. You can also use the programmatic format, `labelType.Cuboid`.

`labelType("ProjectedCuboid")` creates a projected cuboid ROI label type for labeling image and video data. You can also use the programmatic format, `labelType.ProjectedCuboid`.

`labelType("Scene")` creates a scene label type for labeling data across a specified time range. You can also use the programmatic format, `labelType.Scene`.

`labelType('Custom')` creates a custom label type. Labeler apps do not display labels of type `Custom`. You can also use the programmatic format, `labelType.Custom`.

Object Functions

isCustom	Determine if label types are Custom labels
isROI	Determine if label types are ROI labels
isScene	Determine if label types are Scene labels

Examples

Specify Label Type in Label Definition Creator Object

Create a label definition creator object. Add a label named "Road" with the label type specified as a rectangle.

```
ldc = labelDefinitionCreator();
addLabel(ldc,'Road',labelType.Rectangle);
```

Add a label named "Sky" with the label type specified as a scene.

```
addLabel(ldc,'Sky',labelType.Scene);
```

Add a sublabel named "Lanes" to the label "Road". Specify the label type for the sublabel as a line.

```
addSublabel(ldc,'Road','Lanes',labelType.Line);
```

Display the details of the definitions stored in the label definition creator object.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Road with 1 sublabels and 0 attributes and belongs to None group.    (info)
    Sky with 0 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the label "Road" as a structure by using the `info` function. The `Type` field in the structure `labelStruct` indicates the type of label.

```
labelStruct = info(ldc,'Road')
```

```
labelStruct = struct with fields:
           Name: "Road"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
```

```
    Sublabels: "Lanes"
  Description: ' '
```

Version History

Introduced in R2017a

See Also

Apps

Image Labeler | Ground Truth Labeler | Video Labeler

Objects

`groundTruth` | `labelDefinitionCreator` | `labelDefinitionCreatorMultisignal` | `groundTruthMultisignal` | `attributeType`

isCustom

Determine if label types are `Custom` labels

Syntax

```
tf = isCustom(labelTypes)
```

Description

`tf = isCustom(labelTypes)` returns a logical array that indicates which elements in `labelTypes` are `Custom` label types. `isCustom` returns logical `1` (true) for `Custom` labels and otherwise returns logical `0` (false).

Input Arguments

`labelTypes` — Type of labels

enumeration

Types of labels, specified as a `labelType` or `lidarLabelType` enumeration.

Version History

Introduced in R2017a

See Also

`isROI` | `isScene`

isROI

Determine if label types are ROI labels

Syntax

```
tf = isROI(labelTypes)
```

Description

`tf = isROI(labelTypes)` returns a logical array that indicates which elements in `labelTypes` are `ROI` label types. `isROI` returns logical `1` (true) for `ROI` labels and otherwise returns logical `0` (false). Valid `ROI` label types are `Rectangle`, `Cuboid`,`ProjectedCuboid`, `Line`, `Voxel` and `PixelLabel`.

Input Arguments

`labelTypes` — Type of labels

enumeration

Types of labels, specified as a `labelType` or `lidarLabelType` enumeration.

Version History

Introduced in R2017a

See Also

`isCustom` | `isScene`

isScene

Determine if label types are `Scene` labels

Syntax

```
tf = isScene(labelTypes)
```

Description

`tf = isScene(labelTypes)` returns a logical array that indicates which elements in `labelTypes` are `Scene` label types. `isScene` returns logical `1` (true) for `Scene` labels and otherwise returns logical `0` (false).

Input Arguments

`labelTypes` — Type of labels

enumeration

Types of labels, specified as a `labelType` or `lidarLabelType` enumeration.

Version History

Introduced in R2017a

See Also

`isCustom` | `isROI`

attributeType

Attribute type enumerations for labeling

Description

The `attributeType` enumerations enable you to specify the types of attributes used in these labeling apps: Image Labeler, Video Labeler, Lidar Labeler, and Ground Truth Labeler (requires Automated Driving Toolbox). When creating label definitions by using a `labelDefinitionCreator` or `labelDefinitionCreatorMultisignal` object, use these enumerations to create attributes of specific types.

Creation

Syntax

```
attributeType('Numeric')
attributeType('String')
attributeType('Logical')
attributeType('List')
attributeType('None')
```

Description

`attributeType('Numeric')` creates a numeric attribute type enumeration. Attributes of this type are numeric scalar values. You can also use the programmatic format, `attributeType.Numeric`.

`attributeType('String')` creates a string attribute type enumeration. Attributes of this type are string scalars. You can also use the programmatic format, `attributeType.String`.

`attributeType('Logical')` creates a logical attribute type enumeration. Attributes of this type are `true` (logical `1`), `false` (logical `0`), or an empty logical value. You can also use the programmatic format, `attributeType.Logical`.

`attributeType('List')` creates a list attribute type enumeration. Attributes of this type are lists of predefined strings. You can also use the programmatic format, `attributeType.List`.

`attributeType('None')` creates an enumeration whose type is not defined. You can also use the programmatic format, `attributeType.None`.

Object Functions

hasValue Determine if the attribute type is numeric or logical

Examples

Specify Attribute Type in Label Definition Creator Object

Create a label definition creator object and add a label named "Vehicle". Specify the type of label as a rectangle.

```
ldc = labelDefinitionCreator;
addLabel(ldc,'Vehicle',labelType.Rectangle);
```

Add an attribute named "Color" to the label "Vehicle". Specify the attribute type as a list.

```
addAttribute(ldc,'Vehicle','Color',attributeType.List,{'White','Green','Blue'})
```

Display information about the attribute "Color" as a structure by using the `info` function. The `Type` field in the output structure `colorStruct` indicates the type of attribute.

```
colorStruct = info(ldc,'Vehicle/Color')
```

```
colorStruct = struct with fields:
           Name: "Color"
           Type: List
      ListItems: {'White'  'Green'  'Blue'}
    Description: ' '
```

Version History

Introduced in R2018b

See Also

Apps

Image Labeler | Ground Truth Labeler | Video Labeler

Objects

`labelType` | `labelDefinitionCreator` | `labelDefinitionCreatorMultisignal`

hasValue

Determine if the attribute type is numeric or logical

Syntax

```
tf = hasValue(typeOfAttribute)
```

Description

`tf = hasValue(typeOfAttribute)` returns a logical value that indicates if the enumerator in `typeOfAttribute` is either `Numeric` or `Logical`. `hasValue` returns logical `1 (true)` for `Numeric` and `Logical` attributes. Otherwise returns logical `0 (false)`.

Examples

Determine the Type of Attribute

Create an `attributeType` enumeration array.

```
Type = attributeType({'Numeric'; 'String'; 'Logical'; 'List'});
```

Check if any of the enumerator defined in the `attributeType` enumeration array is Numeric or Logical.

```
tf = zeros(0,length(Type));
for i = 1:length(Type)
    tf(i) = hasValue(Type(i));
end
```

Display the output. The value 1 is returned when an enumerator in the `attributeType` enumeration array is either Numeric or Logical. Otherwise, the value 0 is returned.

```
tf
```

```
tf = 1×4

     1     0     1     0
```

Input Arguments

`typeOfAttribute` — Type of attribute

`attributeType` enumeration

Type of attribute, specified as an `attributeType` enumeration. The enumerator in the `attributeType` enumeration can be `Numeric`, `String`, `Logical`, `List`, or `None`.

Version History

Introduced in R2018b

See Also

Objects

`groundTruth` | `groundTruthMultisignal` | `labelDefinitionCreator` | `labelDefinitionCreatorMultisignal` | `attributeType`

cameraParameters

Object for storing camera parameters

Description

The `cameraParameters` object stores the intrinsic, extrinsic, and lens distortion parameters of a camera.

Creation

You can create a `cameraParameters` object using the `cameraParameters` function described here. You can also create a `cameraParameters` object by using the `estimateCameraParameters` with an *M*-by-2-by-*numImages* array of input image points. *M* is the number of keypoint coordinates in each pattern.

Syntax

```
cameraParams = cameraParameters
cameraParams = cameraParameters(Name=Value)
cameraParams = cameraParameters(paramStruct)
```

Description

`cameraParams = cameraParameters` creates a `cameraParameters` object that contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`cameraParams = cameraParameters(Name=Value)` sets properties on page 2-1411 of the `cameraParameters` object by using one or more name-value arguments. Unspecified properties use default values. For example, `WorldUnits='m'` sets the world units to `'m'`.

For example, `cameraParams = cameraParameters("RadialDistortion",[0 10])` sets the radial lens distortion property, `RadialDistortion`, as the vector `[0 10]`.

`cameraParams = cameraParameters(paramStruct)` creates an identical `cameraParameters` object from an existing `cameraParameters` object with parameters stored in `paramStruct`.

Input Arguments

`paramStruct` — Camera parameters
structure

Camera parameters, specified as a camera parameters structure. To get a `paramStruct` from an existing `cameraParameters` object, use the `toStruct` function.

Properties

Intrinsic Camera Parameters:

K — Camera intrinsic matrix

3-by-3 matrix

Camera intrinsic matrix, specified as a 3-by-3 matrix. The matrix has this format:

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The coordinates $[c_x\ c_y]$ represent the optical center (the principal point), in pixels. When the x- and y-axes are exactly perpendicular, the skew parameter s equals `0`.

$f_x = F*s_x$
$f_y = F*s_y$

- F is the focal length in world units, typically expressed in millimeters.
- s_x and s_y are the number of pixels per world unit in the x- and y-direction respectively.
- f_x and f_y are expressed in pixels.

`Intrinsics` — Camera intrinsics object

`cameraIntrinsics` object

This property is read-only.

Camera intrinsics object, stated as a `cameraIntrinsics` object. The object contains information about camera intrinsic calibration parameters, including lens distortion.

Dependency

You must provide an image size (using the `ImageSize` property) for the `Intrinsics` property to be non-empty. The intrinsics for the camera parameters depends on the image size.

`ImageSize` — Image size

two-element vector

Image size, specified as a two-element vector [*mrows ncols*].

Camera Lens Distortion:

`RadialDistortion` — Radial distortion coefficients

`[0 0]` (default) | 2-element vector | 3-element vector | 6-element vector

Radial lens distortion coefficients, specified as a 2-, 3-, or 6-element vector.

- 2-element vector — [*k1 k2*].
- 3-element vector — [*k1 k2 k3*].
- 6-element vector — [*k1 k2 k3 k4 k5 k6*].

The camera parameters object calculates the radial-distorted location of a point, denoted as ($x_{distorted}$, $y_{distorted}$):

$$x_{distorted} = x\left(\frac{1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6}{1 + k_4 * r^2 + k_5 * r^4 + k_6 * r^6}\right)$$

$$y_{distorted} = y\left(\frac{1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6}{1 + k_4 * r^2 + k_5 * r^4 + k_6 * r^6}\right)$$

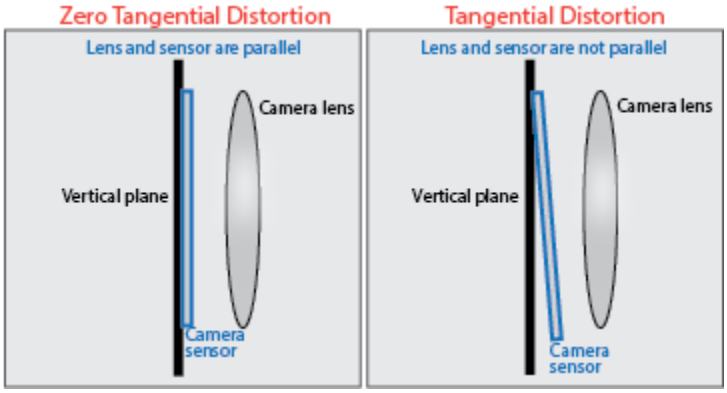
x, y is a undistorted image point in normalized image coordinates in world units, with the origin at the optical center.
$r^2 = x^2 + y^2$
k_1, k_2, ..., k_6 are radial distortion coefficients of the lens. Typically, two coefficients are sufficient and k_3, ..., k_6 are only needed for wide-angle lenses.

`TangentialDistortion` — Tangential distortion coefficients

`[0 0]'` (default) | 2-element vector

Tangential distortion coefficients, specified as a two-element vector. Tangential distortion occurs when the lens and the image plane are not parallel. The camera parameters object calculates the tangential distorted location of a point. You can denote the distorted points as ($x_{distorted}$, $y_{distorted}$). The undistorted pixel locations appear in normalized image coordinates, with the origin at the optical center. The coordinates are expressed in world units.

Tangential distortion occurs when the lens and the image plane are not parallel. The tangential distortion coefficients model this type of distortion.



The distorted points are denoted as ($x_{distorted}$, $y_{distorted}$):

$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$

$y_{distorted} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$

- x, y — Undistorted pixel locations. x and y are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, x and y are dimensionless.
- p_1 and p_2 — Tangential distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

Extrinsic Camera Parameters:

`PatternExtrinsics` — Calibration pattern extrinsics

`[]` (default) | *N*-element vector of `rigidtform3d` objects | *N*-by-*P* matrix of `rigidtform3d` objects

This property is read-only.

Calibration pattern extrinsics, specified as an *N*-element vector or an *N*-by-*P* array of *N* number of `rigidtform3d` objects. Each `rigidtform3d` object represents the pose of the camera's image plane relative to the corresponding calibration pattern.

- The `R` property of each `rigidtform3d` object describes the 3-D rotation of the camera image plane relative to the corresponding calibration pattern.
- The `Translation` property of each `rigidtform3d` object describes the translation *t* of the camera relative to the corresponding calibration pattern, expressed in world units.

For single-pattern calibration, specify `PatternExtrinsics` as an *N*-element vector. For multiple patterns, use an *N*-by-*P* matrix.

This equation provides the transformation that relates a world coordinate in the checkerboard frame [*X Y Z*] and the corresponding image point [*x y*]:

$$w\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R\ t]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- *w*: arbitrary scale factor
- *K*: camera intrinsic matrix
- *R*: matrix representing the 3-D rotation of the camera
- *t*: translation of the camera relative to the world coordinate system

The rigid geometric transformations do not take distortion into consideration. Use the `undistortImage` function to remove distortion.

`RotationVectors` — 3-D camera rotation vectors

`[ ]` (default) | *R*-by-3 matrix | *R*-by-3-by-*P* array

This property is read-only.

3-D camera rotation vectors, specified as an *R*-by-3 matrix or an *R*-by-3-by-*P* array, containing *R* rotation vectors, where *P* is the number of calibration patterns. Each vector describes the 3-D rotation of the camera's image plane relative to the corresponding calibration pattern.

Each vector specifies the axis of rotation in 3-D space, with its magnitude indicating the rotation angle in radians. The corresponding 3-D rotation matrices are specified by the `PatternExtrinsics` property, specifically in the form of `tform.R`, where `tform` is a `rigidtform3d` object. The `PatternExtrinsics` property specifies geometric transformation objects with the corresponding 3-D rotation matrices.

For single-pattern calibration, specify `RotationVectors` as an *R*-by-3 matrix. For multiple patterns, use an *R*-by-3-by-*P* array.

Estimated Camera Parameter Accuracy:

`MeanReprojectionError` — Average Euclidean distance

numeric value

This property is read-only.

Average Euclidean distance between reprojected and detected points, specified as a numeric value in pixels.

`ReprojectionErrors` — Estimated camera parameters accuracy

[ ] (default) | *K*-by-2-by-*V* array | *K*-by-2-by-*V*-by-*P* array

Estimated camera parameters accuracy, specified as a *K*-by-2-by-*V* or *K*-by-2-by-*V*-by-*P* array, of [*x y*] pairs. The pairs represent the translation in *x* and *y* between the reprojected pattern keypoints *K*, and the detected pattern keypoints. The values of this property represent the accuracy of the estimated camera parameters. *V* is the number of pattern views used to estimate camera parameters and *P* is the number of calibration patterns. *K* is the number of keypoints in each image.

For single-pattern calibration, specify `ReprojectionErrors` as an*K*-by-2-by-*V* array. For multiple patterns, use an*K*-by-2-by-*V*-by-*P* array.

`ReprojectedPoints` — World points reprojected onto calibration images

K-by-2-by-*V* array | *K*-by-2-by-*V*-by-*P* array

This property is read-only.

World points reprojected onto calibration images, specified as a *K*-by-2-by-*V* or *K*-by-2-by-*V*-by-*P* array of [*x y*] coordinates. The coordinates represent world points reprojected onto calibration images. The number of keypoints in each image is represented by *K*. The number of pattern views is denoted by *V*, and *P* represents the number of calibration patterns. Missing points in the detected keypoints of the pattern are denoted as `[NaN,NaN]`.

For single-pattern calibration, specify `ReprojectedPoints` as a *K*-by-2-by-*V* array. For multiple patterns, use an*K*-by-2-by-*V*-by-*P* array.

`DetectedKeypoints` — Detected keypoints in the calibration pattern

[ ] (default) | *K*-by-*V* array | *K*-by-*V*-by-*P* array

Detected keypoints in the calibration pattern, specified as a logical *K*-by-*V* or *K*-by-*V*-by-*P* array. The number of keypoints in each image is represented by *K*. The number of pattern views is denoted by *V*, and *P* represents the number of calibration patterns.

For single-pattern calibration, specify `DetectedKeypoints` as a *K*-by-*V* array. For multiple patterns, use an*K*-by-*V*-by-*P* array.

Settings for Camera Parameter Estimation:

`PatternCount` — Number of patterns to estimate camera extrinsics

scalar integer

This property is read-only.

Number of patterns to estimate camera extrinsics, specified as an scalar integer.

`NumViews` — Number of pattern views used to estimate camera extrinsics

scalar integer

This property is read-only.

Number of pattern views used to estimate camera extrinsics, specified as an scalar integer. A view is the perspective from which a camera captures a scene.

`WorldPoints` — World coordinates
M-by-2 array | `[]`

World coordinates of key points on calibration pattern, specified as an *M*-by-2 array. *M* represents the number of key points in the pattern.

`WorldUnits` — World points units
`"mm"` (default) | character vector | string scalar

World points units, specified as a character vector or string scalar. The value describes the units of measure.

`EstimateSkew` — Estimate skew flag
`false` (default) | `true`

Estimate skew flag, specified as a logical scalar. When you set the logical to `true`, the object estimates the image axes skew. When you set the logical to `false`, the image axes are exactly perpendicular.

`NumRadialDistortionCoefficients` — Number of radial distortion coefficients
2 (default) | 3 | 6

Number of radial distortion coefficients, specified as the number 2, 3, or 6.

`EstimateTangentialDistortion` — Estimate tangential distortion flag
`false` (default) | `true`

Estimate tangential distortion flag, specified as the logical scalar `true` or `false`. When you set the logical to `true`, the object estimates the tangential distortion. When you set the logical to `false`, the tangential distortion is negligible.

Examples

Remove Distortion from an Image Using Camera Parameters Object

Use the camera calibration functions to remove distortion from an image. This example creates a `cameraParameters` object manually, but in practice, you would use the `estimateCameraParameters` or the **Camera Calibrator** app to derive the object.

Create a `cameraParameters` object manually.

```
k = [715.2699 0 565.6995; 0 711.5281 355.3466; 0 0 1];
radialDistortion = [-0.3361 0.0921];
cameraParams = cameraParameters("K",k,"RadialDistortion",radialDistortion)
```

```
cameraParams =
  cameraParameters with properties:

   Camera Intrinsics
                        Intrinsics: [0×0 cameraIntrinsics]

   Camera Extrinsics
                PatternExtrinsics: [0×0 rigidtform3d]
```

```
   Accuracy of Estimation
              MeanReprojectionError: NaN
                 ReprojectionErrors: [0×2 double]
                  ReprojectedPoints: [0×2×0×0 double]

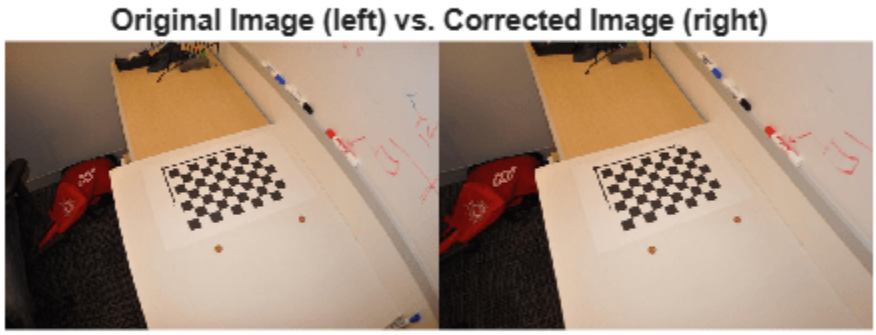
   Calibration Settings
                            NumViews: 0
                         PatternCount: 0
                   DetectedKeypoints: [0×2 double]
                         WorldPoints: [0×2 double]
                          WorldUnits: 'mm'
                        EstimateSkew: 0
    NumRadialDistortionCoefficients: 2
       EstimateTangentialDistortion: 0
```

Remove distortion from the images.

```
I = imread(fullfile(matlabroot,"toolbox","vision","visiondata","calibration","mono","image01.jpg
J = undistortImage(I,cameraParams);
```

Display the original and the undistorted images.

```
montage({I,J})
title("Original Image (left) vs. Corrected Image (right)")
```



Version History

Introduced in R2014a

R2026a: Properties enhanced for multi-camera calibration

New properties for calibration pattern count, `PatternCount`, and the number of views, `NumViews`. These other properties were enhanced to support the pattern count:

- `RotationVectors`
- `PatternExtrinsics`
- `ReprojectionErrors`
- `ReprojectionPoints`
- `DetectedKeyPoints`

R2026a: Removed `NumPatterns` property

The `NumPatterns` property is removed. Use the new `PatternCount` and `NumViews` properties instead.

R2022b: Supports premultiply matrix convention
Behavior changed in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, some properties of the `cameraParameters` object have changed to support the premultiply convention.

- The new `K` property replaces the old `IntrinsicMatrix` property. The value of `K` is the transpose of `IntrinsicMatrix`.
- The new `PatternExtrinsics` property replaces the old `RotationMatrices` and `TranslationVectors` properties. You can access the rotation matrices and translation vectors by querying the `R` and `Translation` properties of the `rigidtform3d` objects stored in the `PatternExtrinsics` property. The `R` property stores a rotation matrix as the transpose of the rotation matrix represented by `RotationMatrices`.

For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2024b: Import OpenCV pinhole camera model with six radial distortion coefficients

The `cameraParameters` has been updated to support OpenCV pinhole camera model with 6 radial distortion coefficients.

References

[1] Zhang, Z. "A Flexible New Technique for Camera Calibration." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, no. 11 (November 2000): 1330–34. https://doi.org/10.1109/34.888718.

[2] Heikkila, J., and O. Silven. “A Four-Step Camera Calibration Procedure with Implicit Image Correction.” In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1106–12. San Juan, Puerto Rico: IEEE Comput. Soc, 1997. https://doi.org/10.1109/CVPR.1997.609468.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use the `toStruct` function to pass a `cameraParameters` object into generated code. See the "Code Generation for Depth Estimation from Stereo Video" example.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Classes

`stereoParameters` | `cameraCalibrationErrors` | `intrinsicsEstimationErrors` | `extrinsicsEstimationErrors` | `cameraIntrinsics`

Functions

`world2img` | `img2world2d` | `estimateCameraParameters` | `showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `patternWorldPoints`

Topics

"Measuring Planar Objects with a Calibrated Camera"
"Code Generation for Depth Estimation from Stereo Video"
"Using the Single Camera Calibrator App"

fisheyeCalibrationErrors

Object for storing standard errors of estimated fisheye camera parameters

Description

`fisheyeCalibrationErrors` contains the standard errors of estimated camera parameters. You can access the standard errors of the intrinsics and extrinsics by using the object properties. To display the standard errors, use the `displayErrors` function.

Creation

Create a `fisheyeCalibrationErrors` object by calling the `estimateFisheyeParameters`,

Properties

`IntrinsicsErrors` — Standard errors of estimated intrinsics
`fisheyeIntrinsicsEstimationErrors` object

Standard errors of the estimated intrinsics for a fisheye camera, specified as a `fisheyeIntrinsicsEstimationErrors` object.

`ExtrinsicsErrors` — Standard errors of estimated rotations and translations
`extrinsicsEstimationErrors` object

Standard errors of the estimated rotations and translations for a fisheye camera relative to the calibration pattern, specified as a `extrinsicsEstimationErrors` object.

Object Functions

displayErrors Display standard errors of camera parameter estimates

Examples

Display Fisheye Camera Calibration Errors

Gather a set of checkerboard calibration images.

```
images = imageDatastore(fullfile(toolboxdir('vision'),'visiondata',...
 'calibration','gopro'));
```

Detect the calibration pattern from the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate world coordinates for the corners of the checkerboard squares.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints('checkerboard',patternDims,squareSize);
```

Estimate the fisheye parameters using image and world points. Use the first image to get the image size. Also, store the errors from the calibration.

```
I = readimage(images,1);
imageSize = [size(I,1) size(I,2)];
[params,~,errors] = estimateFisheyeParameters(imagePoints, ...
                                worldPoints,imageSize);
```

Display the standard errors of the estimated camera parameters.

```
displayErrors(errors,params);
```

```
            Standard Errors of Estimated Camera Parameters
            ----------------------------------------------

Intrinsics
----------
Mapping coefficients:    [  875.0746 +/- 0.9451       -0.0003 +/- -0.0000      -0.0000 +/- 0.0000
Distortion center (pixels):[ 1005.8175 +/- 0.6871      743.0343 +/- 0.5578  ]
Stretch matrix parameters:[    1.0000 +/- 0.0000        0.0000 +/- 0.0000        0.0000 +/- 0.000

Extrinsics
----------
Rotation vectors:
                         [   -0.0699 +/- 0.0010       -0.0267 +/- 0.0009        0.0258 +/- 0.0002
                         [    0.3628 +/- 0.0010        0.2950 +/- 0.0009       -0.1967 +/- 0.0003
                         [   -0.2159 +/- 0.0009        0.3442 +/- 0.0009       -0.1941 +/- 0.0003
                         [    0.0282 +/- 0.0009       -0.3784 +/- 0.0009        0.0829 +/- 0.0003
                         [    0.0146 +/- 0.0008        0.4575 +/- 0.0009       -0.1215 +/- 0.0003
                         [    0.6775 +/- 0.0008        0.1089 +/- 0.0008       -0.0386 +/- 0.0004
                         [   -0.4936 +/- 0.0008        0.0063 +/- 0.0008        0.0486 +/- 0.0003
                         [    0.3823 +/- 0.0008        0.2797 +/- 0.0008        0.1509 +/- 0.0003
                         [    0.5171 +/- 0.0008       -0.3295 +/- 0.0008        0.0541 +/- 0.0003
                         [   -0.1896 +/- 0.0008       -0.3543 +/- 0.0009        0.2637 +/- 0.0003
                         [   -0.2911 +/- 0.0008        0.3680 +/- 0.0008       -0.1329 +/- 0.0003

Translation vectors (mm):
                         [ -132.9184 +/- 0.1609      -82.6066 +/- 0.1356      195.1098 +/- 0.2311
                         [ -178.9934 +/- 0.1905      -15.7749 +/- 0.1712      241.7116 +/- 0.2795
                         [ -183.7960 +/- 0.2168      -56.7377 +/- 0.1884      269.9729 +/- 0.2790
                         [  -17.6297 +/- 0.1315      -70.2875 +/- 0.1041      157.0820 +/- 0.1933
                         [ -161.9827 +/- 0.1807      -46.9680 +/- 0.1569      228.4052 +/- 0.2302
                         [ -122.4242 +/- 0.1309      -16.0260 +/- 0.1153      162.6239 +/- 0.2072
                         [ -112.4270 +/- 0.1745     -125.5876 +/- 0.1428      212.8047 +/- 0.2156
                         [ -148.7139 +/- 0.1387      -72.5409 +/- 0.1260      173.7608 +/- 0.2080
                         [  -49.5393 +/- 0.0919      -24.8329 +/- 0.0745      104.3536 +/- 0.1506
                         [   -3.4047 +/- 0.1274      -93.4073 +/- 0.1010      155.8242 +/- 0.1693
                         [ -160.7347 +/- 0.1855      -51.9152 +/- 0.1600      234.4066 +/- 0.2318
```

Version History

Introduced in R2017b

See Also

Objects
`fisheyeParameters` | `fisheyeIntrinsics` | `fisheyeIntrinsicsEstimationErrors` | `extrinsicsEstimationErrors`

Functions
`estimateFisheyeParameters` | `undistortFisheyePoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `showExtrinsics` | `showReprojectionErrors`

Topics
“Fisheye Calibration Basics”
“Stereo Fisheye Camera Calibration”
“Configure Monocular Fisheye Camera” (Automated Driving Toolbox)

fisheyeIntrinsics

Object for storing intrinsic fisheye camera parameters

Description

The `fisheyeIntrinsics` object stores the camera intrinsics for a fisheye camera. See `fisheyeParameters` for details on other fisheye parameters.

Creation

Syntax

```
intrinsics = fisheyeIntrinsics(mappingCoeffs,imageSize,distortionCenter)
intrinsics = fisheyeIntrinsics( ___ ,stretchMatrix)
```

Description

`intrinsics = fisheyeIntrinsics(mappingCoeffs,imageSize,distortionCenter)` returns a `fisheyeIntrinsics` object with the specified `[a0 a2 a3 a4]` polynomial coefficients, image size, and center of distortion. These input arguments are assigned directly to the corresponding properties of the object. See `fisheyeParameters` for more details.

`intrinsics = fisheyeIntrinsics( ___ ,stretchMatrix)` additionally specifies a 2-by-2 transformation matrix that describes the alignment between the sensor plane and the image plane. The default value is an identity matrix.

Properties

`MappingCoefficients` — Polynomial coefficients for projection function
`[a0 a2 a3 a4]` vector

This property is read-only.

Polynomial coefficients for the projection function described by Scaramuzza's Taylor model, specified as an `[a0 a2 a3 a4]` vector.

`DistortionCenter` — Center of distortion in pixels
`[cx cy]` vector

This property is read-only.

Center of distortion in pixels, specified as a `[cx cy]` vector.

`StretchMatrix` — Transformation from sensor plane to pixel in camera image plane
2-by-2 transformation matrix

This property is read-only.

Transformation from the sensor plane to a pixel in the camera image plane, specified as a 2-by-2 transformation matrix. This misalignment is caused by the lens not being parallel to the sensor and by the digitization process.

`ImageSize` — Image size
`[mrows ncols]` vector

This property is read-only.

Image size, specified as an `[mrows ncols]` vector.

Examples

Create Fisheye Intrinsics

Specify the mapping coefficients, image size, and distortion center parameters of a `fisheyeIntrinsics` object. Ignore optical axis misalignment.

```
mappingCoeffs = [880 -3e-4 0 0];    % mapping polynomial coefficients
imageSize = [1500 2000];            % in [mrows ncols]
distortionCenter = [1000 750];      % in pixels

intrinsics = fisheyeIntrinsics(mappingCoeffs,imageSize,distortionCenter);
```

Version History

Introduced in R2017b

References

[1] Scaramuzza, D., A. Martinelli, and R. Siegwart. "A Toolbox for Easy Calibrating Omnidirectional Cameras." *Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006)*. Beijing, China, October 7–15, 2006.

[2] Urban, S., J. Leitloff, and S. Hinz. "Improved Wide-Angle, Fisheye and Omnidirectional Camera Calibration." *ISPRS Journal of Photogrammetry and Remove Sensing*. Vol. 108, 2015, pp.72–79.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`fisheyeParameters` | `fisheyeIntrinsicsEstimationErrors` | `extrinsicsEstimationErrors`

Functions
`estimateFisheyeParameters` | `undistortFisheyePoints` | `world2img` | `img2world2d`

Topics
“Fisheye Calibration Basics”
“Stereo Fisheye Camera Calibration”
“Configure Monocular Fisheye Camera” (Automated Driving Toolbox)
“3-D Motion Reconstruction Using Multiple Cameras”

fisheyeIntrinsicsEstimationErrors

Object for storing standard errors of estimated fisheye camera intrinsics

Description

`fisheyeIntrinsicsEstimationErrors` contains the standard errors of estimated fisheye camera intrinsics.

Creation

The `estimateFisheyeParameters` function returns the `fisheyeCalibrationErrors` object, which contains `fisheyeIntrinsicsEstimationErrors` and `extrinsicsEstimationErrors` as properties.

Properties

`MappingCoefficientsError` — Standard error of mapping coefficient estimates
4-element vector

Standard error of mapping coefficient estimates, specified as a 4-element vector.

`DistortionCenterError` — Standard error of distortion center estimates
2-element vector

Standard error of distortion center estimates, specified as a two-element vector.

`StretchMatrixError` — Standard error of stretch matrix estimate
3-element vector

Standard error of stretch matrix estimate, specified as a 3-element vector.

Version History

Introduced in R2017b

See Also

Objects
`fisheyeParameters` | `fisheyeIntrinsics` | `extrinsicsEstimationErrors` | `cameraCalibrationErrors` | `stereoCalibrationErrors`

Functions
`estimateFisheyeParameters` | `undistortFisheyePoints` | `world2img` | `img2world2d`

Topics
“Fisheye Calibration Basics”
“Stereo Fisheye Camera Calibration”

“Configure Monocular Fisheye Camera” (Automated Driving Toolbox)

fisheyeParameters

Object for storing fisheye camera parameters

Description

The `fisheyeParameters` object is used to store fisheye camera parameters. Use `estimateFisheyeParameters` to estimate parameters using calibration images.

Creation

Syntax

```
fisheyeParams = fisheyeParameters(intrinsics)
fisheyeParams = fisheyeParameters(intrinsics,Name=Value)
```

Description

`fisheyeParams = fisheyeParameters(intrinsics)` returns an object that contains intrinsic and extrinsic parameters of a fisheye camera. `intrinsics` must be a `fisheyeIntrinsics` object. This syntax sets the `Intrinsics` property of the object.

`fisheyeParams = fisheyeParameters(intrinsics,Name=Value)` sets properties on page 2-1411 of the `cameraParameters` object by using one or more name-value arguments. Unspecified properties use default values. For example, `WorldUnits='m'` sets the world units to `'m'`.

Properties

Intrinsic Camera Parameters

`Intrinsics` — Fisheye intrinsic camera parameters
`fisheyeIntrinsics` object

Fisheye intrinsic camera parameters, specified as a `fisheyeIntrinsics` object.

Extrinsic Camera Parameters

`RotationVectors` — 3-D Camera rotations
`[ ]` (default) | *M*-by-3 matrix | *R*-by-3-by-*P* array

3-D camera rotation vectors, specified as an *R*-by-3 matrix or an *R*-by-3-by-*P* array, containing *R* rotation vectors, where *P* is the number of calibration patterns. Each vector describes the 3-D rotation of the camera's image plane relative to the corresponding calibration pattern.

Each vector specifies the axis of rotation in 3-D space, with its magnitude indicating the rotation angle in radians. The corresponding 3-D rotation matrices are specified by the `PatternExtrinsics` property, specifically in the form of `tform.R`, where `tform` is a `rigidtform3d` object. The `PatternExtrinsics` property specifies geometric transformation objects with the corresponding 3-D rotation matrices.

For single-pattern calibration, specify `RotationVectors` as an *R*-by-3 matrix. For multiple patterns, use an *R*-by-3-by-*P* array.

To ensure that the number of rotation vectors equals the number of translation vectors, you must set the `RotationVectors` and `TranslationVectors` properties together when creating the object. Setting only one property results in an error.

`TranslationVectors` — Camera translations

[ ] (default) | *M*-by-3 matrix

Camera translations, specified as an *M*-by-3 matrix. This matrix contains translation vectors for *M* images. The vectors contain the calibration pattern that estimates the calibration parameters. Each row of the matrix contains a vector that describes the translation of the camera relative to the corresponding pattern, expressed in world units.

To ensure that the number of rotation vectors equals the number of translation vectors, you must set the `RotationVectors` and `TranslationVectors` properties together when creating the object. Setting only one property results in an error.

Accuracy of Estimated Camera Parameters

`ReprojectionErrors` — Estimated camera parameters accuracy

[ ] (default) | *K*-by-2-by-*V* array | *K*-by-2-by-*V*-by-*P* array

Estimated camera parameters accuracy, specified as a *K*-by-2-by-*V* or *K*-by-2-by-*V*-by-*P* array, of [*x y*] pairs. The pairs represent the translation in *x* and *y* between the reprojected pattern keypoints *K*, and the detected pattern keypoints. The values of this property represent the accuracy of the estimated camera parameters. *V* is the number of pattern views used to estimate camera parameters and *P* is the number of calibration patterns. *K* is the number of keypoints in each image.

For single-pattern calibration, specify `ReprojectionErrors` as an*K*-by-2-by-*V* array. For multiple patterns, use an*K*-by-2-by-*V*-by-*P* array.

`ReprojectedPoints` — World points reprojected onto calibration images

K-by-2-by-*V* array | *K*-by-2-by-*V*-by-*P* array

This property is read-only.

World points reprojected onto calibration images, specified as a *K*-by-2-by-*V* or *K*-by-2-by-*V*-by-*P* array of [*x y*] coordinates. The coordinates represent world points reprojected onto calibration images. The number of keypoints in each image is represented by *K*. The number of pattern views is denoted by *V*, and *P* represents the number of calibration patterns. Missing points in the detected keypoints of the pattern are denoted as `[NaN,NaN]`.

For single-pattern calibration, specify `ReprojectedPoints` as a *K*-by-2-by-*V* array. For multiple patterns, use an*K*-by-2-by-*V*-by-*P* array.

`DetectedKeypoints` — Detected keypoints in the calibration pattern

[ ] (default) | *K*-by-*V* array | *K*-by-*V*-by-*P* array

Detected keypoints in the calibration pattern, specified as a logical *K*-by-*V* or *K*-by-*V*-by-*P* array. The number of keypoints in each image is represented by *K*. The number of pattern views is denoted by *V*, and *P* represents the number of calibration patterns.

For single-pattern calibration, specify `DetectedKeypoints` as a *K*-by-*V* array. For multiple patterns, use an*K*-by-*V*-by-*P* array.

Settings Used to Estimate Camera Parameters

`PatternCount` — Number of patterns to estimate camera extrinsics
scalar integer

This property is read-only.

Number of patterns to estimate camera extrinsics, specified as an scalar integer.

`NumViews` — Number of pattern views used to estimate camera extrinsics
scalar integer

This property is read-only.

Number of pattern views used to estimate camera extrinsics, specified as an scalar integer. A view is the perspective from which a camera captures a scene.

`WorldPoints` — World coordinates
`[ ]` (default) | *M*-by-2 matrix

World coordinates of key points on the calibration pattern, specified as an *M*-by-2 matrix. *M* represents the number of key points in the pattern.

`WorldUnits` — World point units
`'mm'` (default) | character vector | string scalar

World point units, specified as the comma-separated pair consisting of `'WorldUnits'` and a character vector or string scalar. This argument is used simply to store the unit type and does not affect any calculations.

`EstimateAlignment` — Estimate axes alignment
`false` (default) | `true`

Estimate axes alignment, specified as `false` or `true`. Set to `true` if the optical axis of the fisheye lens is not perpendicular to the image plane.

Examples

Create Fisheye Parameters Object Manually

Create a fisheye parameters object by specifying the properties manually. Alternatively, you can create this object using the `estimateFisheyeParameters` function.

Specify fisheye intrinsics.

```
mappingCoefficients = rand(1,4);
distortionCenter = [320 240];
imageSize = [480 640];
intrinsics = fisheyeIntrinsics(mappingCoefficients,imageSize,distortionCenter);
```

Create a `fisheyeParameters` object using the specified intrinsics.

```
params = fisheyeParameters(intrinsics);
```

Version History

Introduced in R2017b

R2026a: New `PatternCount` and `NumViews` properties

Adding the `PatternCount` and `NumViews` properties.

R2026a: Removed `NumPatterns` property

The `NumPatterns` property is removed. Use the new `PatternCount` and `NumViews` properties instead.

References

[1] Scaramuzza, D., A. Martinelli, and R. Siegwart. "A Toolbox for Easy Calibrating Omnidirectional Cameras." *Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006)*. Beijing, China, October 7–15, 2006.

[2] Urban, S., J. Leitloff, and S. Hinz. "Improved Wide-Angle, Fisheye and Omnidirectional Camera Calibration." *ISPRS Journal of Photogrammetry and Remove Sensing*. Vol. 108, 2015, pp.72–79.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`fisheyeIntrinsics` | `fisheyeIntrinsicsEstimationErrors` | `extrinsicsEstimationErrors` | `cameraCalibrationErrors` | `stereoCalibrationErrors`

Functions
`estimateFisheyeParameters` | `undistortFisheyePoints` | `world2img` | `img2world2d`

Topics
“Fisheye Calibration Basics”
“Stereo Fisheye Camera Calibration”
“Configure Monocular Fisheye Camera” (Automated Driving Toolbox)

velodyneFileReader

Read point cloud data from Velodyne PCAP file

Description

The `velodyneFileReader` object reads point cloud data from a Velodyne® packet capture (PCAP) file.

Creation

Syntax

```
veloReader = velodyneFileReader(FileName,DeviceModel)
veloReader = velodyneFileReader(FileName,DeviceModel,Name=Value)
```

Description

`veloReader = velodyneFileReader(FileName,DeviceModel)` creates a Velodyne file reader that reads in point cloud data. Specify the PCAP file and the device model that generated the file. The inputs set the `FileName` and `DeviceModel` properties directly. The reader supports the VLP-16, Puck LITE, Puck Hi-Res, VLP-32C, HDL-32E, HDL-64E, VLS-128, and Velarray H800 device models.

`veloReader = velodyneFileReader(FileName,DeviceModel,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, (`OrganizePoints=true`) returns an organized point cloud.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: (`OrganizePoints=true`) returns an organized point cloud.

`CalibrationFile` — Calibration XML file
`''` (default) | string

Calibration XML file, specified as a string. If you do not specify a calibration file, the reader selects a default calibration file containing data from the Velodyne device manual.

Note If using the `"VelarrayH800"` device model, you must specify the calibration file.

`OrganizePoints` — Logical to set the structure for the output point cloud
`true` (default) | `false`

Logical to set the structure for the output point cloud, specified as a numeric or logical `1` (`true`) or `0` (`false`) scalar.

To return an organized point cloud structure, set `OrganizePoints` to `true`. For an organized point cloud, every row represents a separate laser scan, and the number of columns is based on the horizontal angle resolution of the sensor.

To return an unorganized point cloud structure, set `OrganizePoints` to `false`.

Properties

`FileName` — Velodyne PCAP file name

character vector | string scalar

This property is read-only.

Name of Velodyne PCAP file to read lidar data from, specified as a character vector or string scalar.

`DeviceModel` — Velodyne device model name

`"VLP16"` | `"PuckLITE"` | `"PuckHiRes"` | `"VLP32C"` | `"HDL32E"` | `"HDL64E"` | `"VLS128"` | `"VelarrayH800"`

This property is read-only.

Velodyne device model name, specified as `"VLP16"`, `"PuckLITE"`, `"PuckHiRes"`, `"VLP32C"`, `"HDL32E"`, `"HDL64E"`, `"VLS128"`, or `"VelarrayH800"`.

Note Specifying the incorrect device model returns an improperly calibrated point cloud.

`CalibrationFile` — Name of Velodyne calibration XML file

character vector | string scalar

This property is read-only.

Name of the Velodyne calibration XML file, specified as a character vector or string scalar. This calibration file is included with every sensor.

`NumberOfFrames` — Total number of point clouds

positive integer

This property is read-only.

Total number of point clouds in the file, specified as a positive integer.

`Duration` — Total duration of file in seconds

`duration` scalar

This property is read-only.

Total duration of the file in seconds, specified as a `duration` scalar.

`StartTime` — Time of first point cloud reading

`duration` scalar

This property is read-only.

Time of the first point cloud, specified as a `duration` scalar in seconds.

Start and end times are specified relative to the previous whole hour. For instance, if the file is recorded for 7 minutes from 1:58 p.m. to 2:05 p.m., then:

- `StartTime` = 58 min × 60 s = 3480 s
- `EndTime` = `StartTime` + 7 min × 60 s = 3900 s

`EndTime` — Time of last point cloud reading

`duration` scalar

This property is read-only.

Time of the last point cloud reading, specified as a `duration` scalar.

Start and end times are specified relative to the previous whole hour. For instance, if the file is recorded for 7 minutes from 1:58 PM to 2:05 PM, then:

- `StartTime` = 58 min × 60 s = 3480 s
- `EndTime` = `StartTime` + 7 min × 60 s = 3900 s

`CurrentTime` — Time of current point cloud reading

`duration` scalar

Time for the current point cloud reading in seconds, specified as a `duration` scalar. As you read point clouds using `readFrame`, this property is updated with the most recent point cloud reading time. You can use `reset` to reset the value of this property to the default value. The default value matches the `StartTime` property.

`Timestamps` — Start time for each point cloud frame

`duration` vector

This property is read-only.

Start time for each point cloud frame in seconds, specified as a `duration` vector. The length of the vector is equal to the value of `NumberOfFrames` property. The value of first element in the vector is same as that of the `StartTime` property. You can use this property to read point cloud frames captured at different times.

For example, read start time of a point cloud frame from the `Timestamps` property. Pass the start time as an input to the `readFrame` function and read the corresponding point cloud frame.

```
veloReader = velodyneFileReader("lidarData_ConstructionRoad.pcap","HDL32E")
frameTime  = veloReader.Timestamps(10);
ptCloud    = readFrame(veloReader,frameTime);
```

`OrganizePoints` — Logical to set the structure for the output point cloud

`true` | `false`

This property is read-only.

Logical to set the structure for the output point cloud, specified as a numeric or logical `1` (`true`) or `0` (`false`). This read-only property is set using the `OrganizePoints on page 2-0` name-value argument.

HasPositionData — Logical to indicate existence of position data
`true` | `false`

This property is read-only.

Logical to indicate existence of position data in the PCAP file, specified as a logical scalar `1` (`true`) or `0` (`false`).

Object Functions

hasFrame	Determine if another Velodyne point cloud is available
readFrame	Read Velodyne point cloud from file
reset	Reset the CurrentTime property of velodyneFileReader object to the default value

Examples

Display Point Clouds from Velodyne PCAP File

Use the `velodyneFileReader` to read a packet capture (PCAP) file from a Velodyne® sensor. View point clouds using `pcplayer`.

Read in point clouds by using a Velodyne® file reader. Specify the PCAP file to read and the Velodyne® device model.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Define *x-, y-,* and *z-*axes limits for `pcplayer` in meters. Label the axes.

```
xlimits = [-60 60];
ylimits = [-60 60];
zlimits = [-20 20];
```

Create the point cloud player.

```
player = pcplayer(xlimits,ylimits,zlimits);
```

Label the axes.

```
xlabel(player.Axes,'X (m)');
ylabel(player.Axes,'Y (m)');
zlabel(player.Axes,'Z (m)');
```

The first point cloud of interest is captured at 0.3 second into the file. Set the `CurrentTime` property to that time to being reading point clouds from there.

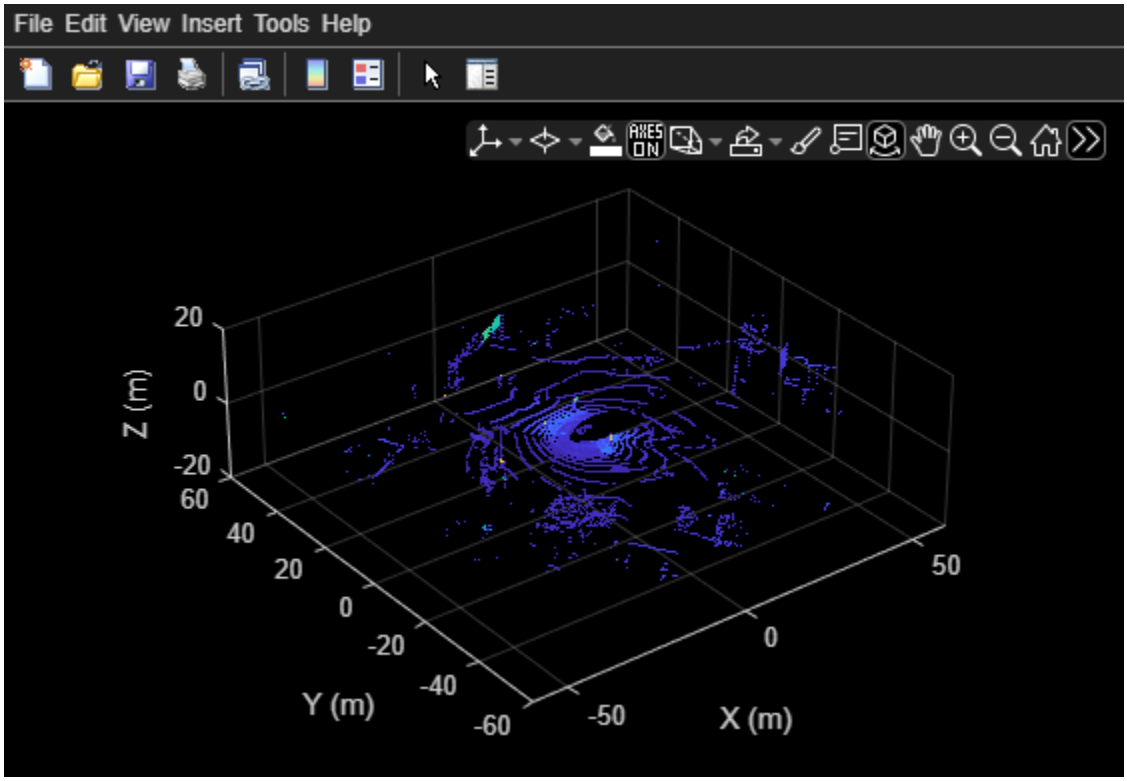
```
veloReader.CurrentTime = veloReader.StartTime + seconds(0.3);
```

Display the point cloud stream for 10 seconds. Remove the last `while` condition to display the full stream.

Use `hasFrame` to check if a new frame is available. Iterate through the file by calling `readFrame` to read in point clouds. Display them using the point cloud player. Remove the last `while` condition to display the full stream.

```
while(hasFrame(veloReader) && player.isOpen() && (veloReader.CurrentTime < veloReader.StartTime +
    ptCloudObj = readFrame(veloReader);
```

```
    view(player,ptCloudObj.Location,ptCloudObj.Intensity);
    pause(0.1);
end
```



Version History

Introduced in R2018a

R2022b: Supports reading from Velarray H800 device model

The `velodyneFileReader` object additionally supports reading point cloud data from the Velarray H800 Velodyne device model.

See Also

Functions

`pcplayer` | `readFrame` | `hasFrame` | `reset` | `pcshow` | `pcread`

Objects

`pcplayer` | `pointCloud`

Topics

“Lidar Data Acquisition and Sensor Simulation” (Lidar Toolbox)
“Get Started with Velodyne LiDAR Sensor Acquisition” (Lidar Toolbox)
“Build a Map from Lidar Data”

External Websites
Velodyne Manuals

hasFrame

Determine if another Velodyne point cloud is available

Syntax

```
isAvailable = hasFrame(veloReader)
```

Description

`isAvailable = hasFrame(veloReader)` determines if another point cloud is available in the packet capture (PCAP) file of the input Velodyne file reader. As you read point clouds using `readFrame`, the point clouds are read sequentially until this function returns `false`.

Examples

Check for Next Point Cloud in Velodyne PCAP File

Create a `velodyneFileReader` object to read a Velodyne® packet capture (PCAP) file and check for available frames using `hasFrame`.

Read in a point cloud by using a Velodyne® file reader. Specify the PCAP file to be read and Velodyne® device model.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Check if the file reader has a point cloud to read.

```
disp(hasFrame(veloReader));
```

```
   1
```

Read the last frame of the file.

```
ptCloudObj = readFrame(veloReader,veloReader.NumberOfFrames);
```

Check again if the file reader has another point cloud available.

```
disp(hasFrame(veloReader));
```

```
   0
```

Input Arguments

`veloReader` — Velodyne file reader

`velodyneFileReader` object

Velodyne file reader, specified as a `velodyneFileReader` object.

Output Arguments

`isAvailable` — Indicator if frame is available
`true` | `false`

Indicator if frame is available, returned as `true` or `false`.

Version History

Introduced in R2018a

See Also

`velodyneFileReader` | `readFrame` | `reset` | `pcplayer` | `pointCloud` | `pcshow`

External Websites
Velodyne Manuals

readFrame

Read Velodyne point cloud from file

Syntax

```
ptCloud = readFrame(veloReader)
ptCloud = readFrame(veloReader,frameNumber)
ptCloud = readFrame(veloReader,frameTime)
[ptCloud,pointTimestamps] = readFrame( ___ )
[ptCloud,pointTimestamps,positionData] = readFrame( ___ )
```

Description

`ptCloud = readFrame(veloReader)` reads the next point cloud in sequence from the Velodyne PCAP file and returns a `pointCloud` object.

`ptCloud = readFrame(veloReader,frameNumber)` reads the point cloud with the specific frame number from the file.

`ptCloud = readFrame(veloReader,frameTime)` reads the first point cloud recorded at or after the given `frameTime`.

`[ptCloud,pointTimestamps] = readFrame( ___ )` returns the timestamps for all points in the point cloud using any combination of input arguments from previous syntaxes.

`[ptCloud,pointTimestamps,positionData] = readFrame( ___ )` returns the positional information for each point cloud frame.

Examples

Read Point Cloud from File Using Time Duration

Create a `velodyneFileReader` object to read a Velodyne® packet capture (PCAP) file and select specific point clouds using a `duration` scalar.

Read in point clouds by using a Velodyne® file reader. Specify the PCAP file to read and the Velodyne® device model.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Create a `duration` scalar that represents one second after the first point cloud reading.

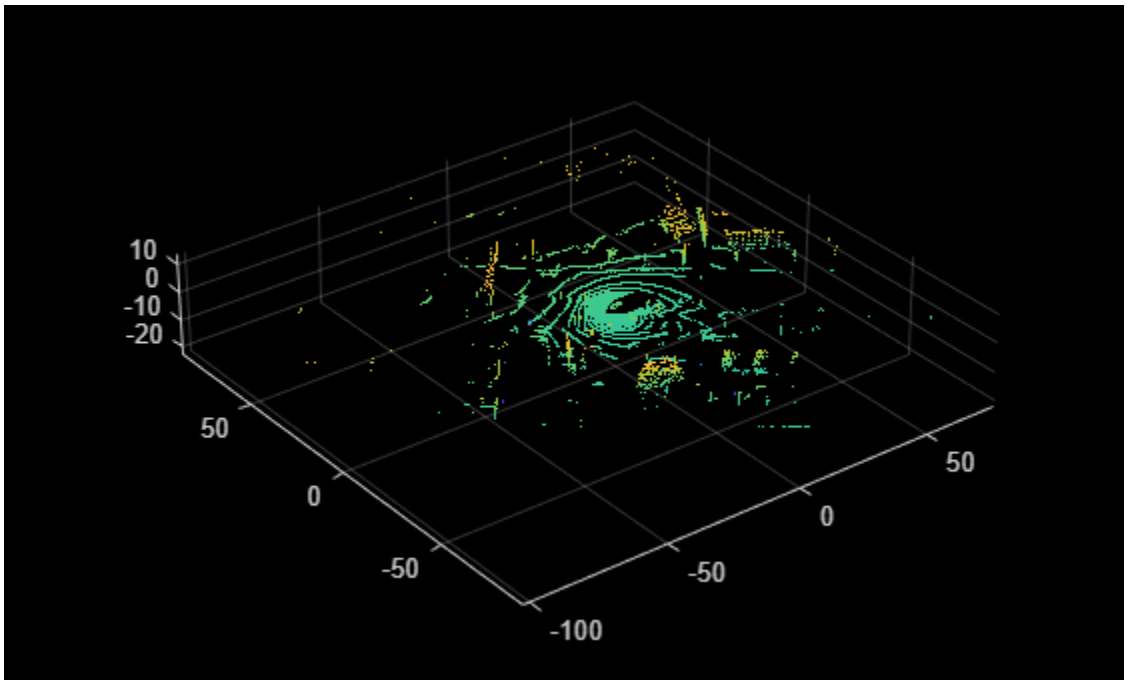
```
timeDuration = veloReader.StartTime + duration(0,0,1,Format='s');
```

Read the first point cloud recorded at or after the given time duration.

```
ptCloudObj = readFrame(veloReader,timeDuration);
```

Display the point cloud using `pcshow`.

```
figure
pcshow(ptCloudObj)
```



Input Arguments

`veloReader` — Velodyne file reader
`velodyneFileReader` object

Velodyne file reader, specified as a `velodyneFileReader` object.

`frameNumber` — Frame number of desired point cloud in the file
positive integer

Frame number of the desired point cloud in file, specified as a positive integer. Frame numbers are sequential.

`frameTime` — Frame time of desired point cloud in file
`duration` scalar

Frame time of the desired point cloud in the file, specified as a `duration` scalar in seconds. The first frame available at or after `frameTime` is given.

Output Arguments

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, returned as a `pointCloud` object.

`pointTimestamps` — Timestamps for all points in point cloud
`duration` vector | `duration` matrix

Timestamps for all points in the point cloud, returned as a `duration` vector or a matrix.

The function returns timestamps as:

- An *M*-element vector for unorganized point clouds. *M* is the total number of points in the point cloud.
- An *M*-by-*N* matrix for organized point clouds. *M* and *N* are the number of rows and columns in the organized point cloud, respectively.

Timestamps are between the values of `StartTime` and `EndTime`, which are properties of the `velodyneFileReader` object. Points in an organized point cloud with a `NaN` value is returned with a time stamp set to `NaN`.

`positionData` — Position data
structure

Position data, returned as a structure with five fields:

Field	Description
`GPSReadings`	GPS reading data, returned as an *M*-by-4 matrix of data type `double`. The number of rows *M* is the number of position readings within a given frame. Each row is of the form `[Latitude Longitude Heading Altitude]`, where : • `Latitude` — Latitude in decimal degrees obtained by the external GPS where north is positive. • `Longitude` — Longitude in decimal degrees obtained by the external GPS where east is positive. • `Heading` — Horizontal direction of motion in degrees from True North, obtained by the external GPS. Data in this column will be `NaN` when an external GPS is connected and a $GPGGA NMEA message is sent. • `Altitude` — Altitude in meters above sea level. Data in this column will be `NaN` when an external GPS is connected and a $GPRMC NMEA message is sent. If the GPS hardware is not connected, then the function returns `GPSReadings` as an empty array, `[]`.

Field	Description
`GyroscopeReadings`	Raw gyroscopic rotation readings in radians per second, collected from the three internal gyroscopes in GPS hardware, returned as an *M*-by-3 matrix of data type `single`. The matrix has these columns: • `Gyro1` — Gyroscopic rotation with respect to the *y*-axis where the positive direction is clockwise around the *y*-axis when viewed from the origin. • `Gyro2` — Gyroscopic rotation with respect to the *x*-axis where the positive direction is counter-clockwise around the *x*-axis when viewed from the origin. • `Gyro3` — Gyroscopic rotation with respect to the *z*-axis where the positive direction is clockwise around the *z*-axis when viewed from the origin. If the Velodyne device model is not HDL-32E, then the function returns `GyroscopeReadings` as an empty array, `[]`.
`AccelerometerReadings`	Raw accelerometer readings, in m/s^2, collected from the three internal accelerometers in GPS hardware, returned as an *M*-by-6 matrix of data type `single`. The columns of the matrix correspond to the *x*-, *y*-, and *z*-axis-axes of the sensor body for both the primary and backup accelerometer readings, as defined in the HDL-32E user manual [1 on page 2-1443]: • `AccelYPrimary` — Increases in the positive *y*-axis direction. • `AccelXPrimary` — Increases in the positive *x*-axis direction. • `AccelZPrimary` — Decreases in the positive *z*-axis direction. • `AccelYBackup` — Increases in the positive *y*-axis direction. • `AccelXBackup` — Increases in the negative *x*-axis direction. • `AccelZBackup` — Decreases in the positive *z*-axis direction. If the Velodyne device model is not HDL-32E, then the function returns `AccelerometerReadings` as an empty array, `[]`.
`PositionTimestamps`	Timestamp for each position reading, returned as an *M*-by-1 `duration` vector.
`GPSTimes`	Time of the GPS in UTC, returned as an *M*-by-1 `duration` vector, used to synchronize the timestamps of the GPS coordinates with their respective point clouds.

`positionData` returns `[]` when there is no position data in the PCAP file.

Version History

Introduced in R2018a

R2023a: Returns positional information for each point cloud frame

The `readFrame` function can return the raw position readings of point cloud data for each frame as a structure with fields for the GPS locations, gyroscope readings, accelerometer readings, position reading timestamps, and GPS reading timestamps.

R2022b: Returns timestamps for all points in point cloud

The `readFrame` function additionally returns the timestamps for all the points in the point cloud as a `duration` vector or matrix.

References

[1] Velodyne LiDAR, Inc. "*HDL-32E User Manual*." 63-9113 Rev. M, 2018.

See Also

Functions
`pcshow` | `reset` | `hasFrame`

Objects
`velodyneFileReader` | `pcplayer` | `pointCloud`

External Websites
Velodyne Manuals

reset

Reset the `CurrentTime` property of `velodyneFileReader` object to the default value

Syntax

```
reset(veloReader)
```

Description

`reset(veloReader)` resets the `CurrentTime` property of `velodyneFileReader` to the default value. The default value is the `StartTime` property of `velodyneFileReader`.

Examples

Reset the velodyneFileReader Object

Construct `velodyneFileReader` object.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','PuckLITE');
```

Inspect the properties of the `velodyneFileReader` object to know the `CurrentTime`. The default value of `CurrentTime` is the value of `StartTime`.

```
veloReader
```

```
veloReader =
  velodyneFileReader with properties:

           FileName: 'B:\runnable\matlab\toolbox\vision\visiondata\lidarData_ConstructionRoad.pca
        DeviceModel: 'PuckLITE'
    CalibrationFile: 'B:\runnable\matlab\toolbox\shared\pointclouds\utilities\velodyneFileReader
     OrganizePoints: 1
     NumberOfFrames: 40
           Duration: 1.9982 sec
          StartTime: 1145.1 sec
            EndTime: 1147.1 sec
         Timestamps: [1145.1 sec    1145.1 sec    1145.2 sec    1145.2 sec    1145.3 sec    1145
    HasPositionData: 1
        CurrentTime: 1145.1 sec
```

Use `readFrame` to read a point cloud frame from the `velodyneFileReader` object.

```
frameNumber = 30;
ptCloud = readFrame(veloReader,frameNumber);
```

Inspect the `CurrentTime` property of `velodyneFileReader` object.

```
veloReader.CurrentTime
```

```
ans = duration
   1146.6 sec
```

Reset the `CurrentTime` property of `velodyneFileReader` object.

```
reset(veloReader)
```

Inspect the `CurrentTime` property and verify that its value is reset to the default value.

```
veloReader.CurrentTime
```

```
ans = duration
   1145.1 sec
```

Input Arguments

`veloReader` — Velodyne file reader

`velodyneFileReader` object

Velodyne file reader, specified as a `velodyneFileReader` object.

Version History

Introduced in R2018a

See Also

`velodyneFileReader` | `readFrame` | `hasFrame`

External Websites

Velodyne Manuals

changeFilePaths

Change file paths in ground truth data

Syntax

```
unresolvedPaths = changeFilePaths(gTruth,alternativePaths)
[unresolvedPaths,unresolvedFolderPaths] = changeFilePaths(gTruth,
alternativePaths)
```

Description

`unresolvedPaths = changeFilePaths(gTruth,alternativePaths)` changes the file paths stored in a `groundTruth` object, `gTruth`, based on pairs of current paths and alternative paths, `alternativePaths`. If `gTruth` is a vector of `groundTruth` objects, the function changes the file paths across all objects. The function returns the unresolved paths in `unresolvedPaths`. An unresolved path is any current path in `alternativePaths` not found in `gTruth` or any alternative path in `alternativePaths` not found at the specified path location. In both cases, `unresolvedPaths` returns only the current paths.

Use this function to update the file paths of ground truth data that changes folder locations. You can change file paths for the ground truth data source and pixel label data.

`[unresolvedPaths,unresolvedFolderPaths] = changeFilePaths(gTruth, alternativePaths)` additionally returns unresolved folder paths `unresolvedFolderPaths`, which lists the folder locations corresponding to the unresolved current paths returned in `unresolvedPaths`. Use `unresolvedFolderPaths` to identify unresolved paths at the folder level when `unresolvedPaths` contains many files from the same folder.

Examples

Change File Paths in Ground Truth Data

Change the file paths to the data source and pixel label data in a `groundTruth` object.

Load a `groundTruth` object containing ground truth data into the workspace. The data source and pixel label data of the object contains file paths corresponding to an image sequence showing a building. MATLAB® displays a warning that the path to the data source cannot be found.

```
load('gTruthSeq.mat');
```

```
Warning: The data source points to a directory that cannot be found.
'C:\CFP\building'
Update the DataSource using <a href="matlab:doc('changeFilePaths')">changeFilePaths</a> method.
```

Display the current path to the data source.

```
gTruth.DataSource
```

```
ans =
'C:\CFP\building'
```

Specify the current path to the data source and an alternative path and store these paths in a cell array. Use the `changeFilePaths` function to update the data source path based on the paths in the cell array. Because the function does not find the pixel label data at the specified new path, it returns the current unresolved paths.

```
currentPathDataSource = "C:\CFP\building";
newPathDataSource = fullfile(matlabroot,"toolbox\vision\visiondata\building");
alternativePaths = {[currentPathDataSource newPathDataSource]};
unresolvedPaths = changeFilePaths(gTruth,alternativePaths)
```

```
unresolvedPaths = 5×1 string array
    "C:\CFP\building\PixelLabelData\Label_1.png"
    "C:\CFP\building\PixelLabelData\Label_2.png"
    "C:\CFP\building\PixelLabelData\Label_3.png"
    "C:\CFP\building\PixelLabelData\Label_4.png"
    "C:\CFP\building\PixelLabelData\Label_5.png"
```

Verify that the paths in the `groundTruth` object match the unresolved paths returned by the `changeFilePaths` function. The unresolved paths are stored in the `LabelData` property of the `groundTruth` object, in the `PixelLabelData` column.

```
gTruth.LabelData.PixelLabelData
```

```
ans = 5×1 cell array
    "'C:\CFP\building\PixelLabelData\Label_1.png'"
    "'C:\CFP\building\PixelLabelData\Label_2.png'"
    "'C:\CFP\building\PixelLabelData\Label_3.png'"
    "'C:\CFP\building\PixelLabelData\Label_4.png'"
    "'C:\CFP\building\PixelLabelData\Label_5.png'"
```

Specify the current path and an alternative path for the pixel label files and change the file paths. The function updates the paths for all pixel labels. Because the function resolves all paths, it returns an empty array of unresolved paths.

```
currentPathPixels = "C:\CFP\building\PixelLabelData";
newPathPixels = fullfile(matlabroot,"toolbox\vision\visiondata\buildingPixellabels");
alternativePaths = {[currentPathPixels newPathPixels]};
unresolvedPaths = changeFilePaths(gTruth,alternativePaths)
```

```
unresolvedPaths =

  0×0 empty string array
```

To view the new data source path, use the `gTruth.DataSource` command. To view the new pixel label data paths, use the `gTruth.LabelData.PixelLabelData` command.

Input Arguments

`gTruth` — Ground truth data

`groundTruth` object | array of `groundTruth` objects

Ground truth data, specified as a `groundTruth` object or array of `groundTruth` objects. You can export these objects from the Ground Truth Labeler app or create them programmatically.

`alternativePaths` — Alternative file paths

1-by-2 string vector | cell array of 1-by-2 string vectors

Alternative file paths, specified as a 1-by-2 string vector or cell array of 1-by-2 string vectors of the form [p_{current} p_{new}].

- p_{current} is a current file path in `gTruth`. This file path can be from the data source or pixel label data of `gTruth`. Specify p_{current} using backslashes as the path separators.
- p_{new} is the new path to which you want to change p_{current}. Specify p_{new} using either forward slashes or backslashes as the path separators.

You can specify alternatives paths to these files:

- Data source — This path is stored in the `DataSource` property of `gTruth`. If you use `changeFilePaths` to update the `DataSource` property of a `groundTruth` object, the new path must already exist; otherwise, the path won't update and the old value remains. Only after the file exists at the new location will the `DataSource` property update to reflect the new path and data type.
- Pixel label data — These paths are stored in the `PixelLabelData` column of the `LabelData` property of `gTruth`.

If `gTruth` is a vector of `groundTruth` objects, the function changes the file paths across all objects.

Example: `["C:\Pixels\PixelLabelData_1" "C:\Pixels\PixelLabelData_2]` changes the path to the pixel label data folder. The function updates the path in all pixel label files stored in that folder.

Example: `{["C:\Pixels\PixelLabelData_1" "C:\Pixels\PixelLabelData_2]; ["B:\Sources\video.mp4" "C:\Sources\video.mp4"]}` changes the path to the pixel label data folder and the drive letter in the path to the data source.

Output Arguments

`unresolvedPaths` — Unresolved file paths

string array

Unresolved file paths, returned as a string array. If the function cannot find either the current path or new path in the string vectors specified by `alternativePaths`, then it returns the unresolved current paths in `unresolvedPaths`.

If the function finds and resolves all file paths, then it returns `unresolvedPaths` as an empty string array.

`unresolvedFolderPaths` — Unresolved folder paths

string array

Unresolved folder paths, returned as a string array. This value lists the folder locations for the unresolved current paths returned in `unresolvedPaths`, providing a shorter list when many unresolved files come from the same folder.

If the function finds and resolves all file paths, then it returns `unresolvedFolderPaths` as an empty string array.

Version History

Introduced in R2018b

R2026a: Additional output argument `unresolvedFolderPaths`

The function can now also return an `unresolvedFolderPaths` output, which lists the folder locations for the unresolved file paths.

See Also

`groundTruth` | `groundTruthDataSource`

Topics

“Share and Store Labeled Ground Truth Data”
“How Labeler Apps Store Exported Pixel Labels”

labelDefinitionCreator

Object for storing, modifying and creating label definitions table

Description

The `labelDefinitionCreator` object stores definitions of labels, sublabels, and attributes for labeling ground truth data. Use "Object Functions" on page 2-1451 to add, remove, modify, or display label definitions. Using the `create` object function, you can create label definitions table from the `labelDefinitionCreator` object. You can use this label definitions table with the Image Labeler, and Video Labeler apps. To create a label definitions table for use with the Ground Truth Labeler app, use a `labelDefinitionCreatorMultisignal` object instead.

Creation

Syntax

```
ldc = labelDefinitionCreator
ldc = labelDefinitionCreator(labelDefs)
```

Description

`ldc = labelDefinitionCreator` creates an empty label definition creator object, `ldc`. Add label definitions by using "Object Functions" on page 2-1451. The details of the stored labels, sublabels, and attributes can be inspected using the `info` object function.

`ldc = labelDefinitionCreator(labelDefs)` creates a label definition creator object `ldc` and stores definitions from the label definitions table `labelDefs`. Use "Object Functions" on page 2-1451 to add new label definitions or modify the existing label definitions. The details of the stored labels, sublabels, and attributes can be inspected using the `info` object function.

Input Arguments

`labelDefs` — Label definitions
table

Label definitions, specified as a table with up to seven columns. The possible columns are *Name*, *Type*, *LabelColor*, *PixelLabelID*, *Group*, *Description*, and *Hierarchy*. This table specifies the definitions of labels, sublabels, and attributes for labeling ground truth data. For more details, see "LabelDefinitions" on page 2-0 .

Output Arguments

`ldc` — Label definition creator
`labelDefinitionCreator` object

Label definition creator, returned as a `labelDefinitionCreator` object that contains information about label definitions associated with ground truth data.

Object Functions

addLabel Add label to label definition creator object
addSublabel Add sublabel to label in label definition creator object
addAttribute Add attributes to label or sublabel in label definition creator object
removeLabel Remove label from label definition creator object
removeSublabel Remove sublabel from label in label definition creator object
removeAttribute Remove attribute from label or sublabel in label definition creator object
editLabelGroup Modify a label group name
editGroupName Change group name
editLabelDescription Modify description of label or sublabel in label definition creator object
editAttributeDescription Modify description of attribute in label definition creator object
create Create label definitions table from the label definition creator object
info Display information about a label, sublabel, or attribute stored in label definition creator object

Examples

Create Label Definition Creator Object and Add Label Definitions

Create an empty label definition creator object.

```
ldc = labelDefinitionCreator
```

```
ldc =

labelDefinitionCreator with 0 labels. Use the addLabel method to add a label.
```

Add a label with the name "Vehicle" and specify the type of label as a rectangle.

```
addLabel(ldc,'Vehicle',labelType.Rectangle)
```

Add a sublabel with the name "Wheel" and an attribute with the name "Color" to the label "Vehicle" stored in `ldc`. Specify the type of sublabel as a rectangle and the attribute type as a string with value `'Red'`.

```
addSublabel(ldc,'Vehicle','Wheel',labelType.Rectangle)
addAttribute(ldc,'Vehicle','Color',attributeType.String,'Red')
```

Display the details of the label definition creator object.

```
ldc
```

```
ldc =

labelDefinitionCreator contains the following labels:

    Vehicle with 1 sublabels and 1 attributes.    (info)

For more details about attributes and sublabels, use the info method.
```

Create Label Definition Creator Object from Existing Label Definitions Table

Load an existing label definitions table into the workspace.

```
labelDefFile = fullfile(toolboxdir('vision'), 'visiondata', 'labelDefsWithAttributes.mat');
ld = load(labelDefFile)
```

```
ld =

  struct with fields:

    labelDefs: [4×4 table]
```

Create a label definition creator object from the label definitions table. Display the details of the label definition creator object.

```
ldc = labelDefinitionCreator(ld.labelDefs)
```

```
ldc =

labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes.    (info)
    Pedestrian with 0 sublabels and 0 attributes.    (info)
    LaneMarker with 0 sublabels and 2 attributes.    (info)
    TrafficLight with 1 sublabels and 0 attributes.    (info)

For more details about attributes and sublabels, use the info method.
```

Version History

Introduced in R2018b

See Also

Apps

Image Labeler | Video Labeler

Objects

`groundTruth` | `labelType` | `attributeType`

addLabel

Add label to label definition creator object

Syntax

```
addLabel(ldc,labelName,typeOfLabel)
addLabel( ___ ,Name,Value)
```

Description

`addLabel(ldc,labelName,typeOfLabel)` adds a label with the specified name and type to the label definition creator object `ldc`.

`addLabel( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

Examples

Add Label Using Label Definition Creator

Create a labelDefinitionCreator.

```
ldc = labelDefinitionCreator;
```

Add a label named 'Car'.

```
addLabel(ldc,'Car',labelType.Rectangle);
```

Inspect the label.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 0 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Add another label named 'StopSign' in a group named 'TrafficSign'. Add a description.

```
addLabel(ldc,'StopSign','Rectangle','Group','TrafficSign','Description','Bounding boxes for stop
```

Inspect the labels.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 0 sublabels and 0 attributes and belongs to None group.    (info)
    StopSign with 0 sublabels and 0 attributes and belongs to TrafficSign group.    (info)
```

```
For more details about attributes and sublabels, use the info method.
```

Add Labels Related to Driving Scene to Label Definition Creator Object

Create an empty label definition creator object.

```
ldc = labelDefinitionCreator
```

```
ldc =
labelDefinitionCreator
```

Add label "Vehicle" to the label definition creator object. Specify the type of label as a rectangle.

```
addLabel(ldc,'Vehicle',labelType.Rectangle)
```

Add label "StopSign" to the label definition creator object. Specify the type of label as a rectangle and add a description to the label.

```
addLabel(ldc,'StopSign',labelType.Rectangle,'Description','Bounding boxes for stop signs')
```

Display the details of the label definition creator object.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 0 attributes and belongs to None group.    (info)
    StopSign with 0 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the label "Vehicle" using the object function `info`.

```
info(ldc,'Vehicle')
```

```
           Name: "Vehicle"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: []
    Description: ' '
```

Display information about the label "StopSign" using the object function `info`.

```
info(ldc,'StopSign')
```

```
           Name: "StopSign"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: []
    Description: 'Bounding boxes for stop signs'
```

Input Arguments

`ldc` — Label definition creator
`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label name
character vector | string scalar

Label name, specified as a character vector or string scalar that uniquely identifies the label to be added.

`typeOfLabel` — Type of label
`labelType` enumeration | character vector | string scalar

Type of label, specified as one of these values:

- `labelType` enumeration — You can use any of these `labelType` enumerators to specify the type of label: `Rectangle`, `Line`, `PixelLabel`, `Scene`, or `Custom`.

Example: `addLabel(ldc,'Car',labelType.Rectangle);`

- Character vector or string scalar — This value must partially or fully match one of the `labelType` enumerators.

Example: `addLabel(ldc,'Car','Rec');`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `addLabel(ldc,'Car',labelType.Rectangle,'Description','Type of Vehicle');`

`Group` — Group name
`'None'` (default) | character vector | string scalar

Group name, specified as a comma-separated pair consisting of `'Group'` and a character vector or string scalar. Use this name-value pair to specify a name for a group of labels.

`Description` — Label description
`''` (default) | character vector | string scalar

Label description, specified as a comma-separated pair consisting of `'Description'` and a character vector or string scalar. Use this name-value pair to describe the label.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator` | `labelType`

Functions
`addSublabel` | `addAttribute` | `editLabelDescription` | `removeLabel`

addSublabel

Add sublabel to label in label definition creator object

Syntax

```
addSublabel(ldc,labelName,sublabelName,typeOfSublabel)
addSublabel( ___ ,Name,Value)
```

Description

`addSublabel(ldc,labelName,sublabelName,typeOfSublabel)` adds a sublabel with the specified name and type to the indicated label. The sublabel is added under the hierarchy for the specified label in the label definition creator object `ldc`.

`addSublabel( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

Examples

Add Sublabels to Labels in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
load('labelDefsTable.mat')
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 0 sublabels and 0 attributes and belongs to None group.    (info)
    TrafficLight with 0 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Add sublabel "Wheel" to the label "Car". Specify the label type for sublabel as a rectangle and add description to the sublabel.

```
addSublabel(ldc,'Car','Wheel',labelType.Rectangle,'Description','Bounding box for the wheel')
```

Add sublabels "RedLight" and "GreenLight" to the label "TrafficLight". Specify the label type for the sublabels as rectangle.

```
addSublabel(ldc,'TrafficLight','RedLight',labelType.Rectangle)
addSublabel(ldc,'TrafficLight','GreenLight',labelType.Rectangle)
```

Display the details of the label definition creator object.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 1 sublabels and 0 attributes and belongs to None group.    (info)
    TrafficLight with 2 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the sublabels under the label "Car" using the `info` object function.

```
info(ldc,'Car')
```

```
           Name: "Car"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: "Wheel"
    Description: 'Bounding box for car'
```

Display information about the sublabels under the label "TrafficLight" using the `info` object function.

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: ["RedLight"    "GreenLight"]
    Description: 'Bounding boxes for traffic light'
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label name

character vector | string scalar

Label name, specified as a character vector or string scalar that uniquely identifies the label to which the sublabel is associated.

`sublabelName` — Sublabel name

character vector | string scalar

Sublabel name, specified as a character vector or string scalar that identifies the sublabel to be added.

`typeOfSublabel` — Type of sublabel

`labelType` enumeration | character vector | string scalar

Type of sublabel, specified as one of these values:

- `labelType` enumeration — The type of sublabel must be one of these `labelType` enumerators: `Rectangle` or `Line`.

Example: `addSublabel(ldc,'Car','Wheel',labelType.Rectangle)`

- Character vector or string scalar — This value must partially or fully match one of these `labelType` enumerators: `Rectangle` or `Line`.

Example: `addSublabel(ldc,'Car','Wheel','Rec')`

Note The `labelType` enumerators `PixelLabel`, `Scene`, and `Custom` are not supported as values for the type of sublabel in labeling apps.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `addSublabel(ldc,'Car','Wheel','Rect','Description','Bounding box for Wheel');`

`Description` — Sublabel description
`' '` (default) | character vector | string scalar

Sublabel description, specified as a comma-separated pair consisting of `'Description'` and a character vector or string scalar. Use this name-value pair to describe the sublabel.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator` | `labelType`

Functions
`addLabel` | `addAttribute` | `removeSublabel`

addAttribute

Add attributes to label or sublabel in label definition creator object

Syntax

```
addAttribute(ldc,labelName,attributeName,typeOfAttribute,attributeDefault)
addAttribute( ___ ,Name,Value)
```

Description

`addAttribute(ldc,labelName,attributeName,typeOfAttribute,attributeDefault)` adds an attribute with specified name and type to the indicated label or sublabel. The attribute is added under the hierarchy for the specified label or sublabel in the label definition creator object `ldc`.

`addAttribute( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

Examples

Add Attributes to Label and Sublabel in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
load('sublabelDefsTable.mat')
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 1 sublabels and 0 attributes and belongs to None group.    (info)
    TrafficLight with 2 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Add attribute "Color" to the label "Car". Specify the attribute type as a string with value `'Red'`.

```
addAttribute(ldc,'Car','Color',attributeType.String,'Red')
```

Display information about the sublabels under the label "TrafficLight".

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
```

```
      Sublabels: ["RedLight"    "GreenLight"]
    Description: 'Bounding boxes for traffic light'
```

Add attribute "IsOn" to the label "TrafficLight/RedLight". Specify the attribute type as logical with value `true`.

```
addAttribute(ldc,'TrafficLight/RedLight','IsOn','logical',true)
```

Display the details of the label definition creator object.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 1 sublabels and 1 attributes and belongs to None group.    (info)
    TrafficLight with 2 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the attributes under the label "Car" using the `info` object function.

```
info(ldc,'Car')
```

```
           Name: "Car"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: "Color"
      Sublabels: "Light"
    Description: 'Bounding box for vehicle'
```

Display information about the attributes for sublabel "RedLight" under the label "TrafficLight" using the `info` object function.

```
info(ldc,'TrafficLight/RedLight')
```

```
           Name: "RedLight"
           Type: Rectangle
     Attributes: "IsOn"
      Sublabels: []
    Description: ' '
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label or sublabel name

character vector | string scalar

Label or sublabel name, specified as a character vector or string scalar that uniquely identifies the label or sublabel to which the attribute is to be added.

- To specify a label, use the form '*labelName*'.
- To specify a sublabel, use the form '*labelName/sublabelName*'. In this case, the attribute associates to the sublabel.

`attributeName` — Attribute name
character vector | string scalar

Attribute name, specified as a character vector or string scalar that identifies the attribute to be added to the label or sublabel.

`typeOfAttribute` — Type of attribute
`attributeType` enumeration | character vector | string scalar

Type of attribute, specified as one of these values:

- `attributeType` enumeration — The type of attribute must be one of these enumerators in `attributeType` enumeration: `Numeric`, `Logical`, `String`, or `List`.

Example: `addAttribute(ldc,'Car','Color',attributeType.String,'Red');`

- Character vector or string scalar — This value must partially or fully match one of the enumerators in `attributeType` enumeration.

Example: `addAttribute(ldc,'Car','Color','Str','Red');`

`attributeDefault` — Default value of attribute
numeric scalar | logical scalar | character vector | string scalar | cell array of character vectors | cell array of string scalars

Default value of the attribute, specified as one of these values:

- Numeric scalar — Specify this value when `typeOfAttribute` is `Numeric`.
- Logical scalar — Specify this value when `typeOfAttribute` is `Logical`.
- Character vector or string scalar — Specify this value when `typeOfAttribute` is `String`.
- Cell array of character vectors or cell array of string scalars — Specify this value when `typeOfAttribute` is `List`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `addAttribute(ldc,'Car/Wheel','Outsidediameter',attributeType.Numeric,740,'Description','Outside diameter in mm');`

`Description` — Attribute description
`''` (default) | character vector | string scalar

Attribute description, specified as a comma-separated pair consisting of `'Description'` and a character vector or string scalar. Use this name-value pair to describe the attribute.

Version History

Introduced in R2018b

See Also

Objects

`labelDefinitionCreator` | `attributeType`

Functions

`addLabel` | `addSublabel` | `removeAttribute` | `editAttributeDescription`

removeLabel

Remove label from label definition creator object

Syntax

```
removeLabel(ldc,labelName)
```

Description

`removeLabel(ldc,labelName)` removes the specified label from the label definition creator object `ldc`.

Note Removing a label also removes any attributes or sublabels associated with that label.

Examples

Remove Label from Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
labelDefFile = fullfile(toolboxdir('vision'),'visiondata','labelDefsWithAttributes.mat');
ld = load(labelDefFile)
```

```
ld = struct with fields:
    labelDefs: [4×4 table]
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(ld.labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Remove the label "Vehicle" from the label definition creator object.

```
removeLabel(ldc,'Vehicle')
```

Display the details of the label definition creator object to confirm that "Vehicle" is removed from the label definitions.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Input Arguments

`ldc` — Label definition creator
`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label name
character vector | string scalar

Label name, specified as a character vector or string scalar that uniquely identifies the label to be removed from the label definition creator object.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator`

Functions
`addLabel` | `removeAttribute`

removeSublabel

Remove sublabel from label in label definition creator object

Syntax

```
removeSublabel(ldc,labelName,sublabelName)
```

Description

`removeSublabel(ldc,labelName,sublabelName)` removes the specified sublabel from the indicated label. This label must be in the label definition creator object `ldc`.

Note Removing a sublabel also removes any attributes associated with that sublabel.

Examples

Remove Sublabel from Label in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
labelDefFile = fullfile(toolboxdir('vision'), 'visiondata', 'labelDefsWithAttributes.mat');
ld = load(labelDefFile)
```

```
ld = struct with fields:
    labelDefs: [4×4 table]
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(ld.labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the label "TrafficLight" defined in the label definition creator object.

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
```

```
      Sublabels: "Light"
    Description: 'Mark a tight bounding box around the traffic light. Use the sublabels to mark e
```

Remove the sublabel "Light" from the label "TrafficLight".

```
removeSublabel(ldc,'TrafficLight','Light')
```

Display the details of the label definition creator object to confirm that a sublabel is removed from the label "TrafficLight".

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 0 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Display information about the label "TrafficLight". Confirm that the sublabel "Light" is removed.

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: []
    Description: 'Mark a tight bounding box around the traffic light. Use the sublabels to mark e
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label name

character vector | string scalar

Label name, specified as a character vector or string scalar that uniquely identifies the label to which the sublabel is associated.

`sublabelName` — Sublabel name

character vector | string scalar

Sublabel name, specified as a character vector or string scalar that identifies the sublabel to be removed from the indicated label `labelName`.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator`

Functions
`removeLabel` | `removeAttribute` | `addLabel` | `addSublabel`

removeAttribute

Remove attribute from label or sublabel in label definition creator object

Syntax

```
removeAttribute(ldc,labelName,attributeName)
```

Description

`removeAttribute(ldc,labelName,attributeName)` removes the specified attribute from the indicated label or sublabel in the label definition creator object `ldc`.

Examples

Remove Attributes from Label and Sublabel in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
labelDefFile = fullfile(toolboxdir('vision'),'visiondata','labelDefsWithAttributes.mat');
ld = load(labelDefFile)
```

```
ld = struct with fields:
    labelDefs: [4×4 table]
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(ld.labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Remove an Attribute from a Label

Display information about the label "Vehicle" defined in the label definition creator object.

```
info(ldc,'Vehicle')
```

```
           Name: "Vehicle"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: ["Class"    "Color"    "View"]
      Sublabels: []
    Description: 'Draw a tight bounding box around the vehicle. Use this label for cars, buses ar
```

Remove the attribute "Color" from the label "Vehicle".

```
removeAttribute(ldc,'Vehicle','Color')
```

Display information about the label "Vehicle". Confirm that the attribute "Color" is removed.

```
info(ldc,'Vehicle')
```

```
           Name: "Vehicle"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: ["Class"    "View"]
      Sublabels: []
    Description: 'Draw a tight bounding box around the vehicle. Use this label for cars, buses ar
```

Remove an Attribute from a Sublabel

Display information about the label "TrafficLight" defined in the label definition creator object.

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: "Light"
    Description: 'Mark a tight bounding box around the traffic light. Use the sublabels to mark e
```

Display information about the sublabel "Light" under the label "TrafficLight".

```
info(ldc,'TrafficLight/Light')
```

```
           Name: "Light"
           Type: Rectangle
     Attributes: ["Active"    "Color"]
      Sublabels: []
    Description: 'Mark a tight bounding box around each light.'
```

Remove the attribute "Active" from the sublabel "TrafficLight/Light".

```
removeAttribute(ldc,'TrafficLight/Light','Active')
```

Display information about the sublabel "TrafficLight/Light". Confirm that the attribute "Active" is removed.

```
info(ldc,'TrafficLight/Light')
```

```
           Name: "Light"
           Type: Rectangle
     Attributes: "Color"
      Sublabels: []
    Description: 'Mark a tight bounding box around each light.'
```

Input Arguments

`ldc` — Label definition creator
`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label or sublabel name
character vector | string scalar

Label or sublabel name, specified as a character vector or string scalar that uniquely identifies the label or sublabel from which the attribute is to be removed.

- To specify a label, use the form '*labelName*'.
- To specify a sublabel, use the form '*labelName/sublabelName*'. In this case, the attribute associates to the sublabel.

`attributeName` — Attribute name
character vector | string scalar

Attribute name, specified as a character vector or string scalar that identifies the attribute to be removed from the indicated label or sublabel `labelName`.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator`

Functions
`removeLabel` | `addLabel` | `addAttribute`

editLabelDescription

Modify description of label or sublabel in label definition creator object

Syntax

```
editLabelDescription(ldc,labelName,description)
```

Description

`editLabelDescription(ldc,labelName,description)` modifies the description of a label or sublabel identified by `labelName`. The label or sublabel must be in the label definition creator object `ldc`.

Examples

Modify Description of Label and Sublabel in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
load('editlabelDefs.mat')
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Modify the Description of a Label

Display information about the label "LaneMarker".

```
info(ldc,'LaneMarker')
```

```
           Name: "LaneMarker"
           Type: Line
     LabelColor: {''}
          Group: "None"
     Attributes: ["BoundaryType"    "Location_wrt_Ego"]
      Sublabels: []
    Description: ''
```

Modify the description for the label "LaneMarker".

```
editLabelDescription(ldc,'LaneMarker','Use 5 or more points for curved lane lines.')
```

Display information about the label "LaneMarker" to verify the modified label description.

```
info(ldc,'LaneMarker')
```

```
           Name: "LaneMarker"
           Type: Line
     LabelColor: {''}
          Group: "None"
     Attributes: ["BoundaryType"    "Location_wrt_Ego"]
      Sublabels: []
    Description: 'Use 5 or more points for curved lane lines.'
```

Modify the Description of a Sublabel

Display information about the label "TrafficLight".

```
info(ldc,'TrafficLight')
```

```
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: "Light"
    Description: 'Bounding box for the traffic light. Use sublabels to mark each individual ligh
```

Display information about the sublabel "Light" under the label "TrafficLight".

```
info(ldc,'TrafficLight/Light')
```

```
           Name: "Light"
           Type: Rectangle
     Attributes: ["Active"    "Color"]
      Sublabels: []
    Description: 'lights'
```

Modify the description for the sublabel "Light".

```
editLabelDescription(ldc,'TrafficLight/Light','Mark a tight bounding box around each light.')
```

Display information about the sublabel "Light" to verify the modified sublabel description.

```
info(ldc,'TrafficLight/Light')
```

```
           Name: "Light"
           Type: Rectangle
     Attributes: ["Active"    "Color"]
      Sublabels: []
    Description: 'Mark a tight bounding box around each light.'
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label or sublabel name

character vector | string scalar

Label or sublabel name, specified as a character vector or string scalar that uniquely identifies the label or sublabel for which the description is to be updated.

- To specify a label, use the form '*labelName*'.
- To specify a sublabel, use the form '*labelName/sublabelName*'.

`description` — Description
character vector | string scalar

Description, specified as a character vector or string scalar that contains the new description for the label or sublabel identified by `labelName`.

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator` | `groundTruth`

Functions
`editAttributeDescription`

editLabelGroup

Modify a label group name

Syntax

```
editLabelGroup(ldc,labelName,groupName)
```

Description

`editLabelGroup(ldc,labelName,groupName)` modifies the group name that corresponds to the label identified by `labelName`. The label must be in the label definition creator object `ldc`.

Examples

Modify Group Name for Labels

Create a labelDefinitionCreator.

```
ldc = labelDefinitionCreator;
```

Add a rectangle label named `Car` in a group named `Vehicle`.

```
addLabel(ldc,'Car',labelType.Rectangle,'Group','Vehicle');
```

Add a rectangle label named `Truck` in a group named `FourWheeler`.

```
addLabel(ldc,'Truck',labelType.Rectangle,'Group','FourWheeler');
```

Move the `Car` label into the `FourWheeler` group.

```
editLabelGroup(ldc,'Car','FourWheeler');
```

Inspect the labels.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 0 sublabels and 0 attributes and belongs to FourWheeler group.    (info)
    Truck with 0 sublabels and 0 attributes and belongs to FourWheeler group.    (info)

For more details about attributes and sublabels, use the info method.
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label name
character vector | string scalar

Label name, specified as a character vector or string scalar that uniquely identifies the label that corresponds to the `groupName` you want to modify.

`groupName` — Group name
character vector | string scalar

Group name, specified as a character vector or string scalar that identifies the group you want to modify, which corresponds to the label specified by `labelName`.

Version History

Introduced in R2019a

See Also

Objects
`labelDefinitionCreator`

Functions
`editLabelDescription` | `editGroupName`

editGroupName

Change group name

Syntax

```
editGroupName(ldc,oldname,newname)
```

Description

`editGroupName(ldc,oldname,newname)` changes the group name from `oldname` to `newname`. This function changes the group name in all the label definitions that have the `oldname`.

Examples

Rename Label Group Created with labelDefinitionCreator

Create a `labelDefinitionCreator`.

```
ldc = labelDefinitionCreator;
```

Add labels named `Car` and `Truck` in a group named `Vehicle`.

```
addLabel(ldc,'Car',labelType.Rectangle,'Group','Vehicle');
addLabel(ldc,'Truck',labelType.Rectangle,'Group','Vehicle');
```

Change the `Vehicle` group name `FourWheeler`.

```
editGroupName(ldc,'Vehicle','FourWheeler');
```

Inspect the labels.

```
ldc
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Car with 0 sublabels and 0 attributes and belongs to FourWheeler group.    (info)
    Truck with 0 sublabels and 0 attributes and belongs to FourWheeler group.    (info)

For more details about attributes and sublabels, use the info method.
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`oldname` — Old group name

character vector | string scalar

Old group name, specified as a character vector or string scalar that uniquely identifies group name you want to modify.

`newname` — New group name
character vector | string scalar

New group name, specified as a character vector or string scalar that uniquely identifies the new group name.

Version History

Introduced in R2019a

See Also

Objects
`labelDefinitionCreator`

Functions
`editLabelDescription` | `editLabelGroup`

editAttributeDescription

Modify description of attribute in label definition creator object

Syntax

`editAttributeDescription(ldc,labelName,attributeName,description)`

Description

`editAttributeDescription(ldc,labelName,attributeName,description)` modifies the description of an attribute under the label or sublabel identified by `labelName`. The label or sublabel must be in the label definition creator object `ldc`.

Examples

Modify Description of Attributes in Label Definition Creator Object

Load an existing label definitions table into the workspace.

```
labelDefFile = fullfile(toolboxdir('vision'), 'visiondata', 'labelDefsWithAttributes.mat');
ld = load(labelDefFile)
```

```
ld = struct with fields:
    labelDefs: [4×4 table]
```

Create a label definition creator object from the label definitions table.

```
ldc = labelDefinitionCreator(ld.labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Modify the Description of Attribute Under a Label

Display information about the label "Vehicle".

```
info(ldc,'Vehicle')
```

```
           Name: "Vehicle"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: ["Class"    "Color"    "View"]
```

```
      Sublabels: []
    Description: 'Draw a tight bounding box around the vehicle. Use this label for cars, buses ar
```

Display information about the attribute "Color" under the label "Vehicle".

```
info(ldc,'Vehicle/Color')
```

```
           Name: "Color"
           Type: String
    DefaultValue: ""
     Description: 'Color of the vehicle, specified as a string, such as red, blue, white.'
```

Modify the description of the attribute "Color" under the label "Vehicle".

```
editAttributeDescription(ldc,'Vehicle','Color','Color of the vehicle, specified as a string, e.g
```

Display information about the label "Vehicle" to verify the modified attribute description.

```
info(ldc,'Vehicle/Color')
```

```
           Name: "Color"
           Type: String
    DefaultValue: ""
     Description: 'Color of the vehicle, specified as a string, e.g. "red".'
```

Modify the Description of Attribute Under a Sublabel

Display information about the label "TrafficLight".

```
info(ldc,'TrafficLight')
```

```
          Name: "TrafficLight"
          Type: Rectangle
     LabelColor: {''}
         Group: "None"
     Attributes: []
      Sublabels: "Light"
    Description: 'Mark a tight bounding box around the traffic light. Use the sublabels to mark e
```

Display information about the sublabel "Light" under the label "TrafficLight".

```
info(ldc,'TrafficLight/Light')
```

```
          Name: "Light"
          Type: Rectangle
     Attributes: ["Active"    "Color"]
      Sublabels: []
    Description: 'Mark a tight bounding box around each light.'
```

Display information about the attribute "Active" under the sublabel "TrafficLight/Light".

```
info(ldc,'TrafficLight/Light/Active')
```

```
           Name: "Active"
           Type: Logical
    DefaultValue: 0
     Description: 'Mark true if this light is active and false if not.'
```

Modify the description of the attribute "Active" under the sublabel "TrafficLight/Light".

```
editAttributeDescription(ldc,'TrafficLight/Light','Active','Is Active: true (DefaultValue: 1), fa
```

Display information about the attribute "Active" to verify the modified attribute description.

```
info(ldc,'TrafficLight/Light/Active')
```

```
            Name: "Active"
            Type: Logical
    DefaultValue: 0
     Description: 'Is Active: true (DefaultValue: 1), false (DefaultValue: 0)'
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`labelName` — Label or sublabel name

character vector | string scalar

Label or sublabel name, specified as a character vector or string scalar that uniquely identifies the label or sublabel to which the attribute is associated.

- To specify a label, use the form '*labelName*'.
- To specify a sublabel, use the form '*labelName/sublabelName*'. In this case, the attribute is associated with the sublabel.

`attributeName` — Attribute name

character vector | string scalar

Attribute name, specified as a character vector or string scalar that identifies the attribute for which the description is to be updated.

`description` — Description

character vector | string scalar

Description, specified as a character vector or string scalar that contains the new description for the attribute identified by `attributeName`.

Version History

Introduced in R2018b

See Also

Objects

`labelDefinitionCreator`

Functions

`editLabelDescription`

create

Create label definitions table from the label definition creator object

Syntax

```
labelDefs = create(ldc)
```

Description

`labelDefs = create(ldc)` creates a label definitions table, `labelDefs`, from the label definition creator object `ldc`. You can import the `labelDefs` table into the Image Labeler and Video Labeler apps to label ground truth data.

Examples

Create Label Definitions Table from Label Definition Creator Object

Create an empty label definition creator object.

```
ldc = labelDefinitionCreator
```

```
ldc =
labelDefinitionCreator
```

Add a label "Vehicle" to the label definition creator object. Specify the type of label as a rectangle and add a description to the label.

```
addLabel(ldc,'Vehicle','Rectangle','Description','Bounding box for the vehicle. Use this label fo
```

Add an attribute "IsCar" to the label "Vehicle". Specify the type of attribute as logical with value `true` and add description to the attribute.

```
addAttribute(ldc,'Vehicle','IsCar','logical',true,'Description','Type of vehicle')
```

Add an attribute "IsBus" to the label "Vehicle". Specify the type of attribute as logical with value `false` and add description to the attribute.

```
addAttribute(ldc,'Vehicle','IsBus','logical',false,'Description','Type of vehicle')
```

Create a label definitions table from the definitions stored in the label definition creator object.

```
labelDefs = create(ldc)
```

```
labelDefs=1×6 table
       Name          Type       LabelColor     Group                                  Description
    ___________    _________    __________    ________    ____________________________________________

    {'Vehicle'}    Rectangle    {0×0 char}    {'None'}    {'Bounding box for the vehicle. Use thi
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object. The object `ldc` defines the labels, sublabels, and attributes used for generating the label definitions table `labelDefs`.

Output Arguments

`labelDefs` — Label definitions

table

Label definitions, returned as a table with up to seven columns. The possible columns are *Name*, *Type*, *LabelColor*, *PixelLabelID*, *Group*, *Description*, and *Hierarchy*. This table specifies the definitions of labels, sublabels, and attributes for labeling ground truth data. For more details, see “LabelDefinitions” on page 2-0 .

Version History

Introduced in R2018b

See Also

Objects

`labelDefinitionCreator`

Functions

`addLabel` | `addSublabel` | `addAttribute` | `info`

info

Display information about a label, sublabel, or attribute stored in label definition creator object

Syntax

```
info(ldc,name)
infoStruct = info(ldc,name)
```

Description

`info(ldc,name)` displays information about the specified label, sublabel or attribute stored in the label definition creator object `ldc`.

`infoStruct = info(ldc,name)` returns the information as a structure.

Examples

Display Information About Definitions Stored in Label Definition Creator Object

Load an existing label definitions table into the workspace. Create a label definition creator object.

```
FilePath = fullfile(toolboxdir('vision'),'visiondata','labelDefsWithAttributes.mat');
Ld = load(FilePath);
ldc = labelDefinitionCreator(Ld.labelDefs)
```

```
ldc =
labelDefinitionCreator contains the following labels:

    Vehicle with 0 sublabels and 3 attributes and belongs to None group.    (info)
    Pedestrian with 0 sublabels and 0 attributes and belongs to None group.    (info)
    LaneMarker with 0 sublabels and 2 attributes and belongs to None group.    (info)
    TrafficLight with 1 sublabels and 0 attributes and belongs to None group.    (info)

For more details about attributes and sublabels, use the info method.
```

Get information about the label "TrafficLight" as a structured data.

```
trafficStruct = info(ldc,'TrafficLight')
```

```
trafficStruct = struct with fields:
           Name: "TrafficLight"
           Type: Rectangle
     LabelColor: {''}
          Group: "None"
     Attributes: []
      Sublabels: "Light"
    Description: 'Mark a tight bounding box around the traffic light. Use the sublabels to mark e
```

Get information about the sublabel "Light" under the label "TrafficLight" as a structured data.

```
lightStruct = info(ldc,'TrafficLight/Light')
```

```
lightStruct = struct with fields:
           Name: "Light"
           Type: Rectangle
     Attributes: ["Active"    "Color"]
      Sublabels: []
    Description: 'Mark a tight bounding box around each light.'
```

Get information about the attribute "Color" under the sublabel "TrafficLight/Light" as a structured data.

```
colorStruct = info(ldc,'TrafficLight/Light/Color')
```

```
colorStruct = struct with fields:
           Name: "Color"
           Type: List
      ListItems: {3×1 cell}
    Description: 'Specify the color of the light as one of Red, Yellow (amber) or Green.'
```

Display the `ListItems` field of the structure `colorStruct`.

```
colorStruct.ListItems
```

```
ans = 3×1 cell
    {'Red'   }
    {'Yellow'}
    {'Green' }
```

Input Arguments

`ldc` — Label definition creator

`labelDefinitionCreator` object

Label definition creator, specified as a `labelDefinitionCreator` object.

`name` — Name of label, sublabel, or attribute

character vector | string scalar

Name of label, sublabel, or attribute in the `ldc` object, specified as a character vector or string scalar whose form depends on the type of name you specify.

- To specify a label, use the form '*labelName*'.
- To specify a sublabel, use the form '*labelName/sublabelName*'.
- To specify an attribute, use the form '*labelName/sublabelName/attributeName*'.

Output Arguments

`infoStruct` — Information structure

structure

Information structure, returned as a structure that contains the fields `Name`, `Type`, `Attributes` (when pertinent), `Sublabels` (when pertinent), and `Description`. If `name` specifies an attribute, then `infoStruct` also contains the fields `DefaultValue` and `ListItems` (for List attributes).

Version History

Introduced in R2018b

See Also

Objects
`labelDefinitionCreator`

Functions
`addLabel` | `create`

vision.DeployableVideoPlayer

Namespace: `vision`

Display video

Description

The `DeployableVideoPlayer` object displays video frames. This player is capable of displaying high definition video at high frame rates. This video player object supports C code generation.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Creation

Syntax

```
depVideoPlayer = vision.DeployableVideoPlayer
depVideoPlayer = vision.DeployableVideoPlayer(Name,Value)
```

Description

`depVideoPlayer = vision.DeployableVideoPlayer` returns a video player `depVideoPlayer`, for displaying video frames. This object, unlike the `vision.VideoPlayer` object, can generate C code.

`depVideoPlayer = vision.DeployableVideoPlayer(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `depVideoPlayer = vision.DeployableVideoPlayer('Name','Caption title')`

Properties

`Location` — Bottom left corner of video window

[ ] (default) | two-element vector

Location of bottom left corner of video frame, specified as a two-element vector. The first and second elements are specified in pixels and represent the horizontal and vertical coordinates respectively. The coordinates `[0 0]` represent the bottom left corner of the screen. The default location depends on the screen resolution, and will result in a window positioned in the center of the screen.

`Name` — Video window title bar caption

`'Deployable Video Player'`

Video window title bar caption, specified as the comma-separated pair consisting of '`Name`' and a character vector.

`Size` — Size of video display window
`True size (1:1)` (default) | `Full-screen` | `Custom`

Size of video display window, specified as the comma-separated pair consisting of '`Size`' and `Full-screen`, `True size (1:1)` or `Custom`. When this property is set to `Full-screen`, use the `Esc` key to exit out of full-screen mode.

`CustomSize` — Custom size for video player window
`[300 410]` (default) | two-element vector

Custom size for video player window, specified as the comma-separated pair consisting of '`CustomSize`' and a two-element vector. The first and second elements are specified in pixels and represent the horizontal and vertical components respectively. The video data will be resized to fit the window. This property applies when you set the `Size` property to `Custom`.

`InputColorFormat` — Color format of input signal
`RGB` (default) | `'YCbCr 4:2:2'`

Color format of input signal, specified as the comma-separated pair consisting of '`InputColorFormat`' and `'RGB'` or `'YCbCr 4:2:2'`. The number of columns in the Cb and Cr components must be half the number of columns in Y.

Usage

Syntax

```
depVideoPlayer(videoFrame)
depVideoPlayer(videoFrame,Y,Cb,Cr)
```

Description

`depVideoPlayer(videoFrame)` displays one grayscale or truecolor RGB video frame in the video player.

`depVideoPlayer(videoFrame,Y,Cb,Cr)` displays one frame of YCbCr 4:2:2 video in the color components Y, Cb, and Cr when you set the `InputColorFormat` property to `YCbCr 4:2:2`. The number of columns in the Cb and Cr components must be half the number of columns in the Y component.

Input Arguments

`videoFrame` — Video frame
truecolor | 2-D grayscale image

Video frame, specified as a truecolor or 2-D grayscale image.

`Y,Cb,Cr` — YCbCr color format
`'YCbCr 4:2:2'`

YCbCr color format, returned in the `YCbCr 4:2:2` format.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to vision.DeployableVideoPlayer

isOpen	Visible or hidden status for player

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Play a Video from a File

Create a video reader and a video player object to read and view a video file.

```
vidReader = VideoReader('atrium.mp4');
depVideoPlayer = vision.DeployableVideoPlayer;
```

Continue to read frames of video until the last frame is read. Exit the loop if the user closes the video player window.

```
while hasFrame(vidReader)
    videoFrame = readFrame(vidReader);
    depVideoPlayer(videoFrame);
end
```

Release the video player object.

```
release(depVideoPlayer);
```

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generated code for this function uses a precompiled platform-specific shared library.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects
`vision.VideoFileWriter` | `VideoReader` | `vision.VideoPlayer`

Functions
`delete`

Simulink.ImageType

Specify image data type

Description

Use a `Simulink.ImageType` object to set and share image data types for signal, state, and parameter data in a model.

1. Create an instance of this class in the MATLAB base workspace.
2. Set the properties of the object to describe the image.
3. Assign the data type to supported blocks, which have a **Data type** parameter, that you want to conform to the data type.

The benefits of the `Simulink.ImageType` data type include:

- Strong typing of image data at model interfaces.
- Carrying image data attributes such as color format and pixel data type, eliminating the need to create and maintain secondary signals.
- Enabling the Simulink environment to interpret the data as an image.
- Enabling custom data type replacement with third party implementation, such as the OpenCV class `cv::Mat`.

Note To simulate a model that contains `Simulink.ImageType` signal, set the model configuration parameter **Language** (`SimTargetLang`) to `C++`.

Creation

To create a `Simulink.ImageType` object programmatically, use the `Simulink.ImageType` function described below.

Syntax

```
imageType = Simulink.ImageType
imageType = Simulink.ImageType(Rows,Columns,Channels)
imageType = Simulink.ImageType(Rows,Columns,Channels,'ClassUnderlying',
ClassUnderlying,'ColorFormat',ColorFormat,'Layout',Layout)
```

Description

`imageType = Simulink.ImageType` returns a `Simulink.Parameter` object with default property values.

`imageType = Simulink.ImageType(Rows,Columns,Channels)` returns a `Simulink.ImageType` object and initializes the `Rows`, `Columns`, and `Channels` property to the specified values.

`imageType = Simulink.ImageType(Rows,Columns,Channels,'ClassUnderlying', ClassUnderlying,'ColorFormat',ColorFormat,'Layout',Layout)` returns a `Simulink.ImageType` object and additionally initializes the optional properties `ClassUnderlying`, `ColorFormat`, and `Layout` to the specified values.

Properties

`Rows` — Number of rows in image data

`480` (default) | positive integer

Number of rows in the image data, specified as a positive integer.

Example: `480`

Data Types: `double`

`Columns` — Number of columns in image data

`640` (default) | positive integer

Number of columns in the image data, specified as a positive integer.

Example: `640`

Data Types: `double`

`Channels` — Number of color channels or samples for each pixel in array

`3` (default) | `1` | `4`

Number of color channels or samples for each pixel in the array, specified as `1`, `3`, or `4`. The number of channels must correspond to the number of color channels in the color format of the image data. These are the values of `Channels` for the supported color formats:

Color Format	Channels
`Grayscale`	1
`RGB`	3
`BGR`	3
`BGRA`	4

Example: `3`

Data Types: `double`

`ClassUnderlying` — Data type of underlying image data

`'uint8'` (default) | `'int8'` | `'uint16'` | `'int16'` | `'uint32'` | `'int32'` | `'single'` | `'double'` | `'boolean'`

Data type of the underlying image data, specified as one of these values:

- `'uint8'`
- `'int8'`
- `'uint16'`
- `'int16'`

- `'uint32'`
- `'int32'`
- `'single'`
- `'double'`
- `'boolean'`

Example: `'uint8'`

Data Types: `char` | `string`

`ColorFormat` — Color format of underlying image data
`'RGB'` (default) | `'Grayscale'` | `'BGR'` | `'BGRA'`

Color format of the underlying image data, specified as `'RGB'`, `'Grayscale'`, `'BGR'`, or `'BGRA'`. The color format determines what each color channel of a pixel in the image represents.

Example: `'RGB'`

Data Types: `char` | `string`

`Layout` — Memory arrangement of matrix data in image
`'ColumnMajor'` (default) | `'RowMajor'`

Memory arrangement of the matrix data in the image, specified as `'ColumnMajor'` or `'RowMajor'`.

Example: `'ColumnMajor'`

Data Types: `char` | `string`

Examples

Configure Data as `Simulink.ImageType` Data Type

1 Create a `Simulink.ImageType` object with default values.

```
imageType = Simulink.ImageType;
```

```
imageType =

  ImageType with properties:

              Rows: 480
              Cols: 640
          Channels: 3
    ClassUnderlying: 'uint8'
        ColorFormat: 'RGB'
             Layout: 'ColumnMajor'
```

2 Set the properties of the object to describe the image.

```
imageType.Rows = 720
imageType.Cols = 860
```

You can then reference this object from supported blocks that have a **Data type** field parameter.

Limitations

These blocks support simulation and code generation of a `Simulink.ImageType` object:

Block Library	Block Name
Sources	• Ground • From Multimedia File
Signal Routing	• Goto • From • Data Store Read • Data Store Write • Data Store Memory • Switch • Multiport Switch • Merge • Variant Source • Variant Merge (internal block added during code generation) • Mux • Demux • Vector Concatenate, Matrix Concatenate • Selector
Sinks	• Terminator • To Workspace (simulation only) • Record (simulation only) • Video Viewer (simulation only)
Ports & Subsystems	• Inport • Outport • Subsystem • Enabled Subsystem • Triggered Subsystem • Function-Call Subsystem • Resettable Subsystem • For Iterator Subsystem • Model
Discrete	• Unit Delay
Signal Attributes	• Signal Conversion • Signal Specification

Block Library	Block Name
User-Defined Functions	• Initialize Function • Reset Function • Terminate Function • Simulink Function block with side I/O and arguments

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Supports C++ code generation. To generate code, set the model configuration parameter **Language** (`TargetLang`) to `C++` and the **Language standard** to `C++11 (ISO)`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Image To Matrix | Matrix To Image | Image Attributes

Topics

“Convert Between Simulink Image Type and Matrices”
“Smile Detection by Using OpenCV Code in Simulink”
“Convert RGB Image to Grayscale Image by Using OpenCV Importer”
“Shadow Detection by Using OpenCV Code in Simulink”

quaternion

Create quaternion array

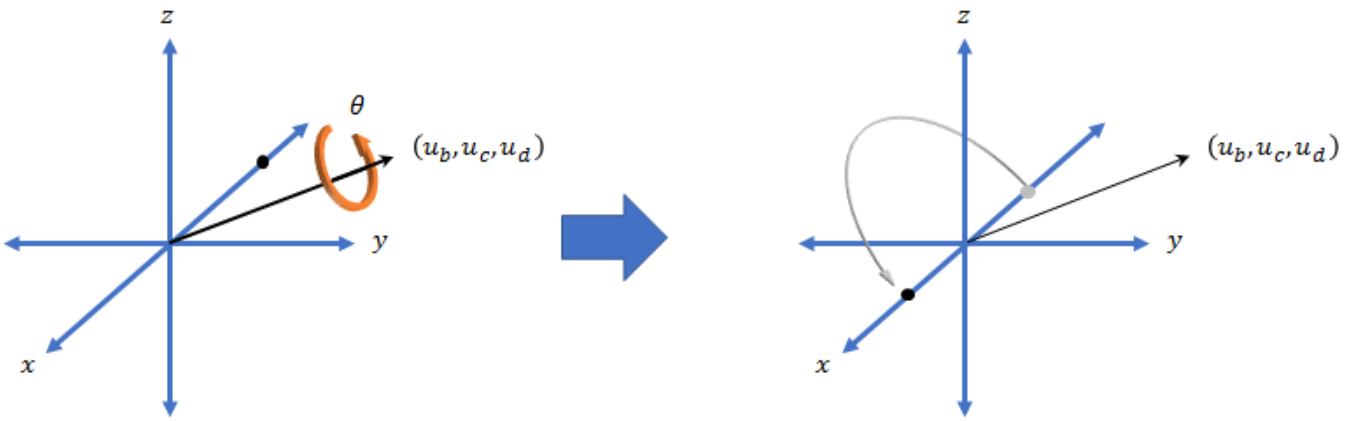
Description

A quaternion is a four-part hyper-complex number used in three-dimensional rotations and orientations.

A quaternion number is represented in the form $a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, where a, b, c, and d parts are real numbers, and i, j, and k are the basis elements, satisfying the equation: $\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1$.

The set of quaternions, denoted by **H**, is defined within a four-dimensional vector space over the real numbers, $\mathbf{R}^4$. Every element of **H** has a unique representation based on a linear combination of the basis elements, i, j, and k.

All rotations in 3-D can be described by an axis of rotation and angle about that axis. An advantage of quaternions over rotation matrices is that the axis and angle of rotation is easy to interpret. For example, consider a point in $\mathbf{R}^3$. To rotate the point, you define an axis of rotation and an angle of rotation.



The quaternion representation of the rotation can be expressed as $q = \cos\left(\theta/2\right) + \sin\left(\theta/2\right)(u_b\mathrm{i} + u_c\mathrm{j} + u_d\mathrm{k})$, where θ is the angle of rotation and [u_b, u_c, and u_d] is the axis of rotation.

Creation

Syntax

```
quat = quaternion()
quat = quaternion(A,B,C,D)
quat = quaternion(matrix)
quat = quaternion(RV,"rotvec")
```

```
quat = quaternion(RV,"rotvecd")
quat = quaternion(RM,"rotmat",PF)
quat = quaternion(E,"euler",RS,PF)
quat = quaternion(E,"eulerd",RS,PF)
```

Description

`quat = quaternion()` creates an empty quaternion.

`quat = quaternion(A,B,C,D)` creates a quaternion array where the four quaternion parts are taken from the arrays `A`, `B`, `C`, and `D`. All the inputs must have the same size and be of the same data type.

`quat = quaternion(matrix)` creates an *N*-by-1 quaternion array from an *N*-by-4 matrix, where each column becomes one part of the quaternion.

`quat = quaternion(RV,"rotvec")` creates an *N*-by-1 quaternion array from an *N*-by-3 matrix of rotation vectors, `RV`. Each row of `RV` represents a rotation vector in radians.

`quat = quaternion(RV,"rotvecd")` creates an *N*-by-1 quaternion array from an *N*-by-3 matrix of rotation vectors, `RV`. Each row of `RV` represents a rotation vector in degrees.

`quat = quaternion(RM,"rotmat",PF)` creates an *N*-by-1 quaternion array from the 3-by-3-by-*N* array of rotation matrices, `RM`. `PF` can be either `"point"` if the Euler angles represent point rotations or `"frame"` for frame rotations.

`quat = quaternion(E,"euler",RS,PF)` creates an *N*-by-1 quaternion array from the *N*-by-3 matrix, `E`. Each row of `E` represents a set of Euler angles in radians. The angles in `E` are rotations about the axes in sequence `RS`.

`quat = quaternion(E,"eulerd",RS,PF)` creates an *N*-by-1 quaternion array from the *N*-by-3 matrix, `E`. Each row of `E` represents a set of Euler angles in degrees. The angles in `E` are rotations about the axes in sequence `RS`.

Input Arguments

`A,B,C,D` — Quaternion parts

comma-separated arrays of the same size

Parts of a quaternion, specified as four comma-separated scalars, matrices, or multi-dimensional arrays of the same size.

Example: `quat = quaternion(1,2,3,4)` creates a quaternion of the form 1 + 2i + 3j + 4k.

Example: `quat = quaternion([1,5],[2,6],[3,7],[4,8])` creates a 1-by-2 quaternion array where `quat(1,1) = 1 + 2i + 3j + 4k` and `quat(1,2) = 5 + 6i + 7j + 8k`

Data Types: `single` | `double`

`matrix` — Matrix of quaternion parts

N-by-4 matrix

Matrix of quaternion parts, specified as an *N*-by-4 matrix. Each row represents a separate quaternion. Each column represents a separate quaternion part.

Example: `quat = quaternion(rand(10,4))` creates a 10-by-1 quaternion array.

Data Types: `single` | `double`

`RV` — Matrix of rotation vectors
N-by-3 matrix

Matrix of rotation vectors, specified as an *N*-by-3 matrix. Each row of `RV` represents the [X Y Z] elements of a rotation vector. A rotation vector is a unit vector representing the axis of rotation scaled by the angle of rotation in radians or degrees.

To use this syntax, specify the first argument as a matrix of rotation vectors and the second argument as the `"rotvec"` or `"rotvecd"`.

Example: `quat = quaternion(rand(10,3),"rotvec")` creates a 10-by-1 quaternion array.

Data Types: `single` | `double`

`RM` — Rotation matrices
3-by-3 matrix | 3-by-3-by-*N* array

Array of rotation matrices, specified by a 3-by-3 matrix or 3-by-3-by-*N* array. Each page of the array represents a separate rotation matrix.

Example: `quat = quaternion(rand(3),"rotmat","point")`

Example: `quat = quaternion(rand(3),"rotmat","frame")`

Data Types: `single` | `double`

`PF` — Type of rotation matrix
`"point"` | `"frame"`

Type of rotation matrix, specified by `"point"` or `"frame"`.

Example: `quat = quaternion(rand(3),"rotmat","point")`

Example: `quat = quaternion(rand(3),"rotmat","frame")`

Data Types: `char` | `string`

`E` — Matrix of Euler angles
N-by-3 matrix

Matrix of Euler angles, specified by an *N*-by-3 matrix. If using the `"euler"` syntax, specify `E` in radians. If using the `"eulerd"` syntax, specify `E` in degrees.

Example: `quat = quaternion(E,"euler","YZY","point")`

Example: `quat = quaternion(E,"euler","XYZ","frame")`

Data Types: `single` | `double`

`RS` — Rotation sequence
character vector | scalar string

Rotation sequence, specified as a three-element character vector:

- `"YZY"`
- `"YXY"`
- `"ZYZ"`

- "ZXZ"
- "XYX"
- "XZX"
- "XYZ"
- "YZX"
- "ZXY"
- "XZY"
- "ZYX"
- "YXZ"

Assume you want to determine the new coordinates of a point when its coordinate system is rotated using frame rotation. The point is defined in the original coordinate system as:

```
point = [sqrt(2)/2,sqrt(2)/2,0];
```

In this representation, the first column represents the *x*-axis, the second column represents the *y*-axis, and the third column represents the *z*-axis.

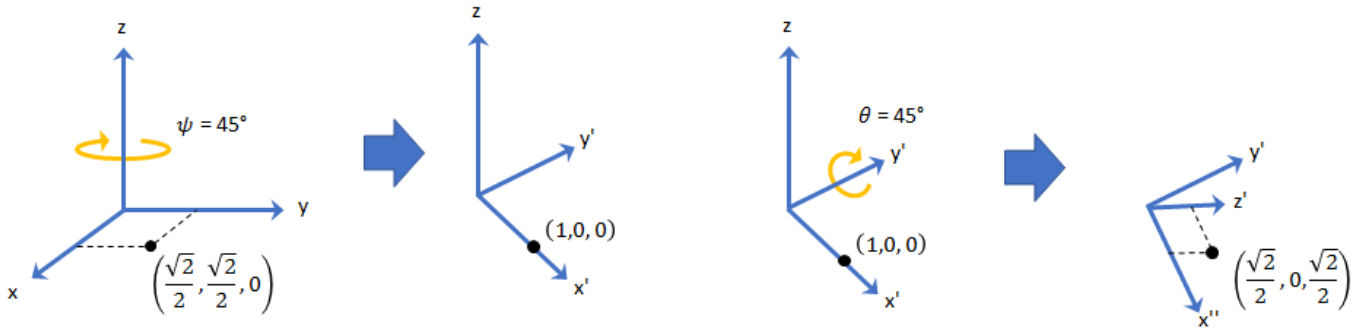
You want to rotate the point using the Euler angle representation [45,45,0]. Rotate the point using two different rotation sequences:

- If you create a quaternion rotator and specify the "ZYX" sequence, the frame is first rotated 45° around the *z*-axis, then 45° around the new *y*-axis.

  ```
  quatRotator = quaternion([45,45,0],"eulerd","ZYX","frame");
  newPointCoordinate = rotateframe(quatRotator,point)
  ```

  ```
  newPointCoordinate =

      0.7071   -0.0000    0.7071
  ```

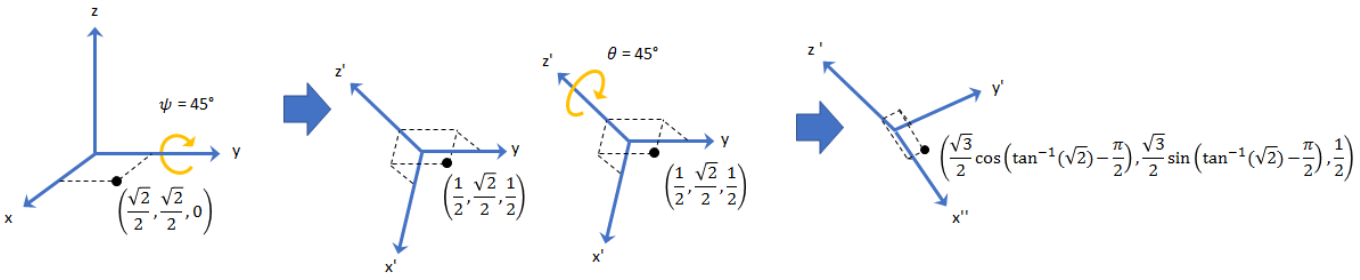


- If you create a quaternion rotator and specify the "YZX" sequence, the frame is first rotated 45° around the *y*-axis, then 45° around the new *z*-axis.

  ```
  quatRotator = quaternion([45,45,0],"eulerd","YZX","frame");
  newPointCoordinate = rotateframe(quatRotator,point)
  ```

  ```
  newPointCoordinate =

      0.8536    0.1464    0.5000
  ```



Data Types: `char` | `string`

Object Functions

Construction and Conversion

compact	Convert quaternion array to N-by-4 matrix
eulerd	Convert quaternion to Euler angles (degrees)
euler	Convert quaternion to Euler angles (radians)
rotmat	Convert quaternion to rotation matrix
rotvec	Convert quaternion to rotation vector (radians)
rotvecd	Convert quaternion to rotation vector (degrees)
ones	Create quaternion array with real parts set to one and imaginary parts set to zero
parts	Extract quaternion parts
randrot	Uniformly distributed random rotations
zeros	Create quaternion array with all parts set to zero

Rotation

rotateframe	Quaternion frame rotation
rotatepoint	Quaternion point rotation

Normalization

norm	Quaternion norm
normalize	Quaternion normalization

Computations on Quaternions

angvel	Angular velocity from quaternion array
dist	Angular distance in radians
exp	Exponential of quaternion array
interp1	Quaternion interpolation (table lookup)
log	Natural logarithm of quaternion array
meanrot	Quaternion mean rotation
slerp	Spherical linear interpolation

Arithmetic Operations

conj	Complex conjugate of quaternion
ctranspose, '	Complex conjugate transpose of quaternion array
ldivide, .\	Element-wise quaternion left division
minus, -	Quaternion subtraction

mldivide, \	Quaternion left division for scalars
mrdivide, /	Quaternion right division for scalars
mtimes, *	Quaternion multiplication
power	Element-wise quaternion power
prod	Product of quaternion array
rdivide, ./	Element-wise quaternion right division
times	Element-wise quaternion multiplication
transpose, .'	Transpose quaternion array
uminus, -	Quaternion unary minus

Examples

Create Empty Quaternion

```
quat = quaternion()
```

```
quat =

  0×0 empty quaternion array
```

By default, the underlying class of the quaternion is a double.

```
classUnderlying(quat)
```

```
ans =
'double'
```

Create Quaternion by Specifying Individual Quaternion Parts

You can create a quaternion array by specifying the four parts as comma-separated scalars, matrices, or multidimensional arrays of the same size.

Define quaternion parts as scalars.

```
A = 1.1;
B = 2.1;
C = 3.1;
D = 4.1;
quatScalar = quaternion(A,B,C,D)
```

```
quatScalar = quaternion
     1.1 + 2.1i + 3.1j + 4.1k
```

Define quaternion parts as column vectors.

```
A = [1.1;1.2];
B = [2.1;2.2];
C = [3.1;3.2];
D = [4.1;4.2];
quatVector = quaternion(A,B,C,D)
```

```
quatVector = 2×1 quaternion array
     1.1 + 2.1i + 3.1j + 4.1k
```

```
     1.2 + 2.2i + 3.2j + 4.2k
```

Define quaternion parts as matrices.

```
A = [1.1,1.3; ...
     1.2,1.4];
B = [2.1,2.3; ...
     2.2,2.4];
C = [3.1,3.3; ...
     3.2,3.4];
D = [4.1,4.3; ...
     4.2,4.4];
quatMatrix = quaternion(A,B,C,D)
```

```
quatMatrix = 2×2 quaternion array
     1.1 + 2.1i + 3.1j + 4.1k     1.3 + 2.3i + 3.3j + 4.3k
     1.2 + 2.2i + 3.2j + 4.2k     1.4 + 2.4i + 3.4j + 4.4k
```

Define quaternion parts as three dimensional arrays.

```
A = randn(2,2,2);
B = zeros(2,2,2);
C = zeros(2,2,2);
D = zeros(2,2,2);
quatMultiDimArray = quaternion(A,B,C,D)
```

```
quatMultiDimArray = 2×2×2 quaternion array
quatMultiDimArray(:,:,1) =

     0.53767 +       0i +       0j +       0k     -2.2588 +       0i +       0j +       0k
      1.8339 +       0i +       0j +       0k     0.86217 +       0i +       0j +       0k


quatMultiDimArray(:,:,2) =

     0.31877 +       0i +       0j +       0k    -0.43359 +       0i +       0j +       0k
     -1.3077 +       0i +       0j +       0k     0.34262 +       0i +       0j +       0k
```

Create Quaternion by Specifying Quaternion Parts Matrix

You can create a scalar or column vector of quaternions by specify an *N*-by-4 matrix of quaternion parts, where columns correspond to the quaternion parts A, B, C, and D.

Create a column vector of random quaternions.

```
quatParts = rand(3,4)
```

```
quatParts = 3×4

    0.8147    0.9134    0.2785    0.9649
    0.9058    0.6324    0.5469    0.1576
    0.1270    0.0975    0.9575    0.9706
```

```
quat = quaternion(quatParts)
```

```
quat = 3×1 quaternion array
     0.81472 + 0.91338i +  0.2785j + 0.96489k
     0.90579 + 0.63236i + 0.54688j + 0.15761k
     0.12699 + 0.09754i + 0.95751j + 0.97059k
```

To retrieve the `quatParts` matrix from quaternion representation, use `compact`.

```
retrievedquatParts = compact(quat)

retrievedquatParts = 3×4

    0.8147    0.9134    0.2785    0.9649
    0.9058    0.6324    0.5469    0.1576
    0.1270    0.0975    0.9575    0.9706
```

Create Quaternion by Specifying Rotation Vectors

You can create an *N*-by-1 quaternion array by specifying an *N*-by-3 matrix of rotation vectors in radians or degrees. Rotation vectors are compact spatial representations that have a one-to-one relationship with normalized quaternions.

Rotation Vectors in Radians

Create a scalar quaternion using a rotation vector and verify the resulting quaternion is normalized.

```
rotationVector = [0.3491,0.6283,0.3491];
quat = quaternion(rotationVector,"rotvec")

quat = quaternion
     0.92124 + 0.16994i + 0.30586j + 0.16994k


norm(quat)

ans =
1.0000
```

You can convert from quaternions to rotation vectors in radians using the `rotvec` function. Recover the `rotationVector` from the quaternion, `quat`.

```
rotvec(quat)

ans = 1×3

    0.3491    0.6283    0.3491
```

Rotation Vectors in Degrees

Create a scalar quaternion using a rotation vector and verify the resulting quaternion is normalized.

```
rotationVector = [20,36,20];
quat = quaternion(rotationVector,"rotvecd")
```

```
quat = quaternion
     0.92125 + 0.16993i + 0.30587j + 0.16993k
```

```
norm(quat)

ans =
1
```

You can convert from quaternions to rotation vectors in degrees using the `rotvecd` function. Recover the `rotationVector` from the quaternion, `quat`.

```
rotvecd(quat)

ans = 1×3

   20.0000   36.0000   20.0000
```

Create Quaternion by Specifying Rotation Matrices

You can create an N-by-1 quaternion array by specifying a 3-by-3-by-N array of rotation matrices. Each page of the rotation matrix array corresponds to one element of the quaternion array.

Create a scalar quaternion using a 3-by-3 rotation matrix. Specify whether the rotation matrix should be interpreted as a frame or point rotation.

```
rotationMatrix = [1 0         0; ...
                  0 sqrt(3)/2 0.5; ...
                  0 -0.5      sqrt(3)/2];
quat = quaternion(rotationMatrix,"rotmat","frame")

quat = quaternion
     0.96593 + 0.25882i +       0j +       0k
```

You can convert from quaternions to rotation matrices using the `rotmat` function. Recover the `rotationMatrix` from the quaternion, `quat`.

```
rotmat(quat,"frame")

ans = 3×3

    1.0000         0         0
         0    0.8660    0.5000
         0   -0.5000    0.8660
```

Create Quaternion by Specifying Euler Angles

You can create an *N*-by-1 quaternion array by specifying an *N*-by-3 array of Euler angles in radians or degrees.

Euler Angles in Radians

Use the `euler` syntax to create a scalar quaternion using a 1-by-3 vector of Euler angles in radians. Specify the rotation sequence of the Euler angles and whether the angles represent a frame or point rotation.

```
E = [pi/2,0,pi/4];
quat = quaternion(E,"euler","ZYX","frame")
```

```
quat = quaternion
     0.65328 +  0.2706i +  0.2706j + 0.65328k
```

You can convert from quaternions to Euler angles using the `euler` function. Recover the Euler angles, `E`, from the quaternion, `quat`.

```
euler(quat,"ZYX","frame")
```

```
ans = 1×3

    1.5708         0    0.7854
```

Euler Angles in Degrees

Use the `eulerd` syntax to create a scalar quaternion using a 1-by-3 vector of Euler angles in degrees. Specify the rotation sequence of the Euler angles and whether the angles represent a frame or point rotation.

```
E = [90,0,45];
quat = quaternion(E,"eulerd","ZYX","frame")
```

```
quat = quaternion
     0.65328 +  0.2706i +  0.2706j + 0.65328k
```

You can convert from quaternions to Euler angles in degrees using the `eulerd` function. Recover the Euler angles, `E`, from the quaternion, `quat`.

```
eulerd(quat,"ZYX","frame")
```

```
ans = 1×3

   90.0000         0   45.0000
```

Quaternion Algebra

Quaternions form a noncommutative associative algebra over the real numbers. This example illustrates the rules of quaternion algebra.

Addition and Subtraction

Quaternion addition and subtraction occur part-by-part, and are commutative:

```
Q1 = quaternion(1,2,3,4)
```

```
Q1 = quaternion
     1 + 2i + 3j + 4k
```

```
Q2 = quaternion(9,8,7,6)

Q2 = quaternion
     9 + 8i + 7j + 6k
```

```
Q1plusQ2 = Q1 + Q2

Q1plusQ2 = quaternion
     10 + 10i + 10j + 10k
```

```
Q2plusQ1 = Q2 + Q1

Q2plusQ1 = quaternion
     10 + 10i + 10j + 10k
```

```
Q1minusQ2 = Q1 - Q2

Q1minusQ2 = quaternion
    -8 - 6i - 4j - 2k
```

```
Q2minusQ1 = Q2 - Q1

Q2minusQ1 = quaternion
     8 + 6i + 4j + 2k
```

You can also perform addition and subtraction of real numbers and quaternions. The first part of a quaternion is referred to as the *real* part, while the second, third, and fourth parts are referred to as the *vector*. Addition and subtraction with real numbers affect only the real part of the quaternion.

```
Q1plusRealNumber = Q1 + 5

Q1plusRealNumber = quaternion
     6 + 2i + 3j + 4k
```

```
Q1minusRealNumber = Q1 - 5

Q1minusRealNumber = quaternion
    -4 + 2i + 3j + 4k
```

Multiplication

Quaternion multiplication is determined by the products of the basis elements and the distributive law. Recall that multiplication of the basis elements, i, j, and k, are not commutative, and therefore quaternion multiplication is not commutative.

```
Q1timesQ2 = Q1 * Q2
```

```
Q1timesQ2 = quaternion
    -52 + 16i + 54j + 32k
```

```
Q2timesQ1 = Q2 * Q1
```

```
Q2timesQ1 = quaternion
    -52 + 36i + 14j + 52k
```

```
isequal(Q1timesQ2,Q2timesQ1)
```

```
ans = logical
   0
```

You can also multiply a quaternion by a real number. If you multiply a quaternion by a real number, each part of the quaternion is multiplied by the real number individually:

```
Q1times5 = Q1*5
```

```
Q1times5 = quaternion
      5 + 10i + 15j + 20k
```

Multiplying a quaternion by a real number is commutative.

```
isequal(Q1*5,5*Q1)
```

```
ans = logical
   1
```

Conjugation

The complex conjugate of a quaternion is defined such that each element of the vector portion of the quaternion is negated.

```
Q1
```

```
Q1 = quaternion
     1 + 2i + 3j + 4k
```

```
conj(Q1)
```

```
ans = quaternion
     1 - 2i - 3j - 4k
```

Multiplication between a quaternion and its conjugate is commutative:

```
isequal(Q1*conj(Q1),conj(Q1)*Q1)
```

```
ans = logical
   1
```

Quaternion Array Manipulation

You can organize quaternions into vectors, matrices, and multidimensional arrays. Built-in MATLAB® functions have been enhanced to work with quaternions.

Concatenate

Quaternions are treated as individual objects during concatenation and follow MATLAB rules for array manipulation.

```
Q1 = quaternion(1,2,3,4);
Q2 = quaternion(9,8,7,6);

qVector = [Q1,Q2]
```

```
qVector = 1×2 quaternion array
     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
```

```
Q3 = quaternion(-1,-2,-3,-4);
Q4 = quaternion(-9,-8,-7,-6);

qMatrix = [qVector;Q3,Q4]
```

```
qMatrix = 2×2 quaternion array
     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
    -1 - 2i - 3j - 4k    -9 - 8i - 7j - 6k
```

```
qMultiDimensionalArray(:,:,1) = qMatrix;
qMultiDimensionalArray(:,:,2) = qMatrix
```

```
qMultiDimensionalArray = 2×2×2 quaternion array
qMultiDimensionalArray(:,:,1) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
    -1 - 2i - 3j - 4k    -9 - 8i - 7j - 6k


qMultiDimensionalArray(:,:,2) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
    -1 - 2i - 3j - 4k    -9 - 8i - 7j - 6k
```

Indexing

To access or assign elements in a quaternion array, use indexing.

```
qLoc2 = qMultiDimensionalArray(2)
```

```
qLoc2 = quaternion
    -1 - 2i - 3j - 4k
```

Replace the quaternion at index two with a quaternion one.

```
qMultiDimensionalArray(2) = ones("quaternion")
```

```
qMultiDimensionalArray = 2×2×2 quaternion array
qMultiDimensionalArray(:,:,1) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
     1 + 0i + 0j + 0k    -9 - 8i - 7j - 6k


qMultiDimensionalArray(:,:,2) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
    -1 - 2i - 3j - 4k    -9 - 8i - 7j - 6k
```

Reshape

To reshape quaternion arrays, use the `reshape` function.

```
qMatReshaped = reshape(qMatrix,4,1)
```

```
qMatReshaped = 4×1 quaternion array
     1 + 2i + 3j + 4k
    -1 - 2i - 3j - 4k
     9 + 8i + 7j + 6k
    -9 - 8i - 7j - 6k
```

Transpose

To transpose quaternion vectors and matrices, use the `transpose` function.

```
qMatTransposed = transpose(qMatrix)
```

```
qMatTransposed = 2×2 quaternion array
     1 + 2i + 3j + 4k    -1 - 2i - 3j - 4k
     9 + 8i + 7j + 6k    -9 - 8i - 7j - 6k
```

Permute

To permute quaternion vectors, matrices, and multidimensional arrays, use the `permute` function.

```
qMultiDimensionalArray
```

```
qMultiDimensionalArray = 2×2×2 quaternion array
qMultiDimensionalArray(:,:,1) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
     1 + 0i + 0j + 0k    -9 - 8i - 7j - 6k


qMultiDimensionalArray(:,:,2) =

     1 + 2i + 3j + 4k     9 + 8i + 7j + 6k
    -1 - 2i - 3j - 4k    -9 - 8i - 7j - 6k
```

```
qMatPermute = permute(qMultiDimensionalArray,[3,1,2])
```

```
qMatPermute = 2×2×2 quaternion array
qMatPermute(:,:,1) =
```

```
     1 + 2i + 3j + 4k     1 + 0i + 0j + 0k
     1 + 2i + 3j + 4k    -1 - 2i - 3j - 4k


qMatPermute(:,:,2) =

     9 + 8i + 7j + 6k    -9 - 8i - 7j - 6k
     9 + 8i + 7j + 6k    -9 - 8i - 7j - 6k
```

Version History

Introduced in R2023b

R2023b: Moved to Computer Vision Toolbox from Automated Driving Toolbox

The `quaternion` object and its object functions no longer require Automated Driving Toolbox.

R2025a: `interp1` object function for quaternion interpolation

Use the `interp1` object function to interpolate quaternions using interpolation methods such as SQUAD and SLERP.

R2025a: `slerp` object function supports natural interpolation

The `slerp` function now supports spherically interpolation using the `"natural"` path option, enabling smoother transitions that respect the orientation of the start and end quaternions.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

classUnderlying

Class of parts within quaternion

Syntax

```
underlyingClass = classUnderlying(quat)
```

Description

`underlyingClass = classUnderlying(quat)` returns the name of the class of the parts of the quaternion `quat`.

Examples

Get Underlying Class of Quaternion

A quaternion is a four-part hyper-complex number used in three-dimensional representations. The four parts of the quaternion are of data type `single` or `double`.

Create two quaternions, one with an underlying data type of `single`, and one with an underlying data type of `double`. Verify the underlying data types by calling `classUnderlying` on the quaternions.

```
qSingle = quaternion(single([1,2,3,4]))
```

```
qSingle = quaternion
     1 + 2i + 3j + 4k
```

```
classUnderlying(qSingle)
```

```
ans =
'single'
```

```
qDouble = quaternion([1,2,3,4])
```

```
qDouble = quaternion
     1 + 2i + 3j + 4k
```

```
classUnderlying(qDouble)
```

```
ans =
'double'
```

You can separate quaternions into their parts using the `parts` function. Verify the parts of each quaternion are the correct data type. Recall that `double` is the default MATLAB® type.

```
[aS,bS,cS,dS] = parts(qSingle)
```

```
aS = single

1

bS = single

2

cS = single

3

dS = single

4
```

```
[aD,bD,cD,dD] = parts(qDouble)

aD =
1

bD =
2

cD =
3

dD =
4
```

Quaternions follow the same implicit casting rules as other data types in MATLAB. That is, a quaternion with underlying data type `single` that is combined with a quaternion with underlying data type `double` results in a quaternion with underlying data type `single`. Multiply `qDouble` and `qSingle` and verify the resulting underlying data type is `single`.

```
q = qDouble*qSingle;
classUnderlying(q)

ans =
'single'
```

Input Arguments

`quat` — Quaternion to investigate

`quaternion` object | array of `quaternion` objects

Quaternion to investigate, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`underlyingClass` — Underlying class of quaternion object

`'single'` | `'double'`

Underlying class of quaternion, returned as the character vector `'single'` or `'double'`.

Data Types: `char`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`compact` | `parts`

Objects

`quaternion`

angvel

Angular velocity from quaternion array

Syntax

```
AV = angvel(quat,dt,"frame")
AV = angvel(quat,dt,"point")
[AV,qf] = angvel(quat,dt,fp,qi)
```

Description

`AV = angvel(quat,dt,"frame")` returns the angular velocity array from an array of quaternions, `quat`. The quaternions in `quat` correspond to frame rotation. The initial quaternion is assumed to represent zero rotation.

`AV = angvel(quat,dt,"point")` returns the angular velocity array from an array of quaternions, `quat`. The quaternions in `quat` correspond to point rotation. The initial quaternion is assumed to represent zero rotation.

`[AV,qf] = angvel(quat,dt,fp,qi)` allows you to specify the initial quaternion, `qi`, and the type of rotation, `fp`. It also returns the final quaternion, `qf`.

Examples

Generate Angular Velocity From Quaternion Array

Create an array of quaternions.

```
eulerAngles = [(0:10:90).',zeros(numel(0:10:90),2)];
q = quaternion(eulerAngles,'eulerd','ZYX','frame');
```

Specify the time step and generate the angular velocity array.

```
dt = 1;
av = angvel(q,dt,'frame') % units in rad/s
```

```
av = 10×3

         0         0         0
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
         0         0    0.1743
```

Input Arguments

`quat` — Quaternions

N-by-1 vector of `quaternion` objects

Quaternions, specified as an *N*-by-1 vector of `quaternion` objects.

`dt` — Time step

nonnegative scalar

Time step, specified as a nonnegative scalar.

Data Types: `single` | `double`

`fp` — Type of rotation

`"frame"` | `"point"`

Type of rotation, specified as `"frame"` or `"point"`.

`qi` — Initial quaternion

`quaternion` object

Initial quaternion, specified as a `quaternion` object.

Output Arguments

`AV` — Angular velocity

N-by-3 real matrix

Angular velocity, returned as an *N*-by-3 real matrix, where *N* is the number of quaternions in the `quat` argument. Each row of the matrix corresponds to an angular velocity vector.

`qf` — Final quaternion

`quaternion` object

Final quaternion, returned as a `quaternion` object. `qf` is the same as the last quaternion in the `quat` argument.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`quaternion`

compact

Convert quaternion array to *N*-by-4 matrix

Syntax

```
matrix = compact(quat)
```

Description

`matrix = compact(quat)` converts the quaternion array, `quat`, to an *N*-by-4 matrix. The columns are made from the four quaternion parts. The i^{th} row of the matrix corresponds to `quat(i)`.

Examples

Convert Quaternion Array to Compact Representation of Parts

Create a scalar quaternion with random parts. Convert the parts to a 1-by-4 vector using `compact`.

```
randomParts = randn(1,4)
```

```
randomParts = 1×4

    0.5377    1.8339   -2.2588    0.8622
```

```
quat = quaternion(randomParts)
```

```
quat = quaternion
     0.53767 +  1.8339i -  2.2588j + 0.86217k
```

```
quatParts = compact(quat)
```

```
quatParts = 1×4

    0.5377    1.8339   -2.2588    0.8622
```

Create a 2-by-2 array of quaternions, then convert the representation to a matrix of quaternion parts. The output rows correspond to the linear indices of the quaternion array.

```
quatArray = [quaternion([1:4;5:8]),quaternion([9:12;13:16])]
```

```
quatArray = 2×2 quaternion array
      1 +  2i +  3j +  4k      9 + 10i + 11j + 12k
      5 +  6i +  7j +  8k     13 + 14i + 15j + 16k
```

```
quatArrayParts = compact(quatArray)
```

```
quatArrayParts = 4×4
```

```
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

Input Arguments

`quat` — Quaternion to convert

`quaternion` object | array of `quaternion` objects

Quaternion to convert, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`matrix` — Quaternion in matrix form

N-by-4 numeric matrix

Quaternion in matrix form, returned as an *N*-by-4 numeric matrix, where *N* is the number of quaternions given in the input `quat`.

Data Types: `single` | `double`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`parts` | `classUnderlying`

Objects

`quaternion`

conj

Complex conjugate of quaternion

Syntax

```
quatConjugate = conj(quat)
```

Description

`quatConjugate = conj(quat)` returns the complex conjugate of the quaternion, `quat`.

If $q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, the complex conjugate of q is $q^* = a - b\mathrm{i} - c\mathrm{j} - d\mathrm{k}$. Considered as a rotation operator, the conjugate performs the opposite rotation. For example,

```
q = quaternion(deg2rad([16 45 30]),"rotvec");
a = q*conj(q);
rotatepoint(a,[0,1,0])
```

```
ans =

     0     1     0
```

Examples

Complex Conjugate of Quaternion

Create a quaternion scalar and get the complex conjugate.

```
q = normalize(quaternion([0.9 0.3 0.3 0.25]))
```

```
q = quaternion
     0.87727 + 0.29242i + 0.29242j + 0.24369k
```

```
qConj = conj(q)
```

```
qConj = quaternion
     0.87727 - 0.29242i - 0.29242j - 0.24369k
```

Verify that a quaternion multiplied by its conjugate returns a quaternion one.

```
q*qConj
```

```
ans = quaternion
     1 + 0i + 0j + 0k
```

Input Arguments

`quat` — Quaternion

`quaternion` object | array of `quaternion` objects

Quaternion to conjugate, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`quatConjugate` — Quaternion conjugate

`quaternion` object | array of `quaternion` objects

Quaternion conjugate, returned as a `quaternion` object or an array of `quaternion` objects of the same size as `quat`.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`norm` | `.*,times`

Objects

`quaternion`

ctranspose, '

Complex conjugate transpose of quaternion array

Syntax

```
quatTransposed = quat'
```

Description

`quatTransposed = quat'` returns the complex conjugate transpose of the quaternion, `quat`.

Examples

Vector Complex Conjugate Transpose

Create a vector of quaternions and compute its complex conjugate transpose.

```
quat = quaternion(randn(4,4))
```

```
quat = 4×1 quaternion array
      0.53767 +  0.31877i +   3.5784j +   0.7254k
       1.8339 -   1.3077i +   2.7694j - 0.063055k
      -2.2588 -  0.43359i -   1.3499j +  0.71474k
      0.86217 +  0.34262i +   3.0349j -  0.20497k
```

```
quatTransposed = quat'
```

```
quatTransposed = 1×4 quaternion array
      0.53767 -  0.31877i -   3.5784j -   0.7254k       1.8339 +   1.3077i -   2.7694j + 0.06305
```

Matrix Complex Conjugate Transpose

Create a matrix of quaternions and compute its complex conjugate transpose.

```
quat = [quaternion(randn(2,4)),quaternion(randn(2,4))]
```

```
quat = 2×2 quaternion array
      0.53767 -   2.2588i +  0.31877j -  0.43359k       3.5784 -   1.3499i +   0.7254j +  0.7147
       1.8339 +  0.86217i -   1.3077j +  0.34262k       2.7694 +   3.0349i - 0.063055j -  0.2049
```

```
quatTransposed = quat'
```

```
quatTransposed = 2×2 quaternion array
      0.53767 +   2.2588i -  0.31877j +  0.43359k       1.8339 -  0.86217i +   1.3077j -  0.3426
       3.5784 +   1.3499i -   0.7254j -  0.71474k       2.7694 -   3.0349i + 0.063055j +  0.2049
```

Input Arguments

`quat` — Quaternion to transpose

`quaternion` object | vector of `quaternion` objects | matrix of `quaternion` objects

Quaternion to transpose, specified as a `quaternion` object or a vector or matrix of `quaternion` objects. The complex conjugate transpose is defined for dimensionality up to 2.

Output Arguments

`quatTransposed` — Conjugate transposed quaternion

`quaternion` object | vector of `quaternion` objects | matrix of `quaternion` objects

Conjugate transposed quaternion, returned as a `quaternion` object or a vector or matrix of `quaternion` objects. The size of `quatTransposed` is *N*-by-*M*, where the size of `quat` is *M*-by-*N*.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`transpose, .'`

Objects

`quaternion`

dist

Angular distance in radians

Syntax

```
distance = dist(quatA,quatB)
```

Description

`distance = dist(quatA,quatB)` returns the angular distance in radians between two quaternions, `quatA` and `quatB`.

Examples

Calculate Quaternion Distance

Calculate the quaternion distance between a single quaternion and each element of a vector of quaternions. Define the quaternions using Euler angles.

```
q = quaternion([0,0,0],"eulerd","ZYX","frame")
```

```
q = quaternion
     1 + 0i + 0j + 0k
```

```
qArray = quaternion([0,45,0;0,90,0;0,180,0;0,-90,0;0,-45,0],"eulerd","ZYX","frame")
```

```
qArray = 5×1 quaternion array
       0.92388 +         0i +   0.38268j +         0k
       0.70711 +         0i +   0.70711j +         0k
    6.1232e-17 +         0i +         1j +         0k
       0.70711 +         0i -   0.70711j +         0k
       0.92388 +         0i -   0.38268j +         0k
```

```
quaternionDistance = rad2deg(dist(q,qArray))
```

```
quaternionDistance = 5×1

   45.0000
   90.0000
  180.0000
   90.0000
   45.0000
```

If both arguments to `dist` are vectors, the quaternion distance is calculated between corresponding elements. Calculate the quaternion distance between two quaternion vectors.

```
angles1 = [30,0,15; ...
           30,5,15; ...
```

```
           30,10,15; ...
           30,15,15];
angles2 = [30,6,15; ...
           31,11,15; ...
           30,16,14; ...
           30.5,21,15.5];

qVector1 = quaternion(angles1,"eulerd","ZYX","frame");
qVector2 = quaternion(angles2,"eulerd","ZYX","frame");

rad2deg(dist(qVector1,qVector2))

ans = 4×1

    6.0000
    6.0827
    6.0827
    6.0287
```

Note that a quaternion represents the same rotation as its negative. Calculate a quaternion and its negative.

```
qPositive = quaternion([30,45,-60],"eulerd","ZYX","frame")

qPositive = quaternion
     0.72332 - 0.53198i + 0.20056j +  0.3919k


qNegative = -qPositive

qNegative = quaternion
    -0.72332 + 0.53198i - 0.20056j -  0.3919k
```

Find the distance between the quaternion and its negative.

```
dist(qPositive,qNegative)

ans =
0
```

The components of a quaternion may look different from the components of its negative, but both expressions represent the same rotation.

Input Arguments

`quatA,quatB` — Quaternions to calculate distance between

`quaternion` objects | arrays of `quaternion` objects

Quaternions to calculate distance between, specified as comma-separated `quaternion` objects or arrays of `quaternion` objects of any dimensionality. `quatA` and `quatB` must have compatible sizes:

- `size(quatA) == size(quatB)`, or
- `numel(quatA) == 1`, or

- `numel(quatB) == 1`, or
- if `[Adim1,…,AdimN] = size(quatA)` and `[Bdim1,…,BdimN] = size(quatB)`, then for `i = 1:N`, either `Adimi==Bdimi` or `Adim==1` or `Bdim==1`.

  If one of the quaternion arguments contains only one quaternion, then this function returns the distances between that quaternion and every quaternion in the other argument.

Output Arguments

`distance` — Angular distance (radians)

numeric scalar | numeric array

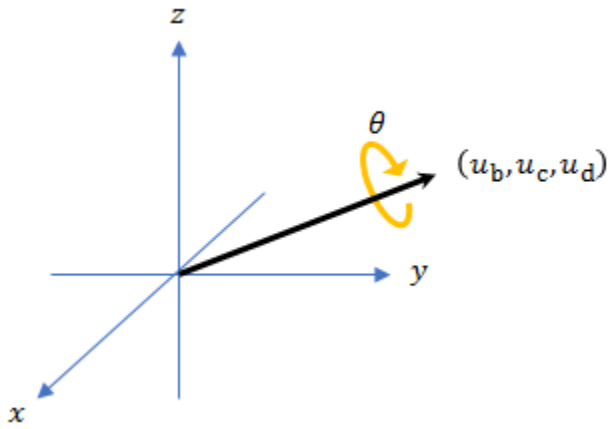
Angular distance in radians, returned as a numeric scalar or numeric array. The dimensions are the maximum of the union of `size(quatA)` and `size(quatB)`.

Data Types: `single` | `double`

Algorithms

The `dist` function returns the angular distance between two quaternions.

A quaternion may be defined by an axis (u_b,u_c,u_d) and angle of rotation θ_q:
$q = \cos\left(\theta_q/2\right) + \sin\left(\theta_q/2\right)(u_b\mathrm{i} + u_c\mathrm{j} + u_d\mathrm{k})$.



Given a quaternion in the form, $q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, where a is the real part, you can solve for the angle of q as $\theta_q = 2\cos^{-1}(a)$.

Consider two quaternions, p and q, and the product $z = p*\text{conjugate}(q)$. As p approaches q, the angle of z goes to 0, and z approaches the unit quaternion.

The angular distance between two quaternions can be expressed as $\theta_z = 2\cos^{-1}(\text{real}(z))$.

Using the `quaternion` data type syntax, the angular distance is calculated as:

```
angularDistance = 2*acos(abs(parts(p*conj(q))));
```

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`parts` | `conj`

Objects

`quaternion`

euler

Convert quaternion to Euler angles (radians)

Syntax

```
eulerAngles = euler(quat,rotationSequence,rotationType)
```

Description

`eulerAngles = euler(quat,rotationSequence,rotationType)` converts the quaternion, `quat`, to an *N*-by-3 matrix of Euler angles.

Examples

Convert Quaternion to Euler Angles in Radians

Convert a quaternion frame rotation to Euler angles in radians using the "ZYX" rotation sequence.

```
quat = quaternion([0.7071 0.7071 0 0]);
eulerAnglesRandians = euler(quat,"ZYX","frame")
```

```
eulerAnglesRandians = 1×3

         0         0    1.5708
```

Input Arguments

quat — Quaternion to convert to Euler angles

`quaternion` object | array of `quaternion` objects

Quaternion to convert to Euler angles, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

rotationSequence — Rotation sequence

`"ZYX"` | `"ZYZ"` | `"ZXY"` | `"ZXZ"` | `"YXZ"` | `"YXY"` | `"YZX"` | `"YZY"` | `"XYZ"` | `"XYX"` | `"XZY"` | `"XZX"`

Rotation sequence of Euler angle representation, specified as one of these values.

- `"YZY"`
- `"YXY"`
- `"ZYZ"`
- `"ZXZ"`
- `"XYX"`
- `"XZX"`
- `"XYZ"`

- "YZX"
- "ZXY"
- "XZY"
- "ZYX"
- "YXZ"

The rotation sequence defines the order of rotations about the axes. For example, if you specify a rotation sequence of "YZX":

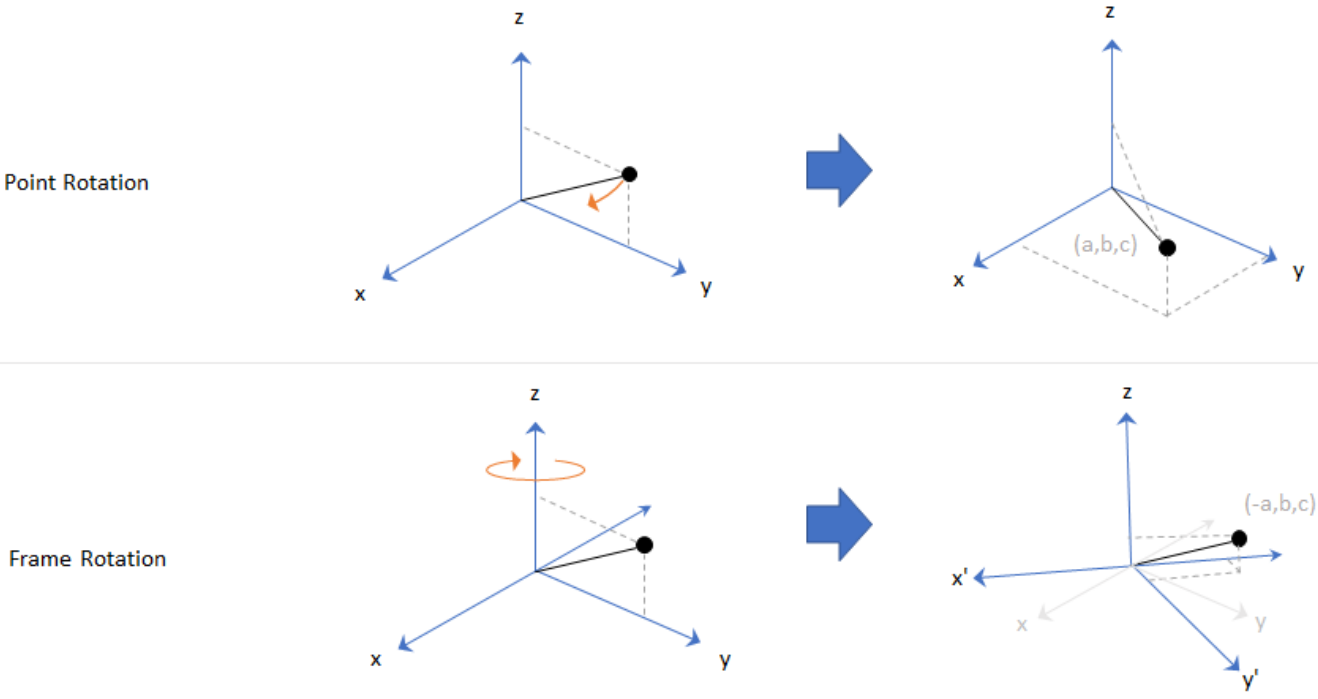
1. The first rotation is about the *y*-axis.
2. The second rotation is about the new *z*-axis.
3. The third rotation is about the new *x*-axis.

Data Types: `char` | `string`

`rotationType` — Type of rotation
`"point"` | `"frame"`

Type of rotation, specified as `"point"` or `"frame"`.

In a point rotation, the frame is static and the point moves. In a frame rotation, the point is static and the frame moves. Point rotation and frame rotation define equivalent angular displacements but in opposite directions.



Data Types: `char` | `string`

Output Arguments

`eulerAngles` — Euler angle representation (radians)

N-by-3 numeric matrix

Euler angle representation in radians, returned as an *N*-by-3 numeric matrix, where *N* is the number of quaternions in the `quat` argument.

For each row of `eulerAngles`, the first element corresponds to the first axis in the rotation sequence, the second element corresponds to the second axis in the rotation sequence, and the third element corresponds to the third axis in the rotation sequence.

The data type of the Euler angles representation is the same as the underlying data type of `quat`.

Data Types: `single` | `double`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`eulerd` | `rotateframe` | `rotatepoint`

Objects

`quaternion`

eulerd

Convert quaternion to Euler angles (degrees)

Syntax

```
eulerAngles = eulerd(quat,rotationSequence,rotationType)
```

Description

`eulerAngles = eulerd(quat,rotationSequence,rotationType)` converts the quaternion, `quat`, to an *N*-by-3 matrix of Euler angles in degrees.

Examples

Convert Quaternion to Euler Angles in Degrees

Convert a quaternion frame rotation to Euler angles in degrees using the "ZYX" rotation sequence.

```
quat = quaternion([0.7071 0.7071 0 0]);
eulerAnglesDegrees = eulerd(quat,"ZYX","frame")
```

```
eulerAnglesDegrees = 1×3

         0         0   90.0000
```

Input Arguments

`quat` — Quaternion to convert to Euler angles

`quaternion` object | array of `quaternion` objects

Quaternion to convert to Euler angles, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

`rotationSequence` — Rotation sequence

`"ZYX"` | `"ZYZ"` | `"ZXY"` | `"ZXZ"` | `"YXZ"` | `"YXY"` | `"YZX"` | `"YZY"` | `"XYZ"` | `"XYX"` | `"XZY"` | `"XZX"`

Rotation sequence of Euler angle representation, specified as one of these values.

- `"YZY"`
- `"YXY"`
- `"ZYZ"`
- `"ZXZ"`
- `"XYX"`
- `"XZX"`
- `"XYZ"`

- "YZX"
- "ZXY"
- "XZY"
- "ZYX"
- "YXZ"

The rotation sequence defines the order of rotations about the axes. For example, if you specify a rotation sequence of "YZX":

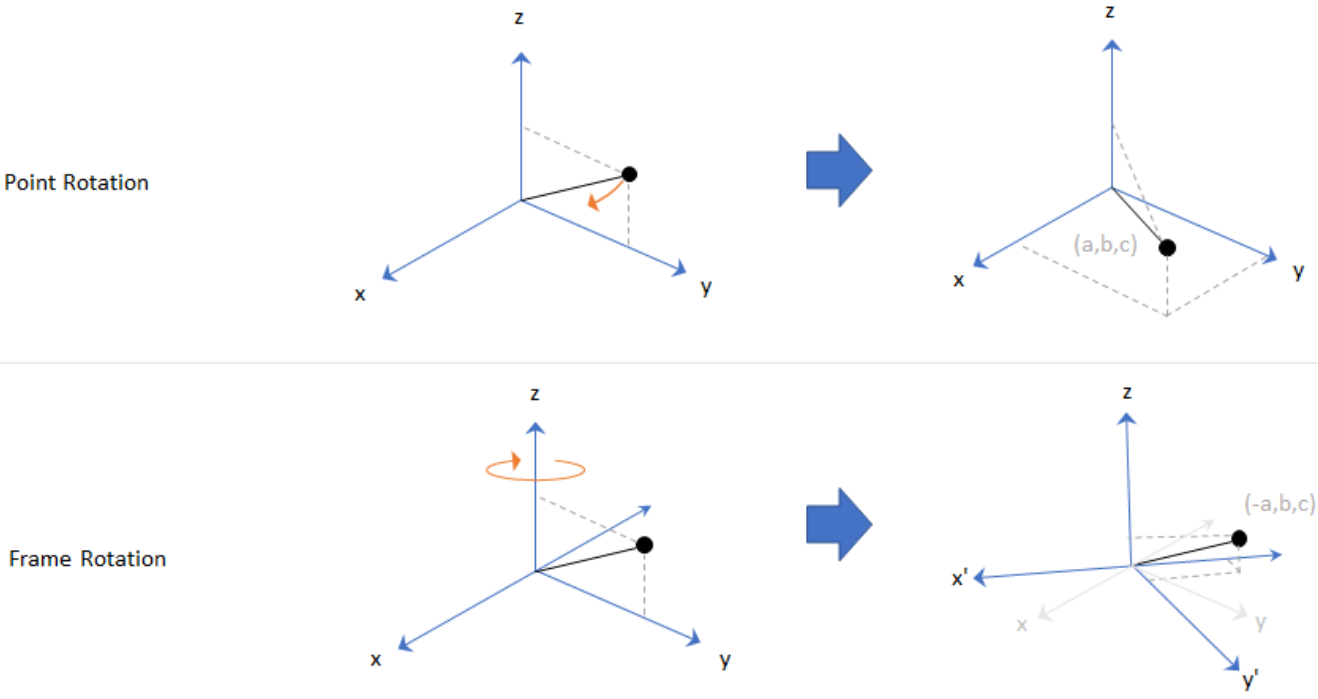
1. The first rotation is about the *y*-axis.
2. The second rotation is about the new *z*-axis.
3. The third rotation is about the new *x*-axis.

Data Types: `char` | `string`

`rotationType` — Type of rotation
`"point"` | `"frame"`

Type of rotation, specified as `"point"` or `"frame"`.

In a point rotation, the frame is static and the point moves. In a frame rotation, the point is static and the frame moves. Point rotation and frame rotation define equivalent angular displacements but in opposite directions.



Data Types: `char` | `string`

Output Arguments

`eulerAngles` — Euler angle representation (degrees)

N-by-3 numeric matrix

Euler angle representation in degrees, returned as an *N*-by-3 numeric matrix, where *N* is the number of quaternions in the `quat` argument.

For each row of `eulerAngles`, the first column corresponds to the first axis in the rotation sequence, the second column corresponds to the second axis in the rotation sequence, and the third column corresponds to the third axis in the rotation sequence.

The data type of the Euler angles representation is the same as the underlying data type of `quat`.

Data Types: `single` | `double`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`euler` | `rotateframe` | `rotatepoint`

Objects

`quaternion`

exp

Exponential of quaternion array

Syntax

```
B = exp(A)
```

Description

`B = exp(A)` computes the exponential of the elements of the quaternion array `A`.

Examples

Exponential of Quaternion Array

Create a 4-by-1 quaternion array A.

```
A = quaternion(magic(4))
```

```
A = 4×1 quaternion array
     16 +  2i +  3j + 13k
      5 + 11i + 10j +  8k
      9 +  7i +  6j + 12k
      4 + 14i + 15j +  1k
```

Compute the exponential of A.

```
B = exp(A)
```

```
B = 4×1 quaternion array
     5.3525e+06 + 1.0516e+06i + 1.5774e+06j + 6.8352e+06k
        -57.359 -     89.189i -     81.081j -     64.865k
        -6799.1 +     2039.1i +     1747.8j +     3495.6k
          -6.66 +     36.931i +     39.569j +     2.6379k
```

Input Arguments

A — Input quaternion

`quaternion` object | array of `quaternion` objects

Input quaternion, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

B — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion exponential, returned as a `quaternion` object or an array of `quaternion` objects of the same size as `A`.

Algorithms

Given a quaternion $A = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k} = a + \bar{v}$, the exponential is computed by

$$\exp(A) = e^{a}\left(\cos\|\bar{v}\| + \frac{\bar{v}}{\|\bar{v}\|}\sin\|\bar{v}\|\right)$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.^,power` | `log`

Objects

`quaternion`

ldivide, .\

Element-wise quaternion left division

Syntax

```
C = A.\B
```

Description

`C = A.\B` performs quaternion element-wise division by dividing each element of `B` by the corresponding element of `A`.

Examples

Divide a Quaternion Array by a Real Scalar

Create a 2-by-1 quaternion array, and divide it element-by-element by a real scalar.

```
A = quaternion([1:4;5:8])
```

```
A = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

```
B = 2;
C = A.\B
```

```
C = 2×1 quaternion array
     0.066667 -  0.13333i -      0.2j -  0.26667k
     0.057471 - 0.068966i -  0.08046j - 0.091954k
```

Divide a Quaternion Array by Another Quaternion Array

Create a 2-by-2 quaternion array, and divide it element-by-element by another 2-by-2 quaternion array.

```
q1 = quaternion([1:4;2:5;4:7;5:8]);
A = reshape(q1,2,2)
```

```
A = 2×2 quaternion array
     1 + 2i + 3j + 4k     4 + 5i + 6j + 7k
     2 + 3i + 4j + 5k     5 + 6i + 7j + 8k
```

```
q2 = quaternion(magic(4));
B = reshape(q2,2,2)
```

```
B = 2×2 quaternion array
     16 +  2i +  3j + 13k      9 +  7i +  6j + 12k
      5 + 11i + 10j +  8k      4 + 14i + 15j +  1k


C = A.\B

C = 2×2 quaternion array
          2.7 -      1.9i -      0.9j -      1.7k       1.5159 -  0.37302i -  0.15079j -  0.0238
       2.2778 +  0.46296i -  0.57407j + 0.092593k       1.2471 +  0.91379i -  0.33908j -   0.109
```

Input Arguments

A — Divisor

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Divisor, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

B — Dividend

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Dividend, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion division, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Division

Given a quaternion $A = a_1 + a_2\mathrm{i} + a_3\mathrm{j} + a_4\mathrm{k}$ and a real scalar p,

$$C = p.\backslash A = \frac{a_1}{p} + \frac{a_2}{p}\mathrm{i} + \frac{a_3}{p}\mathrm{j} + \frac{a_4}{p}\mathrm{k}$$

Note For a real scalar p, $A./p = A.\backslash p$.

Quaternion Division by a Quaternion Scalar

Given two quaternions A and B of compatible sizes, then

$$C = A.\backslash B = A^{-1}.*B = \left(\frac{conj(A)}{norm(A)^2}\right).*B$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.*,times` | `conj` | `norm` | `rdivide, ./` | `mldivide, \`

Objects

`quaternion`

log

Natural logarithm of quaternion array

Syntax

```
B = log(A)
```

Description

`B = log(A)` computes the natural logarithm of the elements of the quaternion array `A`.

Examples

Logarithmic Values of Quaternion Array

Create a 3-by-1 quaternion array A.

```
A = quaternion(randn(3,4))
```

```
A = 3×1 quaternion array
     0.53767 + 0.86217i - 0.43359j +  2.7694k
      1.8339 + 0.31877i + 0.34262j -  1.3499k
     -2.2588 -  1.3077i +  3.5784j +  3.0349k
```

Compute the logarithmic values of A.

```
B = log(A)
```

```
B = 3×1 quaternion array
      1.0925 + 0.40848i - 0.20543j +  1.3121k
      0.8436 + 0.14767i + 0.15872j - 0.62533k
      1.6807 - 0.53829i +   1.473j +  1.2493k
```

Input Arguments

A — Input quaternion

`quaternion` object | array of `quaternion` objects

Input quaternion, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

B — Natural logarithm values

`quaternion` object | array of `quaternion` objects

Natural logarithm values, returned as a `quaternion` object or an array of `quaternion` objects of the same size of the `A` argument.

Algorithms

Given a quaternion $A = a + \bar{v} = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, the logarithm is computed by

$$\log(A) = \log\|A\| + \frac{\bar{v}}{\|\bar{v}\|}\arccos\frac{a}{\|A\|}$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`exp` | `.^,power`

Objects

`quaternion`

interp1

Quaternion interpolation (table lookup)

Syntax

```
vq = interp1(x,v,xq)
vq = interp1(x,v,xq,method)
vq = interp1(x,v,xq,method,extrapolation)

vq = interp1(v,xq)
vq = interp1(v,xq,method)
vq = interp1(v,xq,method,extrapolation)
```

Description

`vq = interp1(x,v,xq)` interpolates the quaternion values of a 1-D function at specific query points using spherical linear interpolation (SLERP). `x` specifies the sample points, and `v` specifies the quaternions that contains the corresponding values *v(x)*. `xq` specifies the query points. By default, the function uses `"slerp-short"` interpolation method.

`vq = interp1(x,v,xq,method)` specifies the interpolation method.

`vq = interp1(x,v,xq,method,extrapolation)` specifies a quaternion value to return for all query points in `xq` that are outside the domain of `x`.

`vq = interp1(v,xq)` interpolates the quaternion values by assuming a default set of sample points. The default points are the sequence of numbers from 1 to *n*, where *n* depends on the shape of `v`:

- When `v` is a vector of quaternions, the default points are `1:length(v)`.
- When `v` is an array of quaternions, the default points are `1:size(v,1)`.

Use this syntax when you are not concerned about the absolute distances between sample points.

`vq = interp1(v,xq,method)` specifies the interpolation method and uses the default sample points.

`vq = interp1(v,xq,method,extrapolation)` specifies an extrapolation value and uses the default sample points.

Examples

Interpolate Quaternion Function Defined Using Sample Points

Define the sample points `x` and corresponding sample values `v`.

```
x = [1 2 5 6];
eul = [-185:45:-50; -20*ones(1,4); zeros(1,4)]';
v = quaternion(eul,"eulerd","ZYX","frame");
```

Define the query points over the range of x.

```
xq = [1.5 3 4 5.4];
```

Interpolate at the query points.

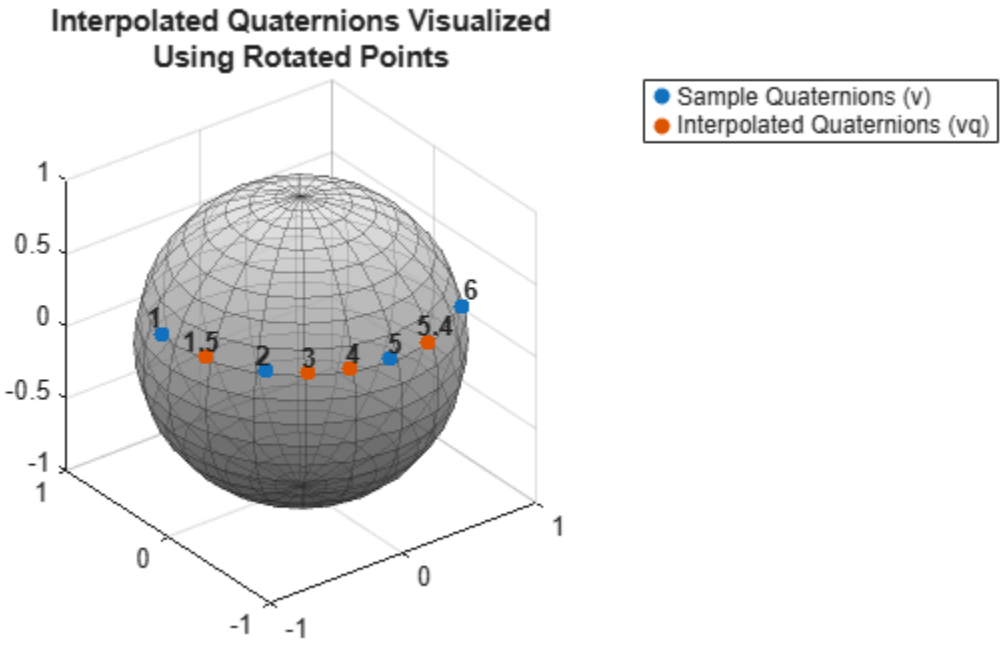
```
vq = interp1(x,v,xq);
```

To visualize the result, rotate the same point using the sample and interpolated quaternions.

```
pts_samples = rotatepoint(v,[1.05 0 0]);
pts_query = rotatepoint(vq,[1.05 0 0]);
```

Plot a unit sphere, and then plot the sample and interpolated quaternions on the sphere.

```
figure
[X,Y,Z] = sphere;
surf(X,Y,Z,FaceAlpha=0.5,EdgeAlpha=0.35)
colormap gray
hold on
scatter3(pts_samples(:,1),pts_samples(:,2),pts_samples(:,3),"filled")
scatter3(pts_query(:,1),pts_query(:,2),pts_query(:,3),"filled")
exampleHelperAnnotateQuats(pts_samples,pts_query,x,xq,10,1.05)
axis equal
title(["Interpolated Quaternions Visualized","Using Rotated Points"])
legend("","Sample Quaternions (v)","Interpolated Quaternions (vq)")
```



Interpolate Quaternions Without Specifying Sample Points

Define a set of function values.

```
eul = [-170:20:-70; -20*sind(0:72:360)-25; zeros(1,6)]';
v = quaternion(eul,"eulerd","ZYX","frame");
```

Define a set of query points that fall between the default points, `1:6`. In this case, the default points are `1:6` because `v` is a six-element quaternion array.

```
xq = [1.5 2.6 3.5 4.5 5.5];
```

Evaluate `v` at `xq` using the SQUAD natural interpolation method. Note that you could use the default `"slerp-short"` interpolation method here, but for the sine-wave rotation that this quaternion function describes, SQUAD produces a smoother interpolated path.

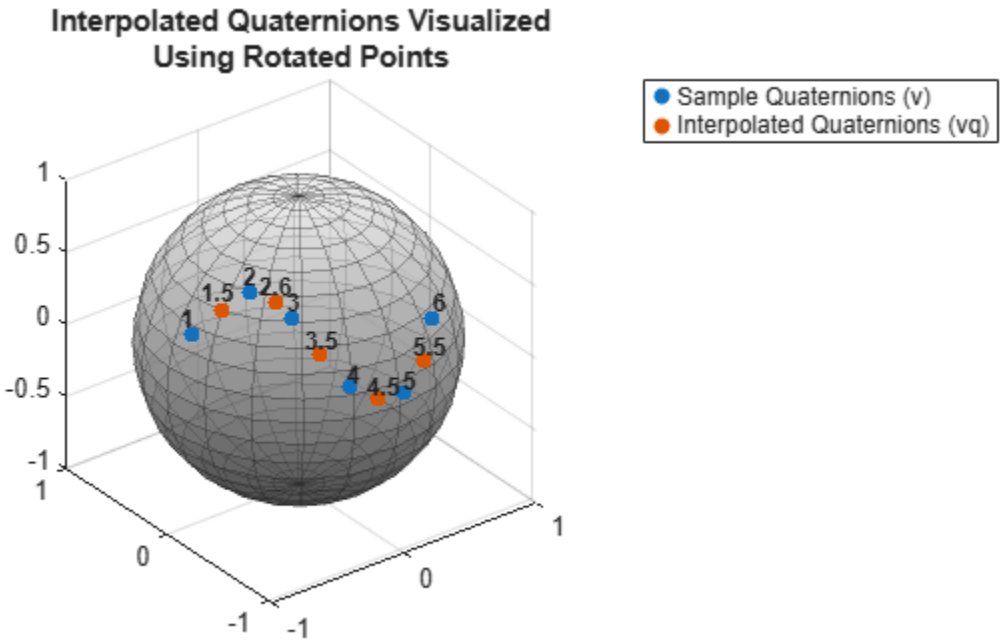
```
vq = interp1(v,xq,"squad-natural");
```

To visualize the result, rotate a point on the *x*-axis using the sample and interpolated quaternions.

```
pts_samples = rotatepoint(v,[1.05 0 0]);
pts_query = rotatepoint(vq,[1.05 0 0]);
```

Plot a unit sphere, and then plot the sample and interpolated quaternions on the sphere.

```
figure
[X,Y,Z] = sphere;
surf(X,Y,Z,FaceAlpha=0.5,EdgeAlpha=0.35)
colormap gray
hold on
scatter3(pts_samples(:,1),pts_samples(:,2),pts_samples(:,3),"filled")
scatter3(pts_query(:,1),pts_query(:,2),pts_query(:,3),"filled")
exampleHelperAnnotateQuats(pts_samples,pts_query,1:6,xq,9,1.05)
axis equal
title(["Interpolated Quaternions Visualized","Using Rotated Points"])
legend("","Sample Quaternions (v)","Interpolated Quaternions (vq)")
```



Get Smoother Angular Velocity Using SQUAD Interpolation

Define three quaternions representing different orientations.

```
q0 = quaternion([0 0 0],"euler","ZYX","frame");
q1 = quaternion([pi/4 pi/6 pi/3],"euler","ZYX","frame");
q2 = quaternion([pi/2 pi/4 pi/2],"euler","ZYX","frame");
```

Create a time vector for the original keyframes

```
x = [1 2 3];
```

Create a time vector for interpolation

```
T = linspace(0,3,300);
```

Interpolate using `interp1` with the `"slerp-natural"` and `"squad-natural"` methods.

```
quats_slerp = interp1(x,[q0 q1 q2],T,"slerp-natural")';
quats_squad = interp1(x,[q0 q1 q2],T,"squad-natural")';
```

Calculate the angular velocities for both interpolations methods.

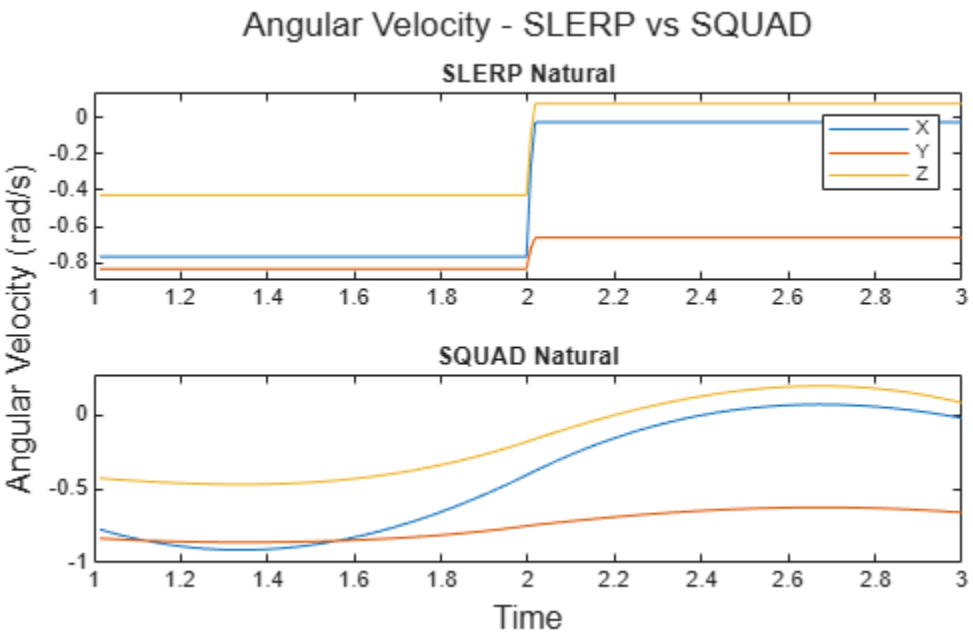
```
ang_vel_slerp = angvel(quats_slerp,T(2)-T(1),"frame");
ang_vel_squad = angvel(quats_squad,T(2)-T(1),"frame");
```

Plot the angular velocities.

```
tl = tiledlayout(2,1);
title(tl,"Angular Velocity - SLERP vs SQUAD")
```

Plot the angular velocities for the SLERP and SQUAD interpolations. Note that the angular velocity from the SQUAD interpolation is smoother than the interpolation using SLERP.

```
nexttile
plot(T,ang_vel_slerp)
ylim padded
title("SLERP Natural")
legend("X","Y","Z")
nexttile
plot(T,ang_vel_squad)
ylim padded
title("SQUAD Natural")
xlabel(tl,"Time")
ylabel(tl,"Angular Velocity (rad/s)")
```



Designate Constant Value for All Queries Outside Domain of x

Define the sample points `x` and corresponding sample values `v`.

```
x = [0 1 2];
eul = [0 30 60; 0 20 60; zeros(1,3)]';
v = quaternion(eul,"eulerd","ZYX","frame");
```

Specify the query points `xq` so that they extend beyond the domain of `x`.

```
xq = [-0.5 0.5 1.5 2.5];
```

Now evaluate `v` at `xq` using the `"slerp-short"` interpolation method, and assign any values outside the domain of `x` to a quaternion with all parts set to `0`.

```
vq = interp1(x,v,xq,"slerp-short",ones("quaternion"))'
```

```
vq = 4×1 quaternion array
           1 +        0i +        0j +        0k
     0.98774 + 0.022751i - 0.084907j -  0.12903k
     0.87097 +    0.151i -  0.30755j -  0.35217k
           1 +        0i +        0j +        0k
```

Input Arguments

x — Sample points

N-element vector

Sample points, specified as an *N*-element vector of real numbers. `x` must contain at least two sample points, and the values in `x` must be distinct. The length of `x` depends on the shape of `v`.

- If `v` is a vector of quaternions, then `length(x)` must equal `length(v)`.
- If `v` is an array of quaternions, then `length(x)` must equal `size(v,1)`.

Example: `[1 2 3 4 5 6 7 8 9 10]`

Example: `1:10`

Example: `[3 7 11 15 19 23 27 31]'`

`v` — Sample quaternions
N-element array of `quaternion` objects | *N*-by-*M* array of `quaternion` objects

Sample quaternions, specified as an *N*-element vector of `quaternion` objects or an *N*-by-*M* array of quaternion objects.

If you have multiple sets of data that are sampled at the same query points, then you can specify `v` as an *N*-by-*M* array. Each column of the array contains a separate set of 1-D sample values and the first dimension of the array must be of length *N*, where *N* is the number of specified sample points `x` and *M* is the number of datasets corresponding to `x`.

`xq` — Query points
scalar | vector | matrix | array

Query points, specified as a scalar, vector, matrix, or array of real numbers.

Example: `5`

Example: `1:0.05:10`

Example: `(1:0.05:10)'`

Example: `[0 1 2 7.5 10]`

`method` — Interpolation method
`"slerp-short"` (default) | `"slerp-natural"` | `"squad-natural"` | `"nearest"` | `"next"` | `"previous"`

Interpolation method, specified as one of the options in this table:

Method	Description	Continuity	Comments
`"slerp-short"`	Spherical linear interpolation between sample quaternions by an interpolation coefficient, picking the shortest interpolation path between the sample quaternions. Note that, at each sample point, the input and output quaternions might be inverted to ensure the shortest path. For more information about the SLERP algorithm, see “SLERP Algorithm” on page 2-1547.	C^0	• Requires at least 2 sample quaternions • Low memory requirements • Efficient computation time • Ensures shortest path interpolation. • Fastest computation time
`"slerp-natural"`	Spherical linear interpolation between sample quaternions by an interpolation coefficient, picking the natural interpolation path between the sample quaternions. The input and output quaternions will have the same sign at the sample points. For more information about the SLERP algorithm, see “SLERP Algorithm” on page 2-1547.	C^0	• Requires at least 2 sample quaternions • Low memory requirements • Efficient computation time • The input and output quaternions will have the same sign at the sample points.

Method	Description	Continuity	Comments
`"squad-natural"`	Spherical quadrangle interpolation between sample quaternions, providing smooth and continuous transitions with gradual changes in acceleration. The input and output quaternions will have the same sign at the sample points. For more information about the SQUAD algorithm, see “SQUAD Algorithm” on page 2-1547.	C^1	• Requires at least 2 sample quaternions • Moderate memory requirements • Efficient computation time • Ensures smooth transitions and gradual acceleration changes
`"nearest"`	Nearest neighbor interpolation. The interpolated value at a query point is the value at the nearest sample quaternion.	Discontinuous	• Requires at least 2 sample quaternions • Modest memory requirements
`"next"`	Next neighbor interpolation. The interpolated value at a query point is the value at the next sample quaternion.	Discontinuous	• Requires at least 2 sample quaternions. • Same memory requirements and computation time as `"nearest"`
`"previous"`	Previous neighbor interpolation. The interpolated value at a query point is the value at the previous sample quaternion.	Discontinuous	• Requires at least 2 sample quaternions. • Same memory requirements and computation time as `"nearest"`

`extrapolation` — Extrapolation quaternion
`nan("quaternion")` (default) | `quaternion` object

Extrapolation quaternion, specified as a `quaternion` object. `interp1` returns this quaternion for any `xq` values that query outside of domain of `x`.

Example: `interp1([0 1 2],quats,2.5,"slerp-natural")` returns a quaternion of value `NaN + NaNi + NaNj + NaNk`.

Example: `interp1([0 1 2],quats,2.5,"slerp-natural",zeros("quaternion"))` returns a quaternion of value `0 + 0i + 0j + 0k`.

Output Arguments

vq — Interpolated quaternions

scalar | vector | matrix | array

Interpolated values, returned as a scalar, vector, matrix, or array. The size of `vq` depends on the shapes of `v` and `xq`.

Shape of v	Shape of xq	Size of vq	Example
Vector	Vector	`size(xq)`	If `size(v) = [1 100]` and `size(xq) = [1 500]`, then `size(vq) = [1 500]`.
Vector	Matrix or N-D Array	`size(xq)`	If `size(v) = [1 100]` and `size(xq) = [50 30]`, then `size(vq) = [50 30]`.
Array	Vector	`[length(xq) size(v,2),...,size(v,n)]`	If `size(v) = [100 3]` and `size(xq) = [1 500]`, then `size(vq) = [500 3]`.
Array	Matrix or N-D Array	`[size(xq,1),...,size(xq, n),... size(v,2),...,size(v,m)]`	If `size(v) = [4 5 6]` and `size(xq) = [2 3 7]`, then `size(vq) = [2 3 7 5 6]`.

Algorithms

SLERP Algorithm

Quaternion spherical linear interpolation (SLERP) is an extension of linear interpolation along a plane to spherical interpolation in three dimensions. The algorithm was first proposed in [1]. Given two quaternions, q_1 and q_2, SLERP interpolates a new quaternion, q_0, along the great circle that connects q_1 and q_2. The interpolation coefficient, T, determines how close the output quaternion is to either q_1 and q_2.

The SLERP algorithm can be described in terms of sinusoids:

$$q_0 = \frac{\sin((1-T)\theta)}{\sin(\theta)} q_1 + \frac{\sin(T\theta)}{\sin(\theta)} q_2$$

where q_1 and q_2 are normalized quaternions, and θ is half the angular distance between q_1 and q_2.

SQUAD Algorithm

Quaternion spherical and quadrangle (SQUAD) interpolation is an extension of the SLERP interpolation method designed for interpolating between a sequence of quaternions [2]. It produces a continuous, smooth, angular velocity across multiple quaternion keyframes by introducing intermediate, control quaternions, much like how Bézier curves use control points to create a smooth path in Cartesian space. The SQUAD algorithm calculates two control quaternions between each consecutive pair of quaternion keyframes, specified using the input argument `v`, forming a quadrangle with each pair of quaternion keyframes. Neighboring quaternion keyframes also influence the calculation of these control quaternions to ensure smooth transitions between the interpolated paths of each quaternion keyframe pair.

The SQUAD interpolation algorithm first uses SLERP between the quaternions q_i and q_{i+1} with the interpolation coefficient h, corresponding to input argument `T` in the `slerp` function. Then, the algorithm uses SLERP to interpolate between the control quaternions s_i and s_{i+1}. Finally, the algorithm uses SLERP between these two SLERP paths, using *2h(1–h)* as the interpolation coefficient to get the SQUAD interpolated path. The *2h(1–h)* interpolation coefficient for the final SLERP is a quadratic function that ensures this interpolation is a smooth curved path instead of a linear straight path.

In equation form, the SQUAD interpolation between q_i and q_{i+1} with the interpolation coefficient h is defined as:

$$Squad(q_i, q_{i+1}, s_i, s_{i+1}, h) = Slerp(Slerp(q_i, q_{i+1}, h), Slerp(s_i, s_{i+1}, h), 2h(1-h)),$$

where s_i and s_{i+1} are the control quaternions defined using the equation:

$$s_i = q_i \exp\left(-\frac{\log(q_i^{-1}q_{i+1}) + \log(q_i^{-1}q_{i-1})}{4}\right)$$

Version History

Introduced in R2025a

References

[1] Shoemake, Ken. "Animating Rotation with Quaternion Curves." *ACM SIGGRAPH Computer Graphics* 19, no. 3 (July 1985): 245–54. https://doi.org/10.1145/325165.325242.

[2] Dam, Erik B., Martin Koch, and Martin Lillholm. *Quaternions, Interpolation and Animation*. Technical Report DIKU-TR-98/5. Department of Computer Science, University of Copenhagen, July 17, 1998. https://web.mit.edu/2.998/www/QuaternionReport1.pdf.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`quaternion` | `slerp`

meanrot

Quaternion mean rotation

Syntax

```
quatAverage = meanrot(quat)
quatAverage = meanrot(quat,dim)
quatAverage = meanrot( ___ ,nanflag)
```

Description

`quatAverage = meanrot(quat)` returns the average rotation of the elements of `quat` along the first array dimension whose size not does equal 1.

- If `quat` is a vector, `meanrot(quat)` returns the average rotation of the elements.
- If `quat` is a matrix, `meanrot(quat)` returns a row vector containing the average rotation of each column.
- If `quat` is a multidimensional array, then `mearot(quat)` operates along the first array dimension whose size does not equal 1, treating the elements as vectors. This dimension becomes 1 while the sizes of all other dimensions remain the same.

The `meanrot` function normalizes the input quaternions, `quat`, before calculating the mean.

`quatAverage = meanrot(quat,dim)` return the average rotation along dimension `dim`. For example, if `quat` is a matrix, then `meanrot(quat,2)` is a column vector containing the mean of each row.

`quatAverage = meanrot( ___ ,nanflag)` specifies whether to include or omit `NaN` values from the calculation for any of the previous syntaxes. `meanrot(quat,"includenan")` includes all `NaN` values in the calculation while `mean(quat,"omitnan")` ignores them.

Examples

Quaternion Mean Rotation

Create a matrix of quaternions corresponding to three sets of Euler angles.

```
eulerAngles = [40 20 10; ...
               50 10 5; ...
               45 70 1];

quat = quaternion(eulerAngles,"eulerd","ZYX","frame");
```

Determine the average rotation represented by the quaternions. Convert the average rotation to Euler angles in degrees for readability.

```
quatAverage = meanrot(quat)
```

```
quatAverage = quaternion
      0.88863 - 0.062598i +  0.27822j +  0.35918k


eulerAverage = eulerd(quatAverage,"ZYX","frame")

eulerAverage = 1×3

   45.7876   32.6452    6.0407
```

Average Out Rotational Noise

Use `meanrot` over a sequence of quaternions to average out additive noise.

Create a vector of 1e6 quaternions whose distance, as defined by the `dist` function, from quaternion(1,0,0,0) is normally distributed. Plot the Euler angles corresponding to the noisy quaternion vector.

```
nrows = 1e6;
ax = 2*rand(nrows,3) - 1;
ax = ax./sqrt(sum(ax.^2,2));
ang = 0.5*randn(size(ax,1),1);
q = quaternion(ax.*ang ,"rotvec");

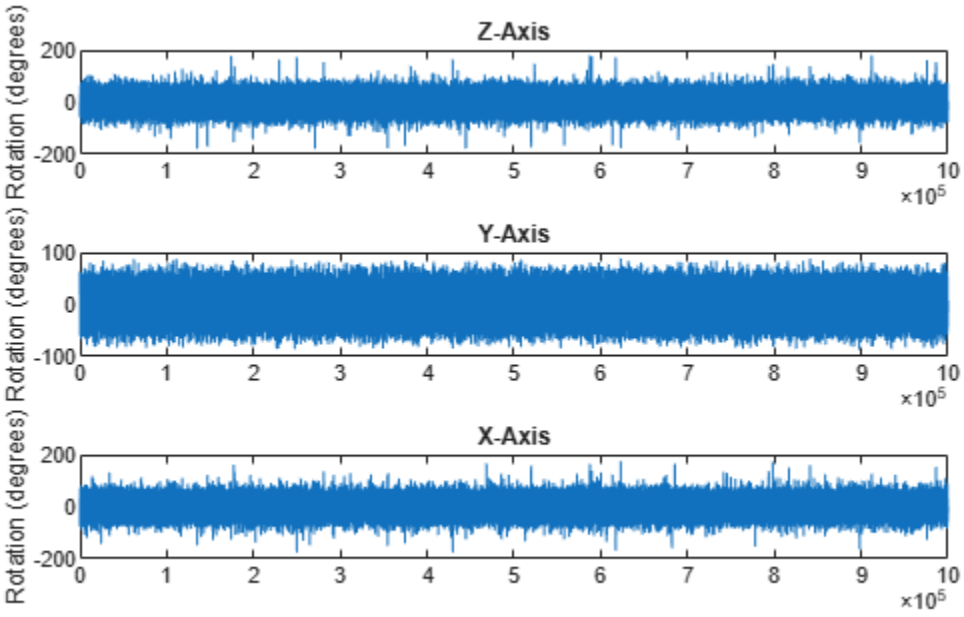
noisyEulerAngles = eulerd(q,"ZYX","frame");

figure(1)

subplot(3,1,1)
plot(noisyEulerAngles(:,1))
title("Z-Axis")
ylabel("Rotation (degrees)")
hold on

subplot(3,1,2)
plot(noisyEulerAngles(:,2))
title("Y-Axis")
ylabel("Rotation (degrees)")
hold on

subplot(3,1,3)
plot(noisyEulerAngles(:,3))
title("X-Axis")
ylabel("Rotation (degrees)")
hold on
```



Use `meanrot` to determine the average quaternion given the vector of quaternions. Convert to Euler angles and plot the results.

```
qAverage = meanrot(q);

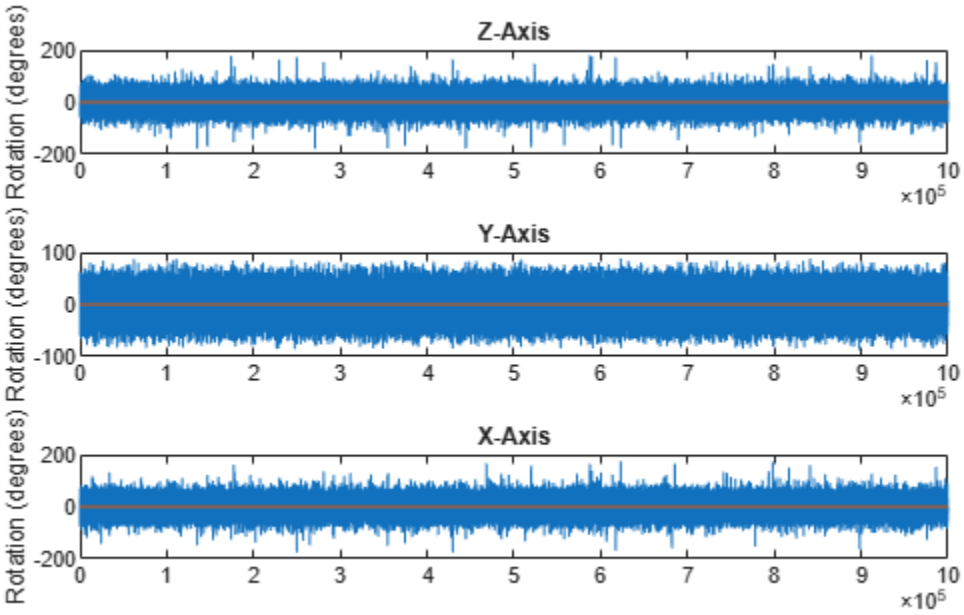
qAverageInEulerAngles = eulerd(qAverage,"ZYX","frame");

figure(1)

subplot(3,1,1)
plot(ones(nrows,1)*qAverageInEulerAngles(:,1))
title("Z-Axis")

subplot(3,1,2)
plot(ones(nrows,1)*qAverageInEulerAngles(:,2))
title("Y-Axis")

subplot(3,1,3)
plot(ones(nrows,1)*qAverageInEulerAngles(:,3))
title("X-Axis")
```



The `meanrot` Algorithm and Limitations

The `meanrot` Algorithm

The `meanrot` function outputs a quaternion that minimizes the squared Frobenius norm of the difference between rotation matrices. Consider two quaternions:

- `q0` represents no rotation.
- `q90` represents a 90 degree rotation about the *x*-axis.

```
q0 = quaternion([0 0 0],"eulerd","ZYX","frame");
q90 = quaternion([0 0 90],"eulerd","ZYX","frame");
```

Create a quaternion sweep, `qSweep`, that represents rotations from 0 to 180 degrees about the *x*-axis.

```
eulerSweep = (0:1:180)';
qSweep = quaternion([zeros(numel(eulerSweep),2),eulerSweep], ...
    "eulerd","ZYX","frame");
```

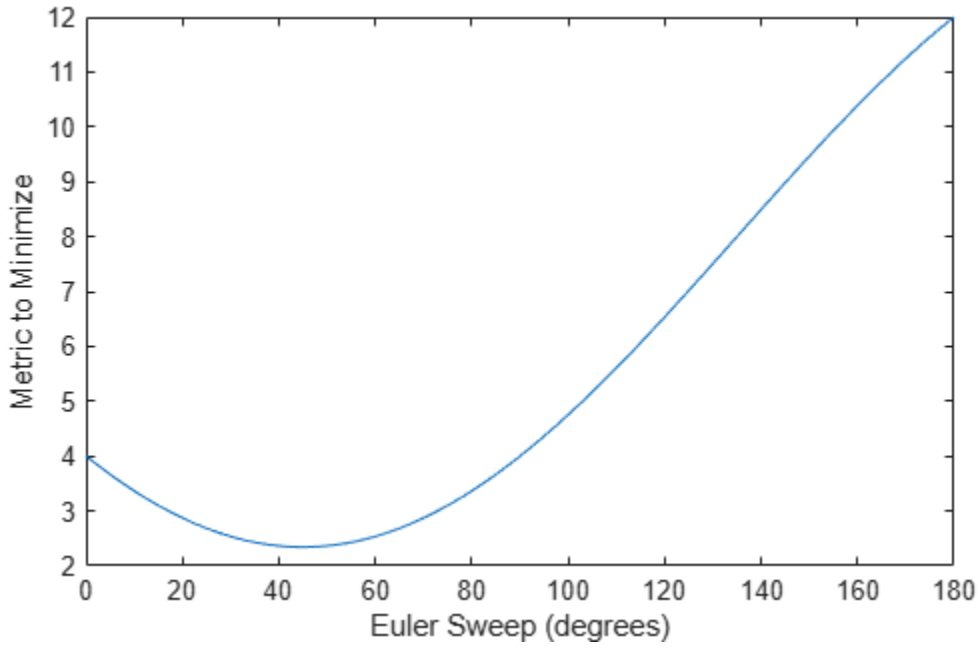
Convert `q0`, `q90`, and `qSweep` to rotation matrices. In a loop, calculate the metric to minimize for each member of the quaternion sweep. Plot the results and return the value of the Euler sweep that corresponds to the minimum of the metric.

```
r0     = rotmat(q0,"frame");
r90    = rotmat(q90,"frame");
rSweep = rotmat(qSweep,"frame");

metricToMinimize = zeros(size(rSweep,3),1);
for i = 1:numel(qSweep)
    metricToMinimize(i) = norm((rSweep(:,:,i) - r0),"fro").^2 + ...
```

```
                        norm((rSweep(:,:,i) - r90),"fro").^2;
end

plot(eulerSweep,metricToMinimize)
xlabel("Euler Sweep (degrees)")
ylabel("Metric to Minimize")
```




```
[~,eulerIndex] = min(metricToMinimize);
eulerSweep(eulerIndex)
```

```
ans =
45
```

The minimum of the metric corresponds to the Euler angle sweep at 45 degrees. That is, `meanrot` defines the average between `quaterion([0 0 0],"ZYX","frame")` and `quaternion([0 0 90],"ZYX","frame")` as `quaternion([0 0 45],"ZYX","frame")`. Call `meanrot` with `q0` and `q90` to verify the same result.

```
eulerd(meanrot([q0,q90]),"ZYX","frame")
```

```
ans = 1×3

         0         0   45.0000
```

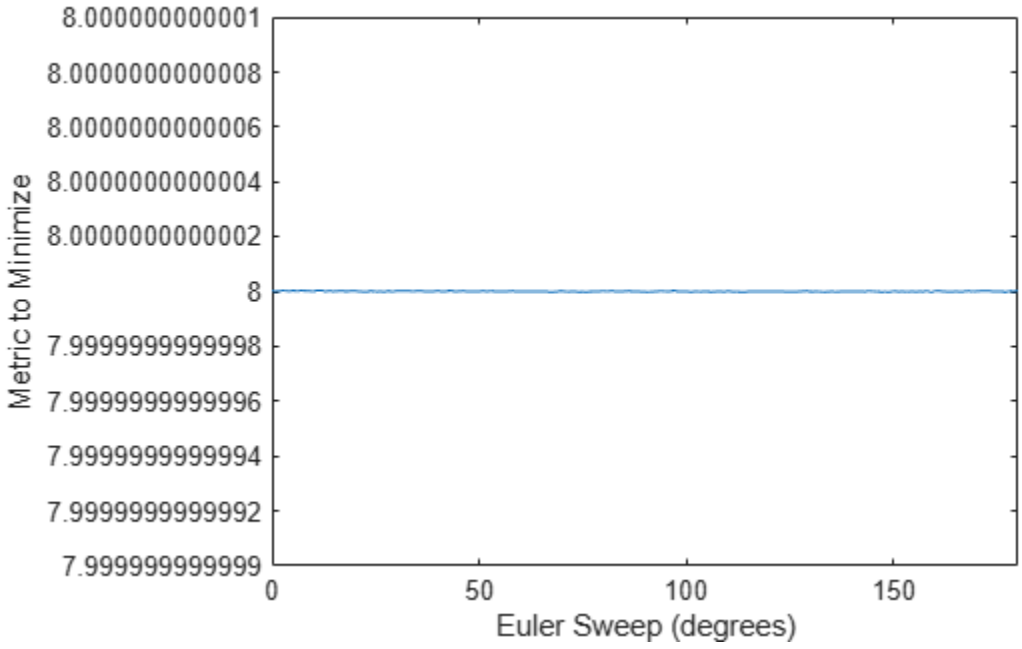
Limitations

The metric that `meanrot` uses to determine the mean rotation is not unique for quaternions significantly far apart. Repeat the experiment above for quaternions that are separated by 180 degrees.

```
q180 = quaternion([0 0 180],"eulerd","ZYX","frame");
r180 = rotmat(q180,"frame");
```

```
for i = 1:numel(qSweep)
    metricToMinimize(i) = norm((rSweep(:,:,i) - r0),"fro").^2 + ...
                          norm((rSweep(:,:,i) - r180),"fro").^2;
end

plot(eulerSweep,metricToMinimize)
xlabel("Euler Sweep (degrees)")
ylabel("Metric to Minimize")
```




```
[~,eulerIndex] = min(metricToMinimize);
eulerSweep(eulerIndex)
```

```
ans =
159
```

Quaternion means are usually calculated for rotations that are close to each other, which makes the edge case shown in this example unlikely in real-world applications. To average two quaternions that are significantly far apart, use the `slerp` function. Repeat the experiment using `slerp` and verify that the quaternion mean returned is more intuitive for large distances.

```
qMean = slerp(q0,q180,0.5);
q0_q180 = eulerd(qMean,"ZYX","frame")
```

```
q0_q180 = 1×3
```

```
         0         0   90.0000
```

Input Arguments

`quat` — Quaternion

`quaternion` object | array of `quaternion` objects

Quaternion for which to calculate the mean, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

`dim` — Dimension to operate along

positive integer scalar

Dimension to operate along, specified as a positive integer scalar. If no value is specified, then the default is the first array dimension whose size does not equal 1.

Dimension `dim` indicates the dimension whose length reduces to 1. The `size(quatAverage,dim)` is `1`, while the sizes of all other dimensions remain the same.

Data Types: `double` | `single`

`nanflag` — `NaN` condition

`"includenan"` (default) | `"omitnan"`

`NaN` condition, specified as one of these values:

- `"includenan"` –– Include `NaN` values when computing the mean rotation, resulting in `NaN`.
- `"omitnan"` –– Ignore all `NaN` values in the input.

Data Types: `char` | `string`

Output Arguments

`quatAverage` — Quaternion average rotation

`quaternion` object | array of `quaternion` objects

Quaternion average rotation, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

`meanrot` determines a quaternion mean, $\bar{q}$, according to [1]. $\bar{q}$ is the quaternion that minimizes the squared Frobenius norm of the difference between rotation matrices:

$$\bar{q} = \arg \min_{q \in \mathrm{S}^3} \sum_{i=1}^{n} \| A(q) - A(q_i) \|_F^2$$

Version History

Introduced in R2023b

References

[1] Markley, F. Landis, Yang Chen, John Lucas Crassidis, and Yaakov Oshman. "Average Quaternions." *Journal of Guidance, Control, and Dynamics*. Vol. 30, Issue 4, 2007, pp. 1193-1197.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`dist` | `slerp`

Objects

`quaternion`

minus, -

Quaternion subtraction

Syntax

```
C = A - B
```

Description

`C = A - B` subtracts quaternion `B` from quaternion `A` using quaternion subtraction. Either `A` or `B` may be a real number, in which case subtraction is performed with the real part of the quaternion argument.

Examples

Subtract a Quaternion from a Quaternion

Quaternion subtraction is defined as the subtraction of the corresponding parts of each quaternion. Create two quaternions and perform subtraction.

```
Q1 = quaternion([1,0,-2,7]);
Q2 = quaternion([1,2,3,4]);

Q1minusQ2 = Q1 - Q2
```

```
Q1minusQ2 = quaternion
     0 - 2i - 5j + 3k
```

Subtract a Real Number from a Quaternion

Addition and subtraction of real numbers is defined for quaternions as acting on the real part of the quaternion. Create a quaternion and then subtract 1 from the real part.

```
Q = quaternion([1,1,1,1])
```

```
Q = quaternion
     1 + 1i + 1j + 1k
```

```
Qminus1 = Q - 1
```

```
Qminus1 = quaternion
     0 + 1i + 1j + 1k
```

Input Arguments

A — Input

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

B — Input

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion subtraction, returned as a `quaternion` object or an array of `quaternion` objects.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`uminus, -` | `times, .*` | `mtimes, *`

Objects

`quaternion`

mldivide, \

Quaternion left division for scalars

Syntax

```
C = A\B
```

Description

`C = A\B` performs left division between a scalar and a quaternion or array of quaternions.

Examples

Left Divide Quaternion Array by Real Scalar

Create a 2-by-1 quaternion array, and left divide it by a real scalar.

```
A = quaternion([1:4;5:8])
```

```
A = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

```
B = 2;
C = A\B
```

```
C = 2×1 quaternion array
     0.066667 -  0.13333i -      0.2j -  0.26667k
     0.057471 - 0.068966i -  0.08046j - 0.091954k
```

Left Divide Quaternion Array by Quaternion Scalar

Create a 2-by-2 quaternion array, and left divide it by a quaternion scalar.

```
q1 = quaternion(magic(4));
A = reshape(q1,2,2)
```

```
A = 2×2 quaternion array
     16 +  2i +  3j + 13k      9 +  7i +  6j + 12k
      5 + 11i + 10j +  8k      4 + 14i + 15j +  1k
```

```
B = quaternion([1:4])
```

```
B = quaternion
     1 + 2i + 3j + 4k
```

```
C = A\B
```

```
C = 2×2 quaternion array
       0.18493 +   0.13014i +  0.061644j +   0.11644k        0.2871 +  0.074194i +  0.080645j +
        0.2871 -  0.054839i +   0.10645j - 0.0032258k       0.18493 -   0.14384i +   0.11644j + 0
```

Input Arguments

A — Divisor

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Divisor, specified as a `quaternion` object, an array of `quaternion` objects, a real numeric scalar, or an array of real numbers. Numeric values must be of data type `single` or `double`.

At least one of `A` and `B` must a scalar.

B — Dividend

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Dividend, specified as a `quaternion` object, an array of `quaternion` objects, a real numeric scalar, or an array of real numbers. Numeric values must be of data type `single` or `double`.

At least one of `A` and `B` must a scalar.

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion left division, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Division

Given a quaternion $A = a_1 + a_2\mathrm{i} + a_3\mathrm{j} + a_4\mathrm{k}$ and a real numeric scalar p,

$$C = p\backslash A = \frac{a_1}{p} + \frac{a_2}{p}\mathrm{i} + \frac{a_3}{p}\mathrm{j} + \frac{a_4}{p}\mathrm{k}$$

Note For a real numeric scalar p, $A\backslash p = p/A$.

Quaternion Division by a Quaternion Scalar

Given two quaternions A and B,

$$C = A\backslash B = A^{-1} * B = \left(\frac{conj(A)}{norm(A)^2}\right) * B$$

Version History

Introduced in R2024a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.*,times` | `conj` | `norm` | `.\,ldivide` | `mrdivide, /`

Objects

`quaternion`

mrdivide, /

Quaternion right division for scalars

Syntax

```
C = A/B
```

Description

`C = A/B` performs right division between a scalar and a quaternion or array of quaternions.

Examples

Divide Quaternion Array by Real Scalar

Create a 2-by-1 quaternion array, and right divide it by a real scalar.

```
A = quaternion([1:4;5:8])
```

```
A = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

```
B = 2;
C = A/B
```

```
C = 2×1 quaternion array
     0.5 +   1i + 1.5j +   2k
     2.5 +   3i + 3.5j +   4k
```

Right Divide Quaternion Array by Quaternion Scalar

Create a 2-by-2 quaternion array, and right divide it by a quaternion scalar.

```
q1 = quaternion(magic(4));
A = reshape(q1,2,2)
```

```
A = 2×2 quaternion array
     16 +  2i +  3j + 13k      9 +  7i +  6j + 12k
      5 + 11i + 10j +  8k      4 + 14i + 15j +  1k
```

```
B = quaternion([1:4])
```

```
B = quaternion
     1 + 2i + 3j + 4k
```

```
C = A/B
```

```
C = 2×2 quaternion array
          2.7 -      0.1i -      2.1j -      1.7k       2.9667 + 0.033333i -  0.56667j -      1.1
       2.9667 -      0.5i +  0.76667j -  0.83333k          2.7 -      1.7i +      1.9j -      0.9
```

Input Arguments

A — Divisor

`quaternion` object | array of `quaternion` objects | real numeric scalar | array of real numbers

Divisor, specified as a `quaternion` object, an array of `quaternion` objects, a real numeric scalar, or an array of real numbers. Numeric values must be of data type `single` or `double`.

At least one of `A` and `B` must a scalar.

B — Dividend

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Dividend, specified as a `quaternion` object, an array of `quaternion` objects, a real numeric scalar, or an array of real numbers. Numeric values must be of data type `single` or `double`.

At least one of `A` and `B` must a scalar.

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion right division, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Division

Given a quaternion $A = a_1 + a_2\mathrm{i} + a_3\mathrm{j} + a_4\mathrm{k}$ and a real numeric scalar p,

$$C = p/A = \frac{a_1}{p} + \frac{a_2}{p}\mathrm{i} + \frac{a_3}{p}\mathrm{j} + \frac{a_4}{p}\mathrm{k}$$

Note For a real numeric scalar p, $A/p = p\backslash A$.

Quaternion Division by a Quaternion Scalar

Given two quaternions A and B,

$$C = A/B = A*B^{-1} = A*\left(\frac{conj(B)}{norm(B)^2}\right)$$

Version History

Introduced in R2024a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.*,times` | `conj` | `norm` | `mldivide, \` | `rdivide, ./`

Objects

`quaternion`

mtimes, *

Quaternion multiplication

Syntax

```
quatC = A*B
```

Description

`quatC = A*B` implements quaternion multiplication if either `A` or `B` is a quaternion. Either `A` or `B` must be a scalar.

You can use quaternion multiplication to compose rotation operators:

- To compose a sequence of frame rotations, multiply the quaternions in the order of the desired sequence of rotations. For example, to apply a p quaternion followed by a q quaternion, multiply in the order pq. The rotation operator becomes $(pq)^*v(pq)$, where v represents the object to rotate specified in quaternion form. * represents conjugation.
- To compose a sequence of point rotations, multiply the quaternions in the reverse order of the desired sequence of rotations. For example, to apply a p quaternion followed by a q quaternion, multiply in the reverse order, qp. The rotation operator becomes $(qp)v(qp)^*$.

Examples

Multiply Quaternion Scalar and Quaternion Vector

Create a 4-by-1 column vector, `A`, and a scalar, `b`. Multiply `A` times `b`.

```
A = quaternion(randn(4,4))
```

```
A = 4×1 quaternion array
      0.53767 +  0.31877i +   3.5784j +   0.7254k
       1.8339 -   1.3077i +   2.7694j - 0.063055k
      -2.2588 -  0.43359i -   1.3499j +  0.71474k
      0.86217 +  0.34262i +   3.0349j -  0.20497k
```

```
b = quaternion(randn(1,4))
```

```
b = quaternion
    -0.12414 +  1.4897i +   1.409j +  1.4172k
```

```
C = A*b
```

```
C = 4×1 quaternion array
      -6.6117 +   4.8105i +  0.94224j -   4.2097k
      -2.0925 +   6.9079i +   3.9995j -   3.3614k
       1.8155 -   6.2313i -    1.336j -     1.89k
      -4.6033 +   5.8317i + 0.047161j -    2.791k
```

Input Arguments

A — Input to multiply

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input to multiply, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

If `B` is nonscalar, then `A` must be scalar.

B — Input to multiply

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input to multiply, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

If `A` is nonscalar, then `B` must be scalar.

Output Arguments

quatC — Quaternion product

`quaternion` object | array of `quaternion` objects

Quaternion product, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Multiplication by a Real Scalar

Given a quaternion

$$q = a_q + b_q\mathrm{i} + c_q\mathrm{j} + d_q\mathrm{k},$$

the product of q and a real scalar β is

$$\beta q = \beta a_q + \beta b_q\mathrm{i} + \beta c_q\mathrm{j} + \beta d_q\mathrm{k}$$

Quaternion Multiplication by a Quaternion Scalar

The definition of the basis elements for quaternions,

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1,$$

can be expanded to populate a table summarizing quaternion basis element multiplication:

	1	**i**	**j**	**k**
1	1	i	j	k
i	i	−1	k	−j

j	j	−k	−1	i
k	k	j	−i	−1

When reading the table, the rows are read first, for example: ij = k and ji = −k.

Given two quaternions, $q = a_q + b_q i + c_q j + d_q k$, and $p = a_p + b_p i + c_p j + d_p k$, the multiplication can be expanded as:

$$\begin{aligned} z = pq &= \left(a_p + b_p i + c_p j + d_p k\right)\left(a_q + b_q i + c_q j + d_q k\right) \\ &= a_p a_q + a_p b_q i + a_p c_q j + a_p d_q k \\ &+ b_p a_q i + b_p b_q i^2 + b_p c_q ij + b_p d_q ik \\ &+ c_p a_q j + c_p b_q ji + c_p c_q j^2 + c_p d_q jk \\ &+ d_p a_q k + d_p b_q ki + d_p c_q kj + d_p d_q k^2 \end{aligned}$$

You can simplify the equation using the quaternion multiplication table:

$$\begin{aligned} z = pq &= a_p a_q + a_p b_q i + a_p c_q j + a_p d_q k \\ &+ b_p a_q i - b_p b_q + b_p c_q k - b_p d_q j \\ &+ c_p a_q j - c_p b_q k - c_p c_q + c_p d_q i \\ &+ d_p a_q k + d_p b_q j - d_p c_q i - d_p d_q \end{aligned}$$

Version History

Introduced in R2023b

References

[1] Kuipers, Jack B. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton, NJ: Princeton University Press, 2007.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.*,times`

Objects

`quaternion`

norm

Quaternion norm

Syntax

```
N = norm(quat)
```

Description

`N = norm(quat)` returns the quaternion's norm, `quat`, as the square root of the sum of its component squared.

Given a quaternion of the form $Q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, the norm of the quaternion is defined as $\mathrm{norm}(Q) = \sqrt{a^2 + b^2 + c^2 + d^2}$.

Examples

Calculate Quaternion Norm

Create a scalar quaternion and calculate its norm.

```
quat = quaternion(1,2,3,4);
norm(quat)
```

```
ans =
5.4772
```

The quaternion norm is defined as the square root of the sum of the quaternion parts squared. Calculate the quaternion norm explicitly to verify the result of the `norm` function.

```
[a,b,c,d] = parts(quat);
sqrt(a^2+b^2+c^2+d^2)
```

```
ans =
5.4772
```

Input Arguments

quat — Quaternion

`quaternion` object | array of `quaternion` objects

Quaternion for which to calculate the norm, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

N — Quaternion norm

real scalar | array of real numbers

Quaternion norm, returned as a real scalar or an array of real numbers of the same size as the `quat` argument. Elements of `N` are of the same data type as the underlying data type of `quat`.

Data Types: `single` | `double`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`normalize` | `parts` | `conj`

Objects

`quaternion`

normalize

Quaternion normalization

Syntax

```
quatNormalized = normalize(quat)
```

Description

`quatNormalized = normalize(quat)` normalizes the quaternion.

Given a quaternion of the form $Q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, the normalized quaternion is defined as $Q/\sqrt{a^2 + b^2 + c^2 + d^2}$.

Examples

Normalize Elements of Quaternion Vector

Quaternions can represent rotations when normalized. You can use `normalize` to normalize a scalar, elements of a matrix, or elements of a multi-dimensional array of quaternions. Create a column vector of quaternions, then normalize them.

```
quatArray = quaternion([1,2,3,4; ...
                        2,3,4,1; ...
                        3,4,1,2]);
quatArrayNormalized = normalize(quatArray)
```

```
quatArrayNormalized = 3×1 quaternion array
     0.18257 + 0.36515i + 0.54772j +  0.7303k
     0.36515 + 0.54772i +  0.7303j + 0.18257k
     0.54772 +  0.7303i + 0.18257j + 0.36515k
```

Input Arguments

quat — Quaternion to normalize

`quaternion` object | array of `quaternion` objects

Quaternion to normalize, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

quatNormalized — Normalized quaternion

`quaternion` object | array of `quaternion` objects

Normalized quaternion, returned as a `quaternion` object or an array of `quaternion` objects of the same size as the `quat` argument.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`norm` | `.*,times` | `conj`

Objects

`quaternion`

ones

Create quaternion array with real parts set to one and imaginary parts set to zero

Syntax

```
quatOnes = ones("quaternion")
quatOnes = ones(n,"quaternion")
quatOnes = ones(sz,"quaternion")
quatOnes = ones(sz1,...,szN,"quaternion")

quatOnes = ones( ___ ,"like",prototype,"quaternion")
```

Description

`quatOnes = ones("quaternion")` returns a scalar quaternion with the real part set to 1 and the imaginary parts set to 0.

Given a quaternion of the form $Q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, a quaternion one is defined as $Q = 1 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}$.

`quatOnes = ones(n,"quaternion")` returns an `n`-by-`n` quaternion matrix with the real parts set to 1 and the imaginary parts set to 0.

`quatOnes = ones(sz,"quaternion")` returns an array of quaternion ones where the size vector, `sz`, defines `size(qOnes)`.

Example: `ones([1,4,2],"quaternion")` returns a 1-by-4-by-2 array of quaternions with the real parts set to `1` and the imaginary parts set to `0`.

`quatOnes = ones(sz1,...,szN,"quaternion")` returns a `sz1`-by-...-by-`szN` array of ones where `sz1,…,szN` indicates the size of each dimension.

`quatOnes = ones( ___ ,"like",prototype,"quaternion")` specifies the underlying class of the returned quaternion array to be the same as the underlying class of the quaternion prototype.

Examples

Quaternion Scalar One

Create a quaternion scalar one.

```
quatOnes = ones("quaternion")
```

```
quatOnes = quaternion
     1 + 0i + 0j + 0k
```

Square Matrix of Quaternion Ones

Create an n-by-n matrix of quaternion ones.

```
n = 3;
quatOnes = ones(n,"quaternion")
```

```
quatOnes = 3×3 quaternion array
     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k
```

Multidimensional Array of Quaternion Ones

Create a multidimensional array of quaternion ones by defining array dimensions in order. In this example, you create a 3-by-1-by-2 array. You can specify dimensions using a row vector or comma-separated integers. Specify the dimensions using a row vector and display the results:

```
dims = [3,1,2];
quatOnesSyntax1 = ones(dims,"quaternion")
```

```
quatOnesSyntax1 = 3×1×2 quaternion array
quatOnesSyntax1(:,:,1) =

     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k


quatOnesSyntax1(:,:,2) =

     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k
```

Specify the dimensions using comma-separated integers, and then verify the equivalency of the two syntaxes:

```
quatOnesSyntax2 = ones(3,1,2,"quaternion");
isequal(quatOnesSyntax1,quatOnesSyntax2)
```

```
ans = logical
   1
```

Underlying Class of Quaternion Ones

A quaternion is a four-part hyper-complex number used in three-dimensional rotations and orientations. You can specify the underlying data type of the parts as `single` or `double`. The default is `double`.

Create a quaternion array of ones with the underlying data type set to `single`.

```
quatOnes = ones(2,"like",single(1),"quaternion")
```

```
quatOnes = 2×2 quaternion array
     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k
     1 + 0i + 0j + 0k     1 + 0i + 0j + 0k
```

Verify the underlying class using the `classUnderlying` function.

```
classUnderlying(quatOnes)
```

```
ans =
'single'
```

Input Arguments

`n` — Size of square quaternion matrix

integer value

Size of square quaternion matrix, specified as an integer value.

If `n` is zero or negative, then `quatOnes` is returned as an empty matrix.

Example: `ones(4,"quaternion")` returns a 4-by-4 matrix of quaternions with the real parts set to `1` and the imaginary parts set to `0`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`sz` — Output size

row vector of integer values

Output size, specified as a row vector of integer values. Each element of `sz` indicates the size of the corresponding dimension in `quatOnes`. If the size of any dimension is `0` or negative, then `quatOnes` is returned as an empty array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`prototype` — Quaternion prototype

`quaternion` object

Quaternion prototype, specified as a `quaternion` object.

Example: `ones(2,"like",quat,"quaternion")` returns a 2-by-2 matrix of quaternions with the same underlying class as the prototype quaternion, `quat`.

`sz1,...,szN` — Size of each dimension

two or more integer values

Size of each dimension, specified as two or more comma-separated integers. If the size of any dimension is `0` or negative, then `quatOnes` is returned as an empty array.

Example: `ones(2,3,"quaternion")` returns a 2-by-3 matrix of quaternions with the real parts set to `1` and the imaginary parts set to `0`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`quatOnes` — Quaternion ones
`quaternion` object | array of `quaternion` objects

Quaternion ones, returned as a `quaternion` object or an array of `quaternion` objects.

Given a quaternion of the form $Q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, a quaternion one is defined as $Q = 1 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}$.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`zeros`

Objects
`quaternion`

parts

Extract quaternion parts

Syntax

```
[a,b,c,d] = parts(quat)
```

Description

`[a,b,c,d] = parts(quat)` returns the parts of the quaternion array as arrays, each the same size as `quat`.

Examples

Convert Quaternion to Matrix of Quaternion Parts

Convert a quaternion representation to parts using the `parts` function.

Create a two-element column vector of quaternions by specifying the parts.

```
quat = quaternion([1:4;5:8])
```

```
quat = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

Recover the parts from the quaternion matrix using the `parts` function. The parts are returned as separate output arguments, each the same size as the input 2-by-1 column vector of quaternions.

```
[qA,qB,qC,qD] = parts(quat)
```

```
qA = 2×1

     1
     5


qB = 2×1

     2
     6


qC = 2×1

     3
     7


qD = 2×1
```

```
     4
     8
```

Input Arguments

`quat` — Quaternion

`quaternion` object | array of `quaternion` objects

Quaternion, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`[a,b,c,d]` — Quaternion parts

numeric array

Quaternion parts, returned as four comma-separated numeric arrays: `a`, `b`, `c`, and `d`. Each part is the same size as `quat`.

Data Types: `single` | `double`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`classUnderlying` | `compact`

Objects

`quaternion`

power, .^

Element-wise quaternion power

Syntax

```
C = A.^b
```

Description

`C = A.^b` raises each element of `A` to the corresponding power in `b`.

Examples

Raise a Quaternion to a Real Scalar Power

Create a quaternion and raise it to a real scalar power.

```
A = quaternion(1,2,3,4)
```

```
A = quaternion
     1 + 2i + 3j + 4k
```

```
b = 3;
C = A.^b
```

```
C = quaternion
     -86 -  52i -  78j - 104k
```

Raise a Quaternion Array to Powers from a Multidimensional Array

Create a 2-by-1 quaternion array and raise it to powers from a 2-D array.

```
A = quaternion([1:4;5:8])
```

```
A = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

```
b = [1 0 2; 3 2 1]
```

```
b = 2×3

     1     0     2
     3     2     1
```

```
C = A.^b
```

```
C = 2×3 quaternion array
        1 +    2i +    3j +    4k        1 +    0i +    0j +    0k      -28 +    4i +    6j +
    -2110 -  444i -  518j -  592k     -124 +   60i +   70j +   80k        5 +    6i +    7j +
```

Input Arguments

A — Base

`quaternion` object | array of `quaternion` objects

Base, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

`A` and `b` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

b — Exponent

real scalar | array of real numbers

Exponent, specified as a real scalar or an array of real numbers.

`A` and `b` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

Data Types: `single` | `double`

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Each element of quaternion `A` raised to the corresponding power in `b`, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

The polar representation of a quaternion $A = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$ is given by

$$A = \|A\|(\cos\theta + \widehat{u}\sin\theta)$$

where θ is the angle of rotation, and $\hat{u}$ is the unit quaternion.

Quaternion A raised by a real exponent b is given by

$$P = A.\hat{}\,b = \|A\|^{b}(\cos(b\theta) + \widehat{u}\sin(b\theta))$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`log` | `exp`

Objects
`quaternion`

prod

Product of quaternion array

Syntax

```
quatProd = prod(quat)
quatProd = prod(quat,dim)
```

Description

`quatProd = prod(quat)` returns the quaternion product of the elements of the array.

`quatProd = prod(quat,dim)` calculates the quaternion product along dimension `dim`.

Examples

Product of Quaternions in Each Column

Create a 3-by-3 array whose elements correspond to their linear indices.

```
A = reshape(quaternion(randn(9,4)),3,3)
```

```
A = 3×3 quaternion array
      0.53767 +   2.7694i +    1.409j -  0.30344k      0.86217 +   0.7254i -   1.2075j +   0.888
       1.8339 -   1.3499i +   1.4172j +  0.29387k      0.31877 - 0.063055i +  0.71724j -   1.147
      -2.2588 +   3.0349i +   0.6715j -  0.78728k      -1.3077 +  0.71474i +   1.6302j -   1.068
```

Find the product of the quaternions in each column. The length of the first dimension is `1`, and the length of the second dimension matches `size(A,2)`.

```
B = prod(A)
```

```
B = 1×3 quaternion array
     -19.837 -  9.1521i +  15.813j -  19.918k     -5.4708 - 0.28535i +   3.077j -  1.2295k      -
```

Product of Specified Dimension of Quaternion Array

You can specify which dimension of a quaternion array to take the product of.

Create a 2-by-2-by-2 quaternion array.

```
A = reshape(quaternion(randn(8,4)),2,2,2);
```

Find the product of the elements in each page of the array. The length of the first dimension matches `size(A,1)`, the length of the second dimension matches `size(A,2)`, and the length of the third dimension is `1`.

```
dim = 3;
B = prod(A,dim)

B = 2×2 quaternion array
     -2.4847 +  1.1659i - 0.37547j +  2.8068k     0.28786 - 0.29876i - 0.51231j -  4.2972k
     0.38986 -  3.6606i -  2.0474j -   6.047k      -1.741 - 0.26782i +  5.4346j +  4.1452k
```

Input Arguments

`quat` — Quaternions

array of `quaternion` objects

Quaternions, specified as an array of `quaternion` objects of any dimensionality.

Example: `qProd = prod(quat)` calculates the quaternion product along the first non-singleton dimension of `quat`.

`dim` — Dimension

positive integer

Dimension along which to calculate the quaternion product, specified as a positive integer. If `dim` is not specified, `prod` operates along the first non-singleton dimension of `quat`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`quatProd` — Quaternion product

array of `quaternion` objects

Quaternion product, returned as an array of `quaternion` objects. `quatProd` has one less non-singleton dimension than `quat`.

For example, if `quat` is a 2-by-2-by-5 array,

- `prod(quat,1)` returns a 1-by-2-by-5 array.
- `prod(quat,2)` returns a 2-by-1-by-5 array.
- `prod(quat,3)` returns a 2-by-2 array.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`mtimes, *` | `times, .*`

Objects
`quaternion`

rdivide, ./

Element-wise quaternion right division

Syntax

```
C = A./B
```

Description

`C = A./B` performs quaternion element-wise division by dividing each element of quaternion `A` by the corresponding element of quaternion `B`.

Examples

Divide a Quaternion Array by a Real Scalar

Create a 2-by-1 quaternion array, and divide it element-by-element by a real scalar.

```
A = quaternion([1:4;5:8])
```

```
A = 2×1 quaternion array
     1 + 2i + 3j + 4k
     5 + 6i + 7j + 8k
```

```
B = 2;
C = A./B
```

```
C = 2×1 quaternion array
     0.5 +   1i + 1.5j +   2k
     2.5 +   3i + 3.5j +   4k
```

Divide a Quaternion Array by Another Quaternion Array

Create a 2-by-2 quaternion array, and divide it element-by-element by another 2-by-2 quaternion array.

```
q1 = quaternion(magic(4));
A = reshape(q1,2,2)
```

```
A = 2×2 quaternion array
     16 +  2i +  3j + 13k      9 +  7i +  6j + 12k
      5 + 11i + 10j +  8k      4 + 14i + 15j +  1k
```

```
q2 = quaternion([1:4;3:6;2:5;4:7]);
B = reshape(q2,2,2)
```

```
B = 2×2 quaternion array
     1 + 2i + 3j + 4k     2 + 3i + 4j + 5k
     3 + 4i + 5j + 6k     4 + 5i + 6j + 7k


C = A./B

C = 2×2 quaternion array
          2.7 -      0.1i -      2.1j -      1.7k       2.2778 + 0.092593i -  0.46296j -  0.5740
       1.8256 - 0.081395i +  0.45349j -  0.24419k       1.4524 -      0.5i +   1.0238j -   0.261
```

Input Arguments

A — Dividend

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Dividend, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

B — Divisor

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Divisor, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of the dimensions is 1.

Output Arguments

C — Result

`quaternion` object | array of `quaternion` objects

Result of quaternion division, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Division

Given a quaternion $A = a_1 + a_2\mathrm{i} + a_3\mathrm{j} + a_4\mathrm{k}$ and a real scalar p,

$$C = A./p = \frac{a_1}{p} + \frac{a_2}{p}\mathrm{i} + \frac{a_3}{p}\mathrm{j} + \frac{a_4}{p}\mathrm{k}$$

Note For a real scalar p, $A./p = p.\backslash A$.

Quaternion Division by a Quaternion Scalar

Given two quaternions A and B of compatible sizes,

$$C = A./B = A.*B^{-1} = A.*\left(\frac{conj(B)}{norm(B)^2}\right)$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`.\,ldivide` | `mrdivide, /` | `.*,times` | `norm` | `conj`

Objects

`quaternion`

randrot

Uniformly distributed random rotations

Syntax

```
R = randrot
R = randrot(m)
R = randrot(m1,...,mN)
R = randrot([m1,...,mN])
```

Description

`R = randrot` returns a unit quaternion drawn from a uniform distribution of random rotations.

`R = randrot(m)` returns an `m`-by-`m` matrix of unit quaternions drawn from a uniform distribution of random rotations.

`R = randrot(m1,...,mN)` returns an `m1`-by-...-by-`mN` array of random unit quaternions, where `m1`, ..., `mN` indicate the size of each dimension. For example, `randrot(3,4)` returns a 3-by-4 matrix of random unit quaternions.

`R = randrot([m1,...,mN])` returns an `m1`-by-...-by-`mN` array of random unit quaternions, where `m1`,..., `mN` indicate the size of each dimension. For example, `randrot([3,4])` returns a 3-by-4 matrix of random unit quaternions.

Examples

Matrix of Random Rotations

Generate a 3-by-3 matrix of uniformly distributed random rotations.

```
r = randrot(3)
```

```
r = 3×3 quaternion array
      0.17446 +  0.59506i -  0.73295j +  0.27976k      0.69704 - 0.060589i +  0.68679j -  0.1969
      0.21908 -  0.89875i -    0.298j +  0.23548k    -0.049744 +  0.59691i +  0.56459j +  0.5678
       0.6375 +  0.49338i -  0.24049j +  0.54068k       0.2979 -  0.53568i +  0.31819j +  0.7232
```

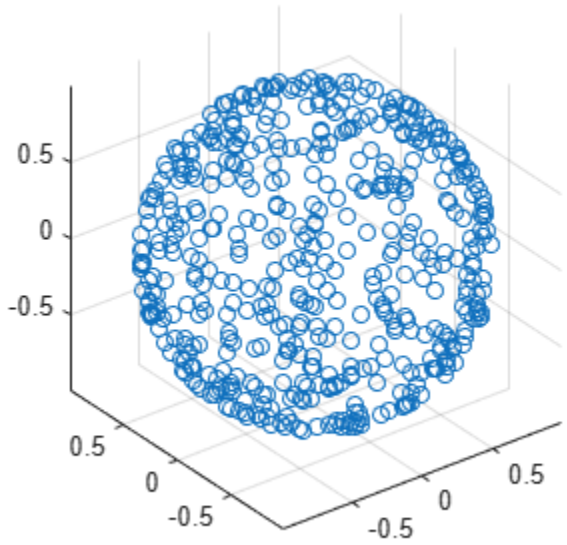
Create Uniform Distribution of Random Rotations

Create a vector of 500 random quaternions. Use `rotatepoint` to visualize the distribution of the random rotations applied to point (1, 0, 0).

```
q = randrot(500,1);

pt = rotatepoint(q, [1 0 0]);
```

```
figure
scatter3(pt(:,1), pt(:,2), pt(:,3))
axis equal
```



Input Arguments

`m` — Size of square matrix

integer

Size of square quaternion matrix, specified as an integer value. If `m` is `0` or negative, then `R` is returned as an empty matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`m1,...,mN` — Size of each dimension

two or more integer values

Size of each dimension, specified as two or more comma-separated integer values. If the size of any dimension is `0` or negative, then `R` is returned as an empty array.

Example: `randrot(2,3)` returns a 2-by-3 matrix of random quaternions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`[m1,...,mN]` — Vector of size of each dimension

row vector of integer values

Vector of size of each dimension, specified as a row vector of two or more integer values. If the size of any dimension is `0` or negative, then `R` is returned as an empty array.

Example: `randrot([2,3])` returns a 2-by-3 matrix of random quaternions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

R — Random quaternions
`quaternion` object | array of `quaternion` objects

Random quaternions, returned as a `quaternion` object or an array of `quaternion` objects.

Version History

Introduced in R2023b

References

[1] Shoemake, K. "Uniform Random Rotations." *Graphics Gems III* (K. David, ed.). New York: Academic Press, 1992.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`quaternion`

rotateframe

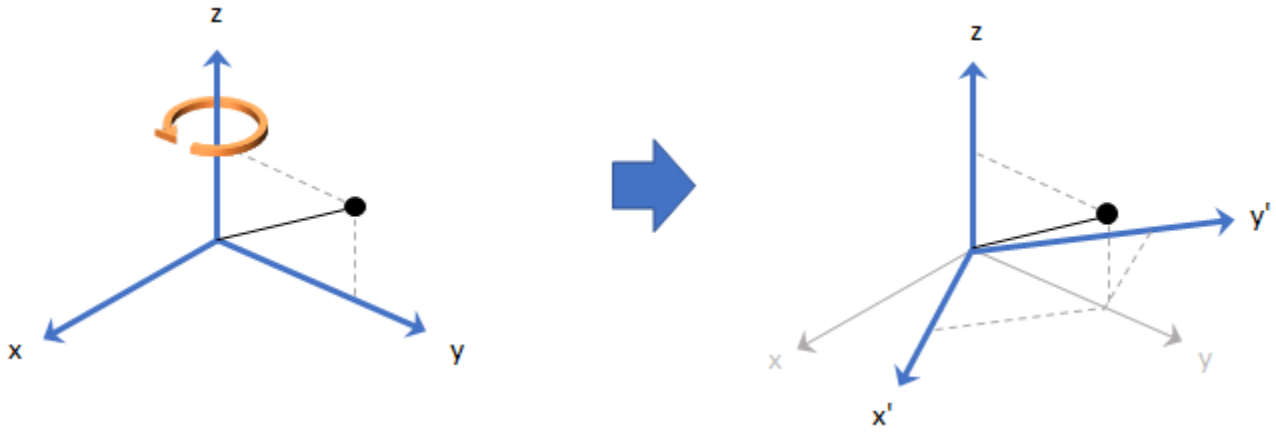
Quaternion frame rotation

Syntax

```
rotationResult = rotateframe(quat,cartesianPoints)
```

Description

`rotationResult = rotateframe(quat,cartesianPoints)` rotates the frame of reference for the Cartesian points using the quaternion, `quat`. The elements of the quaternion are normalized before use in the rotation.

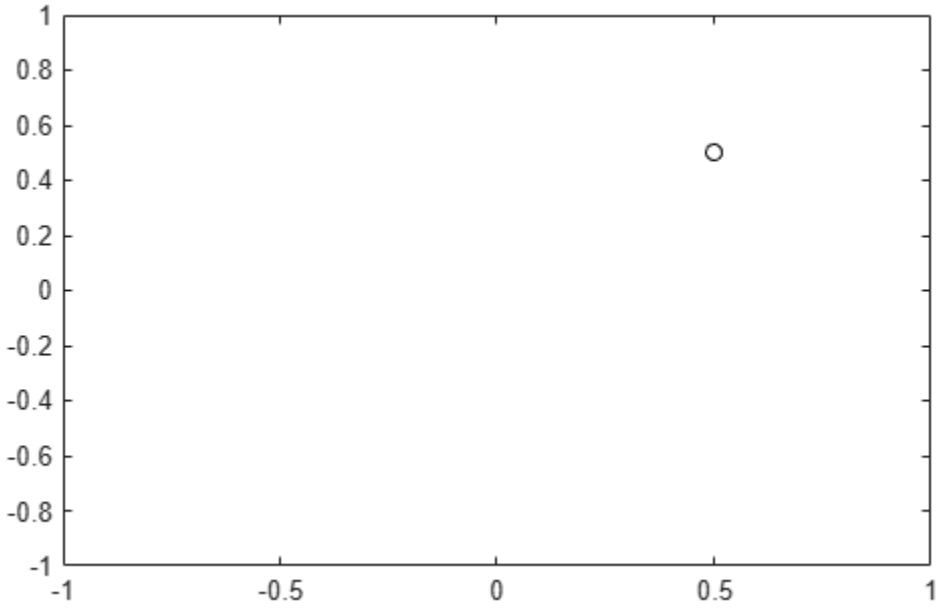


Examples

Rotate Frame Using Quaternion Vector

Define a point in three dimensions. The coordinates of a point are always specified in the order *x*, *y*, and *z*. For convenient visualization, define the point on the *x*-*y* plane.

```
x = 0.5;
y = 0.5;
z = 0;
plot(x,y,"ko")
hold on
axis([-1 1 -1 1])
```



Create a quaternion vector specifying two separate rotations, one to rotate the frame 45 degrees and another to rotate the point -90 degrees about the *z*-axis. Use `rotateframe` to perform the rotations.

```
quat = quaternion([0,0,pi/4; ...
                   0,0,-pi/2],"euler","XYZ","frame");

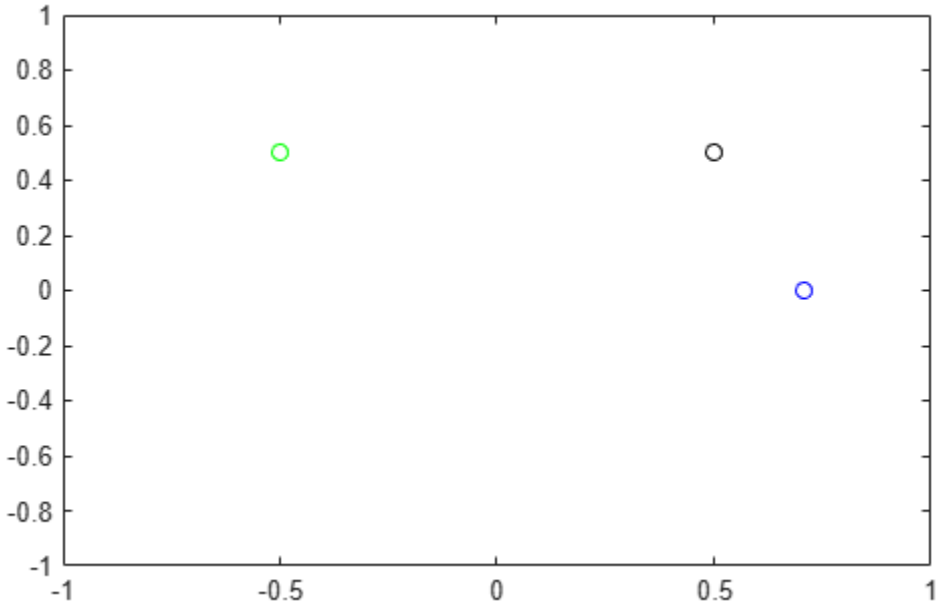
rereferencedPoint = rotateframe(quat,[x,y,z])
```

```
rereferencedPoint = 2×3

    0.7071   -0.0000         0
   -0.5000    0.5000         0
```

Plot the re-referenced points.

```
plot(rereferencedPoint(1,1),rereferencedPoint(1,2),"bo")
plot(rereferencedPoint(2,1),rereferencedPoint(2,2),"go")
```



Re-reference Group of Points using Quaternion

Define two points in three-dimensional space. Define a quaternion to re-reference the points by first rotating the reference frame about the *z*-axis 30 degrees and then about the new *y*-axis 45 degrees.

```
a = [1,0,0];
b = [0,1,0];
quat = quaternion([30,45,0],"eulerd","ZYX","point");
```

Use `rotateframe` to reference both points using the quaternion rotation operator. Display the result.

```
rP = rotateframe(quat,[a;b])
```

```
rP = 2×3

    0.6124   -0.3536    0.7071
    0.5000    0.8660   -0.0000
```

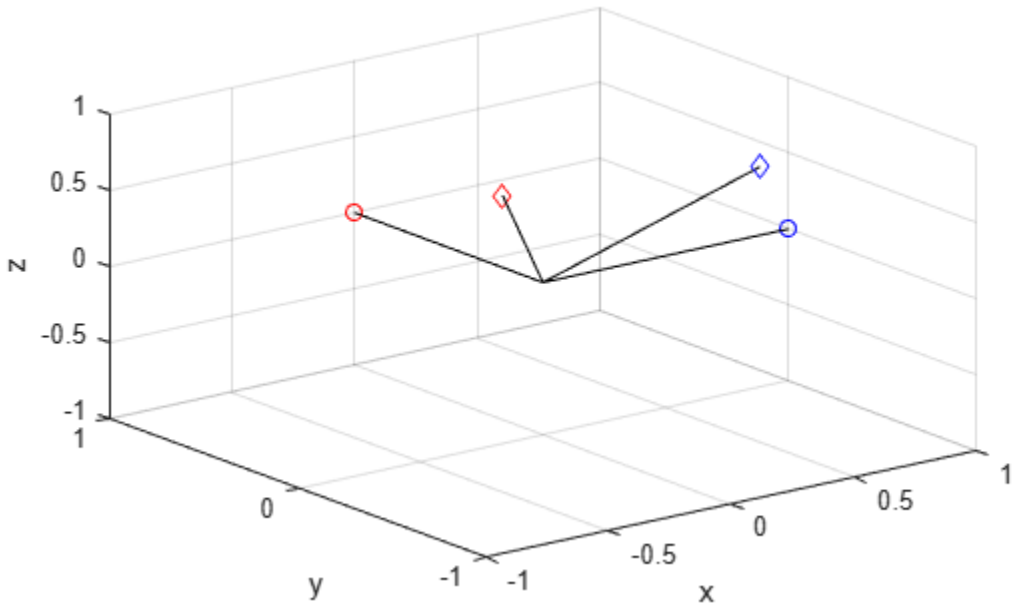
Visualize the original orientation and the rotated orientation of the points. Draw lines from the origin to each of the points for visualization purposes.

```
plot3(a(1),a(2),a(3),"bo");

hold on
grid on
axis([-1 1 -1 1 -1 1])
xlabel("x")
ylabel("y")
zlabel("z")
```

```
plot3(b(1),b(2),b(3),"ro");
plot3(rP(1,1),rP(1,2),rP(1,3),"bd")
plot3(rP(2,1),rP(2,2),rP(2,3),"rd")

plot3([0;rP(1,1)],[0;rP(1,2)],[0;rP(1,3)],"k")
plot3([0;rP(2,1)],[0;rP(2,2)],[0;rP(2,3)],"k")
plot3([0;a(1)],[0;a(2)],[0;a(3)],"k")
plot3([0;b(1)],[0;b(2)],[0;b(3)],"k")
```



Input Arguments

quat — Quaternion that defines rotation

`quaternion` object | vector of `quaternion` objects

Quaternion that defines rotation, specified as a `quaternion` object or a vector of `quaternion` objects. `quat` and `cartesianPoints` must have compatible sizes:

- `length(quat) == size(cartesianPoints,1)`, or
- `length(quat) == 1`, or
- `size(cartesianPoints,1) == 1`

cartesianPoints — Three-dimensional Cartesian points

1-by-3 numeric vector | *N*-by-3 numeric matrix

Three-dimensional Cartesian points, specified as a 1-by-3 numeric vector representing a single point or an *N*-by-3 numeric matrix representing *N* points. `quat` and `cartesianPoints` must have compatible sizes:

- `length(quat) == size(cartesianPoints,1)` or

- `length(quat) == 1`, or
- `size(cartesianPoints,1) == 1`

Data Types: `single` | `double`

Output Arguments

`rotationResult` — Re-referenced Cartesian points

1-by-3 numeric vector | numeric matrix

Cartesian points defined in reference to rotated reference frame, returned as a 1-by-3 numeric vector or a numeric matrix.

`rotationResult` is a 1-by-3 vector when `quat` is a scalar `quaternion` object and `cartesianPoints` is a 1-by-3 vector representing a single point. Otherwise, `rotationResult` is an *M*-by-3 matrix, where *M* is the maximum of `length(quat)` and `size(cartesianPoints,1)`.

Data Types: `single` | `double`

Algorithms

Quaternion frame rotation re-references a point specified in $\mathbf{R}^3$ by rotating the original frame of reference according to a specified quaternion:

$$L_q(u) = q^*uq$$

where q is the quaternion, * represents conjugation, and u is the point to rotate, specified as a quaternion.

For convenience, the `rotateframe` function takes a point in $\mathbf{R}^3$ and returns a point in $\mathbf{R}^3$. Given a function call with some arbitrary quaternion, $q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, and arbitrary coordinate, $[x,y,z]$,

```
point = [x,y,z];
rereferencedPoint = rotateframe(q,point)
```

the `rotateframe` function performs the following operations:

1. Converts point $[x,y,z]$ to a quaternion:

   $$u_q = 0 + xi + yj + zk$$

2. Normalizes the quaternion, q:

   $$q_n = \frac{q}{\sqrt{a^2 + b^2 + c^2 + d^2}}$$

3. Applies the rotation:

   $$v_q = q^*u_qq$$

4. Converts the quaternion output, v_q, back to $\mathbf{R}^3$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`rotatepoint`

Objects

`quaternion`

rotatepoint

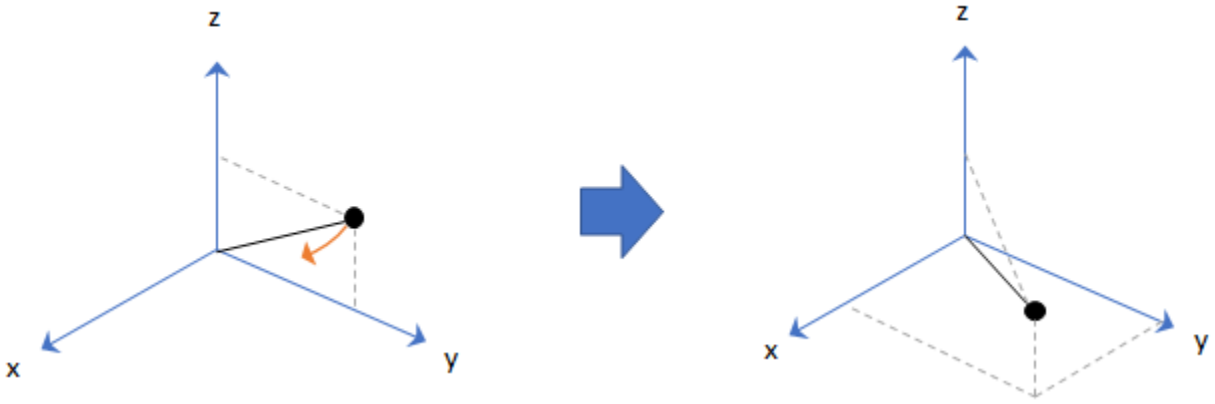
Quaternion point rotation

Syntax

```
rotationResult = rotatepoint(quat,cartesianPoints)
```

Description

`rotationResult = rotatepoint(quat,cartesianPoints)` rotates the Cartesian points using the quaternion, `quat`. The elements of the quaternion are normalized before use in the rotation.



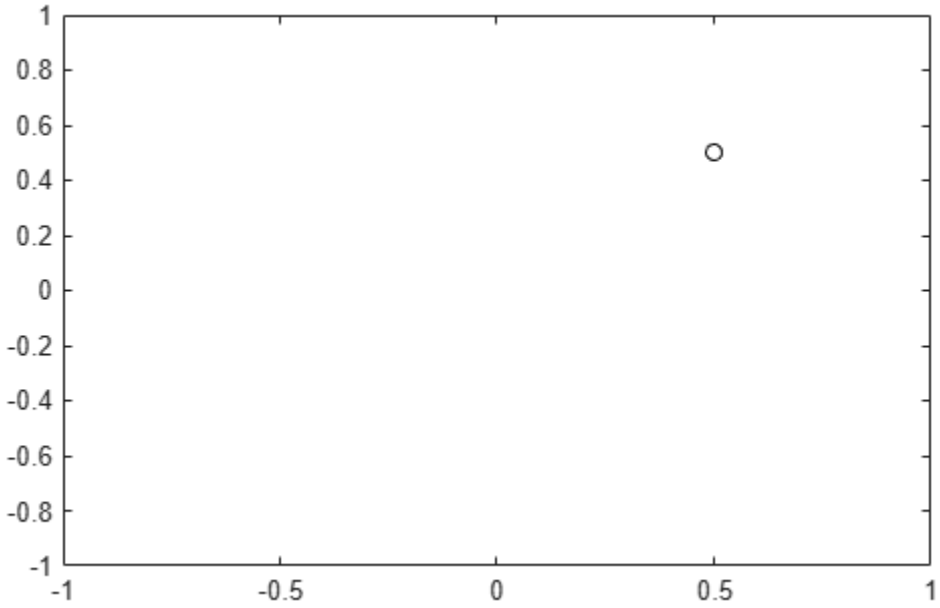
Examples

Rotate Point Using Quaternion Vector

Define a point in three dimensions. The coordinates of a point are always specified in order *x*, *y*, *z*. For convenient visualization, define the point on the *x*-*y* plane.

```
x = 0.5;
y = 0.5;
z = 0;

plot(x,y,"ko")
hold on
axis([-1 1 -1 1])
```



Create a quaternion vector specifying two separate rotations, one to rotate the point 45 and another to rotate the point -90 degrees about the *z*-axis. Use `rotatepoint` to perform the rotation.

```
quat = quaternion([0,0,pi/4; ...
                   0,0,-pi/2],"euler","XYZ","point");

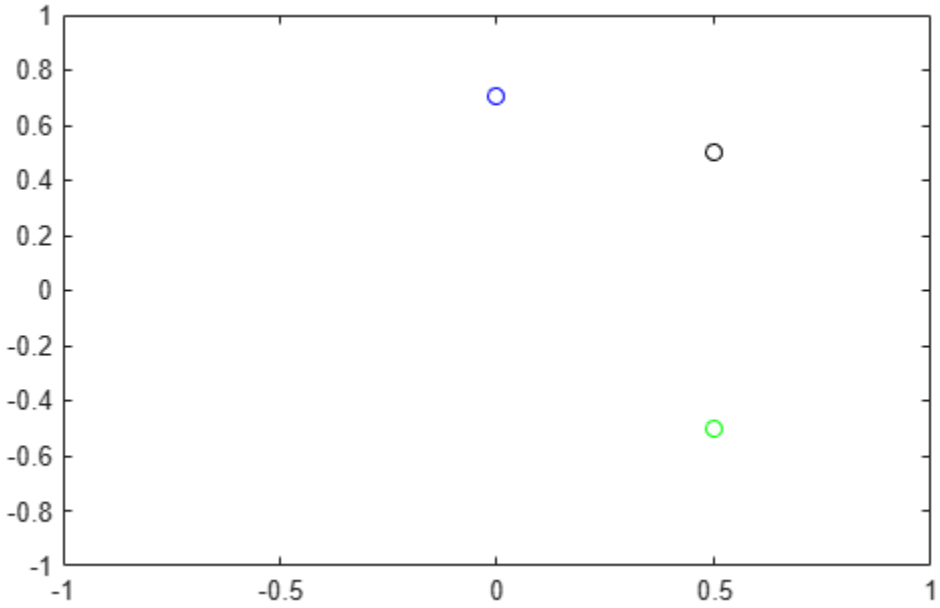
rotatedPoint = rotatepoint(quat,[x,y,z])
```

```
rotatedPoint = 2×3

   -0.0000    0.7071         0
    0.5000   -0.5000         0
```

Plot the rotated points.

```
plot(rotatedPoint(1,1),rotatedPoint(1,2),"bo")
plot(rotatedPoint(2,1),rotatedPoint(2,2),"go")
```



Rotate Group of Points Using Quaternion

Define two points in three-dimensional space. Define a quaternion to rotate the point by first rotating about the *z*-axis 30 degrees and then about the new *y*-axis 45 degrees.

```
a = [1,0,0];
b = [0,1,0];
quat = quaternion([30,45,0],"eulerd","ZYX","point");
```

Use `rotatepoint` to rotate both points using the quaternion rotation operator. Display the result.

```
rP = rotatepoint(quat,[a;b])
```

```
rP = 2×3

    0.6124    0.5000   -0.6124
   -0.3536    0.8660    0.3536
```

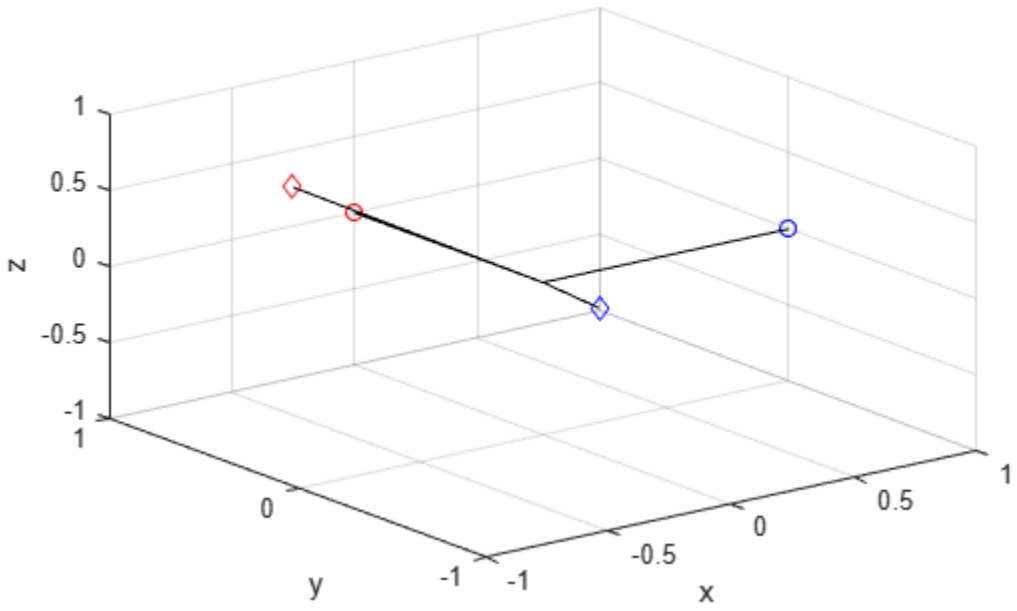
Visualize the original orientation and the rotated orientation of the points. Draw lines from the origin to each of the points for visualization purposes.

```
plot3(a(1),a(2),a(3),"bo");

hold on
grid on
axis([-1 1 -1 1 -1 1])
xlabel("x")
ylabel("y")
zlabel("z")
```

```
plot3(b(1),b(2),b(3),"ro")
plot3(rP(1,1),rP(1,2),rP(1,3),"bd")
plot3(rP(2,1),rP(2,2),rP(2,3),"rd")

plot3([0;rP(1,1)],[0;rP(1,2)],[0;rP(1,3)],"k")
plot3([0;rP(2,1)],[0;rP(2,2)],[0;rP(2,3)],"k")
plot3([0;a(1)],[0;a(2)],[0;a(3)],"k")
plot3([0;b(1)],[0;b(2)],[0;b(3)],"k")
```



Input Arguments

quat — Quaternion that defines rotation

`quaternion` object | vector of `quaternion` objects

Quaternion that defines rotation, specified as a `quaternion` object or a vector of `quaternion` objects. `quat` and `cartesianPoints` must have compatible sizes:

- `length(quat) == size(cartesianPoints,1)`, or
- `length(quat) == 1`, or
- `size(cartesianPoints,1) == 1`

cartesianPoints — Three-dimensional Cartesian points

1-by-3 numeric vector | *N*-by-3 numeric matrix

Three-dimensional Cartesian points, specified as a 1-by-3 numeric vector representing a single point or an *N*-by-3 numeric matrix representing *N* points. `quat` and `cartesianPoints` must have compatible sizes:

- `length(quat) == size(cartesianPoints,1)` or

- `length(quat) == 1`, or
- `size(cartesianPoints,1) == 1`

Data Types: `single` | `double`

Output Arguments

`rotationResult` — Repositioned Cartesian points

1-by-3 numeric vector | numeric matrix

Rotated Cartesian points defined using the quaternion rotation, returned as a 1-by-3 numeric vector or a numeric matrix.

`rotationResult` is a 1-by-3 vector when `quat` is a scalar `quaternion` object and `cartesianPoints` is a 1-by-3 vector representing a single point. Otherwise, `rotationResult` is an *M*-by-3 matrix, where *M* is the maximum of `length(quat)` and `size(cartesianPoints,1)`.

Data Types: `single` | `double`

Algorithms

Quaternion point rotation rotates a point specified in $\mathbf{R}^3$ according to a specified quaternion:

$$L_q(u) = quq^*$$

where q is the quaternion, * represents conjugation, and u is the point to rotate, specified as a quaternion.

For convenience, the `rotatepoint` function takes in a point in $\mathbf{R}^3$ and returns a point in $\mathbf{R}^3$. Given a function call with some arbitrary quaternion, $q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, and arbitrary coordinate, $[x,y,z]$, for example,

```
rereferencedPoint = rotatepoint(q,[x,y,z])
```

the `rotatepoint` function performs the following operations:

1. Converts point $[x,y,z]$ to a quaternion:

   $$u_q = 0 + xi + yj + zk$$

2. Normalizes the quaternion, q:

   $$q_n = \frac{q}{\sqrt{a^2 + b^2 + c^2 + d^2}}$$

3. Applies the rotation:

   $$v_q = qu_qq^*$$

4. Converts the quaternion output, v_q, back to $\mathbf{R}^3$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`rotateframe`

Objects

`quaternion`

rotmat

Convert quaternion to rotation matrix

Syntax

```
rotationMatrix = rotmat(quat,rotationType)
```

Description

`rotationMatrix = rotmat(quat,rotationType)` converts the quaternion, `quat`, to an equivalent rotation matrix representation.

Examples

Convert Quaternion to Rotation Matrix for Point Rotation

Define a quaternion for use in point rotation.

```
theta = 45;
gamma = 30;
quat = quaternion([0,theta,gamma],"eulerd","ZYX","point")
```

```
quat = quaternion
       0.8924 +  0.23912i +  0.36964j + 0.099046k
```

Convert the quaternion to a rotation matrix.

```
rotationMatrix = rotmat(quat,"point")
```

```
rotationMatrix = 3×3

    0.7071   -0.0000    0.7071
    0.3536    0.8660   -0.3536
   -0.6124    0.5000    0.6124
```

To verify the rotation matrix, directly create two rotation matrices corresponding to the rotations about the *y*- and *x*-axes. Multiply the rotation matrices and compare to the output of `rotmat`.

```
theta = 45;
gamma = 30;

ry = [cosd(theta)   0           sind(theta) ; ...
      0             1           0           ; ...
     -sind(theta)   0           cosd(theta)];

rx = [1             0           0           ;      ...
      0             cosd(gamma) -sind(gamma) ;     ...
      0             sind(gamma) cosd(gamma)];

rotationMatrixVerification = rx*ry
```

```
rotationMatrixVerification = 3×3

    0.7071         0    0.7071
    0.3536    0.8660   -0.3536
   -0.6124    0.5000    0.6124
```

Convert Quaternion to Rotation Matrix for Frame Rotation

Define a quaternion for use in frame rotation.

```
theta = 45;
gamma = 30;
quat = quaternion([0,theta,gamma],"eulerd","ZYX","frame")
```

```
quat = quaternion
       0.8924 +  0.23912i +  0.36964j - 0.099046k
```

Convert the quaternion to a rotation matrix.

```
rotationMatrix = rotmat(quat,"frame")
```

```
rotationMatrix = 3×3

    0.7071   -0.0000   -0.7071
    0.3536    0.8660    0.3536
    0.6124   -0.5000    0.6124
```

To verify the rotation matrix, directly create two rotation matrices corresponding to the rotations about the *y*- and *x*-axes. Multiply the rotation matrices and compare to the output of `rotmat`.

```
theta = 45;
gamma = 30;

ry = [cosd(theta)   0           -sind(theta) ; ...
      0             1           0            ; ...
     sind(theta)   0           cosd(theta)];

rx = [1             0           0            ;      ...
      0             cosd(gamma) sind(gamma) ;     ...
      0             -sind(gamma) cosd(gamma)];

rotationMatrixVerification = rx*ry
```

```
rotationMatrixVerification = 3×3

    0.7071         0   -0.7071
    0.3536    0.8660    0.3536
    0.6124   -0.5000    0.6124
```

Convert Quaternion Vector to Rotation Matrices

Create a 3-by-1 normalized quaternion vector.

```
qVec = normalize(quaternion(randn(3,4)));
```

Convert the quaternion array to rotation matrices. The pages of `rotmatArray` correspond to the linear index of `qVec`.

```
rotmatArray = rotmat(qVec,"frame");
```

Assume `qVec` and `rotmatArray` correspond to a sequence of rotations. Combine the quaternion rotations into a single representation, then apply the quaternion rotation to arbitrarily initialized Cartesian points.

```
loc = normalize(randn(1,3));
quat = prod(qVec);
rotateframe(quat,loc)
```

```
ans = 1×3

    0.9524    0.5297    0.9013
```

Combine the rotation matrices into a single representation, then apply the rotation matrix to the same initial Cartesian points. Verify the quaternion rotation and rotation matrix result in the same orientation.

```
totalRotMat = eye(3);
for i = 1:size(rotmatArray,3)
    totalRotMat = rotmatArray(:,:,i)*totalRotMat;
end
totalRotMat*loc'
```

```
ans = 3×1

    0.9524
    0.5297
    0.9013
```

Input Arguments

`quat` — Quaternion to convert

`quaternion` object | array of `quaternion` objects

Quaternion to convert, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

`rotationType` — Type or rotation

`"frame"` | `"point"`

Type of rotation represented by the `rotationMatrix` output, specified as `"frame"` or `"point"`.

Data Types: `char` | `string`

Output Arguments

`rotationMatrix` — Rotation matrix representation
3-by-3 numeric matrix | 3-by-3-by-*N* numeric array

Rotation matrix representation, returned as a 3-by-3 numeric matrix or 3-by-3-by-*N* numeric array.

- If `quat` is a scalar, `rotationMatrix` is returned as a 3-by-3 matrix.
- If `quat` is non-scalar, `rotationMatrix` is returned as a 3-by-3-by-*N* array, where `rotationMatrix(:,:,i)` is the rotation matrix corresponding to `quat(i)`.

The data type of the rotation matrix is the same as the underlying data type of `quat`.

Data Types: `single` | `double`

Algorithms

Given a quaternion of the form

$$q = a + bi + cj + dk\,,$$

the equivalent rotation matrix for frame rotation is defined as

$$\begin{bmatrix} 2a^2 - 1 + 2b^2 & 2bc + 2ad & 2bd - 2ac \\ 2bc - 2ad & 2a^2 - 1 + 2c^2 & 2cd + 2ab \\ 2bd + 2ac & 2cd - 2ab & 2a^2 - 1 + 2d^2 \end{bmatrix}.$$

The equivalent rotation matrix for point rotation is the transpose of the frame rotation matrix:

$$\begin{bmatrix} 2a^2 - 1 + 2b^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & 2a^2 - 1 + 2c^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & 2a^2 - 1 + 2d^2 \end{bmatrix}.$$

Version History

Introduced in R2023b

References

[1] Kuipers, Jack B. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton, NJ: Princeton University Press, 2007.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`rotvec` | `rotvecd` | `euler` | `eulerd`

Objects
`quaternion`

rotvec

Convert quaternion to rotation vector (radians)

Syntax

```
rotationVector = rotvec(quat)
```

Description

`rotationVector = rotvec(quat)` converts the quaternion array, `quat`, to an *N*-by-3 matrix of equivalent rotation vectors in radians. The elements of `quat` are normalized before conversion.

Examples

Convert Quaternion to Rotation Vector in Radians

Convert a random quaternion scalar to a rotation vector in radians

```
quat = quaternion(randn(1,4));
rotvec(quat)
```

```
ans = 1×3

    1.6866   -2.0774    0.7929
```

Input Arguments

`quat` — Quaternion to convert

`quaternion` object | array of `quaternion` objects

Quaternion to convert, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`rotationVector` — Rotation vector (radians)

N-by-3 numeric matrix

Rotation vector representation, in radians, returned as an *N*-by-3 numeric matrix of rotation vectors, where *N* is the number of quaternions in the `quat` argument.

Each row represents the [*X Y Z*] angles of the rotation vectors. The *i*th row of `rotationVector` corresponds to the element `quat(i)`.

The data type of the rotation vector is the same as the underlying data type of `quat`.

Data Types: `single` | `double`

Algorithms

All rotations in 3-D can be represented by a three-element axis of rotation and a rotation angle, for a total of four elements. If the rotation axis is constrained to be unit length, the rotation angle can be distributed over the vector elements to reduce the representation to three elements.

Recall that a quaternion can be represented in axis-angle form

$$q = \cos\left(\theta/2\right) + \sin\left(\theta/2\right)(\mathrm{xi} + y\mathrm{j} + z\mathrm{k}),$$

where θ is the angle of rotation and $[x,y,z]$ represent the axis of rotation.

Given a quaternion of the form

$$q = a + bi + cj + dk\,,$$

you can solve for the rotation angle using the axis-angle form of quaternions:

$$\theta = 2\cos^{-1}(a)\,.$$

Assuming a normalized axis, you can rewrite the quaternion as a rotation vector without loss of information by distributing θ over the parts b, c, and d. The rotation vector representation of q is

$$q_{\mathrm{rv}} = \frac{\theta}{\sin\left(\theta/2\right)}[b, c, d]\,.$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`rotvecd` | `euler` | `eulerd`

Objects

`quaternion`

rotvecd

Convert quaternion to rotation vector (degrees)

Syntax

```
rotationVector = rotvecd(quat)
```

Description

`rotationVector = rotvecd(quat)` converts the quaternion array, `quat`, to an *N*-by-3 matrix of equivalent rotation vectors in degrees. The elements of `quat` are normalized before conversion.

Examples

Convert Quaternion to Rotation Vector in Degrees

Convert a random quaternion scalar to a rotation vector in degrees.

```
quat = quaternion(randn(1,4));
rotvecd(quat)
```

```
ans = 1×3

   96.6345 -119.0274   45.4312
```

Input Arguments

`quat` — Quaternion to convert

`quaternion` object | array of `quaternion` objects

Quaternion to convert, specified as a `quaternion` object or an array of `quaternion` objects of any dimensionality.

Output Arguments

`rotationVector` — Rotation vector representation (degrees)

N-by-3 numeric matrix

Rotation vector representation, in degrees, returned as an *N*-by-3 numeric matrix of rotation vectors, where *N* is the number of quaternions in the `quat` argument.

Each row represents the [*X Y Z*] angles of the rotation vectors in degrees. The *i*th row of `rotationVector` corresponds to the element `quat(i)`.

The data type of the rotation vector is the same as the underlying data type of `quat`.

Data Types: `single` | `double`

Algorithms

All rotations in 3-D can be represented by four elements: a three-element axis of rotation and a rotation angle. If the rotation axis is constrained to be unit length, the rotation angle can be distributed over the vector elements to reduce the representation to three elements.

Recall that a quaternion can be represented in axis-angle form

$$q = \cos\left(\theta/2\right) + \sin\left(\theta/2\right)(\mathrm{xi} + y\mathrm{j} + z\mathrm{k}),$$

where θ is the angle of rotation in degrees, and $[x,y,z]$ represent the axis of rotation.

Given a quaternion of the form

$$q = a + bi + cj + dk\,,$$

you can solve for the rotation angle using the axis-angle form of quaternions:

$$\theta = 2\cos^{-1}(a)\,.$$

Assuming a normalized axis, you can rewrite the quaternion as a rotation vector without loss of information by distributing θ over the parts b, c, and d. The rotation vector representation of q is

$$q_{\mathrm{rv}} = \frac{\theta}{\sin\left(\theta/2\right)}[b, c, d]\,.$$

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`rotvec` | `euler` | `eulerd`

Objects

`quaternion`

slerp

Spherical linear interpolation

Syntax

```
q0 = slerp(q1,q2,T)
q0 = slerp(q1,q2,T,method)
```

Description

`q0 = slerp(q1,q2,T)` spherically interpolates between `q1` and `q2` by the interpolation coefficient `T`. This syntax chooses the shorter interpolation path, between `q1` and `q2`.

`q0 = slerp(q1,q2,T,method)` interpolates between `q1` and `q2` using the specified interpolation method.

Examples

Interpolate Between Two Quaternions

Create two quaternions with the following interpretation:

1. `a` = 45 degree rotation around the *z*-axis
2. `c` = -45 degree rotation around the *z*-axis

```
a = quaternion([45,0,0],"eulerd","ZYX","frame");
c = quaternion([-45,0,0],"eulerd","ZYX","frame");
```

Call `slerp` with the quaternions `a` and `c` and specify an interpolation coefficient of 0.5.

```
interpolationCoefficient = 0.5;

b = slerp(a,c,interpolationCoefficient);
```

The output of `slerp`, `b`, represents an average rotation of `a` and `c`. To verify, convert `b` to Euler angles in degrees.

```
averageRotation = eulerd(b,"ZYX","frame")
```

```
averageRotation = 1×3

     0     0     0
```

The interpolation coefficient is specified as a normalized value between `0` and `1`, inclusive. An interpolation coefficient of `0` corresponds to the `a` quaternion, and an interpolation coefficient of `1` corresponds to the `c` quaternion. Call `slerp` with coefficients `0` and `1` to confirm.

```
b = slerp(a,c,[0,1]);
eulerd(b,"ZYX","frame")
```

```
ans = 2×3

   45.0000         0         0
  -45.0000         0         0
```

You can create smooth paths between quaternions by specifying arrays of equally spaced interpolation coefficients.

```
path = 0:0.1:1;

interpolatedQuaternions = slerp(a,c,path);
```

For quaternions that represent rotation only about a single axis, specifying interpolation coefficients as equally spaced results in quaternions equally spaced in Euler angles. Convert `interpolatedQuaternions` to Euler angles and verify that the difference between the angles in the path is constant.

```
k = eulerd(interpolatedQuaternions,"ZYX","frame");
abc = abs(diff(k))
```

```
abc = 10×3

    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
    9.0000         0         0
```

Alternatively, you can use the `dist` function to verify that the distance between the interpolated quaternions is consistent. The `dist` function returns angular distance in radians; convert to degrees for easy comparison.

```
def = rad2deg(dist(interpolatedQuaternions(2:end),interpolatedQuaternions(1:end-1)))
```

```
def = 1×10

    9.0000    9.0000    9.0000    9.0000    9.0000    9.0000    9.0000    9.0000    9.0000    9.0
```

SLERP Minimizes Great Circle Path

The SLERP algorithm interpolates along a great circle path connecting two quaternions. This example shows how the SLERP algorithm minimizes the great circle path.

Define four quaternions:

1. `q0` - quaternion indicating no rotation from the global frame

2. `q179` - quaternion indicating a 179 degree rotation about the *z*-axis
3. `q180` - quaternion indicating a 180 degree rotation about the *z*-axis
4. `q181` - quaternion indicating a 181 degree rotation about the *z*-axis

```
q0 = ones(1,"quaternion");
q179 = quaternion([179,0,0],"eulerd","ZYX","frame");
q180 = quaternion([180,0,0],"eulerd","ZYX","frame");
q181 = quaternion([181,0,0],"eulerd","ZYX","frame");
```

Use `slerp` to interpolate between `q0` and the three quaternion rotations. Specify that the paths are traveled in 10 steps.

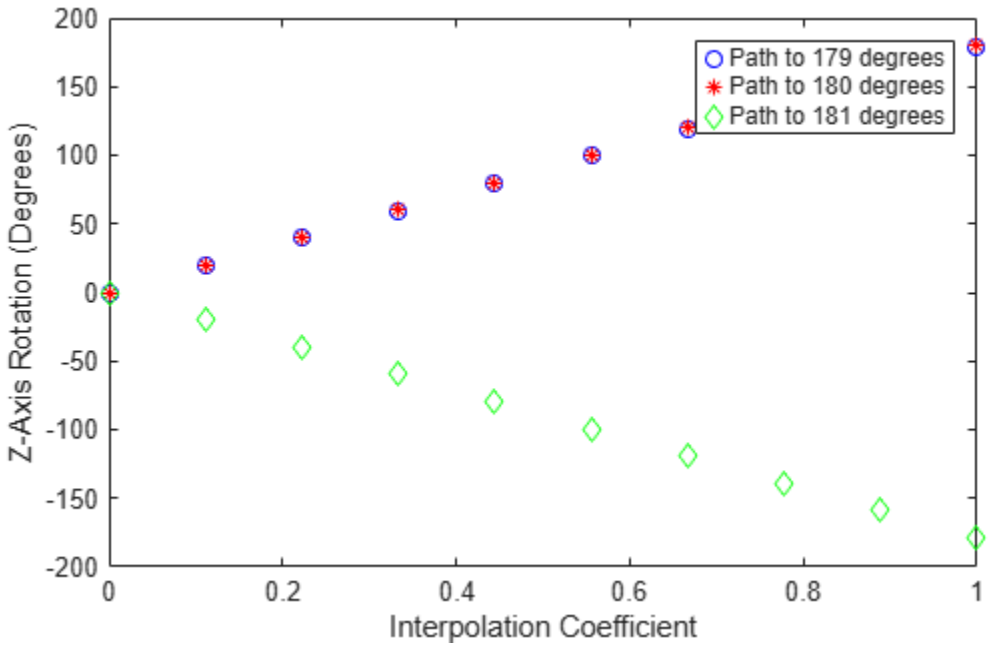
```
T = linspace(0,1,10);

q179path = slerp(q0,q179,T);
q180path = slerp(q0,q180,T);
q181path = slerp(q0,q181,T);
```

Plot each path in terms of Euler angles in degrees.

```
q179pathEuler = eulerd(q179path,"ZYX","frame");
q180pathEuler = eulerd(q180path,"ZYX","frame");
q181pathEuler = eulerd(q181path,"ZYX","frame");

plot(T,q179pathEuler(:,1),"bo", ...
     T,q180pathEuler(:,1),"r*", ...
     T,q181pathEuler(:,1),"gd");
legend("Path to 179 degrees", ...
       "Path to 180 degrees", ...
       "Path to 181 degrees")
xlabel("Interpolation Coefficient")
ylabel("Z-Axis Rotation (Degrees)")
```



The path between `q0` and `q179` is clockwise to minimize the great circle distance. The path between `q0` and `q181` is counterclockwise to minimize the great circle distance. The path between `q0` and `q180` can be either clockwise or counterclockwise, depending on numerical rounding.

Show Interpolated Quaternions on Sphere

Create two quaternions.

```
q1 = quaternion([0 0 0],"eulerd","ZYX","frame");
q2 = quaternion([-180 90 180],"eulerd","ZYX","frame");
```

Define the interpolation coefficient.

```
T = 0:0.01:1;
```

Interpolate quaternions using both the `"short"` and `"natural"` interpolation methods.

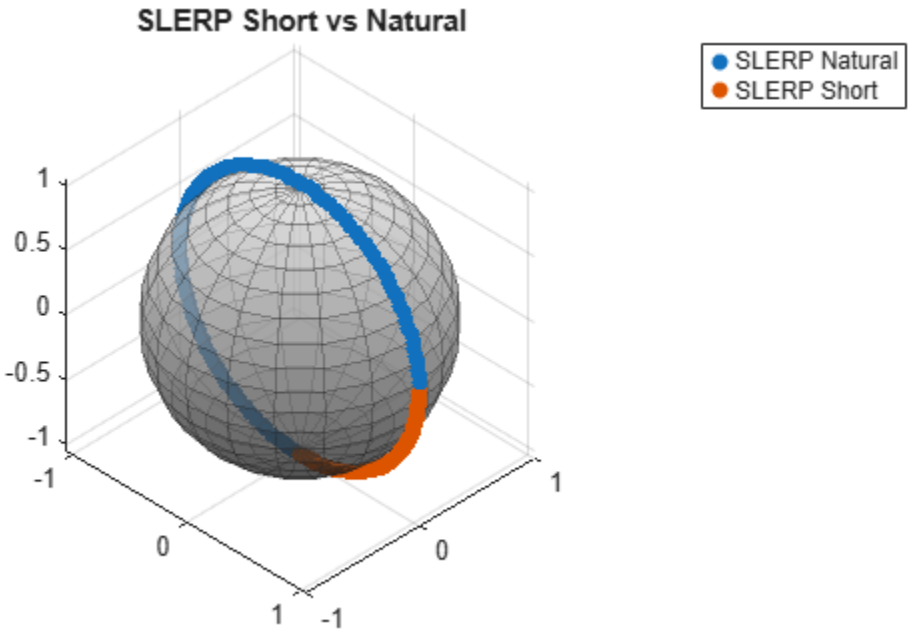
```
quats_natural = slerp(q1,q2,T,"natural");
quats_short = slerp(q1,q2,T,"short");
```

Obtain the corresponding rotate points.

```
pts_natural = rotatepoint(quats_natural,[1.05 0 0]);
pts_short = rotatepoint(quats_short,[1.05 0 0]);
```

Show the interpolated quaternions on a unit sphere.

```
figure
[X,Y,Z] = sphere;
surf(X,Y,Z,FaceAlpha=0.5,EdgeAlpha=0.35)
colormap gray
hold on
scatter3(pts_natural(:,1),pts_natural(:,2),pts_natural(:,3),"filled")
scatter3(pts_short(:,1),pts_short(:,2),pts_short(:,3),"filled")
view([45 36.60])
axis equal
title("SLERP Short vs Natural")
legend("","SLERP Natural","SLERP Short")
```



Input Arguments

q1 — Quaternion to interpolate

`quaternion` object | array of `quaternion` objects

Quaternion to interpolate, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality.

`q1`, `q2`, and `T` must have compatible sizes. In the simplest cases, they can be the same size or any one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of them is 1.

q2 — Quaternion to interpolate

`quaternion` object | array of `quaternion` objects

Quaternion to interpolate, specified as a scalar, vector, matrix, or multidimensional array of quaternions.

`q1`, `q2`, and `T` must have compatible sizes. In the simplest cases, they can be the same size or any one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of the dimension sizes is 1.

T — Interpolation coefficient

scalar | vector | matrix | multidimensional array

Interpolation coefficient, specified as a scalar, vector, matrix, or multidimensional array of numbers with each element in the range [0, 1].

`q1`, `q2`, and `T` must have compatible sizes. In the simplest cases, they can be the same size or any one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are either the same or one of the dimension sizes is 1.

Data Types: `single` | `double`

`method` — Spherical interpolation method
`"short"` (default) | `"natural"`

Spherical interpolation method, specified as either `"short"` or `"natural"`.

- `"short"` — Interpolates along the shortest path on the great circle between `q1` and `q2`, ensuring the minimal angular distance.
- `"natural"` — Skips shortest path optimization, following the natural path determined by the dot product of `q1` and `q2`, which may not be the shortest.

Data Types: `single` | `double`

Output Arguments

`q0` — Interpolated quaternion
`quaternion` object | array of `quaternion` objects

Interpolated quaternion, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion spherical linear interpolation (SLERP) is an extension of linear interpolation along a plane to spherical interpolation in three dimensions. The algorithm was first proposed in [1]. Given two quaternions, q_1 and q_2, SLERP interpolates a new quaternion, q_0, along the great circle that connects q_1 and q_2. The interpolation coefficient, T, determines how close the output quaternion is to either q_1 and q_2.

The SLERP algorithm can be described in terms of sinusoids:

$$q_0 = \frac{\sin((1-T)\theta)}{\sin(\theta)}q_1 + \frac{\sin(T\theta)}{\sin(\theta)}q_2$$

where q_1 and q_2 are normalized quaternions, and θ is half the angular distance between q_1 and q_2.

Version History

Introduced in R2023b

R2025a: Natural interpolation path

Spherically interpolate using the `"natural"` path option, enabling smoother transitions that respect the orientation of the start and end quaternions. Use the `method` argument to specify the interpolation method.

References

[1] Shoemake, Ken. "Animating Rotation with Quaternion Curves." *ACM SIGGRAPH Computer Graphics* 19, no. 3 (July 1985): 245–54. https://doi.org/10.1145/325165.325242.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`dist` | `meanrot`

Objects

`quaternion`

times, .*

Element-wise quaternion multiplication

Syntax

```
quatC = A.*B
```

Description

`quatC = A.*B` returns the element-by-element quaternion multiplication of quaternion arrays.

You can use quaternion multiplication to compose rotation operators:

- To compose a sequence of frame rotations, multiply the quaternions in the same order as the desired sequence of rotations. For example, to apply a p quaternion followed by a q quaternion, multiply in the order pq. The rotation operator becomes $(pq)^*v(pq)$, where v represents the object to rotate in quaternion form. * represents conjugation.
- To compose a sequence of point rotations, multiply the quaternions in the reverse order of the desired sequence of rotations. For example, to apply a p quaternion followed by a q quaternion, multiply in the reverse order, qp. The rotation operator becomes $(qp)v(qp)^*$.

Examples

Multiply Two Quaternion Vectors

Create two vectors, `A` and `B`, and multiply them element by element.

```
A = quaternion([1:4;5:8]);
B = A;
C = A.*B
```

```
C = 2×1 quaternion array
     -28 +   4i +   6j +   8k
    -124 +  60i +  70j +  80k
```

Multiply Two Quaternion Arrays

Create two 3-by-3 arrays, `A` and `B`, and multiply them element by element.

```
A = reshape(quaternion(randn(9,4)),3,3);
B = reshape(quaternion(randn(9,4)),3,3);
C = A.*B
```

```
C = 3×3 quaternion array
     0.60169 +  2.4332i -  2.5844j + 0.51646k    -0.49513 +  1.1722i +  4.4401j -   1.217k      2
     -4.2329 +  2.4547i +  3.7768j + 0.77484k    -0.65232 - 0.43112i -  1.4645j - 0.90073k     -1
```

```
     -4.4159 +  2.1926i +  1.9037j -  4.0303k     -2.0232 +  0.4205i - 0.17288j +  3.8529k     -2
```

Note that quaternion multiplication is not commutative:

```
isequal(C,B.*A)
```

```
ans = logical
   0
```

Multiply Quaternion Row and Column Vectors

Create a row vector `a` and a column vector `b`, then multiply them. The 1-by-3 row vector and 4-by-1 column vector combine to produce a 4-by-3 matrix with all combinations of elements multiplied.

```
a = [zeros("quaternion"),ones("quaternion"),quaternion(randn(1,4))]
```

```
a = 1×3 quaternion array
          0 +       0i +       0j +       0k           1 +       0i +       0j +       0k     0
```

```
b = quaternion(randn(4,4))
```

```
b = 4×1 quaternion array
      0.31877 +   3.5784i +   0.7254j -  0.12414k
      -1.3077 +   2.7694i - 0.063055j +   1.4897k
     -0.43359 -   1.3499i +  0.71474j +    1.409k
      0.34262 +   3.0349i -  0.20497j +   1.4172k
```

```
a.*b
```

```
ans = 4×3 quaternion array
            0 +        0i +        0j +        0k      0.31877 +   3.5784i +   0.7254j -  0.1241
            0 +        0i +        0j +        0k      -1.3077 +   2.7694i - 0.063055j +   1.489
            0 +        0i +        0j +        0k     -0.43359 -   1.3499i +  0.71474j +    1.40
            0 +        0i +        0j +        0k      0.34262 +   3.0349i -  0.20497j +   1.417
```

Input Arguments

A — Input to multiply

`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input to multiply, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of them is 1.

B — Input to multiply
`quaternion` object | array of `quaternion` objects | real scalar | array of real numbers

Input to multiply, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality, a real scalar, or an array of real numbers of any dimensionality. Numeric values must be of data type `single` or `double`.

`A` and `B` must have compatible sizes. In the simplest cases, they can be the same size or one can be a scalar. Two inputs have compatible sizes if, for every dimension, the dimension sizes of the inputs are the same or one of them is 1.

Output Arguments

quatC — Quaternion product
`quaternion` object | array of `quaternion` objects

Quaternion product, returned as a `quaternion` object or an array of `quaternion` objects.

Algorithms

Quaternion Multiplication by a Real Scalar

Given a quaternion,

$$q = a_q + b_q\mathrm{i} + c_q\mathrm{j} + d_q\mathrm{k},$$

the product of q and a real scalar β is

$$\beta q = \beta a_q + \beta b_q\mathrm{i} + \beta c_q\mathrm{j} + \beta d_q\mathrm{k}$$

Quaternion Multiplication by a Quaternion Scalar

The definition of the basis elements for quaternions,

$$\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1,$$

can be expanded to populate a table summarizing quaternion basis element multiplication:

	1	**i**	**j**	**k**
1	1	i	j	k
i	i	−1	k	−j
j	j	−k	−1	i
k	k	j	−i	−1

When reading the table, the rows are read first, for example: ij = k and ji = −k.

Given two quaternions, $q = a_q + b_q\mathrm{i} + c_q\mathrm{j} + d_q\mathrm{k}$, and $p = a_p + b_p\mathrm{i} + c_p\mathrm{j} + d_p\mathrm{k}$, the multiplication can be expanded as:

$$
\begin{aligned}
z = pq &= \left(a_p + b_p\mathrm{i} + c_p\mathrm{j} + d_p\mathrm{k}\right)\left(a_q + b_q\mathrm{i} + c_q\mathrm{j} + d_q\mathrm{k}\right) \\
&= a_p a_q + a_p b_q\mathrm{i} + a_p c_q\mathrm{j} + a_p d_q\mathrm{k} \\
&+ b_p a_q\mathrm{i} + b_p b_q\mathrm{i}^2 + b_p c_q\mathrm{ij} + b_p d_q\mathrm{ik} \\
&+ c_p a_q\mathrm{j} + c_p b_q\mathrm{ji} + c_p c_q\mathrm{j}^2 + c_p d_q\mathrm{jk} \\
&+ d_p a_q k + d_p b_q\mathrm{ki} + d_p c_q\mathrm{kj} + d_p d_q\mathrm{k}^2
\end{aligned}
$$

You can simplify the equation using the quaternion multiplication table.

$$
\begin{aligned}
z = pq &= a_p a_q + a_p b_q\mathrm{i} + a_p c_q\mathrm{j} + a_p d_q\mathrm{k} \\
&+ b_p a_q\mathrm{i} - b_p b_q + b_p c_q\mathrm{k} - b_p d_q\mathrm{j} \\
&+ c_p a_q\mathrm{j} - c_p b_q\mathrm{k} - c_p c_q + c_p d_q\mathrm{i} \\
&+ d_p a_q k + d_p b_q\mathrm{j} - d_p c_q\mathrm{i} - d_p d_q
\end{aligned}
$$

Version History

Introduced in R2023b

References

[1] Kuipers, Jack B. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton, NJ: Princeton University Press, 2007.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`prod` | `mtimes, *`

Objects

`quaternion`

transpose, .'

Transpose quaternion array

Syntax

```
Y = quat.'
```

Description

`Y = quat.'` returns the non-conjugate transpose of the quaternion array, `quat`.

Examples

Vector Transpose

Create a vector of quaternions and compute its nonconjugate transpose.

```
quat = quaternion(randn(4,4))
```

```
quat = 4×1 quaternion array
      0.53767 +  0.31877i +   3.5784j +   0.7254k
       1.8339 -   1.3077i +   2.7694j - 0.063055k
      -2.2588 -  0.43359i -   1.3499j +  0.71474k
      0.86217 +  0.34262i +   3.0349j -  0.20497k
```

```
quatTransposed = quat.'
```

```
quatTransposed = 1×4 quaternion array
      0.53767 +  0.31877i +   3.5784j +   0.7254k       1.8339 -   1.3077i +   2.7694j - 0.06305
```

Matrix Transpose

Create a matrix of quaternions and compute its nonconjugate transpose.

```
quat = [quaternion(randn(2,4)),quaternion(randn(2,4))]
```

```
quat = 2×2 quaternion array
      0.53767 -   2.2588i +  0.31877j -  0.43359k       3.5784 -   1.3499i +   0.7254j +  0.7147
       1.8339 +  0.86217i -   1.3077j +  0.34262k       2.7694 +   3.0349i - 0.063055j -  0.2049
```

```
quatTransposed = quat.'
```

```
quatTransposed = 2×2 quaternion array
      0.53767 -   2.2588i +  0.31877j -  0.43359k       1.8339 +  0.86217i -   1.3077j +  0.3426
       3.5784 -   1.3499i +   0.7254j +  0.71474k       2.7694 +   3.0349i - 0.063055j -  0.2049
```

Input Arguments

quat — Quaternion array to transpose

vector of `quaternion` objects | matrix of `quaternion` objects

Quaternion array to transpose, specified as a vector or matrix of `quaternion` objects. `transpose` is defined for 1-D and 2-D arrays. For higher-order arrays, use `permute`.

Output Arguments

Y — Transposed quaternion array

vector of `quaternion` objects | matrix of `quaternion` objects

Transposed quaternion array, returned as a vector or matrix of `quaternion` objects. The size of `Y` is *N*-by-*M*, where the size of `quat` is *M*-by-*N*.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`ctranspose, '`

Objects

`quaternion`

uminus, -

Quaternion unary minus

Syntax

```
mQuat = -quat
```

Description

`mQuat = -quat` negates the elements of `quat` and stores the result in `mQuat`.

Examples

Negate Elements of Quaternion Matrix

Unary minus negates each part of a the quaternion. Create a 2-by-2 matrix, `Q`.

```
Q = quaternion(randn(2),randn(2),randn(2),randn(2))
```

```
Q = 2×2 quaternion array
      0.53767 +  0.31877i +   3.5784j +   0.7254k      -2.2588 -  0.43359i -   1.3499j +  0.7147
       1.8339 -   1.3077i +   2.7694j - 0.063055k      0.86217 +  0.34262i +   3.0349j -  0.2049
```

Negate the parts of each quaternion in `Q`.

```
R = -Q
```

```
R = 2×2 quaternion array
     -0.53767 -  0.31877i -   3.5784j -   0.7254k       2.2588 +  0.43359i +   1.3499j -  0.7147
      -1.8339 +   1.3077i -   2.7694j + 0.063055k     -0.86217 -  0.34262i -   3.0349j +  0.2049
```

Input Arguments

`quat` — Input quaternion

`quaternion` object | array of `quaternion` objects

Input quaternion, specified as a `quaternion` object, an array of `quaternion` objects of any dimensionality.

Output Arguments

`mQuat` — Negated quaternion

`quaternion` object | array of `quaternion` objects

Negated quaternion, returned as a `quaternion` object or an array of `quaternion` objects of the same size as `quat`.

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`minus, -`

Objects

`quaternion`

zeros

Create quaternion array with all parts set to zero

Syntax

```
quatZeros = zeros("quaternion")
quatZeros = zeros(n,"quaternion")
quatZeros = zeros(sz,"quaternion")
quatZeros = zeros(sz1,...,szN,"quaternion")

quatZeros = zeros( ___ ,"like",prototype,"quaternion")
```

Description

`quatZeros = zeros("quaternion")` returns a scalar quaternion with all parts set to zero.

`quatZeros = zeros(n,"quaternion")` returns an `n`-by-`n` matrix of quaternions.

`quatZeros = zeros(sz,"quaternion")` returns an array of quaternions where the size vector, `sz`, defines `size(quatZeros)`.

`quatZeros = zeros(sz1,...,szN,"quaternion")` returns a `sz1`-by-...-by-`szN` array of quaternions where `sz1,…,szN` indicates the size of each dimension.

`quatZeros = zeros( ___ ,"like",prototype,"quaternion")` specifies the underlying class of the returned quaternion array to be the same as the underlying class of the quaternion prototype.

Examples

Quaternion Scalar Zero

Create a quaternion scalar zero.

```
quatZeros = zeros("quaternion")
```

```
quatZeros = quaternion
     0 + 0i + 0j + 0k
```

Square Matrix of Quaternions

Create an n-by-n array of quaternion zeros.

```
n = 3;
quatZeros = zeros(n,"quaternion")
```

```
quatZeros = 3×3 quaternion array
     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k
```

```
     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k
```

Multidimensional Array of Quaternion Zeros

Create a multidimensional array of quaternion zeros by defining array dimensions in order. In this example, you create a 3-by-1-by-2 array. You can specify dimensions using a row vector or comma-separated integers.

Specify the dimensions using a row vector and display the results:

```
dims = [3,1,2];
quatZerosSyntax1 = zeros(dims,"quaternion")
```

```
quatZerosSyntax1 = 3×1×2 quaternion array
quatZerosSyntax1(:,:,1) =

     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k


quatZerosSyntax1(:,:,2) =

     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k
```

Specify the dimensions using comma-separated integers, and then verify the equivalence of the two syntaxes:

```
quatZerosSyntax2 = zeros(3,1,2,"quaternion");
isequal(quatZerosSyntax1,quatZerosSyntax2)
```

```
ans = logical
   1
```

Underlying Class of Quaternion Zeros

A quaternion is a four-part hyper-complex number used in three-dimensional representations. You can specify the underlying data type of the parts as `single` or `double`. The default is `double`.

Create a quaternion array of zeros with the underlying data type set to `single`.

```
quatZeros = zeros(2,"like",single(1),"quaternion")
```

```
quatZeros = 2×2 quaternion array
     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k
     0 + 0i + 0j + 0k     0 + 0i + 0j + 0k
```

Verify the underlying class using the `classUnderlying` function.

```
classUnderlying(quatZeros)
```

```
ans =
'single'
```

Input Arguments

`n` — Size of square quaternion matrix

integer value

Size of square quaternion matrix, specified as an integer value. If `n` is `0` or negative, then `quatZeros` is returned as an empty matrix.

Example: `zeros(4,"quaternion")` returns a 4-by-4 matrix of quaternion zeros.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`sz` — Output size

row vector of integer values

Output size, specified as a row vector of integer values. Each element of `sz` indicates the size of the corresponding dimension in `quatZeros`. If the size of any dimension is `0` or negative, then `quatZeros` is returned as an empty array.

Example: `zeros([1,4,2],"quaternion")` returns a 1-by-4-by-2 array of quaternion zeros.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`prototype` — Quaternion prototype

`quaternion` object

Quaternion prototype, specified as a `quaternion` object.

Example: `zeros(2,"like",quat,"quaternion")` returns a 2-by-2 matrix of quaternions with the same underlying class as the prototype quaternion, `quat`.

`sz1,...,szN` — Size of each dimension

two or more integer values

Size of each dimension, specified as two or more comma-separated integers.

- If the size of any dimension is `0`, then `quatZeros` is returned as an empty array.
- If the size of any dimension is negative, then it is treated as `0`.

Example: `zeros(2,3,"quaternion")` returns a 2-by-3 matrix of quaternion zeros.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`quatZeros` — Quaternion zeros

`quaternion` object | array of `quaternion` objects

Quaternion zeros, returned as a `quaternion` object or an array of `quaternion` objects.

Given a quaternion of the form $Q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$, a quaternion zero is defined as $Q = 0 + 0\mathrm{i} + 0\mathrm{j} + 0\mathrm{k}$.

Data Types: `quaternion`

Version History

Introduced in R2023b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`ones`

Objects

`quaternion`

efficientADAnomalyDetector

Detect anomalies using EfficientAD network

Description

The `efficientADAnomalyDetector` object detects images of anomalies using an EfficientAD anomaly detector network. Train the detector using the `trainEfficientADAnomalyDetector` function. To detect anomalous images, pass the trained detector to the `classify` function.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = efficientADAnomalyDetector
detector = efficientADAnomalyDetector(Name=Value)
```

Description

`detector = efficientADAnomalyDetector` creates an EfficientAD anomaly detector from a base patch description network (PDN).

`detector = efficientADAnomalyDetector(Name=Value)` specifies options for the EfficientAD anomaly detector creation using one or more name-value arguments.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `efficientADAnomalyDetector(PDNnet="pdn-small")` specifies the patch description network as `"pdn-small"`.

`PDNnet` — Name of patch description network
`"pdn-small"` (default) | `"pdn-medium"`

Name of the patch description network (PDN), a convolutional network that serves as the base feature extraction network, specified as one of these values:

- `"pdn-small"` — PDN of reduced depth that has a teacher and student network architecture.
- `"pdn-medium"` — PDN that uses twice as many filters in the middle layers of both its teacher and student networks. This network has a higher capacity to extract and learn detailed and nuanced features from image patches, at the expense of more computation time.

For images of variable size, the PDN generates all feature vectors in a single forward pass.

Data Types: `char` | `string`

`UseGlobalAnomalyMap` — Global anomaly map autoencoder toggle for tile-based training
`true` or `1` (default) | `false` or `0`

Global anomaly map autoencoder toggle for tile-based training, specified as a numeric or logical `1` (`true`) or `0` (`false`). If you specify `UseGlobalAnomalyMap` as `false`, the `efficientADAnomalyDetector` object creates an EfficientAD anomaly detector that uses the student-teacher model to generate the local anomaly map, and turns off the autoencoder, which primarily detects global anomalies. To perform inference on full-size images after tile-based training on patches of images, turn off the autoencoder by specifying `UseGlobalAnomalyMap` as `false`.

Tip When you perform tile-based training and GPU memory is limited, reduce the mini-batch size using the `MiniBatchSize` argument of the `trainingOptions` function, at the expense of slower computation speed.

`OptimizeScoreForLogicalAnomalies` — Optimize anomaly score for logical anomalies
`false` or `0` (default) | `true` or `1`

Optimize anomaly score for logical anomalies, specified as a numeric or logical `0` (`false`) or `1` (`true`) . If you specify `OptimizeScoreForLogicalAnomalies` as `true`, the `efficientADAnomalyDetector` object creates an EfficientAD anomaly detector that uses both picturable and unpicturable anomaly scores to optimize logical anomaly detection. Logical anomalies result from missing, misplaced, extraneous, or geometrically faulty features. Specify this option to improve detector performance in images with logical anomalies at the expense of increased computation cost from calculating normalization statistics and additional anomaly scores. To learn more, see “Logical Anomaly Detection Method” on page 2-1632.

Properties

`Threshold` — Anomaly threshold
`[]` (default) | numeric scalar

Anomaly threshold, stored as a numeric scalar. You must set the value of the threshold before calling the `classify` function. You can calculate a threshold using the `anomalyThreshold` function.

`ImageSize` — Image size
`[]` (default) | three-element numeric row vector

This property is read-only.

Image size of images used for training and inference, stored as a three-element numeric row vector. The `trainEfficientADAnomalyDetector` function sets the `ImageSize` property during training, which indicates the size of the training images. The image size can be different from the input layer size of the base feature extraction PDN.

Note When using the detector, the image size at inference must match the image size used during training.

Object Functions

predict	Predict unnormalized anomaly scores
classify	Classify image as normal or anomalousJAB
anomalyMap	Predict per-pixel anomaly score map

Examples

Create EfficientAD Anomaly Detector

Create an `efficientADAnomalyDetector` object from a medium-sized base patch description network (PDN).

```
detector = efficientADAnomalyDetector(Network="pdn-medium")
```

```
detector =
  efficientADAnomalyDetector with properties:

    Threshold: []
    ImageSize: []
      Network: "pdn-medium"
```

More About

EfficientAD Student-teacher Network Architecture

The EfficientAD anomaly detector operates on the principle of a student-teacher model, where the *teacher* is a pre-trained deep neural network that understands normal data behavior, and the *student* is a smaller network that learns to mimic the teacher's responses on normal data. During training, only normal data (without anomalies) is used, allowing the student to learn a compact representation of what is considered normal. At inference, both the student and teacher evaluate incoming data; anomalies are detected based on the discrepancy between the student's and teacher's outputs, as the student will likely fail to accurately replicate the teacher's behavior on anomalous data, highlighting instances that deviate from the learned normal behavior.

Logical Anomaly Detection Method

Picturable, or structural, anomalies are anomalies whose location can be clearly depicted on a ground truth map. *Unpicturable* anomalies are logical anomalies which result from missing, misplaced, extraneous, or geometrically faulty features, and are not easily represented by spatial feature maps. The Picturable and Unpicturable Anomaly Scores (PUAD) method uses EfficientAD to detect picturable anomalies, and a feature-based method to detect unpicturable anomalies. The out-of-distribution approach calculates the unpicturable anomalies as Mahalanobis distances using the student’s former half output [2].

Version History

Introduced in R2024b

R2025a: Option to specify EfficientAD detector to optimize anomaly score for logical anomaly detection

Specify the `OptimizeScoreForLogicalAnomalies` name-value argument as `true` (or `1`) to optimize the anomaly score for logical anomaly detection.

References

[1] Batzner, Kilian, Lars Heckler, and Rebecca König. “EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies.” arXiv, February 8, 2024. https://doi.org/10.48550/arXiv.2303.14535.

[2] Sugawara, Shota, and Ryuji Imamura. “PUAD: Frustratingly Simple Method for Robust Anomaly Detection.” In *2024 IEEE International Conference on Image Processing (ICIP)*, 842–48. Abu Dhabi, United Arab Emirates: IEEE, 2024. https://doi.org/10.1109/ICIP51287.2024.10647438.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare an `efficientADAnomalyDetector` object for code generation, use `coder.loadDeepLearningNetwork`.
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare an `efficientADAnomalyDetector` object for code generation, use `coder.loadDeepLearningNetwork`.
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

See Also

`trainEfficientADAnomalyDetector` | `anomalyThreshold` | `evaluateAnomalyDetection`

Topics

“Detect Defects Using Tiled Training of EfficientAD Anomaly Detector”
“Identify Defects in Air Compressors Using Spectrogram Images”
“Getting Started with Anomaly Detection Using Deep Learning”

fastFlowAnomalyDetector

Detect anomalies using FastFlow network

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `fastFlowAnomalyDetector` object detects images of anomalies using a FastFlow anomaly detector network. Train the detector using the `trainFastFlowAnomalyDetector` function. To detect anomalous images, pass the trained detector to the `classify` function.

Note This functionality requires Deep Learning Toolbox. This functionality also requires the Deep Learning Toolbox Model for ResNet-18 Network and the Automated Visual Inspection Library for Computer Vision Toolbox, which you can install from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = fastFlowAnomalyDetector
detector = fastFlowAnomalyDetector(Name=Value)
```

Description

`detector = fastFlowAnomalyDetector` creates a FastFlow anomaly detector from a ResNet-18 backbone network.

`detector = fastFlowAnomalyDetector(Name=Value)` specifies the `Backbone`, `NumFlowSteps` and `FlowModelChannelRatio` properties of the FastFlow anomaly detector using name-value arguments.

For example, `NumFlowSteps=10` sets the number of steps in the flow network of the FastFlow detector to 10.

Properties

Backbone — Pretrained backbone network

`dlnetwork` object

This property is read-only.

Pretrained backbone network that serves as the base feature extraction network, specified as a `dlnetwork` object. The network must accept a single input and produce one or more outputs. If you do not specify the `Backbone` property, `fastFlowAnomalyDetector` uses a ResNet-18 network as the backbone feature extraction network.

To set the value of this property, you must specify it at object creation.

Data Types: `dlnetwork`

`NumFlowSteps` — Number of steps in flow network
`8` (default) | positive integer

This property is read-only.

Number of steps in the flow network of the FastFlow detector, specified as a positive integer. Increase the number of flow network steps to obtain more trainable parameters at the possible expense of increasing memory cost and using more computing resources.

To set the value of this property, you must specify it at object creation.

`FlowModelChannelRatio` — Ratio of input channels to hidden channels
`1.0` (default) | positive numeric scalar

This property is read-only.

Ratio of input channels to hidden channels used in the input and output subnet layers after the channels are split in the flow network, specified as a positive numeric scalar. Increase the value of `FlowModelChannelRatio` to improve detection metrics at the possible expense of increasing memory cost and using more computing resources.

To set the value of this property, you must specify it at object creation.

`Threshold` — Anomaly threshold
`[]` (default) | numeric scalar

Anomaly threshold, specified as a numeric scalar. You must set the value of the threshold before calling the `classify` function. You can calculate a threshold using the `anomalyThreshold` function.

`Layers` — Network layers
array of `Layer` objects

This property is read-only.

Network layers, specified as an array of `Layer` objects.

`InputNames` — Network input layer names
`[]` (default) | cell array of character vectors

This property is read-only.

Network input layer names, specified as a cell array of character vectors.

`OutputNames` — Network output layer names
`[]` (default) | cell array of character vectors

This property is read-only.

Network output layer names, specified as a cell array of character vectors.

`State` — Network state
table

This property is read-only.

Network state, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar
- `Parameter` — State parameter name, specified as a string scalar
- `Value` — Value of state parameter, specified as a `dlarray` object

Layer states contain information calculated during the layer operation to be retained for use in subsequent forward passes of the layer. For example, the cell state and hidden state of LSTM layers, or running statistics in batch normalization layers.

`Learnables` — Network learnable parameters
table

This property is read-only.

Network learnable parameters, specified as a table with three columns.

- `Layer` — Layer name, specified as a string scalar
- `Parameter` — Parameter name, specified as a string scalar
- `Value` — Value of parameter, specified as a `dlarray` object

The network learnable parameters contain the features that the network learns. For example, the parameters include the weights of convolution and fully connected layers.

Object Functions

predict	Predict unnormalized anomaly scores
classify	Classify image as normal or anomalousJAB
anomalyMap	Predict per-pixel anomaly score map

Examples

Create FastFlow Anomaly Detector

Create a `fastFlowAnomalyDetector` object.

```
detector = fastFlowAnomalyDetector
```

```
detector =
  fastFlowAnomalyDetector with properties:

    Threshold: []
```

Version History

Introduced in R2023a

R2024b: Use `coder.loadDeepLearningNetwork` for code generation

Starting in R2024b, you can use the `coder.loadDeepLearningNetwork` function to load a `fastFlowAnomalyDetector` object for CPU and GPU code generation.

R2024a: Option to specify custom backbone network, number of steps in flow network, and ratio of input channels to hidden channels used in subnetwork

Specify the backbone feature extraction network as a `dlnetwork` object by using the `Backbone` name-value argument.

Specify the number of steps in the flow network by using the `NumFlowSteps` name-value argument. For example, `NumFlowSteps=10` sets the number of steps in the flow network of the FastFlow detector to 10.

Specify the ratio of input channels to hidden channels in the input and output subnet layers of the FastFlow detector by using the `FlowModelChannelRatio` name-value argument.

References

[1] Yu, Jiawei, Ye Zheng, Xiang Wang, Wei Li, Yushuang Wu, Rui Zhao, and Liwei Wu. "FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows." arXiv, November 16, 2021. https://doi.org/10.48550/arXiv.2111.07677.

[2] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition.” In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–78. Las Vegas, NV, USA: IEEE, 2016. https://doi.org/10.1109/CVPR.2016.90.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `fastFlowAnomalyDetector` object for code generation, you can use either the `vision.loadFastFlowAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `fastFlowAnomalyDetector` object for GPU code generation, you can use either the `vision.loadFastFlowAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (GPU Coder).
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

See Also

`trainFastFlowAnomalyDetector` | `anomalyThreshold` | `evaluateAnomalyDetection`

Topics

“Getting Started with Anomaly Detection Using Deep Learning”

fcddAnomalyDetector

Detect anomalies using fully convolutional data description (FCDD) network for anomaly detection

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `fcddAnomalyDetector` object detects images of anomalies using a fully convolutional data description (FCDD) network. To detect anomalous images, pass the trained detector to the `classify` function.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = fcddAnomalyDetector(network)
```

Description

`detector = fcddAnomalyDetector(network)` creates an FCDD anomaly detector from a base feature extraction backbone network.

Input Arguments

`network` — Network
`dlnetwork` object

Network that serves as the base feature extraction backbone, specified as a `dlnetwork` object. You can create a `dlnetwork` for several common named pretrained networks, such as GoogLeNet and ResNet-18, using the `pretrainedEncoderNetwork` function.

Properties

`Threshold` — Anomaly threshold
`[]` (default) | numeric scalar

Anomaly threshold, specified as a numeric scalar. You must set the value of the threshold before calling the `classify` function. You can calculate a threshold using the `anomalyThreshold` function.

`Layers` — Network layers
array of `Layer` objects

This property is read-only.

Network layers, specified as an array of `Layer` objects.

`InputNames` — Network input layer names
`[ ]` (default) | cell array of character vectors

This property is read-only.

Network input layer names, specified as a cell array of character vectors.

`OutputNames` — Network output layer names
`[ ]` (default) | cell array of character vectors

This property is read-only.

Network output layer names, specified as a cell array of character vectors.

`State` — Network state
table

This property is read-only.

Network state, specified as a table with three columns:

- `Layer` – Layer name, specified as a string scalar.
- `Parameter` – State parameter name, specified as a string scalar.
- `Value` – Value of state parameter, specified as a `dlarray` object.

Layer states contain information calculated during the layer operation to be retained for use in subsequent forward passes of the layer. For example, the cell state and hidden state of LSTM layers, or running statistics in batch normalization layers.

`Learnables` — Network learnable parameters
table

This property is read-only.

Network learnable parameters, specified as a table with three columns:

- `Layer` – Layer name, specified as a string scalar.
- `Parameter` – Parameter name, specified as a string scalar.
- `Value` – Value of parameter, specified as a `dlarray` object.

The network learnable parameters contain the features learned by the network. For example, the weights of convolution and fully connected layers.

Object Functions

predict	Predict unnormalized anomaly scores
classify	Classify image as normal or anomalousJAB
anomalyMap	Predict per-pixel anomaly score map

Examples

Create FCDD Anomaly Detector

Create an `fcddAnomalyDetector` object with a ResNet-18 backbone.

```
backbone = pretrainedEncoderNetwork("resnet18",3);
detector = fcddAnomalyDetector(backbone)
```

```
detector =
  fcddAnomalyDetector with properties:

      Threshold: []
         Layers: [43×1 nnet.cnn.layer.Layer]
     InputNames: {'data'}
    OutputNames: {'fcddHeadPseudoHuber'}
          State: [4×3 table]
     Learnables: [10×3 table]
```

Version History

Introduced in R2022b

R2024b: Use `coder.loadDeepLearningNetwork` for code generation

Starting in R2024b, you can use the `coder.loadDeepLearningNetwork` function to load a `fcddAnomalyDetector` object for CPU and GPU code generation.

R2023a: Support for C and GPU code generation

`fcddAnomalyDetector` now supports the generation of C code (requires MATLAB Coder) and optimized CUDA code (requires GPU Coder).

References

[1] Liznerski, Philipp, Lukas Ruff, Robert A. Vandermeulen, Billy Joe Franks, Marius Kloft, and Klaus-Robert Müller. "Explainable Deep One-Class Classification." Preprint, submitted March 18, 2021. https://arxiv.org/abs/2007.01760.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To create an `fcddAnomalyDetector` object for code generation, you can use either `vision.loadFCDDAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how

to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).

- The `predict`, `classify`, and `anomalyMap` object functions are supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To create an `fcddAnomalyDetector` object for GPU code generation, you can use either `vision.loadFCDDAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (GPU Coder).
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

See Also

`trainFCDDAnomalyDetector` | `anomalyThreshold` | `evaluateAnomalyDetection`

Topics
“Detect Image Anomalies Using Explainable FCDD Network”
“Getting Started with Anomaly Detection Using Deep Learning”

patchCoreAnomalyDetector

Detect anomalies using PatchCore network

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `patchCoreAnomalyDetector` object detects images of anomalies using a PatchCore anomaly detector network. Train the detector using the `trainPatchCoreAnomalyDetector` function. To detect anomalous images, pass the trained detector to the `classify` function.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
detector = patchCoreAnomalyDetector
detector = patchCoreAnomalyDetector(Backbone=backbone)
```

Description

`detector = patchCoreAnomalyDetector` creates a PatchCore anomaly detector from a ResNet-18 backbone network.

`detector = patchCoreAnomalyDetector(Backbone=backbone)` specifies the base feature extraction backbone network from which to create the PatchCore anomaly detector.

Input Arguments

backbone — Name of pretrained backbone network
`"resnet18"` (default) | `"resnet50"` | `"resnet101"` | `dlnetwork` object

Name of the pretrained backbone network that serves as the base feature extraction network, specified as a `dlnetwork` object or one of these values:

- `"resnet18"` — ResNet-18 convolutional neural network.
- `"resnet50"` — ResNet-50 convolutional neural network.
- `"resnet101"` — ResNet-101 convolutional neural network.

To learn more about each network, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox). You must install the associated add-on for the selected pretrained network.

If you specify the network as a `dlnetwork` object, it must accept a single input and return one or more outputs.

Data Types: `char` | `string` | `dlnetwork`

Properties

`Threshold` — Anomaly threshold

`[]` (default) | numeric scalar

Anomaly threshold, specified as a numeric scalar. You must set the value of the threshold before calling the `classify` function. You can calculate a threshold using the `anomalyThreshold` function.

`ImageSize` — Image size

`[]` (default) | three-element numeric row vector

This property is read-only.

Image size of images used for training and inference, stored as a three-element numeric row vector. The `trainPatchCoreAnomalyDetector` function sets the `ImageSize` property during training, which indicates the size of the training images. The image size can be different from the input layer size of `backbone`.

Note When using the detector, the image size at inference must match the image size used during training.

Object Functions

predict	Predict unnormalized anomaly scores
classify	Classify image as normal or anomalousJAB
anomalyMap	Predict per-pixel anomaly score map

Examples

Create PatchCore Anomaly Detector

Create a `patchCoreAnomalyDetector` object from a ResNet-50 backbone network.

```
detector = patchCoreAnomalyDetector(Backbone="resnet50")
```

```
detector =
  patchCoreAnomalyDetector with properties:

    Threshold: []
    ImageSize: []
```

Version History

Introduced in R2023a

R2024b: Use `coder.loadDeepLearningNetwork` for code generation

Starting in R2024b, you can use the `coder.loadDeepLearningNetwork` function to load a `patchCoreAnomalyDetector` object for CPU and GPU code generation.

R2026a: Option to specify ResNet-101 backbone network

You can now specify the backbone feature extraction network as a ResNet-101 convolutional neural network by specifying the `backbone` input argument as `"resnet101"`.

R2024a: Option to specify backbone network as a `dlnetwork` object

Specify the backbone feature extraction network as a `dlnetwork` object by using the `backbone` input argument. For example, given a custom `dlnetwork` object, `dlnet`, the argument `Backbone` specifies `dlnet` as the backbone feature extraction network.

References

[1] Roth, Karsten, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. “Towards Total Recall in Industrial Anomaly Detection.” arXiv, May 5, 2022. https://arxiv.org/abs/2106.08265.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `patchCoreAnomalyDetector` object for code generation, you can use either `vision.loadPatchCoreAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `patchCoreAnomalyDetector` object for GPU code generation, you can use either `vision.loadPatchCoreAnomalyDetector` or `coder.loadDeepLearningNetwork` function. The recommended approach is to use `coder.loadDeepLearningNetwork`. For information on how to load the detector for code generation, see “Load Pretrained Networks for Code Generation” (GPU Coder).
- The `predict`, `classify`, and `anomalyMap` object functions are supported.

See Also

`trainPatchCoreAnomalyDetector` | `anomalyThreshold` | `evaluateAnomalyDetection`

Topics

“Getting Started with Anomaly Detection Using Deep Learning”
“Localize Industrial Defects Using PatchCore Anomaly Detector”

percentileNormalizer

Create percentile normalizer for anomaly map normalization

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `percentileNormalizer` object creates a percentile anomaly map normalizer for a specified anomaly detector using the calculated percentile statistics of normal, or non-anomalous, images in a datastore. Use this object along with its object function `normalize`, to normalize anomaly score maps from different detectors to the same range, or to normalize anomaly scores for an anomaly map or a set of anomaly maps to a specified range.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
normalizer = percentileNormalizer(dsNormal,detector)
normalizer = percentileNormalizer(dsNormal,detector,Name=Value)
```

Description

`normalizer = percentileNormalizer(dsNormal,detector)` creates a percentile anomaly map normalizer for a specified anomaly detector, `detector`, using the calculated percentile statistics of normal, or non-anomalous, images in a datastore, `dsNormal`.

`normalizer = percentileNormalizer(dsNormal,detector,Name=Value)` specifies additional options for normalization parameters using name-value arguments. For example, `NormalPercentileRange=[0.95 0.99]` sets the lower and upper percentile statistics range for normal images to `[0.95, 0.99]`.

Input Arguments

`dsNormal` — Normal image data
datastore

Normal image data, specified as a datastore that contains image data. Do not include anomalous images in this datastore.

`detector` — Anomaly detector
`efficientADAnomalyDetector` object | `fcddAnomalyDetector` object | `fastFlowAnomalyDetector` object | `patchCoreAnomalyDetector` object

Anomaly detector, specified as an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, a `fastFlowAnomalyDetector` object, or a `patchCoreAnomalyDetector` object.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `percentileNormalizer(dsNormal,detector,NormalPercentileRange=[0.95 0.99])` sets the lower and upper percentile statistics range for normal images to `[0.95, 0.99]`.

`NormalPercentileRange` — Normal image percentile range
`[0.9 0.99]` (default) | vector of form $[p_1\ p_2]$

Normal image percentile range, specified as a vector of the form $[p_1\ p_2]$, where p_1 and p_2 are the lower and upper percentile bounds, respectively. The lower and upper percentile values must be in the range [0, 1].

p_1 and p_2 define the slope of the linear mapping from the anomaly score input to the normalized anomaly score values. To learn more, see “Linear Mapping for Anomaly Map Normalization” on page 2-1648.

`Verbose` — Training progress information display
`true` or `1` (default) | `false` or `0`

Training progress information display in the command window, specified as a logical `1` (`true`) or `0` (`false`).

Properties

`PercentileStatistics` — Anomaly map percentile values
vector of form [*P1 P2*]

Anomaly map percentile values, *P1* and *P2*, stored as a two-element numeric vector of the form [*P1 P2*].

Object Functions

normalize Normalize anomaly score map using percentiles

More About

Linear Mapping for Anomaly Map Normalization

The `percentileNormalizer` object maps the upper range of the normal image percentile to the bottom end of the anomaly map output values. The object considers image pixels with anomaly scores higher than the upper percentile of the normal image to be anomalous. p_1 and p_2 are the normal image upper percentile range lower and upper bounds, respectively, and the object bounds the anomaly map output by the lower and upper values v_1 and v_2, respectively. Therefore, this function defines the linear mapping between the anomaly score input x and the normalized anomaly score output y:

$$y = \frac{x(v2 - v1)}{(p2 - p1)} - \frac{p1(v2 - v1)}{(p2 - p1)} + v1$$

The upper range values of the normal image percentile, p_2 and p_2, define the slope of the linear mapping from the anomaly score input to the normalized anomaly score values.

The normalized anomaly score from the linear mapping, y, is unbounded. To confine the normalization to a range of values, specify the range using the `OutputClipRange` name-value argument of the `normalize` object function.

Version History

Introduced in R2024b

See Also

`anomalyMap` | `efficientADAnomalyDetector` | `fcddAnomalyDetector` | `fastFlowAnomalyDetector` | `patchCoreAnomalyDetector`

Topics

“Getting Started with Anomaly Detection Using Deep Learning”

counTRObjectCounter

Count objects using CounTR deep learning model

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `counTRObjectCounter` object creates a Counting TRansformer (CounTR) object counter model for counting the number of objects from arbitrary semantic categories in an image, without the need for training. Using this object, you can:

- Create a CounTR object counter model using exemplar patches. Exemplar patches are small image patches that contain examples of the objects to be counted. These patches serve as reference examples that help the model identify and count similar objects in the larger input image.
- Create a CounTR object counter model using an exemplar image with exemplar bounding box locations from which the model extracts exemplar patches.

After creating the CounTR model using the `counTRObjectCounter` object, count the number of objects in an image based on the exemplar patches you specify using the `countObjects` object function. Compute the density map of the object count in the image, based on specified exemplar patches, using the `densityMap` object function.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
counter = counTRObjectCounter(exemplarPatches)
counter = counTRObjectCounter(exemplarImage,exemplarBBoxes)
```

Description

`counter = counTRObjectCounter(exemplarPatches)` creates a CounTR object counter model using exemplar patches containing objects to count, `exemplarPatches`.

`counter = counTRObjectCounter(exemplarImage,exemplarBBoxes)` creates a CounTR object counter model using exemplar patches of objects obtained from an exemplar image, `exemplarImage`, at the exemplar bounding box locations, `exemplarBBoxes`.

Input Arguments

`exemplarPatches` — Exemplar patches

cell array

Exemplar patches containing examples of objects of interest, specified as *numPatches*-by-1 cell array where each cell contains a patch of size *P*-by-*Q*-by-3. The `counTRObjectCounter` object resizes patches to 64-by-64-by-3 and stacks them to form an input of size 64-by-64-by-3-by-*numPatches* to the object counter model. *numPatches* is the number of patches.

Tip To select exemplar patches, crop sections of an image that contain clear, non-overlapping instances of the objects of interest. The patches you specify should be representative of the object class, capturing key features that distinguish the objects from the background or other classes.

`exemplarImage` — Exemplar image

H-by-*W* matrix | *H*-by-*W*-by-*3* numeric array

Exemplar image containing examples of objects of interest, specified as one of these options:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
Binary	*H*-by-*W* logical matrix
RGB	*H*-by-*W*-by-*3* numeric array

`exemplarBBoxes` — Exemplar bounding boxes

numShots-by-4 numeric matrix

Exemplar bounding boxes containing sample objects of interest to count, specified as a *numShots*-by-4 numeric matrix. *numShots* is the number of axis-aligned rectangles. The rows of the matrix have the form [*x y w h*]. *x* and `y` specify the upper-left corner of the rectangle, *w* specifies the width of the rectangle, which is its length along the *x*-axis, and *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Properties

`ExemplarPatches` — Exemplar patches

64-by-64-by-3-by-*numPatches*

This property is read-only.

Exemplar patches used by the CounTR model, represented as a 64-by-64-by-3-by-*numPatches* array. *numPatches* is the number of created patches.

You can create exemplar parches from an image using the `patchextractor` function.

`NumShots` — Number of exemplar bounding boxes

positive numeric scalar

Number of exemplar bounding boxes, represented as a positive integer.

Object Functions

countObjects	Count objects using CounTR deep learning model
densityMap	Density map of object count generated using CounTR deep learning model

Examples

Count Objects in Cell Microscopy Image Using CounTR

Load an exemplar image that contains objects to count, cell nuclei, into the workspace. The exemplar image and the test image in this example are cropped from an image of lymph node cells (`patient_000_node_1.tif`) in the CAMELYON-17 data set.

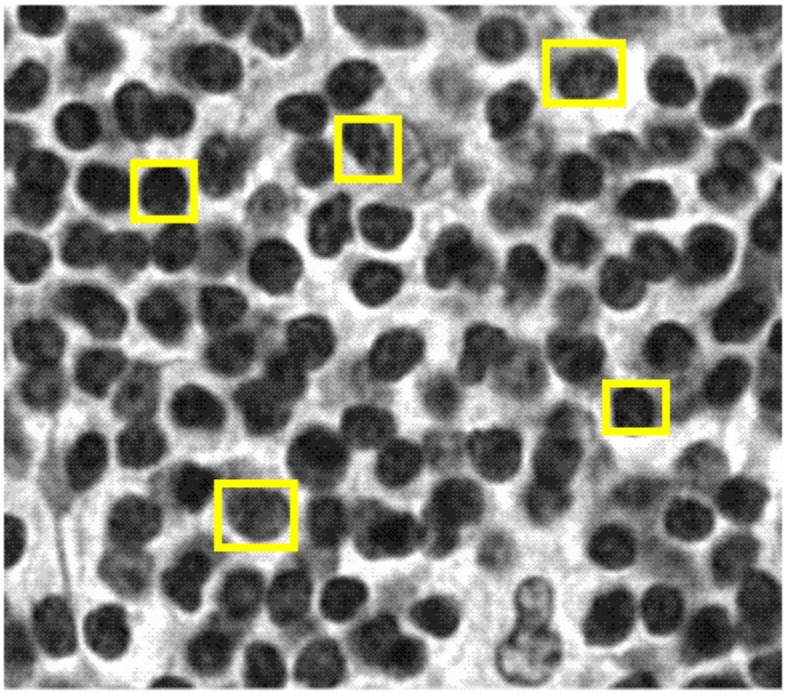
```
exemplarI = imread("nuclei.png");
```

Define exemplar bounding boxes, which signify the locations of sample cells to count.

```
exemplarBoxes = ...
         [272   23   38   30
          302  193   30   24
          66   83   30   28
          108  243   38   32
          168   61   30   30];
```

Display the exemplar image together and the exemplar bounding boxes using the `insertShape` function.

```
annotatedImage = insertShape(exemplarI,"rectangle",exemplarBoxes,"LineWidth",4);
imshow(annotatedImage)
```



Create CounTR Model

Create a CounTR object counter model using the exemplar image and exemplar bounding boxes.

```
counter = counTRObjectCounter(exemplarI,exemplarBoxes);
```

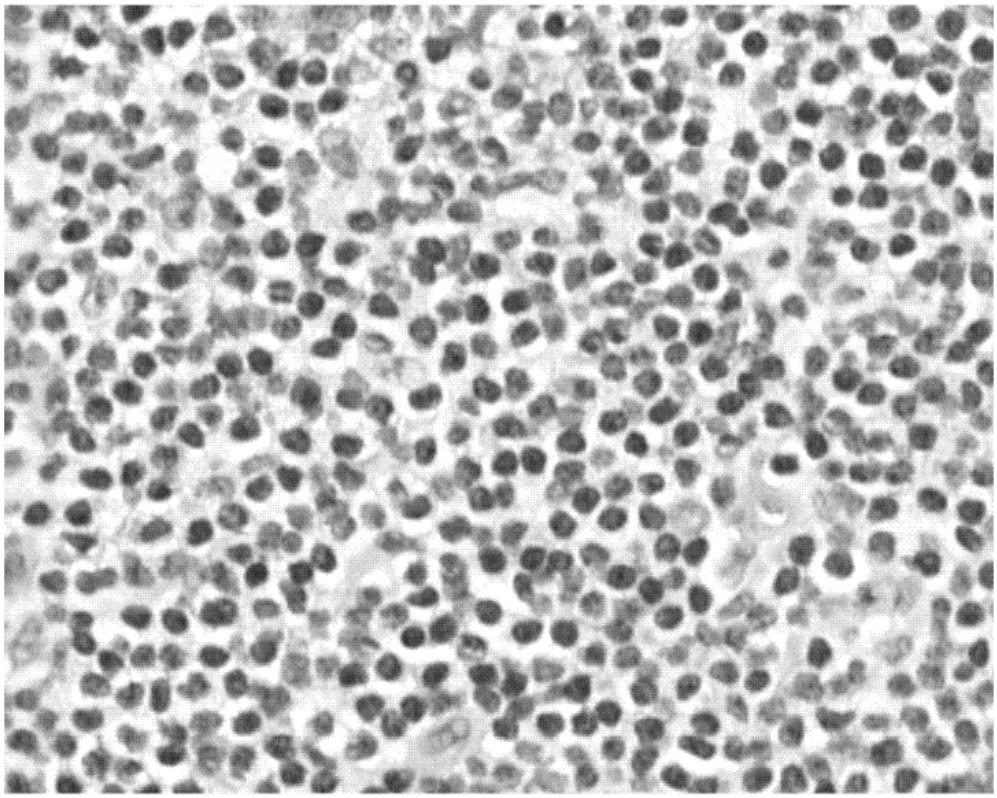
Count Objects in Test Image

Load a test image containing more cell nuclei into the workspace.

```
I = imread("nucleiTest.png");
```

Display the test timage in which to count objects.

```
imshow(I)
```



Compute the number of objects in the test image using the `countObjects` object function.

```
count = countObjects(counter,I);
count
```

```
count = single

498.1617
```

Create Density Map of Object Count

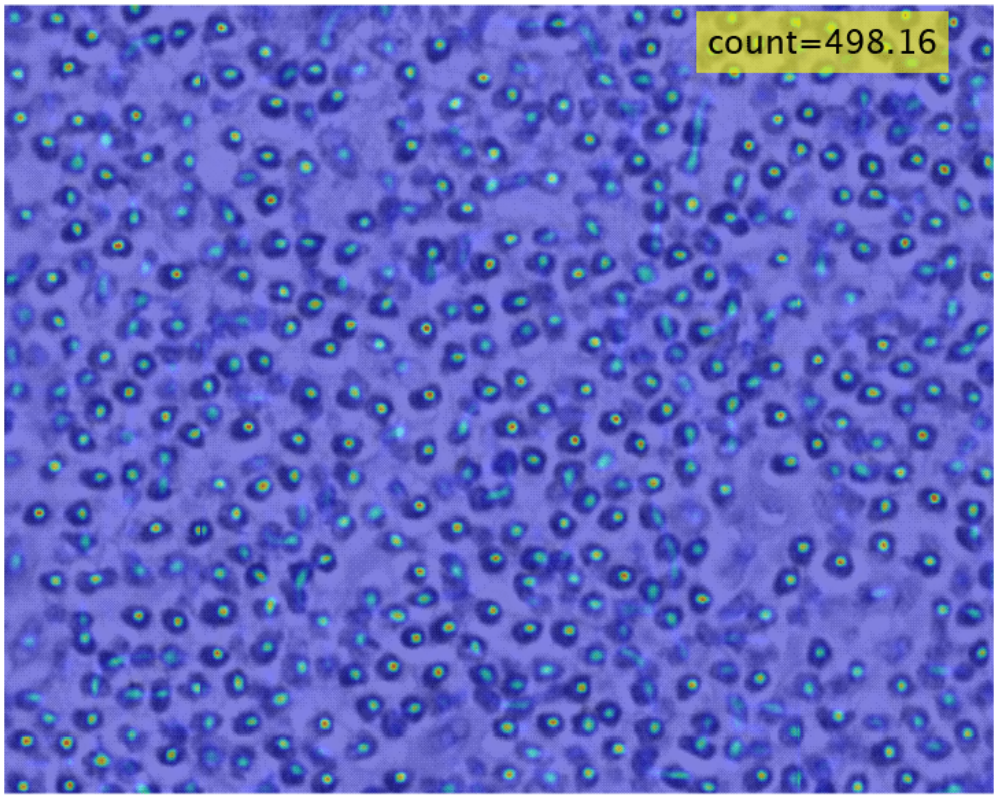
Compute the density map of the object count in the test image using the `densityMap` object function.

```
densMap = densityMap(counter,I);
```

Display the object count density map overlaid over the test image using the `anomalyMapOverlay` function. Display the approximate object count on the overlaid image using the `insertText` function.

```
densityOverlayImage = anomalyMapOverlay(I,densMap,"Blend","equal");
fontSize = min(floor(size(densityOverlayImage,2)/30),200);
```

```
gutterWidth = fontSize*9;
densityOverlayImageWText = insertText(densityOverlayImage,[size(densityOverlayImage,2)-gutterWid
sprintf("count=%0.2f",count),FontSize=fontSize);
imshow(densityOverlayImageWText)
```



Tips

- To select exemplar patches from many images for the CountTR model creation, specify the patches to the `exemplarPatches` input argument. To select exemplar patches from a single image, specify the exemplar image and the object bounding boxes it contains to the `exemplarImage` and `exemplarBBoxes` input argument, respectively.
- To count objects in a datastore of images using the `countObjects` object function, or compute a count density map using the `densityMap` object function, use exemplars in the form of patches or images and bounding boxes. For best model performance, specify exemplars to the `counTRObjectCounter` object that are general enough to apply to the series of images in a datastore.
- For best model performance, use images with well-separated, non-overlapping, objects. To improve performance in images that contain overlapping or occluded objects, crop the image to include only the non-overlapping objects of interest before running the CounTR model. This

ensures that CounTR counts objects only in this region of interest and provides a more accurate count for your specific target objects.

Version History

Introduced in R2025a

References

[1] Liu, Chang, Yujie Zhong, Andrew Zisserman, and Weidi Xie. “CounTR: Transformer-Based Generalised Visual Counting.” *arXiv*, June 2, 2023. https://doi.org/10.48550/arXiv.2208.13721.

[2] grand-challenge.org. “CAMELYON17 - Grand Challenge.” Accessed November 8, 2024. https://camelyon17.grand-challenge.org/Data/.

See Also

`countObjects` | `densityMap` | `uiboxselector` | `patchextractor`

Topics

“Count Objects Using CounTR Model”

countObjects

Count objects using CounTR deep learning model

Syntax

```
count = countObjects(counter,I)
count = countObjects(counter,ds)
count = countObjects( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`count = countObjects(counter,I)` counts objects within an image `I` based on exemplar image patches containing objects.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`count = countObjects(counter,ds)` counts the number of objects within images in a datastore `ds` based on exemplar patches containing objects.

`count = countObjects( ___ ,Name=Value)` specifies options using additional name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MiniBatchSize=4` specifies the mini-batch size as `4`.

Input Arguments

`counter` — CounTR object counter model

`counTRObjectCounter` object

CounTR object counter model, specified as a `counTRObjectCounter` object.

`I` — Image

H-by-*W* matrix | *H*-by-*W*-by-*3* numeric array

Image in which to count objects, specified as one of these values:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
Binary	*H*-by-*W* logical matrix
RGB	*H*-by-*W*-by-*3* numeric array

`ds` — Datastore of images

datastore

Datastore of images, specified as a datastore such as an `ImageDatastore` or `CombinedDatastore` object, in which to count objects. If calling the datastore with the `read` function returns a cell array, then the image data must be in the first cell.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `countObjects(counter,ds,MiniBatchSize=4)` specifies the mini-batch size as 4.

Options for All Image Formats

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to process images with the network, specified as one of the execution environment options in this table.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Options for Datastore Inputs

`MiniBatchSize` — Number of observations returned in each batch
128 (default) | positive integer

Number of observations returned in each batch, specified as a positive integer. If you set a higher `MiniBatchSize`, computation requires more memory, which can cause errors if your system does not have sufficient memory.

You can specify this argument only when you specify an input datastore of images, `ds`.

`Verbose` — Visible progress display
`true` or `1` (default) | `false` or `0`

Visible progress display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

You can specify this argument only when you specify an input datastore of images, `ds`.

Output Arguments

`count` — Object count
positive numeric scalar | *R*-by-1 vector

Object count, returned as a positive numeric scalar for a single image input, or an *R*-by-1 vector for a datastore input. *R* is the number of images in the input datastore, `ds`.

Version History

Introduced in R2025a

See Also

`counTRObjectCounter` | `densityMap`

Topics
“Count Objects Using CounTR Model”

densityMap

Density map of object count generated using CounTR deep learning model

Syntax

```
densMap = densityMap(counter,I)
dsDensMap = densityMap(counter,ds)
densMap = densityMap( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`densMap = densityMap(counter,I)` computes the object count density map for an image `I`.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`dsDensMap = densityMap(counter,ds)` computes the object count density map for datastores of images, `ds`.

`densMap = densityMap( ___ ,Name=Value)` specifies options using additional name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MiniBatchSize=4` specifies the mini-batch size as 4 when you specify an input datastore of images, `ds`.

Examples

Count Objects in Cell Microscopy Image Using CounTR

Load an exemplar image that contains objects to count, cell nuclei, into the workspace. The exemplar image and the test image in this example are cropped from an image of lymph node cells (`patient_000_node_1.tif`) in the CAMELYON-17 data set.

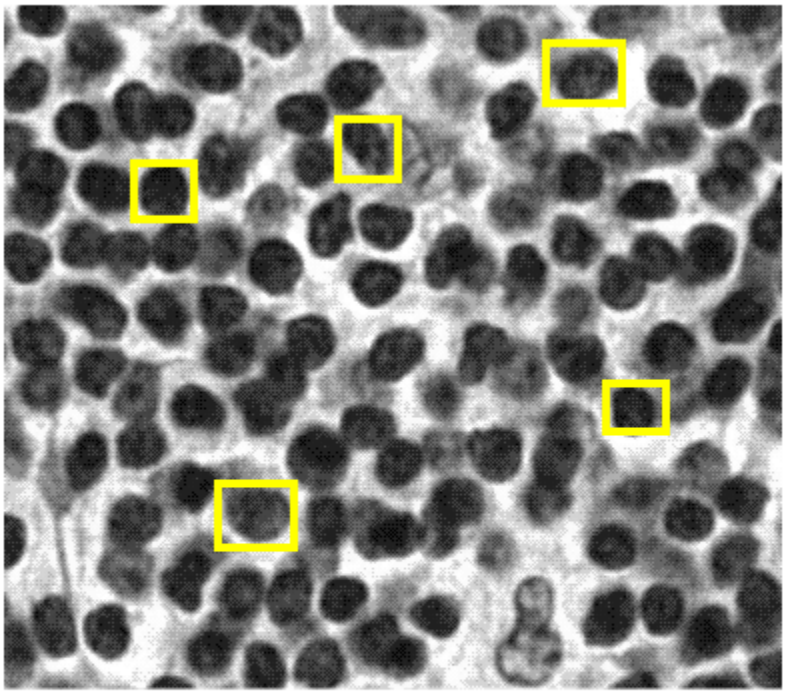
```
exemplarI = imread("nuclei.png");
```

Define exemplar bounding boxes, which signify the locations of sample cells to count.

```
exemplarBoxes = ...
         [272   23   38   30
          302  193   30   24
          66   83   30   28
          108  243   38   32
          168   61   30   30];
```

Display the exemplar image together and the exemplar bounding boxes using the `insertShape` function.

```
annotatedImage = insertShape(exemplarI,"rectangle",exemplarBoxes,"LineWidth",4);
imshow(annotatedImage)
```



Create CounTR Model

Create a CounTR object counter model using the exemplar image and exemplar bounding boxes.

```
counter = counTRObjectCounter(exemplarI,exemplarBoxes);
```

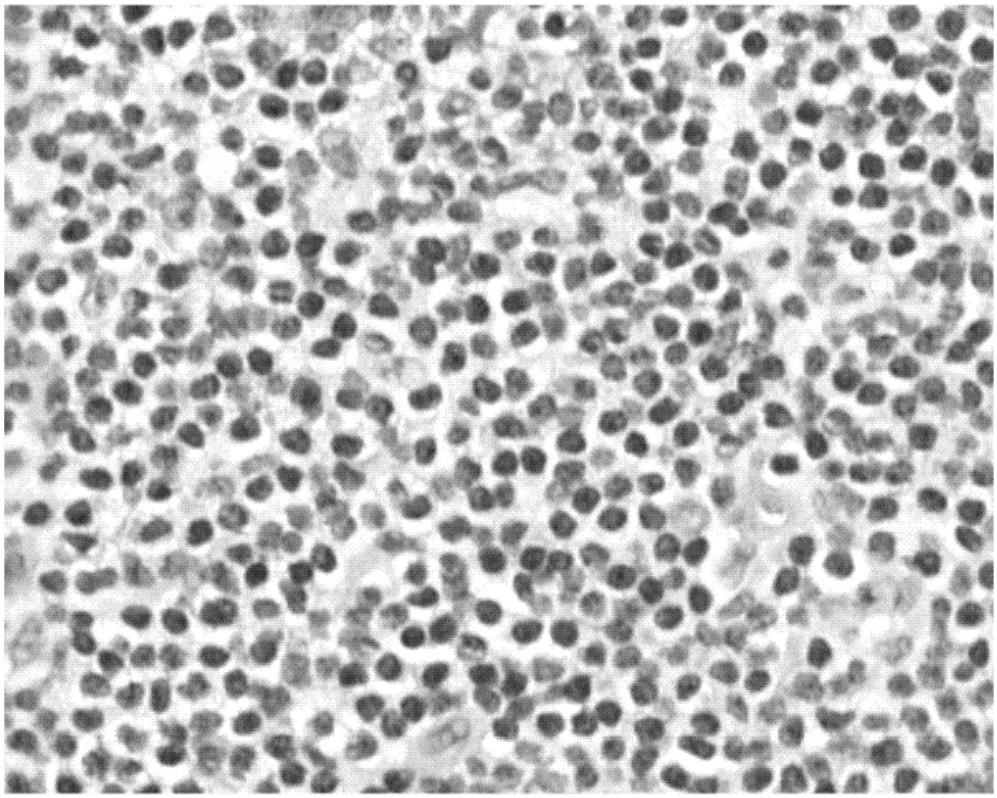
Count Objects in Test Image

Load a test image containing more cell nuclei into the workspace.

```
I = imread("nucleiTest.png");
```

Display the test timage in which to count objects.

```
imshow(I)
```



Compute the number of objects in the test image using the `countObjects` object function.

```
count = countObjects(counter,I);
count
```

```
count = single

498.1617
```

Create Density Map of Object Count

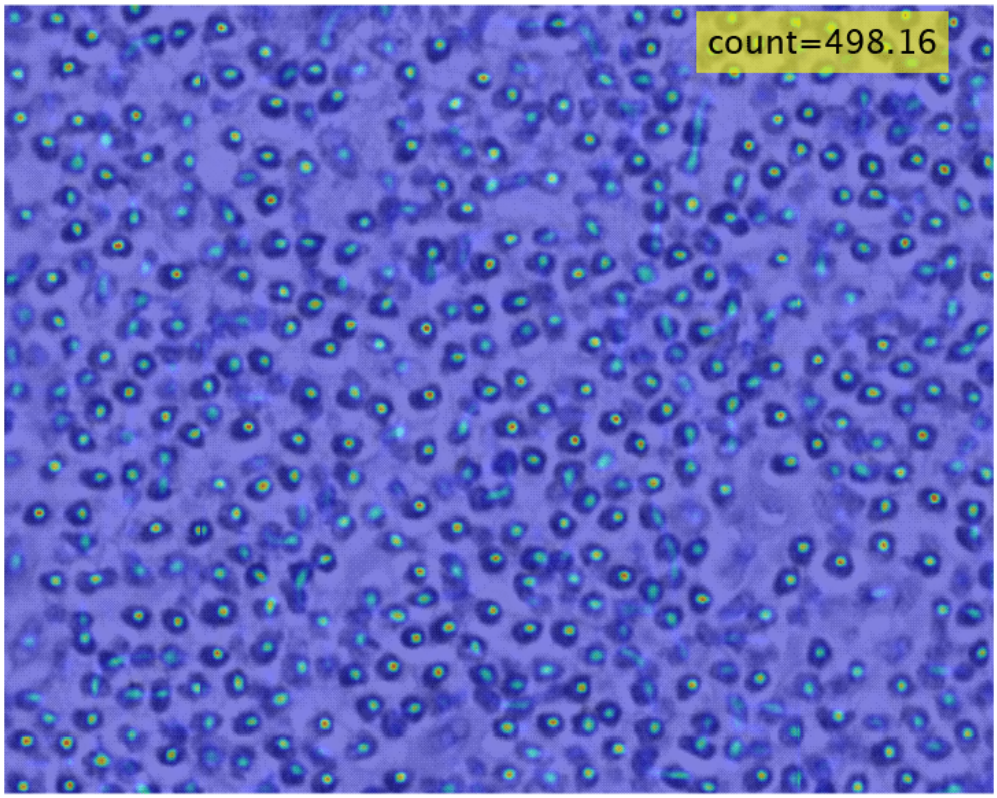
Compute the density map of the object count in the test image using the `densityMap` object function.

```
densMap = densityMap(counter,I);
```

Display the object count density map overlaid over the test image using the `anomalyMapOverlay` function. Display the approximate object count on the overlaid image using the `insertText` function.

```
densityOverlayImage = anomalyMapOverlay(I,densMap,"Blend","equal");
fontSize = min(floor(size(densityOverlayImage,2)/30),200);
```

```
gutterWidth = fontSize*9;
densityOverlayImageWText = insertText(densityOverlayImage,[size(densityOverlayImage,2)-gutterWid
sprintf("count=%0.2f",count),FontSize=fontSize);
imshow(densityOverlayImageWText)
```



Input Arguments

`counter` — CounTR object counter model

`counTRObjectCounter` object

CounTR object counter model, specified as a `counTRObjectCounter` object.

`I` — Image

H-by-*W* matrix | *H*-by-*W*-by-*3* numeric array

Image for which to create an object count density map, specified as one of these values:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix

Image Type	Data Format
Binary	*H*-by-*W* logical matrix
RGB	*H*-by-*W*-by-*3* numeric array

`ds` — Datastore of images
datastore

Datastore of images, specified as a datastore such as an `ImageDatastore` or `CombinedDatastore` object, for which to create object count density maps. If calling the datastore with the `read` function returns a cell array, then the image data must be in the first cell.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `densityMap(counter,ds,MiniBatchSize=4)` specifies the mini-batch size as 4 when you specify an input datastore of images, `ds`.

Options for All Image Formats

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to process images with the network, specified as one of the execution environment options in this table.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Options for Datastore Inputs

`MiniBatchSize` — Number of observations returned in each batch
128 (default) | positive integer

Number of observations returned in each batch, specified as a positive integer. If you set a higher `MiniBatchSize`, density map computation requires more memory, which can cause errors if your system does not have sufficient memory.

You can specify this argument only when you specify an input datastore of images, `ds`.

`WriteLocation` — Location to store writable data
string scalar | character vector

Location to store writable data, specified as a string scalar or character vector. The specified folder must have write permissions. If the folder already exists, the `densityMap` function creates a new folder and adds a suffix to the folder name with the next available number. The default write location is `fullfile(pwd,OutputFolderName)`, where `pwd` is the current working directory and `OutputFolderName` is specified by the `OutputFolderName` name-value argument.

You can specify this argument only when you specify an input datastore of images, `ds`.

Data Types: `char` | `string`

`OutputFolderName` — Output folder name
`"densityMapOutput"` (default) | string scalar | character vector

Output folder name for density map results, specified as a string scalar or a character vector. This folder is in the location specified by the value of the `WriteLocation` name-value argument.

If the output folder already exists, the function creates a new folder with the string `"_1"` appended to the end of the name. Set `OutputFoldername` to `""` to write all the results to the folder specified by `WriteLocation`.

You can specify this argument only when you specify an input datastore of images, `ds`.

`NamePrefix` — Prefix applied to output filenames
`"densityMap"` (default) | string scalar | character vector

Prefix applied to output filenames, specified as a string scalar or character vector. The image files are named as follows:

- *<prefix>*`_`*<N>*`.png`, where *N* corresponds to the index of the input image file, `imds.Files`(*N*).

You can specify this argument only when you specify an input datastore of images, `ds`.

`NameSuffix` — Suffix to add to the output image filename
string scalar | character vector

Suffix to add to the output image filename, specified as a string scalar or a character vector. The function appends the specified suffix to the output filename as:

- *<prefix>*`_`*<N>*`<`*suffix*`>.png`, where *N* corresponds to the index of the input image file, `ds.Files`(*N*).

If you do not specify the suffix, the function uses the input filenames as the output file suffixes. The function extracts the input filenames from the `info` output of the `read` object function of the datastore. When the datastore does not provide the filename, the function does not add a suffix.

You can specify this argument only when you specify an input datastore of images, `ds`.

`Verbose` — Visible progress display
`true` or `1` (default) | `false` or `0`

Visible progress display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

You can specify this argument only when you specify an input datastore of images, `ds`.

Output Arguments

densMap — Density map

P-by-*Q* numeric matrix | datastore

Density map of object count, returned as a *P*-by-*Q* numeric matrix when you specify an input image `I`, or a datastore containing *R* density maps of size *P*-by-*Q* when you specify a datastore input `ds`. *R* is the number of images in the datastore.

Version History

Introduced in R2025a

References

[1] Liu, Chang, Yujie Zhong, Andrew Zisserman, and Weidi Xie. “CounTR: Transformer-Based Generalised Visual Counting.” *arXiv*, June 2, 2023. https://doi.org/10.48550/arXiv.2208.13721.

[2] grand-challenge.org. “CAMELYON17 - Grand Challenge.” Accessed November 8, 2024. https://camelyon17.grand-challenge.org/Data/.

See Also

`counTRObjectCounter` | `countObjects`

Topics

“Count Objects Using CounTR Model”
“Get Started with Image Preprocessing and Augmentation for Deep Learning”

objectInsertionDatastore

Create synthetic labeled datastore for training an instance segmentation or object detection network

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `objectInsertionDatastore` object blends datastores of labeled images containing foreground objects with destination images containing the background. Using this object, you can create a synthetic labeled datastore for training an instance segmentation or object detection network.

To insert an object into a single background image at a randomized or specified location, use the `insertObjectInImage` function.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
dsNew = objectInsertionDatastore(dsDestination,dsObject,numNewImages)
dsNew = objectInsertionDatastore( ___ ,PropertyName=Value)
```

Description

`dsNew = objectInsertionDatastore(dsDestination,dsObject,numNewImages)` creates an augmented datastore `dsNew` by inserting objects from the source datastore `dsObject` into the destination datastore containing image data, `dsDestination`. The objects from the source images are inserted at randomized locations on the destination images. You must specify the number of images synthetic images to create, `numNewImages`.

`dsNew = objectInsertionDatastore( ___ ,PropertyName=Value)` sets writeable properties using one or more name-value arguments, in addition to the input argument from the previous syntax. For example, `NumObjectsToInsert=2` specifies the number of objects to insert as 2.

Input Arguments

`dsDestination` — Destination datastore
datastore

Destination datastore of images, specified as a datastore such as an `ImageDatastore` or `CombinedDatastore` object.

To insert objects at specific image locations using a region constraint mask, specify the `dsDestination` argument as a datastore that returns a cell array with two columns, `{Image`

`ObjectMask}`, when called with the `read` function. `Image` is an array of RGB or grayscale images, and `ObjectMask` is a logical array of masks which span object regions to insert.

`dsObject` — Datastore of source images containing objects
datastore

Datastore of source images containing objects, specified as a datastore such as an `ImageDatastore` or `CombinedDatastore` object. You must set up your data so that calling the `read` and `readall` functions on the datastore returns a cell array with these columns.

Image Data	Boxes	Labels	Masks
RGB or grayscale image containing objects to insert, specified as an *H*-by-*W*-by-3 or *H*-by-*W*-by-1 numeric array, respectively.	Bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.	Object class names, specified as an *M*-by-1 categorical vector, where *M* is the number of specified bounding boxes. All categorical data read from the datastore must contain the same categories.	Binary masks, specified as a logical array of size *H*-by-*W*-by-*M*, where *M* is the number of specified bounding boxes. Each channel is a mask, and each mask is the segmentation of one object in the image.

`numNewImages` — Number of augmented images
positive integer

Number of augmented images to create, specified as a positive integer. In every call to the datastore with the `read` function, the `objectInsertionDatastore` object inserts random object from the datastore `dsObject` into a random image from the datastore `dsDestination`.

Properties

`NumObjectsToInsert` — Number of objects to insert
1 (default) | positive integer | range of positive integers

Number of objects to insert from the source datastore, `dsObject`, specified as a positive integer or a range of positive integers [*minNum*,*maxNum*]. If you specify a range of positive integers [*minNum*,*maxNum*], the `objectInsertionDatastore` object inserts a random number of objects into each image in the destination datastore `dsDestination`.

`OutputFormat` — Output format
`"InstanceSegmentation"` (default) | `"ObjectDetection"` | `"SceneClassification"`

Output format of the data returned by calling the augmented datastore `dsNew` with the `read` function, specified as one of these options:

- `"InstanceSegmentation"` – Instance segmentation data containing image data, bounding boxes, masks, and labels
- `"ObjectDetection"` – Object detection data containing image data, bounding boxes and labels
- `"SceneClassification"` – Scene classification, or semantic segmentation data, containing image data and masks

`ObjectsInSceneMaxOverlap` — Maximum object overlap
`1` (default) | positive scalar in range `[0, 1]`

Maximum object overlap between an object in an image from the datastore `dsObject` and a previously inserted object, specified as a positive scalar in range `[0, 1]`. Specify the `ObjectsInSceneMaxOverlap` value as `1` to insert new objects on top of existing objects. Decrease this value to decrease the overlap between the newly inserted objects and existing objects.

`BoundaryConstraintMode` — Boundary constraint mode
`"inbounds-bbox"` (default) | `"inbounds-exact"` | `"center"`

Boundary constraint mode used to constrain objects at the boundaries of images in the destination datastore `dsDestination`, specified as one of these options:

Value	Edge Detection Mechanism
`"inbounds-bbox"`	To insert an object, the entire bounding box containing the object must be completely within the destination image.
`"inbounds-exact"`	To insert an object, the object mask must be completely within the destination image.
`"center"`	Objects are inserted when the object centroid lies inside the destination image, so objects can be partially visible and clipped at the image boundary.

`MaxInsertionAttempts` — Maximum insertion attempts
`10` (default) | positive scalar

Maximum insertion attempts to satisfy the overlap value, specified as a positive scalar. The overlap value is represented by the `ObjectsInSceneMaxOverlap` property. If no insertion location can be found on the destination image after `MaxInsertionAttempts` iterations, the object from the source datastore image will not be inserted into the destination datastore image.

`BlendMethod` — Blend method
`"guidedfilter"` (default) | `"poisson"` | `"none"`

Blend method to use for inserting objects from the images in the source datastore `dsObject` into the images in the destination datastore `dsDestination`, specified as a one of these options:

BlendMethod Value	Blend Method
"guidedfilter"	Guided filter blending [1] – Blur the object mask of the source image at the mask-object boundary. Guided image filtering uses edge context from the source image to filter the mask prior to blending the object image with the destination image.
"poisson"	Poisson blending [2] – Blend the source object into the destination image by matching the color distribution of the source object to that of the boundary between the destination image and source object mask.
"none"	No blending – At insertion locations of the destination image, replace pixels of the destination image by the source image pixel values.

`GeometricAugmentation` — Geometric augmentation
`[]` (default) | function handle

Geometric augmentation, specified as a function handle that returns an `affinetform2d` object. The geometric augmentation warps a source image and the mask associated with each object in the image before it is inserted into the destination image. For example, specify the `GeometricAugmentation` value as `@() randomAffine2d("Scale",[0.9,1.1])` to apply a randomized scale transformation to each inserted object.

Version History
Introduced in R2025a

References

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. “Guided Image Filtering.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, no. 6 (June 2013): 1397–1409. doi:10.1109/TPAMI.2012.213.

[2] Pérez, Patrick, Michel Gangnet, and Andrew Blake. “Poisson Image Editing.” In *ACM SIGGRAPH 2003 Papers*, 313–18. San Diego California: ACM, 2003. doi:10.1145/1201775.882269.

See Also
`insertObjectInImage` | `ImageDatastore` | `imblend` | `affinetform2d` | `insertObjectMask` | `imguidedfilter`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Get Started with Image Preprocessing and Augmentation for Deep Learning”

uicaliper

Interactively measure distances in images using caliper tool

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

The `uicaliper` object displays a caliper tool on an image object that you create using the `imageshow` function. You can use the caliper tool to interactively detect edges and measure distances in the image.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Syntax

```
hCaliper = uicaliper(hImage)
hCaliper = uicaliper(hImage,PropertyName=Value)
```

Description

`hCaliper = uicaliper(hImage)` configures a caliper tool on the displayed image `hImage`.

`hCaliper = uicaliper(hImage,PropertyName=Value)` sets writable properties using one or more name-value arguments.

For example, `Width=10` specifies the width of the profile scan as 10 pixels.

Input Arguments

`hImage` — Input image

`Image` object

Input image, specified as an `Image` object. To create an `Image` object, use the `imageshow` function.

Properties

`Position` — Profile center line position

2-by-2 numeric matrix

Profile center line position, in pixels, specified as a 2-by-2 numeric matrix of the form $[x_1\ y_1; x_2\ y_2]$, where the $[x_1\ ,\ y_1]$ and $[x_2\ ,y_2]$ are the first and last coordinates defining the line, respectively.

You can specify `Position` programmatically in the MATLAB Command Window, or interactively by clicking a start point in the specified image `hImage` and dragging to your desired end point.

`Width` — Profile scan width
non-negative scalar

Profile scan width, specified as a positive scalar. The width is the area within which the `uicaliper` object computes profile scan lines measurements. The caliper tool scans in lines orthogonal to the profile centerline and using these measurements, computes a mean 1-D profile for gradient detection. If you specify `Width` as `0`, the caliper tool scans the profile along a single line, with no additional sampling scan lines that smooth out the results. By default, the profile scan width is 10% of the length of the caliper tool.

To interactively adjust the profile scan width, configure the caliper tool on a displayed image by dragging the tool between a start and end point. Once you release the tool, you can click and drag the handles perpendicular to the tool to adjust the scan width.

`EdgeTransition` — Edge transition type
`"both"` (default) | `"rising"` | `"falling"`

Edge transition type used in edge detection, specified as one of these options:

- `"rising"` — Detect darker-to-brighter edge transitions.
- `"falling"` — Detect brighter-to-darker edge transitions.
- `"both"` — Detect both brighter-to-darker and darker-to brighter-edge transitions.

Tip When you set the `MeasurementMode` property to `"edge-pairs"`, specify the edge transition with respect to the first edge in the profile scan line. For example, to detect edge pairs in which the first edge is rising and the second edge is falling, specify `EdgeTransition` as `"rising"`.

`GradientThreshold` — Gradient threshold selection
`0.1` (default) | positive scalar in range [0, 1]

Gradient threshold for a gradient to be considered an edge, specified as a positive scalar in range [0, 1]. The gradient at each scan point is the normalized average gradient across the profile line width, `Width`. The `uicaliper` object function evaluates this value at each point along the scan line. Increase the gradient threshold value to detect fewer false positive edges.

`MeasurementMode` — Measurement mode
`"edge-pair"` (default) | `"single-edge"`

Measurement mode used to compute edge data, specified as `"edge-pair"` or `"single-edge"`. This argument determines how the `measurementData` argument outputs distance.

Value	Edge Detection Mode
`"edge-pair"`	Detect edge pairs of the type specified by the `EdgeTransition` property. When the profile scan line spans multiple edges, the caliper tool pairs each starting edge with the spatially closest ending edge to form a pair.

Value	Edge Detection Mode
`"single-edge"`	Detect single edges of the type specified by the `EdgeTransition` property.

`Sigma` — Standard deviation for Gaussian smoothing of 1-D profile data
`1.0` (default) | positive scalar

Standard deviation for Gaussian smoothing of the 1-D profile data, specified as a positive scalar. Increase this value to increase the amount of Gaussian smoothing and reduce measurement noise at the possible expense of lower accuracy in detecting edge positions.

`Color` — Caliper tool color
`[0.0 0.5610 1.0]` (default) | short or long color name | RGB triplet

Caliper tool color, specified as an RGB triplet, or a long or short color name.

`DisplayProfile` — Option to display 1-D profile scan lines
`true` or `1` (default) | `false` or `0`

Option to display 1-D profile scan lines, specified as a numeric or logical `1` (`true`) or `0` (`false`). By default, the `uicaliper` function overlays the 1-D profile scan line over the displayed image.

`DisplayDistance` — Option to display distance measurement value as text
`true` or `1` (default) | `false` or `0`

Option to display the distance measurement value as text overlaid on the image, specified as a numeric or logical `1` (`true`) or `0` (`false`).

`EdgeDistance` — Distance from each detected edge to profile scan line origin
row vector

This property is read-only.

Distance from each detected edge profile scan line origin, represented as a row vector.

`IntraEdgeDistance` — Distance between start and end of each edge pair
row vector

This property is read-only.

Distance between the start and end of each edge pair, represented as a row vector.

`InterEdgeDistance` — Distance between edge pairs
row vector

This property is read-only.

Distance between edge pairs, represented as a row vector. The caliper tool measures this distance from the end of each edge pair to the start of the next edge pair.

`GradientValue` — Gradient value corresponding to each detected edge
N-element row vector

This property is read-only.

Gradient value corresponding to each detected edge, represented as an *N*-element row vector. Each element of the row vector is the gradient value associated with each detected edge. *N* depends on the `MeasurementMode` value.

- `"single-edge"` — *N* is the number of total detected edges.
- `"edge-pair"` — *N* is the number of edges that comprise edge pairs.

`ProfileData` — 1-D profile data that determines detected edges
row vector

This property is read-only.

1-D profile data that determines detected edges, represented as an *M*-element numeric vector. *M* is the integer length of the profile scan, in pixels, and each element of the vector is the mean gradient value at each scan point.

This profile data determines the gradient values for each edge, stored in the `GradientValue` property.

Note Specify the amount of Gaussian smoothing of the profile data using the `Sigma` property.

Examples

Interactively Measure Distances on Image Using Caliper Tool

Load an image into the workspace.

```
I = imread("coins.png");
```

Display the image using the `imageshow` function.

```
hIm = imageshow(I);
```

To interactively measure distances, configure the caliper tool on the displayed image using the `uicaliper` object. Click a start point in the image and drag the end point of the caliper tool to a desired location.

```
hCal = uicaliper(hIm)
```

This GIF shows how to perform multiple measurements by dragging the end points of the caliper tool to different positions on the image.



Version History

Introduced in R2025a

See Also

Functions

`caliper` | `imageshow`

anomalyDetectionMetrics

Anomaly detection metrics

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

An `anomalyDetectionMetrics` object encapsulates anomaly detection quality metrics for a set of images. Images of anomalies belong to the positive (`true`) class. Normal images belong to the negative (`false`) class.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Creation

Create an `anomalyDetectionMetrics` object using the `evaluateAnomalyDetection` function.

Properties

`ConfusionMatrix` — Confusion matrix
table

This property is read-only.

Confusion matrix, specified as a 2-by-2 table. Each table element (*i*, *j*) is the count of images known to belong to class *i* but predicted to belong to class *j*.

`NormalizedConfusionMatrix` — Normalized confusion matrix
table

This property is read-only.

Normalized confusion matrix, specified as a 2-by-2 table. `NormalizedConfusionMatrix` represents a confusion matrix normalized by the number of images known to belong to each class. Each table element (*i*, *j*) is the count of images known to belong to class *i* but predicted to belong to class *j*, divided by the total number of images predicted in class *i*. Elements are in the range [0, 1].

`DataSetMetrics` — Metrics aggregated over data set
table

This property is read-only.

Metrics aggregated over the data set, specified as a 1-by-8 table with columns corresponding to these metrics:

- `GlobalAccuracy` — Ratio of correctly classified images to total images, regardless of class
- `MeanAccuracy` — Ratio of correctly classified images of each class to total images, averaged over all classes
- `Precision` — Positive predictive value
- `Recall` — True positive (TP) rate
- `Specificity` — True negative (TN) rate
- `F1Score` — Harmonic mean of precision and recall
- `FalsePositiveRate` — False positive (FP) rate, or fallout
- `FalseNegativeRate` — False negative (FN) rate, or miss

`ClassMetrics` — Class metrics
table

This property is read-only.

Anomaly detection metrics for each class, specified as a 2-by-2 table. The table has rows "Normal" and "Anomaly" and columns corresponding to these metrics:

- `Accuracy` — Fraction of correctly classified images for each class (normal or anomaly).
- `AccuracyPerSubClass` — Fraction of correctly classified images for each subclass within the normal and anomaly classes. The value of `AccuracyPerSubClass` is a table with *C* rows and one column, where *C* is the number of subclasses.

Examples

Calculate Metrics for Trained FCDD Anomaly Detector

Load test images and corresponding labels, then create a datastore that reads the test data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
dsTest = arrayDatastore(Xtest,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of anomaly labels as the set of digits between 0 and 9, excluding 8.

```
load("digit8AnomalyDetector.mat");
anomalyLabels = setdiff(string(0:9),"8");
```

Classify each test image as normal or anomalous.

```
predLabels = classify(detector,dsTest);
```

Calculate metrics for the anomaly detection results.

```
metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels)
```

```
Evaluating anomaly detection results
------------------------------------
* Finalizing... Done.
* Data set metrics:
```

```
    GlobalAccuracy    MeanAccuracy    Precision    Recall    Specificity    F1Score    FalsePosi
    ______________    ____________    _________    ______    ___________    _______    _________

        0.7662            0.839         0.98971     0.748        0.93        0.85204          0.0

metrics =
  anomalyDetectionMetrics with properties:

              ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
               DataSetMetrics: [1×8 table]
                 ClassMetrics: [2×2 table]
```

Display the confusion matrix.

```
cm = metrics.ConfusionMatrix
```

```
cm=2×2 table
                Normal    Anomaly
                ______    _______

    Normal        465         35
    Anomaly      1134       3366
```

Version History

Introduced in R2022b

See Also

`evaluateAnomalyDetection`

clipNetwork

Create pretrained CLIP deep learning neural network for vision-language tasks

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for OpenAI CLIP Network add-on.

The `clipNetwork` object configures a pretrained Contrastive Language-Image Pre-Training (CLIP) network. Use the CLIP network to connect and compare images and text for tasks like image classification, retrieval, and zero-shot learning.

Use the CLIP network for image retrieval and image classification.

- Image retrieval — To search for images that best match a text query, first extract embeddings for each of the images in the data set using the `extractImageEmbeddings` object function. Then, extract the embeddings for the text search terms using the `extractTextEmbeddings` object function, compute the similarity between the text embeddings and the image embeddings, and select the classes with the closest match.
- Zero-shot classification without retraining — To classify images by comparing extracted image features to the features of candidate class names or descriptions, use the `classify` object function and select the classes with the closest match.

To perform a forward pass on the image and text encoders prior to training the CLIP network, use the `forward` object function.

Note This functionality requires Deep Learning Toolbox.

Creation

Syntax

```
net = clipNetwork(backbone)
net = clipNetwork(backbone,Name=Value)
```

Description

`net = clipNetwork(backbone)` creates a pretrained CLIP network with a pretrained image encoder, `backbone`.

`net = clipNetwork(backbone,Name=Value)` sets writable properties using one or more name-value arguments. For example, `ModelName="TrainedCLIP"` specifies the CLIP network name as `"TrainedCLIP"`.

Input Arguments

backbone — Pretrained image encoder backbone
`"vit-b-16"` | `"vit-l-14"` | `"resnet50`

Pretrained image encoder backbone of the CLIP network, specified as one of these options:

- `"vit-b-16"` — ViT-B/16 (Vision Transformer Base with 16-by-16 input patches).
- `"vit-l-14"` — ViT-L/14 (Vision Transformer Large with 14-by-14 input patches).
- `"resnet50"` — ResNet-50 deep convolutional neural network, scaled by 16 times.

The `clipNetwork` function uses the backbone image encoder to process input images and convert them into high-dimensional feature vectors.

Data Types: `char` | `string`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `clipNetwork("resnet50",ModelName="trainedCLIP")` specifies the CLIP network model name as `"TrainedCLIP"`.

`ModelName` — Name of pretrained CLIP network model
`""` (default) | string scalar | character vector

Name of the pretrained CLIP network model, specified as a string scalar or a character vector.

`ImageNormalizationStatistics` — *Z*-score normalization statistics
`[]` (default) | structure

Z-score normalization statistics, specified as a structure with these fields:

Field	Description	Default Value
`Mean`	1-by-*C* vector of means per channel.	`[122.7709 116.7460 104.0937]`
`StandardDeviation`	1-by-*C* vector of standard deviations per channel.	`[68.5005 66.6321 70.3232]`

C is the number of channels in each image.

Properties

`ModelName` — Name of pretrained CLIP network model
`""` (default) | string scalar | character vector

This property is read-only after object creation. To set this property, use the `ModelName` argument during object creation.

Name of the pretrained CLIP network model, represented as a string scalar or a character vector.

`ImageNormalizationStatistics` — *Z*-score normalization statistics
`[]` (default) | structure

This property is read-only after object creation. To set this property, use the `ImageNormalizationStatistics` argument during object creation.

Z-score normalization statistics, represented as a structure with these fields:

Field	Description	Default Value
`Mean`	1-by-*C* vector of means per channel.	`[122.7709 116.7460 104.0937]`
`StandardDeviation`	1-by-*C* vector of standard deviations per channel.	`[68.5005 66.6321 70.3232]`

The number of channels *C* must match the number of channels in each image.

`TextEncoderNetwork` — Network containing CLIP text encoder parameters
`dlnetwork` object

This property is read-only.

Network containing the CLIP text encoder parameters, represented as a `dlnetwork` object.

`ImageEncoderNetwork` — Network containing CLIP image encoder parameters
`dlnetwork` object

This property is read-only.

Network containing the CLIP image encoder parameters, represented as a `dlnetwork` object.

Object Functions

classify	Classify image using CLIP network
extractImageEmbeddings	Extract feature embeddings from image using CLIP network image encoder
extractTextEmbeddings	Extract text embeddings from search text using CLIP network text encoder
forward	Run forward pass on CLIP network

Examples

Create CLIP Network

Create a pretrained CLIP network.

```
clip = clipNetwork("resnet50")
```

```
clip =
  clipNetwork with properties:

                     ModelName: "resnet50"
            ImageEncoderNetwork: [1×1 dlnetwork]
             TextEncoderNetwork: [1×1 dlnetwork]
   ImageNormalizationStatistics: [1×1 struct]
```

Perform Image Retrieval using CLIP Network

Create a pretrained CLIP network with the ViT-B/16 backbone.

```
clip = clipNetwork("vit-b-16");
```

Create a datastore of images, `imds`, and display a montage of the images.

```
pathToImages = fullfile(toolboxdir("vision"),"visiondata","imageSets");
imds = imageDatastore(pathToImages,IncludeSubfolders=true);
montage(imds)
```



Extract Image Embeddings

Extract the feature embeddings for each image in the datastore.

```
imageEmbeddings = extractImageEmbeddings(clip,imds);
```

Extract Text Embeddings

Define a search text search term, and extract the text embeddings using the CLIP network encoder. You can modify the search term to empirically test image retrieval.

```
search = "A photo of a children's book.";
textEmbeddings = extractTextEmbeddings(clip,search);
```

Retrieve Images Related to Search Text

Compute the cosine similarity between the text embeddings and all image embeddings.

```
function simScores = cosineSim(x,y)
    simScores = (x'*y)'./(vecnorm(x)*norm(y));
end
simScores = cosineSim(imageEmbeddings,textEmbeddings)
```

```
simScores = 1×12 single row vector

    0.3019    0.3202    0.2359    0.2261    0.1954    0.2838    0.1628    0.1735    0.2029    0.1
```

Identify the three highest similarity scores.

```
[~,topIdx] = maxk(simScores,3);
```

Display the images with the highest similarity scores related to the search text.

```
imageFiles = imds.Files(topIdx);
montage(imageFiles)
```



Version History

Introduced in R2026a

References

[1] Radford, Alec, et al. Learning Transferable Visual Models From Natural Language Supervision. arXiv, 2021. DOI.org (Datacite), https://doi.org/10.48550/ARXIV.2103.00020.

See Also

`extractImageEmbeddings` | `extractTextEmbeddings` | `classify` | `forward` | `visionTransformer`

Topics

“Get Started with Vision-Language Models”

classify

Classify image using CLIP network

Syntax

```
classes = classify(clip,I,classNames)
[classes,scores] = classify(clip,I,classNames)
[___] = classify(___,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for OpenAI CLIP Network add-on.

`classes = classify(clip,I,classNames)` assigns each image in `I` to one of the suggested classes `classNames` using a Contrastive Language-Image Pre-Training (CLIP) network.

Note This functionality requires Deep Learning Toolbox.

`[classes,scores] = classify(clip,I,classNames)` additionally returns the CLIP network prediction scores corresponding to the predicted classes `classes`.

`[___] = classify(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MiniBatchSize=32` limits the batch size to 32 images.

Examples

Classify Images in Datastore Using CLIP Network

Create a pretrained CLIP network.

```
clip = clipNetwork("vit-b-16");
```

Create a datastore of test images.

```
imageFiles = ["kobi.png","baby.jpg","flamingos.jpg","saturn.png"];
imds = imageDatastore(imageFiles);
```

Define the list of class suggestions for the test images.

```
classNames = ["baby","dog","flamingo","planet"];
```

Obtain the predicted classes for each image in the datastore.

```
class = classify(clip,imds,classNames);
```

Display the images along with their predicted classes.

```
figure
tiledlayout(2,2)
```

```
for i = 1:length(imageFiles)
    nexttile
    imshow(read(imds))
    title(class(i))
end
```



Classify Image Using CLIP Network

Create a pretrained CLIP network with a RestNet-50 backbone.

```
clip = clipNetwork("resnet50")
```

```
clip =
  clipNetwork with properties:

                      ModelName: "resnet50"
             ImageEncoderNetwork: [1×1 dlnetwork]
              TextEncoderNetwork: [1×1 dlnetwork]
    ImageNormalizationStatistics: [1×1 struct]
```

Load an image that contains the object to classify into the workspace, and display the image.

```
I = imread("kobi.png");
imshow(I)
```



Define the list of potential classes for the image.

```
classNames = ["aardvark","bee","cat","dog"];
```

Obtain the predicted class and prediction scores from the image.

```
[class,scores] = classify(clip,I,classNames)
```

```
class = categorical
     dog


scores = 1×4 single row vector

    0.5309    0.5131    0.5337    0.7217
```

Classify Image with Custom Descriptions Using CLIP Network

Create a pretrained CLIP network.

```
clip = clipNetwork("vit-l-14");
```

Load a satellite photo of the town of Concord, Massachusetts into the workspace, and display the image.

```
I = imread("concordaerial.png");
imshow(I)
```



Define the list of class suggestions for the image. These classes are town or city names.

```
classNames = ["Boston","Concord","Plymouth","Falmouth"];
```

Define class descriptions that provide more context to the CLIP model for more accurate classification.

```
classDescriptions = [ ...
    "A satellite photo of Boston, a city in Massachusetts."
    "A satellite photo of Concord, a suburb in Massachusetts."
    "A satellite photo of Plymouth, a town on the coast of Massachusetts."
    "A satellite photo of Falmouth, a town on Cape Cod in Massachusetts."
    ];
```

Specify the suggested class names for each of the towns, as well as the more detailed class descriptions, to predict the town shown in the image using the CLIP network.

```
class = classify(clip,I,classNames,ClassDescriptions=classDescriptions)
```

```
class = categorical
     Concord
```

Input Arguments

`clip` — CLIP network

`clipNetwork` object

CLIP network, specified as a `clipNetwork` object.

`I` — Image data

numeric array | datastore | formatted `dlarray` object

Image data, specified in one of these formats:

- *H*-by-*W*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.
- *H*-by-*W*-by-1-by-*B* numeric array representing a batch of *B* grayscale images.
- Datastore that reads and returns truecolor images.
- Formatted `dlarray` object with two spatial dimensions of the format `"SSCB"`. You can specify multiple test images by including a batch dimension.

`classNames` — Names of class suggestions
categorical vector | vector of strings

Names of class suggestions, specified as a vector of strings or a categorical vector. You must specify class names in English using ASCII characters. The function automatically pads or truncates each text input so that it contains exactly 77 tokens.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `classify(clip,I,classNames,MiniBatchSize=32)` limits the batch size to 32 images.

`MiniBatchSize` — Size of batches
8 (default) | positive integer

Size of batches for processing large collections of images, specified as a positive integer. Larger batch sizes reduce processing time, but require more memory.

`ClassDescriptions` — Class descriptions
`"auto"` (default) | *C*-element string array

Class descriptions used for classification by the CLIP network, specified as a *C*-element string array. *C* is the number of classes in `classNames`. By default, the CLIP model generates class descriptions from the labels specified by the `classNames` input argument.

Use the `ClassDescriptions` name-value argument to create custom class descriptions. The `classify` function pads each description string with zeros or shortens it so that it contains exactly 77 tokens.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.

Resource	Action
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`classes` — Predicted classes

B-element categorical vector

Predicted classes, returned as a *B*-element categorical vector. *B* is the number of images in the batch.

`scores` — Prediction scores

B-by-*C* numeric matrix

Prediction scores, returned as a *B*-by-*C* numeric matrix. *B* is the number of images in the batch, and *C* is the number of suggested classes specified using the `classNames` input argument.

The `classify` function computes the scores using the CLIPScore algorithm. For an input image *I* and associated text *T*, the algorithm computes the score using the equation

$$\text{CLIPScore}(I, T) = 2.5 \cdot \max(\cos(I, T), 0) .$$

Version History

Introduced in R2026a

See Also

`clipNetwork`

extractImageEmbeddings

Extract feature embeddings from image using CLIP network image encoder

Syntax

```
imageEmbeddings = extractImageEmbeddings(clip,I)
imageEmbeddings = extractImageEmbeddings(clip,I,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for OpenAI CLIP Network add-on.

`imageEmbeddings = extractImageEmbeddings(clip,I)` extracts the feature embeddings from the image data `I` using the image encoder of a Contrastive Language-Image Pre-Training (CLIP) network, `clip`, by running a forward pass on the neural network.

Note This functionality requires Deep Learning Toolbox.

`imageEmbeddings = extractImageEmbeddings(clip,I,Name=Value)` specifies options using one or more name-value arguments. For example, `MiniBatchSize=32` limits the batch size to 32 images.

Examples

Extract Image Embeddings Using CLIP Network Encoder

Create a pretrained CLIP network with a ViT-B/16 backbone.

```
clip = clipNetwork("vit-b-16");
```

Create a datastore of images, `imds`, and display a montage of the images.

```
pathToImages = fullfile(toolboxdir("vision"),"visiondata","imageSets");
imds = imageDatastore(pathToImages,IncludeSubfolders=true);
montage(imds)
```



Extract the feature embeddings for each image in the datastore.

```
imageEmbeddings = extractImageEmbeddings(clip,imds);
```

Display the size of the feature embeddings.

```
size(imageEmbeddings)
```

```
ans = 1×2

   512    12
```

Input Arguments

`clip` — CLIP network

`clipNetwork` object

CLIP network, specified as a `clipNetwork` object.

`I` — Input image data

numeric array | datastore | formatted `dlarray` object

Input image data, specified in one of these formats:

- *H*-by-*W*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.
- *H*-by-*W*-by-1-by-*B* numeric array representing a batch of *B* grayscale images.
- Datastore that reads and returns truecolor images.
- Formatted `dlarray` object with two spatial dimensions of the format `"SSCB"`. You can specify multiple test images by including a batch dimension.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `extractImageEmbeddings(clip,I,MiniBatchSize=32)` limits the batch size to 32 images.

`MiniBatchSize` — Size of batches
8 (default) | positive integer

Size of batches for processing large collections of images, specified as a positive integer. Larger batch sizes reduce processing time, but require more memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`imageEmbeddings` — Feature embeddings
512-by-*B* matrix | 768-by-*B* matrix

Feature embeddings extracted from the CLIP model encoder, returned as a 512-by-*B* or 768-by-*B* matrix, depending on the value of the `backbone` argument of the `clipNetwork` object.

Image Encoder backbone Value	Image Embeddings Format
`"vit-b-16"`	512-by-*B* matrix
`"vit-l-14"` or `"resnet50"`	768-by-*B* matrix

The `extractImageEmbeddings` function resizes the input images to 224-by-224 pixels for the ViT-L/14 and ViT-B/16 backbones, or to 384-by-384 pixels for the ResNet-50 backbone.

Version History

Introduced in R2026a

See Also

`clipNetwork` | `extractTextEmbeddings` | `classify`

extractTextEmbeddings

Extract text embeddings from search text using CLIP network text encoder

Syntax

```
textEmbeddings = extractTextEmbeddings(clip,text)
textEmbeddings = extractTextEmbeddings(clip,text,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for OpenAI CLIP Network add-on.

`textEmbeddings = extractTextEmbeddings(clip,text)` extracts text embeddings from the search text `text` using the text encoder of a Contrastive Language-Image Pre-Training (CLIP) network, `clip`, by running a forward pass on the neural network.

Note This functionality requires Deep Learning Toolbox.

`textEmbeddings = extractTextEmbeddings(clip,text,Name=Value)` specifies options using one or more name-value arguments. For example, `MiniBatchSize=32` limits the batch size to 32 text strings.

Examples

Extract Text Embeddings Using CLIP Network Encoder

Create a pretrained CLIP network with a ViT-B/16 backbone.

```
clip = clipNetwork("vit-b-16");
```

Define a text search term.

```
search = "A photo of a children's book.";
```

Extract the text embeddings for the search term using the CLIP network encoder.

```
textEmbeddings = extractTextEmbeddings(clip,search);
```

Display the size of the text embeddings.

```
size(textEmbeddings)
```

```
ans = 1×2

   512     1
```

Input Arguments

`clip` — CLIP network

`clipNetwork` object

CLIP network, specified as a `clipNetwork` object.

`text` — Input text

B-element string array | datastore

Input text, specified as a *B*-element string array or a datastore containing *B* strings. *B* is the number of strings in the batch. You must specify the text in English using ASCII characters. The function automatically pads or truncates each text input so that it contains exactly 77 tokens.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `extractTextEmbeddings(clip,text,MiniBatchSize=32)` limits the batch size to 32 text strings.

`MiniBatchSize` — Number of strings in each batch

64 (default) | positive integer

Number of strings in each batch, specified as a positive integer. Larger batch sizes reduce processing time, but require more memory.

`ExecutionEnvironment` — Hardware resource

`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`textEmbeddings` — Text embeddings

512-by-*B* matrix | 768-by-*B* matrix

Text embeddings extracted from the CLIP model encoder, returned as a 512-by-*B* or 768-by-*B* matrix, depending on the value of the `backbone` argument of the `clipNetwork` object.

Image Encoder Backbone backbone Value	Text Embeddings Format
`"vit-b-16"`	512-by-*B* matrix
`"vit-l-14"` or `"resnet50"`	768-by-*B* matrix

Version History

Introduced in R2026a

See Also

`clipNetwork` | `classify` | `extractImageEmbeddings`

forward

Run forward pass on CLIP network

Syntax

```
[imageEmbeddings,textEmbeddings,state] = forward(clip,I,text)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for OpenAI CLIP Network add-on.

`[imageEmbeddings,textEmbeddings,state] = forward(clip,I,text)` computes the feature embeddings of the images `I` and text embeddings of the text `text` from the output layers of the CLIP network, and returns the updated network state `state`.

Note This functionality requires Deep Learning Toolbox.

Input Arguments

`clip` — CLIP network

`clipNetwork` object

CLIP network, specified as a `clipNetwork` object.

`I` — Image data

numeric array | datastore | formatted `dlarray` object

Image data, specified in one of these formats:

- *H*-by-*W*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.
- *H*-by-*W*-by-1-by-*B* numeric array representing a batch of *B* grayscale images.
- Datastore that reads and returns truecolor images.
- Formatted `dlarray` object with two spatial dimensions of the format `"SSCB"`. You can specify multiple test images by including a batch dimension.

`text` — Input text corresponding to each image

B-element string array

Input text corresponding to each image, specified as a *B*-element string array. *B* is the number of images in the batch. You must specify the text in English using ASCII characters. The function automatically pads or truncates each text input so that it contains exactly 77 tokens.

Output Arguments

`imageEmbeddings` — Image feature embeddings

`dlarray` object

Image feature embeddings extracted from the CLIP model encoder, returned as a `dlarray` object. The dimensions of the `dlarray` are 512-by-*B* or 768-by-*B*, depending on the value of the `backbone` argument of the `clipNetwork` object.

Image Encoder Backbone `backbone` Value	Image Embeddings `dlarray` Size
`"vit-b-16"`	512-by-*B*
`"vit-l-14"` or `"resnet50"`	768-by-*B*

`textEmbeddings` — Text embeddings
`dlarray` object

Text embeddings extracted from the CLIP model encoder, returned as a `dlarray` object. The dimensions of the `dlarray` are 512-by-*B* or 768-by-*B*, depending on the value of the `backbone` argument of the `clipNetwork` object.

Image Encoder Backbone `backbone` Value	Text Embeddings `dlarray` Size
`"vit-b-16"`	512-by-*B*
`"vit-l-14"` or `"resnet50"`	768-by-*B*

`state` — Updated network state
table

Updated network state, returned as a table. The network state is a table with these columns:

- `Layer` – Layer name, returned as a string scalar.
- `Parameter` – Parameter name, returned as a string scalar.
- `Value` – Parameter value, returned as a numeric array.

The network state contains information remembered by the network between iterations. Use the state to update the network parameters for training.

Version History

Introduced in R2026a

See Also

`clipNetwork` | `classify`

moondream

Create pretrained Moondream vision-language model (VLM)

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for Moondream Vision Language Model add-on.

The `moondream` object configures a pretrained Moondream™ vision-language model (VLM).

Use the Moondream model to quickly understand image content by generating descriptive captions. Due to the lightweight design and speed of the model, you can use it for downstream low-latency tasks like alt-text generation, image-text retrieval, and basic scene description.

To generate image captions using Moondream, use the `captionImage` object function.

Creation

Syntax

```
mdModel = moondream
```

Description

`mdModel = moondream` loads a pretrained Moondream vision-language model with 2 billion parameters.

Properties

`ModelName` — Name of pretrained Moondream vision-language model
`""` (default) | string scalar | character vector

Name of the pretrained Moondream vision-language model, specified as a string scalar or a character vector.

Object Functions

captionImage Caption images using Moondream vision-language model (VLM)

Examples

Caption Image Using Moondream Vision-Language Model

Load the Moondream vision-language model.

```
mdModel = moondream;
```

Load an image to caption into the workspace, and display the image.

```
I = imread("peppers.png");
imshow(I)
```



Caption the image using the `captionImage` object function.

```
captions = captionImage(mdModel,I);
```

Display the generated image caption.

```
display(captions)
```

```
captions =
" A purple tablecloth holds a vibrant array of red, green, yellow, and white peppers, onions, and
```

Tips

- The quality of Moondream outputs can vary across different data domains. Validate its predictions using a data set from a domain similar to your intended application.

Version History

Introduced in R2026a

References

[1] “Moondream.” Accessed September 2, 2025. https://moondream.ai/.

See Also

`captionImage` | `clipNetwork` | `visionTransformer`

Topics

“Get Started with Vision-Language Models”

captionImage

Caption images using Moondream vision-language model (VLM)

Syntax

```
captions = captionImage(mdModel,I)
captionsDS = captionImage(mdModel,imds)
[___] = captionImage(___,CaptionVerbosity=verbosity)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Model for Moondream Vision Language Model add-on.

`captions = captionImage(mdModel,I)` generates a caption for image input `I` using the Moondream™ vision-language model `mdModel`.

`captionsDS = captionImage(mdModel,imds)` generates captions for images in the input image datastore `imds` using the Moondream vision-language model `mdModel`.

`[___] = captionImage(___,CaptionVerbosity=verbosity)` specifies the length of the generated captions, in addition to the arguments from previous syntaxes.

Examples

Caption Image Using Moondream Vision-Language Model

Load the Moondream vision-language model.

```
mdModel = moondream;
```

Load an image to caption into the workspace, and display the image.

```
I = imread("peppers.png");
imshow(I)
```



Caption the image using the `captionImage` object function.

```
captions = captionImage(mdModel,I);
```

Display the generated image caption.

```
display(captions)
```

```
captions =
" A purple tablecloth holds a vibrant array of red, green, yellow, and white peppers, onions, and
```

Generate Descriptive Image Caption Using Moondream Vision-Language Model

Load the Moondream vision-language model.

```
mdModel = moondream;
```

Load an image to caption into the workspace, and display the image.

```
I = imread("visionteam.jpg");
imshow(I)
```

Generate a detailed caption for the image by specifying the `CaptionVerbosity` argument of the `captionImage` object function.

```
captions = captionImage(mdModel,I,CaptionVerbosity="detail");
```

Display the generated image caption.

```
display(captions)
```

```
captions =
" The image shows six individuals standing in a row in what appears to be an office setting. The
```

Input Arguments

`mdModel` — Moondream vision-language model

`moondream` object

Moondream vision-language model, specified as a `moondream` object.

`I` — Input RGB image data

H-by-*W*-by-3 numeric array | *H*-by-*W*-by-3-by-*B* numeric array

Input RGB image data, specified as one of these options:

- *H*-by-*W*-by-3 numeric array representing a single truecolor image.
- *H*-by-*W*-by-3-by-*B* numeric array representing a batch of *B* truecolor images. *B* is the number of images in the batch.

`imds` — Datastore of images

datastore

Datastore of images, specified as any type of datastore that returns image data. If calling the datastore with the `read` function returns a cell array, then the image data must be in the first cell.

`verbosity` — Caption length

`"brief"` (default) | `"detail"`

Caption length, specified as one of these options:

- `"brief"` — Returns a caption containing approximately 25 words or less.
- `"detail"` — Returns a caption containing up to 60 words.

Output Arguments

`captions` — Image captions

string scalar | 1-by-*B* string array

Image captions, returned as one of these options, depending on the format of the input image `I`.

- `I` is a single RGB image — String scalar.
- `I` is a batch of RGB images — 1-by-*B* string array, in which each element is the caption for the corresponding image from the batch. *B* is the number of images in the batch.

`captionsDS` — Datastore image captions

N-element string array

Datastore image captions, returned as an *N*-element string array. *N* is the number of images in the image datastore `imds`.

Tips

- The quality of Moondream outputs can vary across different data domains. Validate its predictions using a data set from a domain similar to your intended application.

Version History

Introduced in R2026a

See Also

`moondream`

Topics

“Get Started with Vision-Language Models”

nerfacto

Dense reconstruction and novel view synthesis using Nerfacto Neural Radiance Field (NeRF) model

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

Use the `nerfacto` object to reconstruct a 3-D representation of a scene, as a dense, colored point cloud, from 2-D images by using the Nerfacto Neural Radiance Field (NeRF) model [1] from the Nerfstudio library [2]. To generate a dense, colored point cloud of a scene, use the `generatePointCloud` function. You can synthesize novel views of the scene by using the `generateViews` function.

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

Creation

Create and train a `nerfacto` object on a collection of 2-D images taken from different camera poses by using the `trainNerfacto` function. The `trainNerfacto` function also generates the model folder that contains the model weights and other parameters in the path that is returned in the `ModelFolder` property.

Properties

`ModelFolder` — Model folder path

string scalar | character vector

This property is read-only.

Model folder path, represented as a string scalar or character vector in accordance with the `outputFolder` argument of the `trainNerfacto` that you use to train the `nerfacto` object. The model folder must be present and accessible for the `nerfacto` object to generate a dense, colored point cloud and create novel views of a scene.

Tip Use the `changePath` function to update the `ModelFolder` property of a trained `nerfacto` object, such as if you move the model folder to a new location.

`MaxIterations` — Maximum number of training iterations

positive integer

This property is read-only.

Maximum number of training iterations, represented as a positive integer in accordance with the `MaxIterations` argument of the `trainNerfacto` that you use to train the `nerfacto` object.

Data Types: `single` | `double`

`Intrinsics` — Camera intrinsic parameters of training images
`cameraIntrinsics` object

This property is read-only.

Camera intrinsic parameters of the training images, represented as a `cameraIntrinsics` object.

Object Functions

generatePointCloud	Generate dense, colored point cloud using trained Nerfacto Neural Radiance Field (NeRF) model
generateViews	Generate novel views using Nerfacto Neural Radiance Field (NeRF) model
writeViews	Write novel views from Nerfacto Neural Radiance Field (NeRF) model to image files
changePath	Update model folder property of trained Nerfacto NeRF model

Examples

Generate Point Cloud Using Neural Radiance Field Model

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto` object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 2-1709], and the `nerfactoModelFolder` folder, which contains the data and configuration files required for the Nerfstudio Library [2 on page 2-1709] to load and execute the trained NeRF model [3 on page 2-1709].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
    unzip("tum_rgbd_nerfacto.zip",pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd, "trainedNerfactoTUMRGBD");
load(fullfile(modelRoot, "saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot, "nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

      ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex9089
    MaxIterations: 30000
       Intrinsics: [1×1 cameraIntrinsics]
```

Generate Point Cloud with Pretrained NeRF Model

Use the `generatePointCloud` function to generate a dense, colored point cloud of the scene. Set the maximum number of points in the point cloud to 100,000 by specifying the `MaxNumPoints` name-value argument. This value achieves a balance between the quality of the point cloud, the computing time, and storage space.

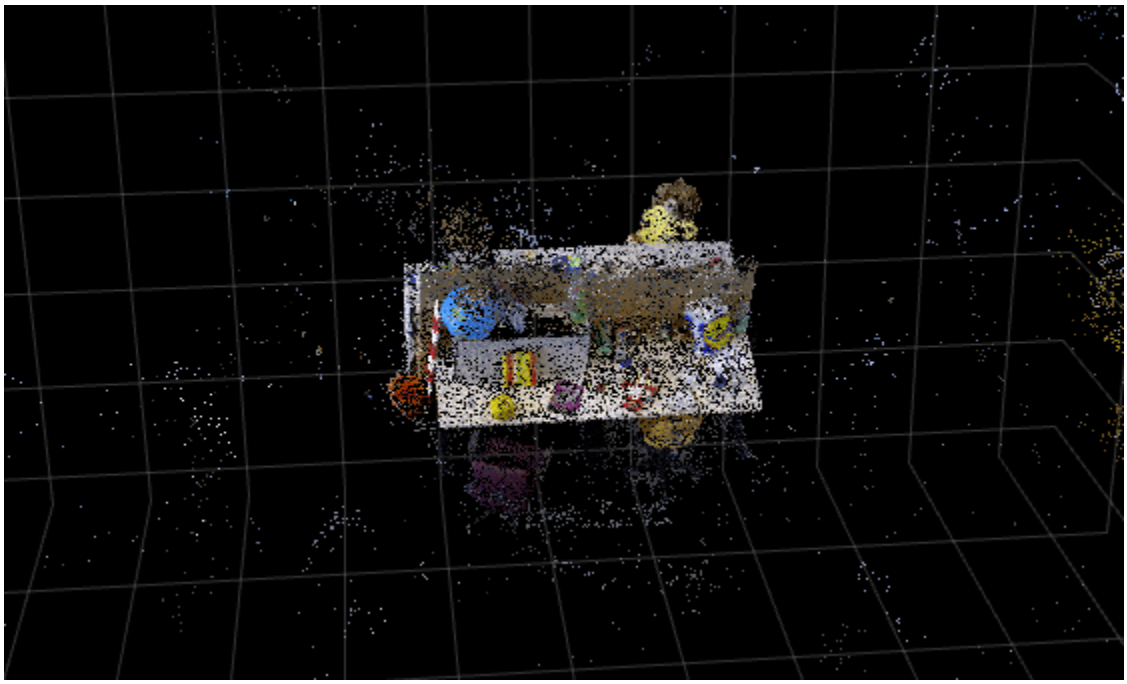
```
ptCloud = generatePointCloud(nerf, MaxNumPoints=100000);
```

```
Generating point cloud from nerfacto object.
Loading latest checkpoint from load_dir
Setting up the Nerfstudio pipeline...
[12:27:23 PM] Auto image downscale factor of 1                                                  nerfs
Setting up training dataset...
Caching all 104 images.
Loading latest checkpoint from load_dir
nerfacto object point cloud generation complete.
```

Visualize Point Cloud

Visualize the point cloud by using the `pcshow` function, and specify the vertical axis and vertical axis direction to match the coordinate system of the TUM RGB-D data set used to train the `nerfacto` object. Modify the view of the point cloud visualization to focus on the scene region of interest by specifying the of low-level camera properties of the axes object. For more information on configuring these properties, see “Low-Level Camera Properties”.

```
figure
ax = pcshow(ptCloud, VerticalAxis="y",VerticalAxisDir="down");
ax.CameraPosition = [-1.5185 -6.5178 -11.9106];
ax.CameraUpVector = [0.0520 -0.8878 0.4572];
ax.CameraViewAngle = 15;
xlabel("X")
ylabel("Y")
zlabel("Z")
title("Point Cloud from NeRF")
```



Tip: You can use the `pcdenoise` function to clean up the outliers and noise in the point cloud.

References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Generate Novel Views using Neural Radiance Field Model

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto` object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 2-1712], and the `nerfactoModelFolder` folder, which contains the data and configuration files

required for the Nerfstudio Library [2 on page 2-1712] to load and execute the trained NeRF model [3 on page 2-1712].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
    unzip("tum_rgbd_nerfacto.zip", pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd, "trainedNerfactoTUMRGBD");
load(fullfile(modelRoot, "saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot, "nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

      ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex3949
    MaxIterations: 30000
       Intrinsics: [1×1 cameraIntrinsics]
```

Load Camera Poses

This example contains a set of predefined camera poses that are stored as an array of `rigidtform3d` objects in the `camera-poses-tum.mat` file. These represent the position and orientation of the camera in the 3-D world coordinate system of the scene.

Load the `camera-poses-tum.mat` file into workspace, which contains the camera poses `camPoses`.

```
load("camera-poses-tum.mat","camPoses");
```

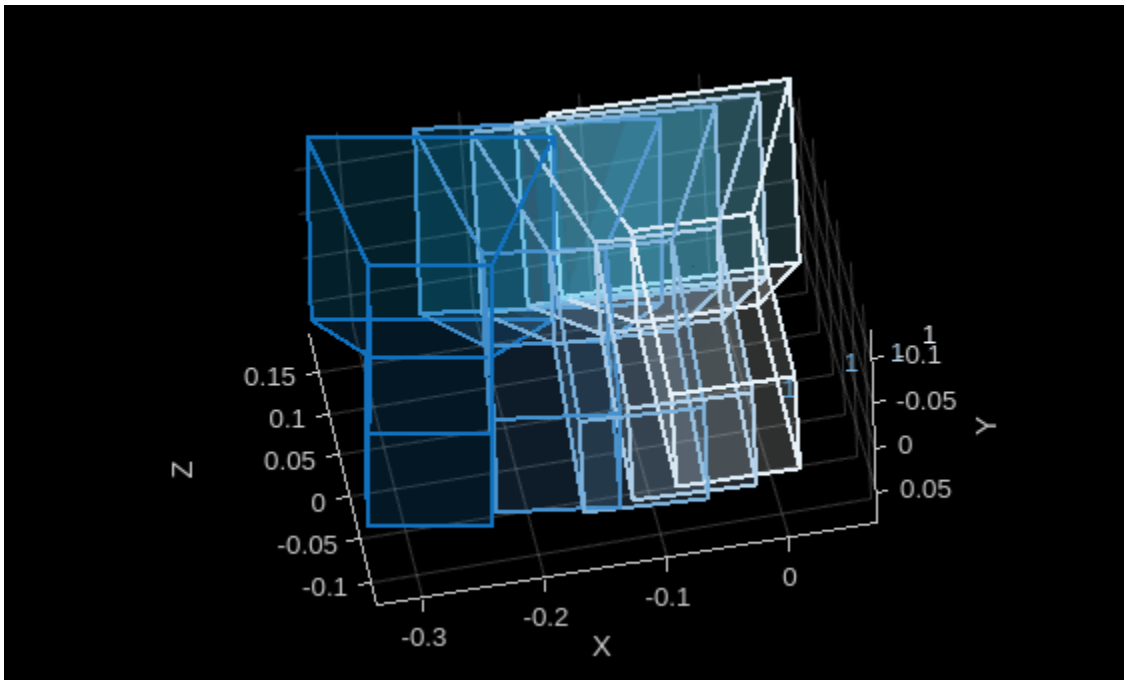
Display the camera poses by using the `plotCamera` function.

```
figure
pcshow([NaN NaN NaN], VerticalAxis="y", VerticalAxisDir="down");
xlabel("X")
```

```
ylabel("Y")
zlabel("Z")
hold on

colors = sky(length(camPoses));
for i = 1:length(camPoses)
plotCamera(camPoses(i), Size=0.1, Color=colors(i,:), Opacity=0.1);
end

hold off
```



Generate Novel Views with Pretrained NeRF Model

Generate the views captured by the camera poses `camPoses` by using the `generateViews` function.

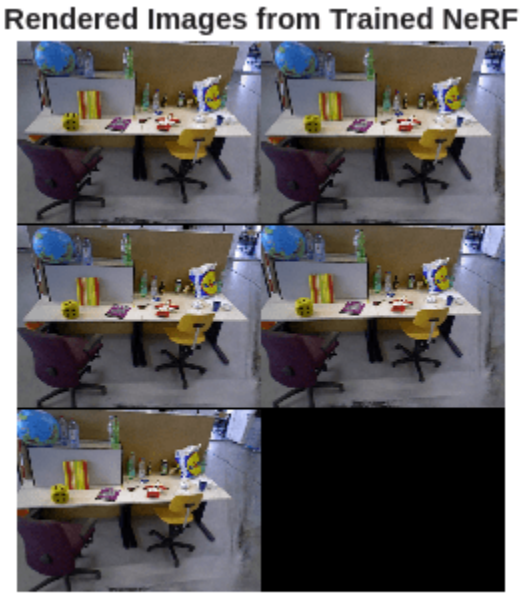
```
images = generateViews(nerf,camPoses);
```

```
Generating views from nerfacto object.
Loading latest checkpoint from load_dir
nerfacto object view generation complete.
```

Tip: If you want to generate a large number of novel views, use the `writeViews` function instead of `generateViews` to avoid an out-of-memory error.

Display the generated images. The images show the globe and chairs in front of the desk that are present in the images used to train this NeRF model.

```
figure
montage(images, BorderSize=[2 2])
title("Rendered Images from Trained NeRF")
```



References

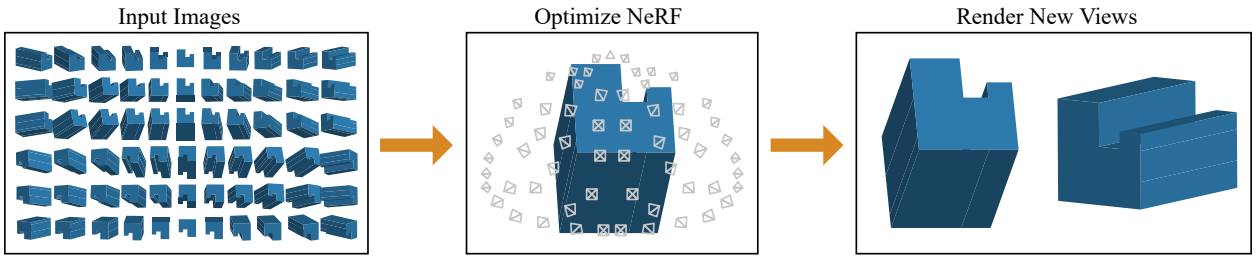
[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

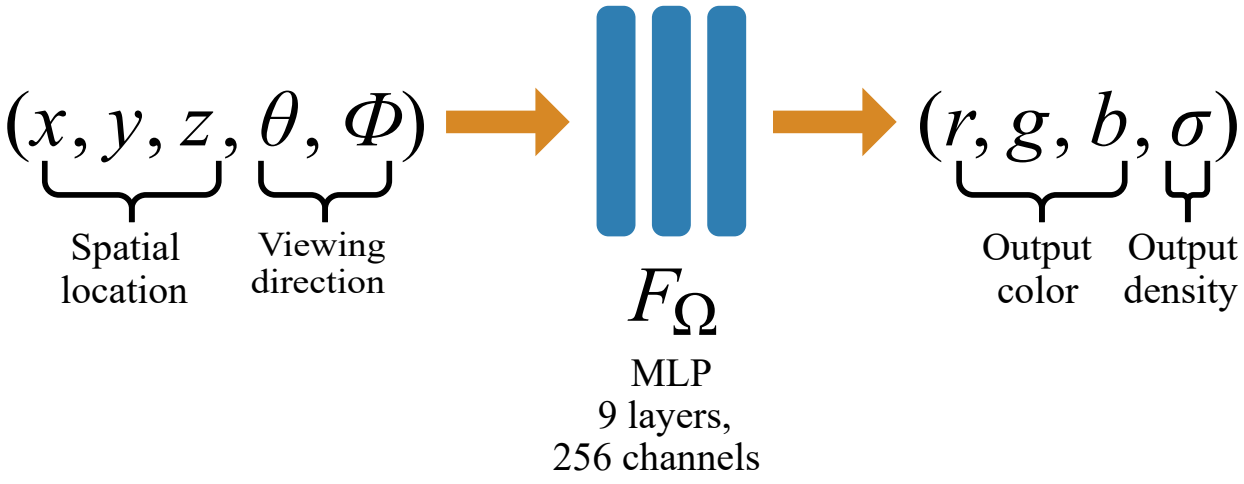
[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Algorithms

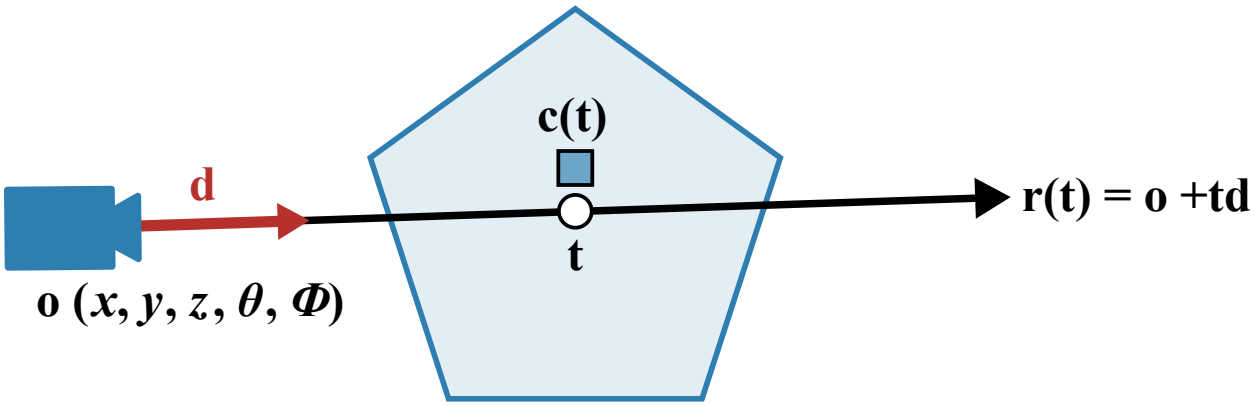
By training a NeRF model on a set of sparse input images and camera poses, you can create an internal representation of a 3D scene. You can use a trained NeRF model to generate a dense, colored point cloud, and render images from novel viewpoints.

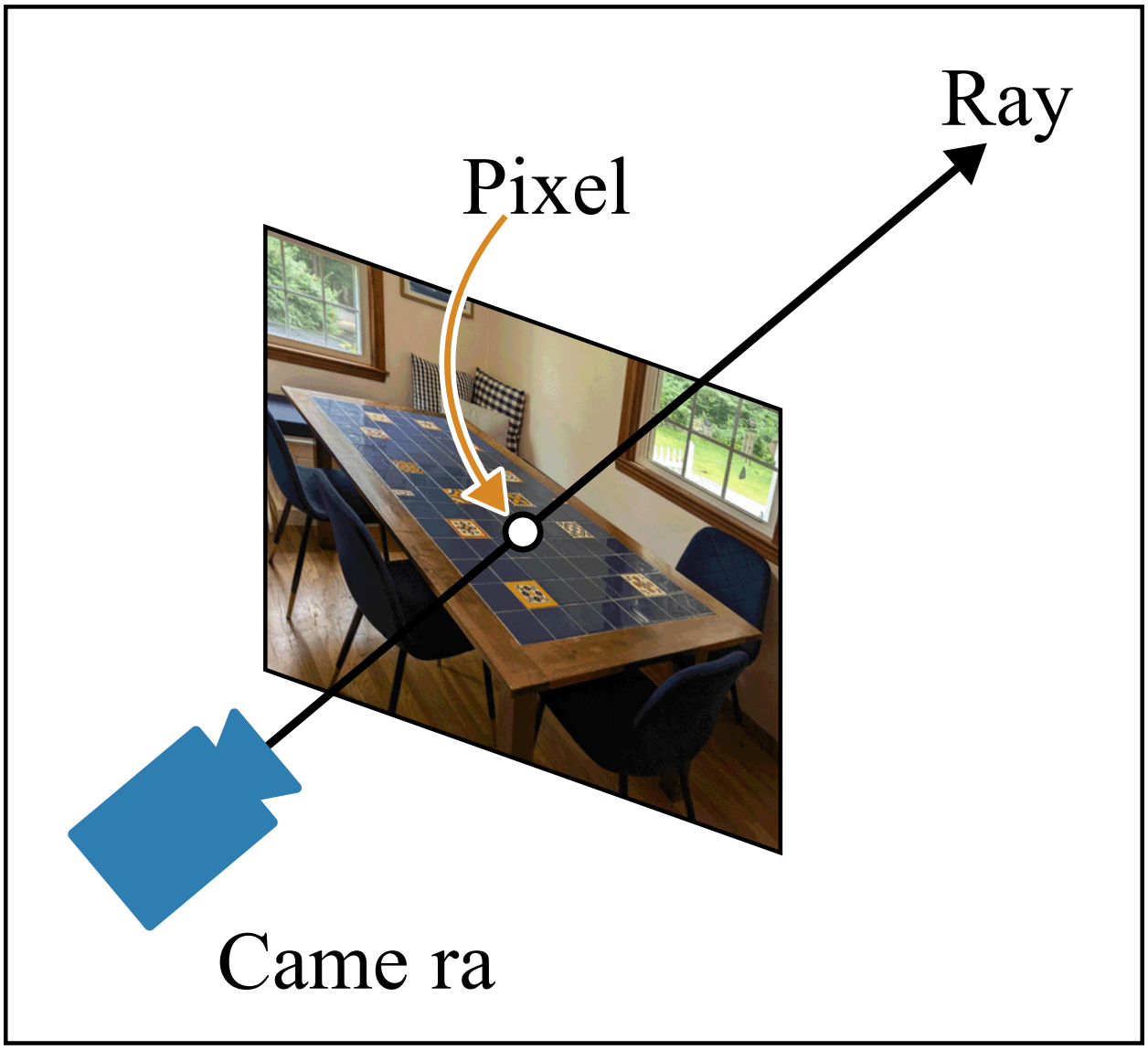

Input Images
Optimize NeRF
Render New Views

A NeRF model represents a scene as a continuous 5D vector-valued function instantiated as a neural network with weights Ω. The function accepts the position and direction of a light ray and returns the color and density of 3D points along the ray.

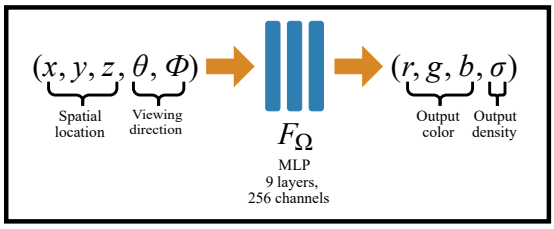

(x, y, z, θ, Φ)
Spatial location
Viewing direction
F_Ω
MLP
9 layers,
256 channels
(r, g, b, σ)
Output color
Output density

When generating a novel view, the NeRF model first uses the neural network to calculate the color and density of 3D points along each ray from a virtual camera. The ray is defined as $r(t) = o + td$, where o is the origin coordinates of the ray, and d is the unit vector in the ray direction defined by the camera angles θ and ϕ. The neural network return the color $c(t)$ if the ray $r(t)$ hits a particle at distance t along the ray. The model then uses volumetric rendering to project the output colors and densities into an image.

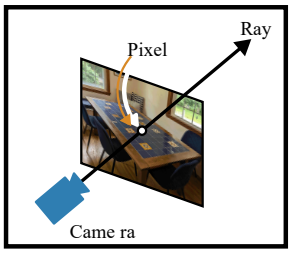

c(t)
d
r(t) = o +td
t
o (x, y, z, θ, Φ)


Ray
Pixel
Came ra

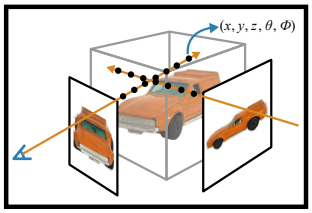
During training, a NeRF model first samples a batch of rays that consist of multiple spatial locations and viewing directions. The neural network F_Ω then predicts the color and density at 3D points along each ray, and the NeRF model uses volumetric rendering to project the output colors and densities onto an image. The model calculates the training loss by matching the image pixels of the rendered image against the captured images in the training data.



Predict color and density at 3D points along ray using neu ral net



Calculate color at each pixel using volumetric rendering



Calculate t raining loss by comparing rendered vs actual image pixels

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`trainNerfacto` | `monovslam` | `cameraIntrinsics` | `generateViews`

Topics

“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”
“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites

Nerfstudio Library GitHub Repository
Nerfstudio Library Documentation

Functions

axang

Convert transformation or rotation into axis-angle rotations

Syntax

```
angles = axang(transformation)
angles = axang(rotation)
```

Description

`angles = axang(transformation)` converts the rotation of the transformation `transformation` to the axis-angle rotations `angles`.

`angles = axang(rotation)` converts the rotation `rotation` to the axis-angle rotations `angles`.

Examples

Convert SE(3) Transformation to Axis-Angle Rotation

Create SE(3) transformation with no translation but with a rotation defined by an axis-angle rotation. Define the axis-rotation with vector of `[0.5 0.25 0.5]` to be the axis and a `pi/2` rotation about that axis.

```
axa1 = [0.5 0.25 0.5 pi/2];
T = se3(axa1,"axang");
```

Get the axis-angle rotation from the transformation.

```
axa2 = axang(T)
```

```
axa2 = 1×4

    0.6667    0.3333    0.6667    1.5708
```

Convert SO(3) Rotation to Axis-Angle Rotation

Create SO(3) transformation with a rotation defined by an axis-angle rotation. Define the axis-rotation with vector of `[0.5 0.25 0.5]` to be the axis and a `pi/2` rotation about that axis.

```
axa1 = [0.5 0.25 0.5 pi/2];
R = so3(axa1,"axang");
```

Get the axis-angle rotation from the transformation. Note that the vector of the axis-angle rotation has a different magnitude from the axis-angle rotation specified to the transformation but the defined axis and rotation are the same.

```
axa2 = axang(R)
```

```
axa2 = 1×4

    0.6667    0.3333    0.6667    1.5708
```

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of `se3` objects

Transformation, specified as an `se3` object or as an *N*-element array of `se3` objects. *N* is the total number of transformations.

`rotation` — Rotation

`so3` object | *N*-element array of `so3` objects

Rotation, specified as an `so3` object or as an *N*-element array of `so3` objects. *N* is the total number of rotations.

Output Arguments

`angles` — Axis-angle rotation angles

N-by-4 matrix

Axis-angle rotation angles, specified as an *N*-by-4 matrix of *N* axis-angle rotations. The first three elements of every row specify the rotation axes, and the last element defines the rotation angle, in radians.

See Also

`se3` | `so3`

detectMultiPatternPoints

Detect keypoints of multiple calibration patterns

Syntax

```
imagePoints = detectMultiPatternPoints(imageFileNames,patternCount,"charuco-
board",patternDims,markerFamily,checkerSize,markerSize)
imagePoints = detectMultiPatternPoints(imageFileNames,
patternCount,"aprilgrid",patternDims,tagFamily)

imagePoints = detectMultiPatternPoints(imageFileNames,patternCount,
patternDetector,numKeyPoints)

imagePoints = detectMultiPatternPoints( ___ ,Name=Value)
```

Description

Natively Supported Patterns

`imagePoints = detectMultiPatternPoints(imageFileNames,patternCount,"charuco-board",patternDims,markerFamily,checkerSize,markerSize)` detects the keypoints of two or more ChArUco board calibration patterns in images `imageFileNames`, with dimensions specified by `patternDims`, `markerFamily`, and `checkerSize`. `patternCount` specifies the number of patterns to detect.

`imagePoints = detectMultiPatternPoints(imageFileNames, patternCount,"aprilgrid",patternDims,tagFamily)` detects the keypoints of two or more AprilGrid calibration patterns in images `imageFileNames`, with dimensions specified by `patternDims`.

Custom Pattern Detector

`imagePoints = detectMultiPatternPoints(imageFileNames,patternCount, patternDetector,numKeyPoints)` detects the specified number of uniquely detectable keypoints `numKeyPoints` of two or more custom calibration patterns of same type that can be detected in an image using the specified detector function `patternDetector`.

Optional Arguments

`imagePoints = detectMultiPatternPoints( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Verbose=true` displays the progress of the detection process.

Examples

Calibrate a Multi-Camera System using Multiple ChArUco Boards

Estimate the relative position and orientation of eight cameras with overlapping fields of view by using calibration images that contain multiple ChArUco boards.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-multiple
calibImagesDir = fullfile(pwd,"overlapping-cameras-multiple-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-multiple-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (86 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 8;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
```

Detect the keypoints of the eight ChArUco boards captured in the calibration images using the `detectMultiPatternPoints` function. Specify the MinMarkerID for each pattern.

```
patternCount = 8;
patternIDs = 144:12:228;

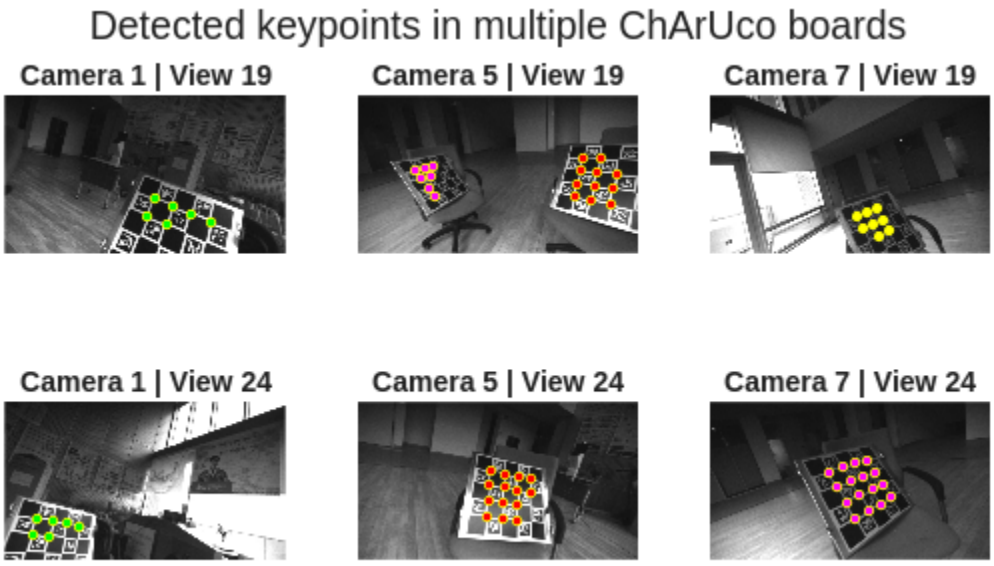
imagePoints = detectMultiPatternPoints(imageFiles,patternCount,"charuco-board",patternDims,marke
    checkerSize,markerSize,MinMarkerID=patternIDs);
```

```
[==================================================] 100%
Elapsed time: 00:00:14
Estimated time remaining: 00:00:00
```

Display the detected keypoints in a couple of views and three of the eight cameras.

```
t = tiledlayout(2,3,TileSpacing="compact",Padding="compact");
colors = ["red","green","cyan","black","magenta","yellow","blue","white"];
for viewIdx = [19,24]
    for camIdx = [1,5,7]
        nexttile
        imshow(imageFiles{viewIdx,camIdx})
        title("Camera " + camIdx + " | View " + viewIdx)
        hold on
        for patternIdx = 1:patternCount
            plot(imagePoints(:,1,viewIdx,camIdx,patternIdx),imagePoints(:,2,viewIdx,camIdx,patte
                MarkerSize=3,MarkerFaceColor=colors(patternIdx))
        end
```

```
    end
end
title(t,"Detected keypoints in multiple ChArUco boards")
```



Generate the world coordinates of each pattern points in their respective pattern coordinate system.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the eight cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

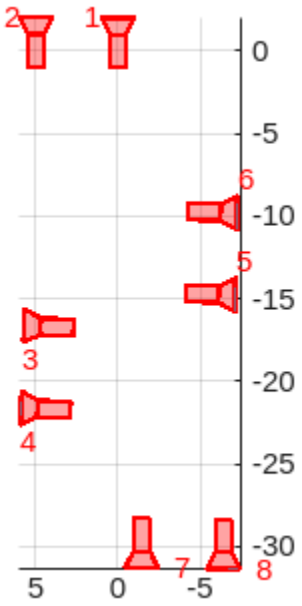
```
ld = load("eightCameraIntrinsics.mat");
```

Perform multi-camera calibration.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

Plot the estimated camera poses.

```
figure
plotCamera(params.CameraPoses)
axis equal
grid on
view([0 1 0])
```



Multi-Camera Calibration with Overlapping and Non-Overlapping Cameras

Calibrate multiple cameras that have some cameras sharing fields-of-view and some cameras not sharing any field-of-view.

Download the calibration images containing two ChArUco boards that were captured from six cameras as per the guidelines outlined in “Prepare Cameras and Capture Images for Multi-Camera Calibration”. The calibration images were captured by moving the multi-camera system around the stationary ChArUco boards.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/non-overlapping-cameras-cha
calibImagesDir = fullfile(pwd,"non-overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"non-overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (68 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

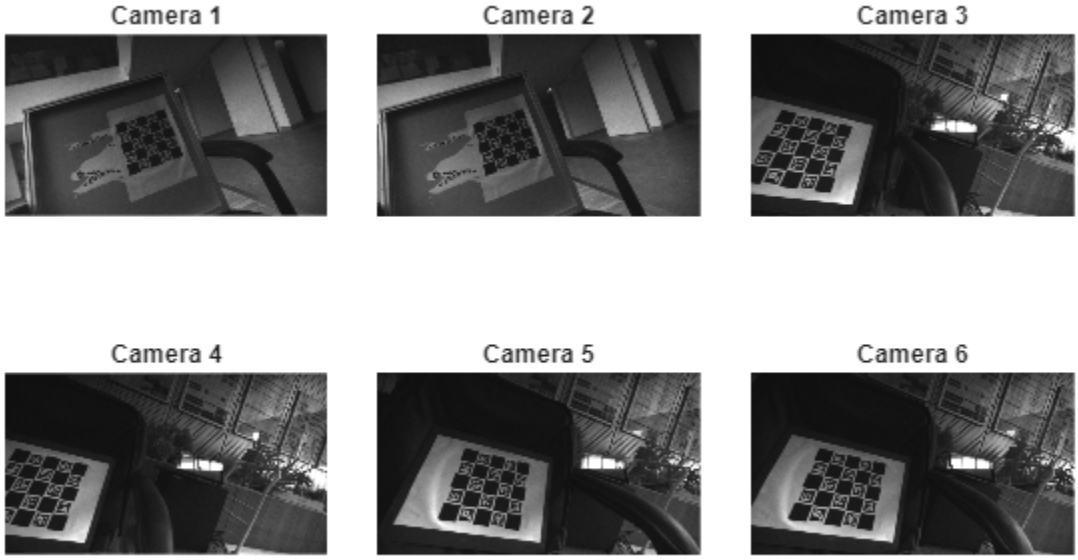
Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
```

```
end
imageFiles = [imageFiles{:}];
```

Display images captured by the six cameras from one of the views.

```
figure
t = tiledlayout(2,3,TileSpacing="tight", Padding="tight");
for i = 1:numCameras
    nexttile
    imshow(imageFiles{25,i})
    title("Camera " + i)
end
```



Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 2.55;   % in cm
checkerSize = 3.40; % in cm
```

Detect the key points of the two ChArUco boards captured in the calibration images using the `detectMultiPatternPoints` function.

```
patternCount = 2;
imagePoints = detectMultiPatternPoints(imageFiles,patternCount,"charuco-board",patternDims,marke
    checkerSize,markerSize);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world coordinates of each pattern points in their respective pattern coordinate system.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the "Using the Single Camera Calibrator App".

```
ld = load("sixCameraIntrinsicsNoOverlap.mat");
```

Perform multi-camera calibration. Enable verbose mode to track the progress of the calibration process.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm",Verb
```

```
Estimating multi-camera extrinsic parameters
--------------------------------------------
* Number of cameras: 6
* Number of patterns: 2
* Number of views: 30
* Number of images used: 110

* Undistorting image points...done.

* Grouping cameras with overlapping field-of-view....done.
* Group 1: camera 3 4 5 6
* Group 2: camera 1 2

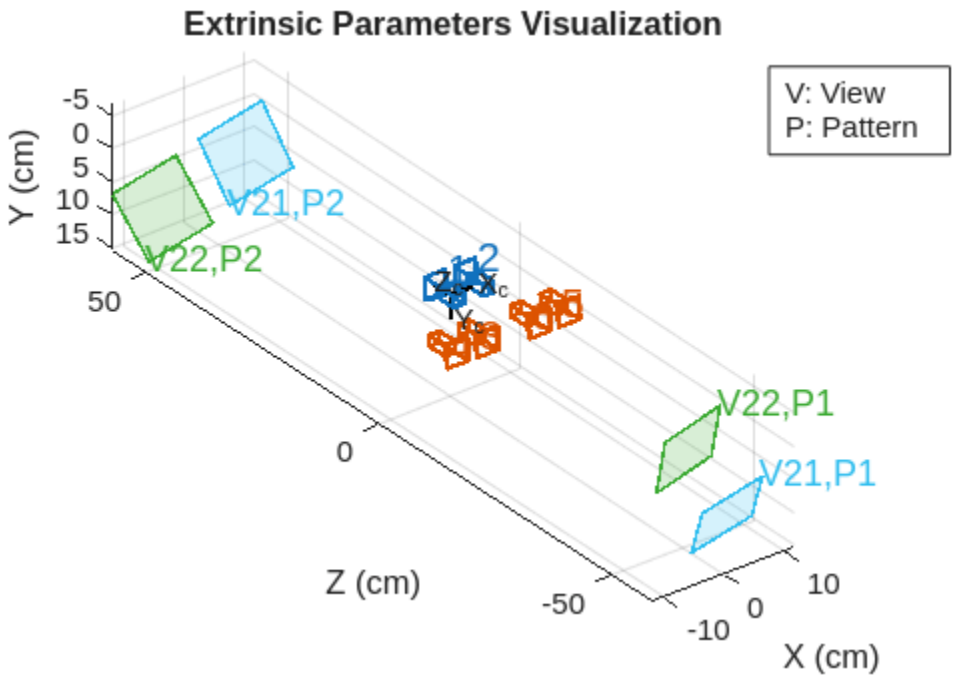
* Refining camera poses of each group:
* Group 1...done.
* Group 2...done.

* Estimating relative pose between camera groups:
* [Pair 1/1] Group 1 and 2...done.

* Refining all camera poses and pattern poses...done.
* Finished calibrating cameras.
```

Visualize the camera extrinsic parameters along with the patterns from a couple of views.

```
figure
showExtrinsics(params,"CameraCentric",ViewIndex=21:22)
```



Input Arguments

`imageFileNames` — Filenames of camera calibration patterns

numViews-by-*numCameras* string array | *numViews*-by-*numCameras* cell array

Filenames of the camera calibration patterns, specified as a *numViews*-by-*numCameras* string or cell array. *numViews* is the number of calibration pattern views for each camera, and *numCameras* is the number of cameras. Because you must specify more than two cameras, *numCameras* must be greater than 2. Any missing views can be specified by an empty string in the array.

Organize your calibration data into folders that contain synchronized images for each camera, ensuring that the filenames are consistent across all camera folders. This arrangement aligns images from different cameras that correspond to the same scene or subject. By using identical filenames in the folder for each camera, you can ensure that the images are properly synchronized.


CalibrationData
Camera-1
image1.ext
image2.ext
image N.ext
Camera-2
image1.ext
image2.ext
image N.ext
Camera-numCameras
image1.ext
image2.ext
image N.ext

`patternCount` — Number of unique calibration patterns captured in calibration images
scalar

Number of unique calibration patterns captured in the calibration images, specified as a scalar. The `patternCount` value must be greater than 1. For single pattern, you can use the `detectPatternPoints` function to detect keypoints.

`patternDims` — Pattern dimensions
two-element vector

Pattern dimensions, specified as a two-element vector that represents the number of checkers in the rows (*dim1*) and columns (*dim2*) of the pattern. For an AprilGrid pattern, ensure that the pattern is oriented with the origin at the top-left as you count the rows and columns. For more information on patterns, see “Calibration Patterns”.

`checkerSize` — ChArUco board checker box side length
scalar

ChArUco board checker box side length, specified as a scalar in world units, such as millimeters or inches. You must specify `checkerSize` and `markerSize` in the same world units, and `checkerSize` must be a larger value than `markerSize`.

`markerSize` — Size of ArUco marker
positive integer

Size of ArUco marker, specified as a positive integer. This value sets the length of one side of a square ArUco marker in world units such as millimeters, centimeters, meters, or inches. The value of `checkerSize` must be greater than the value of `markerSize`.

`markerFamily` — ArUco marker family
`"DICT_ARUCO_ORIGINAL"` | `"DICT_4X4_1000"` | `"DICT_5X5_1000"` | `"DICT_6X6_1000"` | `"DICT_7X7_1000"`

ArUco marker family, specified as one of these options:

- `"DICT_ARUCO_ORIGINAL"`
- `"DICT_4X4_1000"`
- `"DICT_5X5_1000"`
- `"DICT_6X6_1000"`
- `"DICT_7X7_1000"`

`tagFamily` — AprilTag family
`"tag36h11"` | `"tag25h9"` | `"tag16h5"`

AprilTag family, specified as `"tag36h11"`, `"tag25h9"`, or `"tag16h5"`.

`patternDetector` — Custom pattern detector
function handle

Custom pattern detector, specified as a function handle with this syntax:

`[imagePoints,hasDetected] = patternDetector(filename,patternID)`.

Input

- `filename` — Name of the image to be processed, specified as a string scalar or character vector.
- `patternID` — Scalar identifier assigned to the pattern to be detected. This serves the same purpose as the `minMarkerID` of a ChArUco board or an AprilGrid pattern.

Output

- `imagePoints` — Coordinates of the detected keypoints in the image, returned as a *numKeyPoints*-by-2 matrix. *N* is the number of keypoints.
- `hasDetected` — Logical that indicates whether the function detected the pattern, returned as `true` when the function detects the pattern and `false` when it does not.

`numKeyPoints` — Number of keypoints that can be detected

positive integer

Number of keypoints that can be detected in the custom pattern, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectMultiPatternPoints(imageFileNames,patternCount,"charuco-board",patternDims,markerFamily,checkerSize,markerSize,Verbose=1)` enables progress display to the screen.

`MinMarkerID` — Minimum marker ID to detect in each pattern

N-element vector

Minimum marker ID to detect in each pattern, specified as an *N*-element vector of non-negative integers. The number of elements *N* is equal to the number of calibration patterns, `patternCount`.

`Verbose` — Display detection progress information

`true` or `1` (default) | `false` or `0`

Display detection progress information in the Command Window, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set `Verbose` to `true` to display progress information.

Output Arguments

`imagePoints` — Image coordinates for keypoints found in each image

numKeyPoints-by-*2*-by-*numViews*-by-*numCameras*-by-*patternCount* array

Image coordinates for the keypoints found in each image, returned as a *numKeyPoints*-by-2-*numViews*-by-*numCameras*-by-*patternCount* array.

- *numKeyPoints* is the number of keypoints found for each image.
- *numViews* is the number of views with respect to the camera.
- *numCameras* is the number of cameras. For example, `imagePoints(:,:,i,j,k)` contains the points from camera *j* when the k^{th} pattern is located at the i^{th} view.

- *patternCount* is the number of unique calibration patterns captured in the calibration images.

For images where a camera fails to detect a pattern, the corresponding keypoint values are assigned as NaNs (Not a Number).

Version History

Introduced in R2026a

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectMultiPatternPoints` function supports thread-based environments with these usage notes and limitations:

Support for running in a thread-based environment on a background pool is available only when you set `Verbose` to `false`. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Objects

`multiCameraParameters`

Functions

`patternWorldPoints` | `estimateMultiCameraParameters` | `detectPatternPoints`

Topics

“What Is Camera Calibration?”
“What Is Multi-Camera Calibration?”
“What Is Robot Hand-Eye Calibration?”
“Calibration Patterns”
“Prepare Cameras and Capture Images for Multi-Camera Calibration”
“Using the Single Camera Calibrator App”
“Coordinate Systems”
“Evaluating the Accuracy of Single Camera Calibration”
“3-D Motion Reconstruction Using Multiple Cameras”

groundTruthFromCOCO

Convert COCO JSON file into ground truth object

Syntax

```
gTruth = groundTruthFromCOCO(pathToJSON,pathToImages)
```

Description

`gTruth = groundTruthFromCOCO(pathToJSON,pathToImages)` converts the COCO-formatted JSON file `pathToJSON` and the image files in the folder `pathToImages` into a ground truth object, `gTruth`.

You can use this function to convert data stored in the COCO JSON format into a ground truth object, and then use the ground truth object to create a datastore for training deep learning models for semantic segmentation, object detection, and instance segmentation tasks.

Examples

Import JSON File to Ground Truth Object

Define the location for annotations and images.

```
pathToJSON = "cupGtruthCOCO.json";
pathToImages = fullfile(matlabroot,"toolbox","vision","visiondata","imageSets","cups");
```

Import JSON file to a `groundTruth` object.

```
gTruth = groundTruthFromCOCO(pathToJSON,pathToImages);
```

Open the Image Labeler app with the imported data.

```
imageLabeler(gTruth)
```

Input Arguments

`pathToJSON` — Path to JSON file

character vector | string scalar

Path to a JSON file, specified as a character vector or string scalar. The JSON file contains the annotations for the image data.

`pathToImages` — Folder containing images

character vector | string scalar

Folder containing images, specified as a character vector or string scalar.

Output Arguments

`gTruth` — Ground truth object
`groundTruth` object

Ground truth object, returned as a `groundTruth`.

Version History

Introduced in R2025a

See Also

Objects
`groundTruth`

Functions
`pixelLabelTrainingData` | `objectDetectorTrainingData`

estimateMultiCameraParameters

Calibrate extrinsic parameters of multiple cameras with overlapping views

Syntax

```
[cameraParams,imagesUsed] = estimateMultiCameraParameters(imagePoints,
worldPoints,intrinsics)
cameraParams = estimateMultiCameraParameters( ___ ,Name=Value)
```

Description

`[cameraParams,imagesUsed] = estimateMultiCameraParameters(imagePoints, worldPoints,intrinsics)` estimates extrinsic parameters of multiple cameras using images of one or more calibration patterns. The keypoints of the patterns in each image are specified in `imagePoints`. The keypoints of the calibration patterns in world coordinates is specified in `worldPoints`. The function adjusts all image points for distortion according to the lens distortion parameters specified in the `intrinsics` input. The function also returns the images you used, `imagesUsed`, to estimate the camera parameters.

Use this function to calibrate the extrinsic parameters of two cameras without overlapping fields of view. To calibrate the extrinsic parameters of two cameras with overlapping fields of view, use the `estimateStereoBaseline` function.

`cameraParams = estimateMultiCameraParameters( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `WorldUnits="mm"` sets the world units to millimeters.

Examples

Calibrate a Multi-Camera System using a ChArUco board

Estimate the relative position and orientation of six cameras with overlapping fields of view by using calibration images that contain a single ChArUco board.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-charuco
calibImagesDir = fullfile(pwd,"overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (52 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (52 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
numKeyPoints = prod(patternDims - 1);
minMarkerId = 144;
```

Detect the key points of the ChArUco board in the calibration images.

```
imagePoints = detectPatternPoints(imageFiles,"charuco-board",patternDims, ...
    markerFamily,checkerSize,markerSize,MinMarkerID=minMarkerId);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world points for the pattern.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

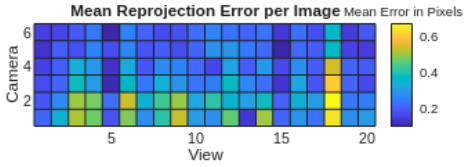
```
ld = load("sixCameraIntrinsics.mat");
```

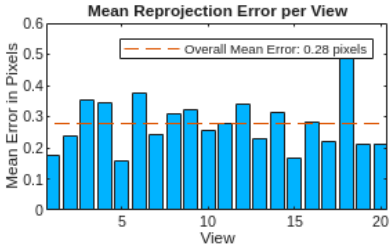
Perform multi-camera calibration.

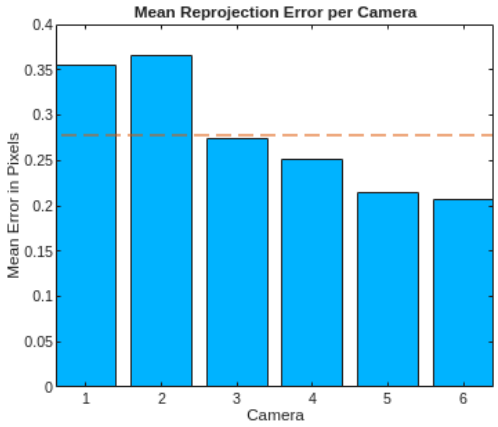
```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

Visualize the calibration accuracy.

```
figure(Position=[100,100,1000,400])
showReprojectionErrors(params)
```

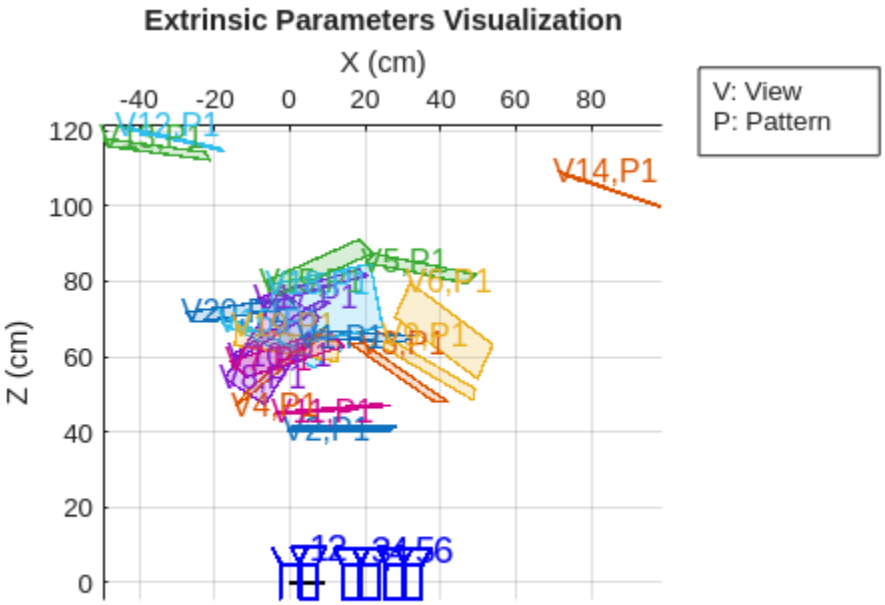






Visualize the camera extrinsic parameters.

```
figure
showExtrinsics(params)
view(2)
```



Multi-Camera Calibration with Overlapping and Non-Overlapping Cameras

Calibrate multiple cameras that have some cameras sharing fields-of-view and some cameras not sharing any field-of-view.

Download the calibration images containing two ChArUco boards that were captured from six cameras as per the guidelines outlined in “Prepare Cameras and Capture Images for Multi-Camera Calibration”. The calibration images were captured by moving the multi-camera system around the stationary ChArUco boards.

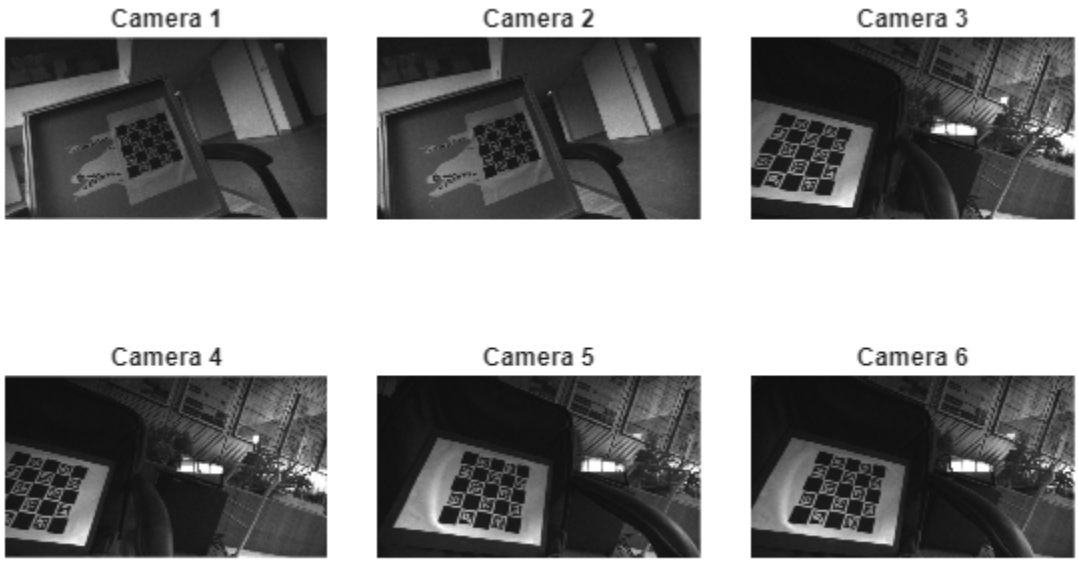
```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/non-overlapping-cameras-cha
calibImagesDir = fullfile(pwd,"non-overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"non-overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (68 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Display images captured by the six cameras from one of the views.

```
figure
t = tiledlayout(2,3,TileSpacing="tight", Padding="tight");
for i = 1:numCameras
    nexttile
    imshow(imageFiles{25,i})
    title("Camera " + i)
end
```



Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 2.55;   % in cm
checkerSize = 3.40; % in cm
```

Detect the key points of the two ChArUco boards captured in the calibration images using the `detectMultiPatternPoints` function.

```
patternCount = 2;
imagePoints = detectMultiPatternPoints(imageFiles,patternCount,"charuco-board",patternDims,marke
    checkerSize,markerSize);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world coordinates of each pattern points in their respective pattern coordinate system.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

```
ld = load("sixCameraIntrinsicsNoOverlap.mat");
```

Perform multi-camera calibration. Enable verbose mode to track the progress of the calibration process.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm",Verb
```

```
Estimating multi-camera extrinsic parameters
--------------------------------------------
```

```
* Number of cameras: 6
* Number of patterns: 2
* Number of views: 30
* Number of images used: 110

* Undistorting image points...done.

* Grouping cameras with overlapping field-of-view....done.
* Group 1: camera 3 4 5 6
* Group 2: camera 1 2

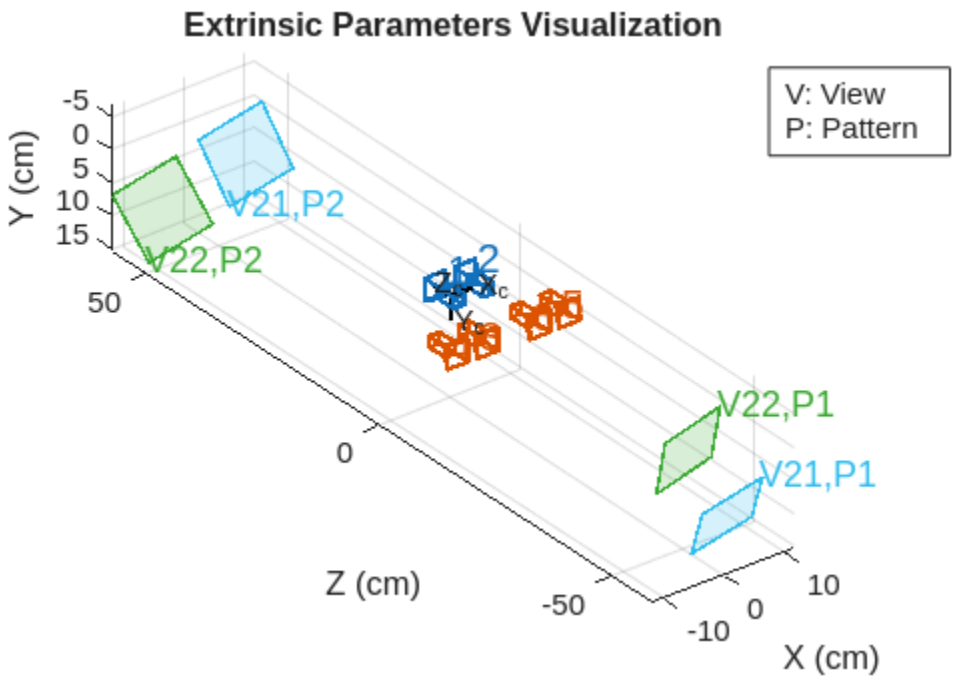
* Refining camera poses of each group:
* Group 1...done.
* Group 2...done.

* Estimating relative pose between camera groups:
* [Pair 1/1] Group 1 and 2...done.

* Refining all camera poses and pattern poses...done.
* Finished calibrating cameras.
```

Visualize the camera extrinsic parameters along with the patterns from a couple of views.

```
figure
showExtrinsics(params,"CameraCentric",ViewIndex=21:22)
```



Calibrate a Multi-Camera System using Multiple ChArUco Boards

Estimate the relative position and orientation of eight cameras with overlapping fields of view by using calibration images that contain multiple ChArUco boards.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-multiple
calibImagesDir = fullfile(pwd,"overlapping-cameras-multiple-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-multiple-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (86 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 8;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
```

Detect the keypoints of the eight ChArUco boards captured in the calibration images using the `detectMultiPatternPoints` function. Specify the MinMarkerID for each pattern.

```
patternCount = 8;
patternIDs = 144:12:228;

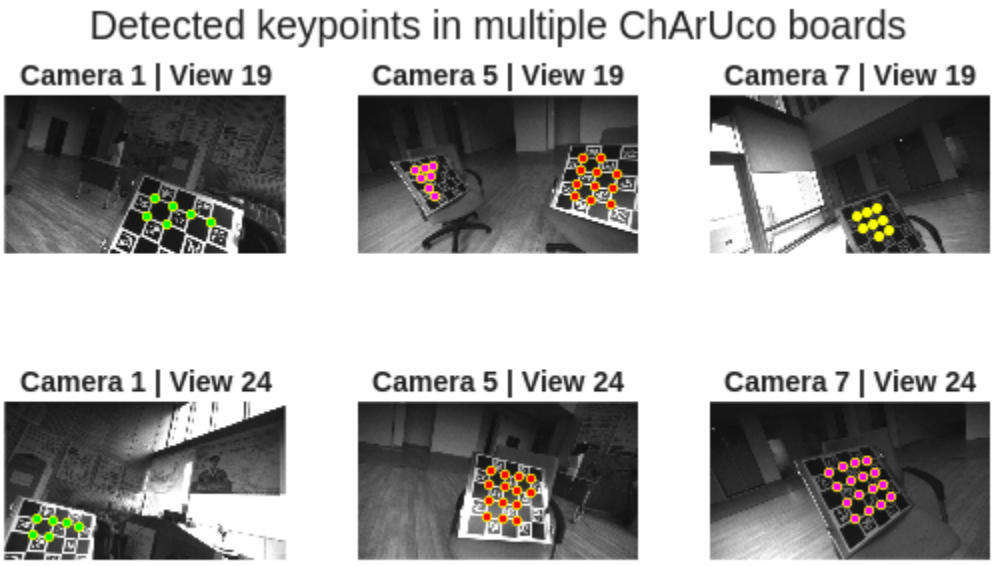
imagePoints = detectMultiPatternPoints(imageFiles,patternCount,"charuco-board",patternDims,marke
    checkerSize,markerSize,MinMarkerID=patternIDs);
```

```
[==================================================] 100%
Elapsed time: 00:00:14
Estimated time remaining: 00:00:00
```

Display the detected keypoints in a couple of views and three of the eight cameras.

```
t = tiledlayout(2,3,TileSpacing="compact",Padding="compact");
colors = ["red","green","cyan","black","magenta","yellow","blue","white"];
for viewIdx = [19,24]
    for camIdx = [1,5,7]
        nexttile
        imshow(imageFiles{viewIdx,camIdx})
        title("Camera " + camIdx + " | View " + viewIdx)
        hold on
        for patternIdx = 1:patternCount
            plot(imagePoints(:,1,viewIdx,camIdx,patternIdx),imagePoints(:,2,viewIdx,camIdx,patte
                MarkerSize=3,MarkerFaceColor=colors(patternIdx))
        end
```

```
    end
end
title(t,"Detected keypoints in multiple ChArUco boards")
```



Generate the world coordinates of each pattern points in their respective pattern coordinate system.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the eight cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

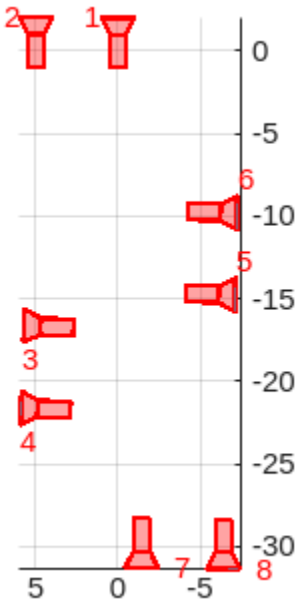
```
ld = load("eightCameraIntrinsics.mat");
```

Perform multi-camera calibration.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

Plot the estimated camera poses.

```
figure
plotCamera(params.CameraPoses)
axis equal
grid on
view([0 1 0])
```



Input Arguments

`imagePoints` — Image coordinates of keypoints in calibration pattern

numKeyPoints-by-2-by-*numViews*-by-*numCameras* array | *numKeyPoints*-by-2-by-*numViews*-by-*numCameras*-by-*patternCount* array

Image coordinates of keypoints in calibration pattern, specified as a *numKeyPoints*-by-2-by-*numViews*-by-*numCameras* 4-D array or a *numKeyPoints*-by-2-by-*numViews*-by-*numCameras*-by-*patternCount* 5-D array of *x*-*y* image coordinates.

- *numKeyPoints* — Number of keypoints in each view, which must be greater than 3.
- *numViews* — Number of unique views of the calibration patterns with respect to each camera. At each view, the pattern must be visible to at least two cameras. Each camera should observe the pattern at least once along with some other cameras. This index represents the points from camera *j* when the pattern is located at the i^{th} view.
  - `imagePoints(:,:,i,j)`
- *numCameras* — Number of cameras to calibrate, which must be greater than 2.
- *patternCount* — Number of calibration patterns. When using multiple calibration patterns, each pattern must either be visible to at least two cameras in a view or be seen by a camera along with another pattern in the same view. Additionally, each camera must observe at least one pattern that is also seen by another camera in one of the views.

To include partially detected patterns, use `[NaN,NaN]` as x-y coordinates for missing keypoints. For calibration with a single pattern, you can create a 4-D array of keypoints using the `detectPatternPoints` function. For calibration with multiple patterns, you can create a 5-D array of keypoints using the `detectMultiPatternPoints` function.

Data Types: `double`

`worldPoints` — World coordinates of keypoints on calibration pattern
numKeyPoints-by-2 array

World coordinates of keypoints on calibration pattern, specified as an *numKeyPoints*-by-2 array. *numKeyPoints* is the number of keypoints. Each keypoint is of the form [*x*,*y*]. The pattern must be planar, so all *z*-coordinates are assumed to be `0`.

Data Types: `double`

`intrinsics` — Intrinsic parameters of each camera
numCameras-element array

Intrinsic parameters of each camera, specified as a *numCameras*-element array. *numCameras* is the number of cameras and each element is either a `cameraIntrinsics` or `fisheyeIntrinsics` object.

- If all cameras share the same model (lens characteristics), the array is contains `cameraIntrinsics` or `fisheyeIntrinsics` objects.
- If different models are used across the cameras, the array is a cell array, with each cell containing a different type of camera intrinsics object.

The intrinsic parameters of the cameras are not refined by this function. By default, all the keypoints are corrected for distortion based on the lens distortion specified in the `intrinsics` input. To enhance speed, you can use the `undistortImage` or `undistortFisheyeImage` function to correct lens distortion of the images. After correcting the images, detect keypoints on these undistorted images. Then, use the undistorted keypoints along with the corresponding intrinsics object to perform calibration.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `estimateMultiCameraParameters(imagePoints,worldPoints,intrinsics,WorldUnits="mm")` sets the world units to millimeters.

`WorldUnits` — Units of world points
`"mm"` (default) | character vector | string scalar

Units of world points, specified as a character vector or string scalar.

`ReferenceCameraIndex` — Index of camera that defines reference coordinate system
`1` (default) | integer

Index of the camera that defines the reference coordinate system, specified as an integer. All other camera poses are defined relative to the reference camera. Use the `ReferenceCameraPose` name-value argument to specify the reference camera pose.

`ReferenceCameraPose` — Pose of reference camera
`rigidtform3d(eye(4))` (default) | `rigidtform3d` scalar

Pose of reference camera with respect to a world coordinate system, specified as a `rigidtform3d` scalar.

`Verbose` — Display detection progress information
`false` or`0` (default) | `true` or `1`

Display detection progress information in the Command Window, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set `Verbose` to `true` to display progress information.

Output Arguments

`cameraParams` — Parameters of multi-camera system
`multiCameraParameters` object

Parameters of a multi-camera system, returned as a `multiCameraParameters` object. The extrinsic parameters include the rotation and translation that describe the position and orientation of each camera with respect to the reference camera.

`imagesUsed` — Images used to estimate camera parameters
numViews-by-*numCameras* logical array

Images you use to estimate camera parameters, returned as a *numViews*-by-*numCameras* logical array. *numViews* corresponds to the number of camera views used and *numCameras* is the number of cameras. The array indicates which images you used to estimate the camera parameters. A logical `true` value in the array indicates which images you used to estimate the camera parameters.

Version History

Introduced in R2025a

R2026a: Added support for multiple calibration patterns and cameras with non-overlapping fields of view

You can now specify a 5-D array for `imagePoints` to include the number of calibration patterns and from multiple cameras with or without overlapping fields of view. Use the `ReferenceCameraPose` name-value argument to specify the camera pose with respect to a world coordinate system.

References

[1] Rameau, Francois, Jinsun Park, Oleksandr Bailo, and In So Kweon. "MC-Calib: A generic and robust calibration toolbox for multi-camera systems."*Computer Vision and Image Understanding*"217 (2022): 103353.

[2] Tsai, Roger Y., and Reimar K. Lenz. "A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration." *IEEE Transactions on robotics and automation* 5, no. 3 (1989): 345-358.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Objects
`multiCameraParameters` | `stereoParameters` | `cameraParameters` | `fisheyeParameters` | `cameraIntrinsics`

Functions
`detectPatternPoints` | `detectMultiPatternPoints` | `showExtrinsics` | `patternWorldPoints` | `showReprojectionErrors` | `plotCamera`

Topics
“What Is Camera Calibration?”
“What Is Multi-Camera Calibration?”
“What Is Robot Hand-Eye Calibration?”
“Calibration Patterns”
“Prepare Cameras and Capture Images for Multi-Camera Calibration”
“Using the Single Camera Calibrator App”
“Coordinate Systems”
“Evaluating the Accuracy of Single Camera Calibration”
“3-D Motion Reconstruction Using Multiple Cameras”

detectPatternPoints

Detect keypoints of calibration pattern in images from multiple cameras

Syntax

```
imagePoints = detectPatternPoints(imageFileNames,"charuco-board",patternDims,
markerFamily,checkerSize,markerSize)
imagePoints = detectPatternPoints(imageFileNames,"aprilgrid",patternDims,
tagFamily)
imagePoints = detectPatternPoints(imageFileNames,"checkerboard",patternDims)
imagePoints = detectPatternPoints(imageFileNames,"circle-grid-asymmetric",
patternDims)

imagePoints = detectPatternPoints(imageFileNames,patternDetector,
numKeyPoints)

imagePoints = detectPatternPoints( ___ ,Name=Value)
```

Description

Natively Supported Patterns

`imagePoints = detectPatternPoints(imageFileNames,"charuco-board",patternDims, markerFamily,checkerSize,markerSize)` detects the keypoints of a ChArUco board calibration pattern in images `imageFileNames`, with dimensions specified by `patternDims`, `markerFamily`, and `checkerSize`.

`imagePoints = detectPatternPoints(imageFileNames,"aprilgrid",patternDims, tagFamily)` detects the keypoints of an AprilGrid calibration pattern in images `imageFileNames`, with dimensions specified by `patternDims`.

`imagePoints = detectPatternPoints(imageFileNames,"checkerboard",patternDims)` detects the keypoints of a fully visible checkerboard calibration pattern in images `imageFileNames`, with dimensions specified by `patternDims`. The values of the elements of `patternDims` must not be equal.

`imagePoints = detectPatternPoints(imageFileNames,"circle-grid-asymmetric", patternDims)` detects keypoints of a circle grid calibration pattern in images `imageFileNames`, with dimensions specified by `patternDims`.

Custom Pattern Detector

`imagePoints = detectPatternPoints(imageFileNames,patternDetector, numKeyPoints)` detects the specified number of uniquely detectable keypoints `numKeyPoints` of a custom calibration pattern that can be detected in an image using the specified detector function `patternDetector`.

Optional Arguments

`imagePoints = detectPatternPoints( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Verbose=true` displays the progress of the detection process.

Examples

Detect AprilGrid in Images from Multiple Cameras

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-aprilgr
calibImagesDir = fullfile(pwd,"overlapping-cameras-aprilgrid");
calibImagesZip = fullfile(pwd,"overlapping-cameras-aprilgrid.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (54 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (54 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the AprilGrid properties.

```
patternDims = [6 6];
tagFamily = "tag36h11";
```

Detect the key points of the AprilGrid in the calibration images.

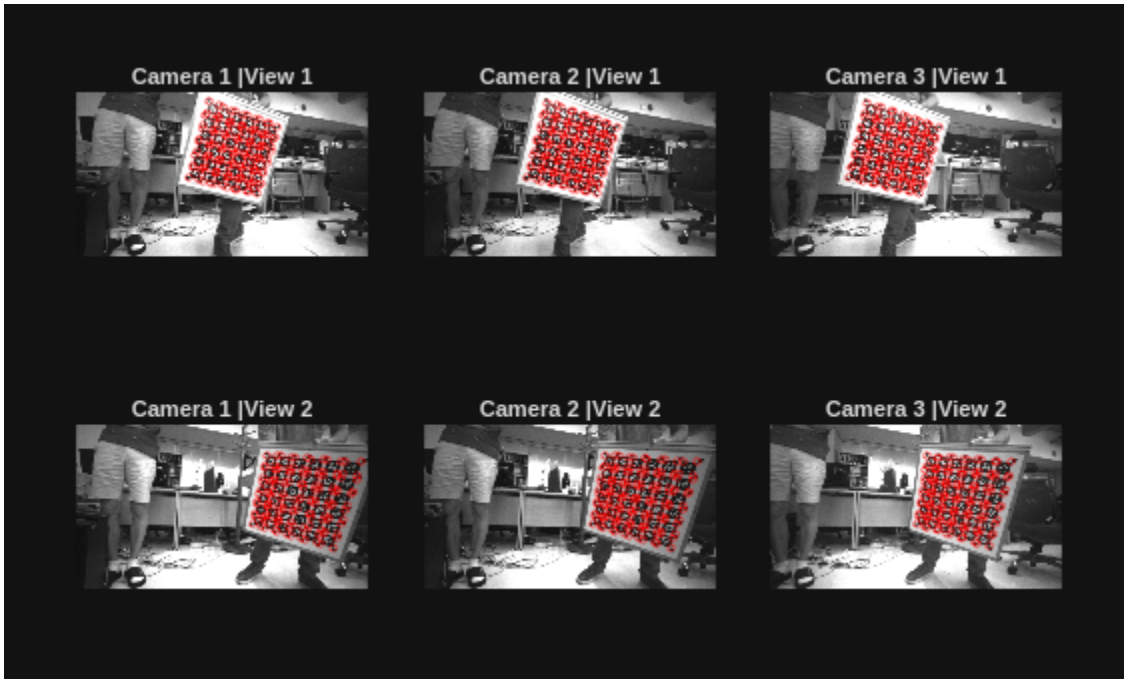
```
imagePoints = detectPatternPoints(imageFiles, "aprilgrid", ...
    patternDims,tagFamily);
```

```
[==================================================] 100%
Elapsed time: 00:00:32
Estimated time remaining: 00:00:00
```

Display the detected keypoints of the AprilGrid in the first two views for three of the six cameras.

```
tiledlayout(2,3,TileSpacing="compact",Padding="compact");
for viewIdx = 1:2
    for camIdx = 1:3
        nexttile
        imshow(imageFiles{viewIdx,camIdx})
        title("Camera " + camIdx + "|View " + viewIdx)
        hold on
        plot(imagePoints(:,1,viewIdx,camIdx),imagePoints(:,2,viewIdx,camIdx), ...
            "ro",MarkerSize=3)
```

```
    end
end
```



Calibrate a Multi-Camera System using a ChArUco board

Estimate the relative position and orientation of six cameras with overlapping fields of view by using calibration images that contain a single ChArUco board.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-charuco
calibImagesDir = fullfile(pwd,"overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (52 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (52 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
```

```
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
numKeyPoints = prod(patternDims - 1);
minMarkerId = 144;
```

Detect the key points of the ChArUco board in the calibration images.

```
imagePoints = detectPatternPoints(imageFiles,"charuco-board",patternDims, ...
    markerFamily,checkerSize,markerSize,MinMarkerID=minMarkerId);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world points for the pattern.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

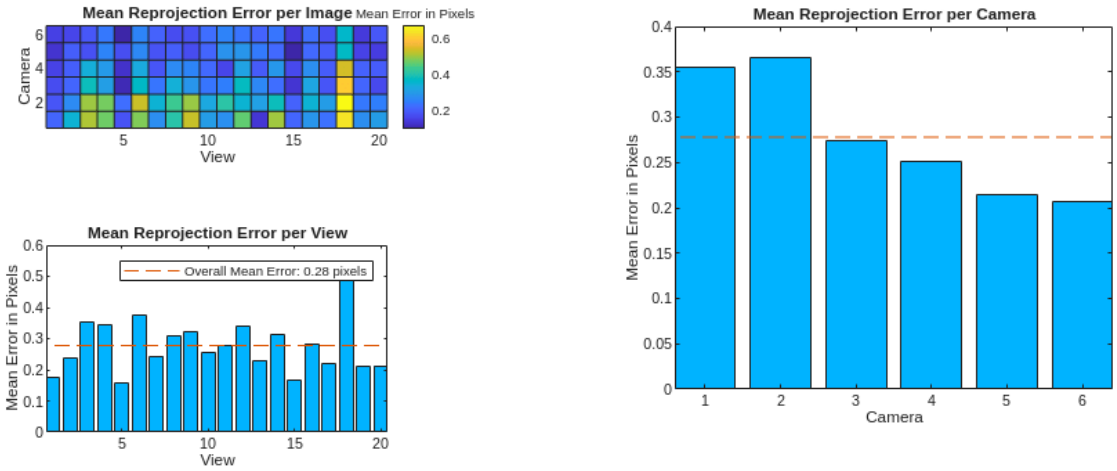
```
ld = load("sixCameraIntrinsics.mat");
```

Perform multi-camera calibration.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

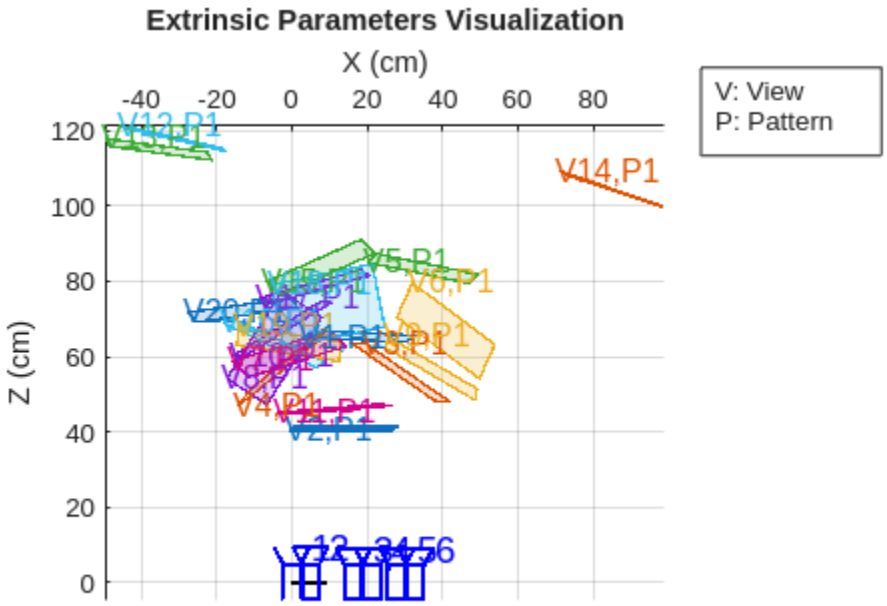
Visualize the calibration accuracy.

```
figure(Position=[100,100,1000,400])
showReprojectionErrors(params)
```



Visualize the camera extrinsic parameters.

```
figure
showExtrinsics(params)
view(2)
```



Input Arguments

`imageFileNames` — Filenames of camera calibration patterns

numViews-by-*numCameras* string array | *numViews*-by-*numCameras* cell array

Filenames of the camera calibration patterns, specified as a *numViews*-by-*numCameras* string or cell array. *numViews* is the number of calibration pattern views for each camera, and *numCameras* is the number of cameras. Because you must specify more than two cameras, *numCameras* must be greater than 2.

Organize your calibration data into folders that contain synchronized images for each camera, ensuring that the filenames are consistent across all camera folders. This arrangement aligns images from different cameras that correspond to the same scene or subject. By using identical filenames in the folder for each camera, you can ensure that the images are properly synchronized.


CalibrationData
Camera-1
image1.ext
image2.ext
image N.ext
Camera-2
image1.ext
image2.ext
image N.ext
Camera-numCameras
image1.ext
image2.ext
image N.ext

`patternDims` — Pattern dimensions
two-element vector

Pattern dimensions, specified as a two-element vector that represents the number of checker boxes or circles along the rows (*dim1*) and columns (*dim2*) of the pattern. For an AprilGrid pattern, ensure that the pattern is oriented with the origin at the top-left as you count the rows and columns. For more information on patterns, see “Calibration Patterns”.

`markerFamily` — ArUco marker family
`"DICT_ARUCO_ORIGINAL"` | `"DICT_4X4_1000"` | `"DICT_5X5_1000"` | `"DICT_6X6_1000"` | `"DICT_7X7_1000"`

ArUco marker family, specified as one of these options:

- `"DICT_ARUCO_ORIGINAL"`
- `"DICT_4X4_1000"`
- `"DICT_5X5_1000"`
- `"DICT_6X6_1000"`
- `"DICT_7X7_1000"`

`checkerSize` — ChArUco board checker box side length
scalar

ChArUco board checker box side length, specified as a scalar in world units, such as millimeters or inches. You must specify `checkerSize` and `markerSize` in the same world units, and `checkerSize` must be a larger value than `markerSize`.

`markerSize` — Size of ArUco marker
positive integer

Size of ArUco marker, specified as a positive integer. This value sets the length of one side of a square ArUco marker in world units such as millimeters, centimeters, meters, or inches. The value of `checkerSize` must be greater than the value of `markerSize`.

`tagFamily` — AprilTag family
`"tag36h11"` | `"tag25h9"` | `"tag16h5"`

AprilTag family, specified as `"tag36h11"`, `"tag25h9"`, or `"tag16h5"`.

`patternDetector` — Custom pattern detector
function handle

Custom pattern detector, specified as a function handle with this syntax:

`[imagePoints,hasDetected] = patternDetector(filename)`.

Input

- `filename` — Name of the image to be processed, specified as a string scalar or character vector.

Output

- `imagePoints` — Coordinates of the detected keypoints in the image, returned as a *numKeyPoints*-by-2 matrix. *N* is the number of keypoints.

- `hasDetected` — Logical that indicates whether the function detected the pattern, returned as `true` when the function detects the pattern and `false` when it does not.

`numKeyPoints` — Number of keypoints that can be detected
positive integer

Number of keypoints that can be detected in the custom pattern, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectPatternPoints(imageFileNames,"circle-grid-asymmetric",patternDims,Verbose=1)` enables progress display to the screen.

`Verbose` — Display detection progress information
`true` or `1` (default) | `false` or `0`

Display detection progress information in the Command Window, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set `Verbose` to `true` to display progress information.

`MinMarkerID` — Minimum marker ID of pattern to detect
`0` (default) | non-negative integer

Minimum marker ID of the pattern to detect, specified as a non-negative integer. This argument applies only to ChArUco and AprilGrid patterns.

Output Arguments

`imagePoints` — Image coordinates for keypoints found in each image
numKeyPoints-by-2-by-*numViews*-by-*numCameras* array

Image coordinates for the keypoints found in each image, returned as a *numKeyPoints*-by-2-by-*numViews*-by-*numCameras* matrix. *numKeyPoints* is the number of keypoints found for each image. *numViews* is the number of views with respect to the camera, and *numCameras* is the number of cameras. For example, `imagePoints(:,:,i,j)` contains the points from camera *j* when the pattern is located at the i^{th} view. For images in which a camera does not detect a pattern, the corresponding keypoint values are `NaN`s.

Version History

Introduced in R2025a

R2026a: Specify minimum marker ID for ChArUco and AprilGrid patterns

Use the `MinMarkerID` name-value argument to specify the minimum marker ID for ChArUco or AprilGrid patterns.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectPatternPoints` function supports thread-based environments with these usage notes and limitations:

Support for running in a thread-based environment on a background pool is available only when you set `Verbose` to `false`. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Objects

`multiCameraParameters`

Functions

`patternWorldPoints` | `estimateMultiCameraParameters` | `detectMultiPatternPoints`

Topics

“What Is Camera Calibration?”
“What Is Robot Hand-Eye Calibration?”
“Calibration Patterns”
“Prepare Cameras and Capture Images for Multi-Camera Calibration”
“Using the Single Camera Calibrator App”
“Coordinate Systems”
“Evaluating the Accuracy of Single Camera Calibration”
“3-D Motion Reconstruction Using Multiple Cameras”

estimateCameraRobotTransform

Estimate pose of camera relative to robot using hand-eye calibration

Syntax

```
relativePose = estimateCameraRobotTransform(cameraExtrinsics,gripperPose,
configuration)
```

Description

`relativePose = estimateCameraRobotTransform(cameraExtrinsics,gripperPose, configuration)` estimates the pose `relativePose` of a camera relative to a robot frame using hand-eye calibration for a moving or stationary `configuration`. The value of `cameraExtrinsics` is the pose of the calibration object relative to the camera. `gripperPose` is the pose of the robot gripper relative to the robot base.

Examples

Robot Hand-Eye Calibration for Stationary-Camera Configuration

Load a MAT file that contains this robot hand-eye calibration data.

- `intrinsics`— Intrinsic parameters of the camera, stored as a `cameraIntrinsics` object. These parameters have been estimated using the process from the “Using the Single Camera Calibrator App” topic.
- `camExtrinsics`: Positions and orientations of the calibration pattern relative to the camera in 15 calibration images, stored as a 15-by-1 vector of `rigidtform3d` objects. These extrinsic parameters have been computed from the images containing the calibration pattern by using the `estimateExtrinsics` function with the estimated `intrinsics`.
- `gripperPoses`: Positions and orientations of the robot gripper, relative to the robot base, while capturing the 15 calibration images, stored as a 15-by-1 vector of `rigidtform3d` objects.

```
ld = load("stationaryCameraCalibrationData.mat");
```

Perform robot hand-eye calibration for the stationary-camera configuration to estimate the camera pose relative to robot base.

```
camPose = estimateCameraRobotTransform(ld.camExtrinsics,ld.gripperPoses,"stationary-camera");
```

To verify the calibration results, follow these steps:

1. Place a detectable object, such a checkerboard calibration pattern, near the robot and record the distance from the object to the robot base.
2. Capture an image of the object from the camera.
3. Estimate the camera extrinsic parameters for the captured image.
4. Transform object keypoints detected in image coordinates to world coordinates.
5. Verify that the transformed world coordinates match the distance recorded in the first step.

This stationary camera configuration has already been used to capture an image of a calibration checkerboard. Load the captured checkerboard image. Load the captured checkerboard image.

```
I = imread("stationaryCameraSampleImage.jpg");
```

Detect checkerboard keypoints, generate corresponding world points in the checkerboard coordinate system, and estimate the camera extrinsic parameters for the captured image.

```
[imagePts,patternDims] = detectCheckerboardPoints(I);
squareSize = 0.029; % in meters
patternWorldPts2D = patternWorldPoints("checkerboard",patternDims,squareSize);
cameraExtrinsics = estimateExtrinsics(imagePts,patternWorldPts2D,ld.intrinsics);
```

Using the cameraExtrinsics and `camPose` objects, transform any image coordinates on the same plane as the calibration pattern to world coordinates in the robot coordinate system. Use the checkerboard keypoints is used as the object of interest.

Transform object points from the image coordinate system to the camera coordinate system.

```
objectWorldPts2D = img2world2d(imagePts,cameraExtrinsics,ld.intrinsics);
objectWorldPts3D = padarray(objectWorldPts2D,[0 1],"post");
```

Use `camPose` to find the transformation of the camera relative to the robot base frame, and transform the object points to the robot base frame.

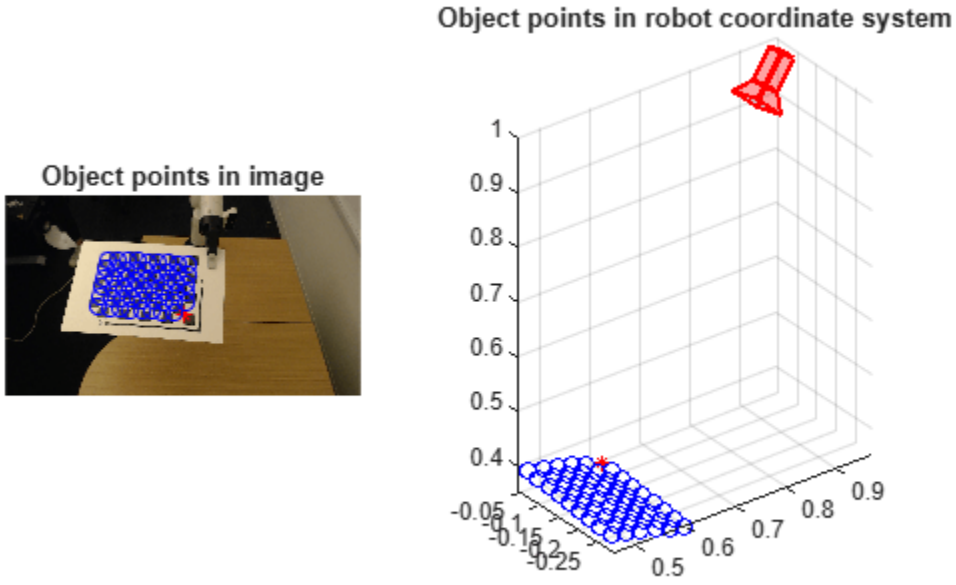
```
pointsToBasePose = rigidtform3d(camPose.A * cameraExtrinsics.A);
objectPts = transformPointsForward(pointsToBasePose,objectWorldPts3D);
```

Visualize the object keypoints detected in the image and transformed to the robot coordinate system. Display the first checkerboard keypoint with a different marker and color.

```
tiledlayout(1,2)

nexttile
imshow(I)
hold on
plot(imagePts(1,1),imagePts(1,2),"r*");
plot(imagePts(2:end,1),imagePts(2:end,2),"bo");
title("Object points in image")

nexttile
plotCamera(AbsolutePose=camPose,Size=.03)
hold on
plot3(objectPts(1,1),objectPts(1,2),objectPts(1,3),"r*")
plot3(objectPts(2:end,1),objectPts(2:end,2),objectPts(2:end,3),"bo")
axis equal
grid on
title("Object points in robot coordinate system")
```



Robot Hand-Eye Calibration for Moving-Camera Configuration

Load a MAT file that contains this robot hand-eye calibration data.

- `intrinsics`— Intrinsic parameters of the camera, stored as a `cameraIntrinsics` object. These parameters have been estimated using the process from the “Using the Single Camera Calibrator App” topic.
- `camExtrinsics`: Positions and orientations of the calibration pattern relative to the camera in 15 calibration images, stored as a 15-by-1 vector of `rigidtform3d` objects. These extrinsic parameters have been computed from the images containing the calibration pattern by using the `estimateExtrinsics` function with the estimated `intrinsics`.
- `gripperPoses`: Positions and orientations of the robot gripper, relative to the robot base, while capturing the 15 calibration images, stored as a 15-by-1 vector of `rigidtform3d` objects.

```
ld1 = load("movingCameraCalibrationData.mat");
```

Perform robot hand-eye calibration for the moving-camera configuration to estimate the camera pose relative to the robot gripper.

```
camPose = estimateCameraRobotTransform(ld1.camExtrinsics,ld1.gripperPoses,"moving-camera");
```

To verify the calibration results, follow these steps:

1. Place a detectable object, such a checkerboard calibration pattern, near the robot and record the distance from the object to the robot base.
2. Capture an image of the object from the camera.
3. Record the robot gripper pose relative to the robot base while capturing that image.

4. Estimate the camera extrinsic parameters for the captured image.
5. Transform object keypoints detected in image coordinates to world coordinates.
6. Verify that the transformed world coordinates match the distance recorded in the first step.

This moving camera configuration has already been used to capture an image of a calibration checkerboard and record the pose of the robot gripper relative to the robot base. Load the captured checkerboard image and the corresponding robot gripper pose.

```
I = imread("movingCameraSampleImage.jpg");
ld2 = load("movingCameraSamplePose.mat");
```

Detect checkerboard keypoints, generate corresponding world points in the checkerboard coordinate system, and estimate the camera extrinsic parameters for the captured image.

```
[imagePts,patternDims] = detectCheckerboardPoints(I);
squareSize = 0.029; % in meters
patternWorldPts2D = patternWorldPoints("checkerboard",patternDims,squareSize);
cameraExtrinsics = estimateExtrinsics(imagePts,patternWorldPts2D,ld1.intrinsics);
```

Using the cameraExtrinsics and `camPose` objects, transform any image coordinates on the same plane as the calibration pattern to world coordinates in the robot coordinate system. Use the checkerboard keypoints is used as the object of interest.

Transform object points from the image coordinate system to the camera coordinate system.

```
objectWorldPts2D = img2world2d(imagePts,cameraExtrinsics,ld1.intrinsics);
objectWorldPts3D = padarray(objectWorldPts2D,[0 1],"post");
```

Use `camPose` to find the transformation of the camera relative to the robot base frame, and transform the object points to the robot base frame.

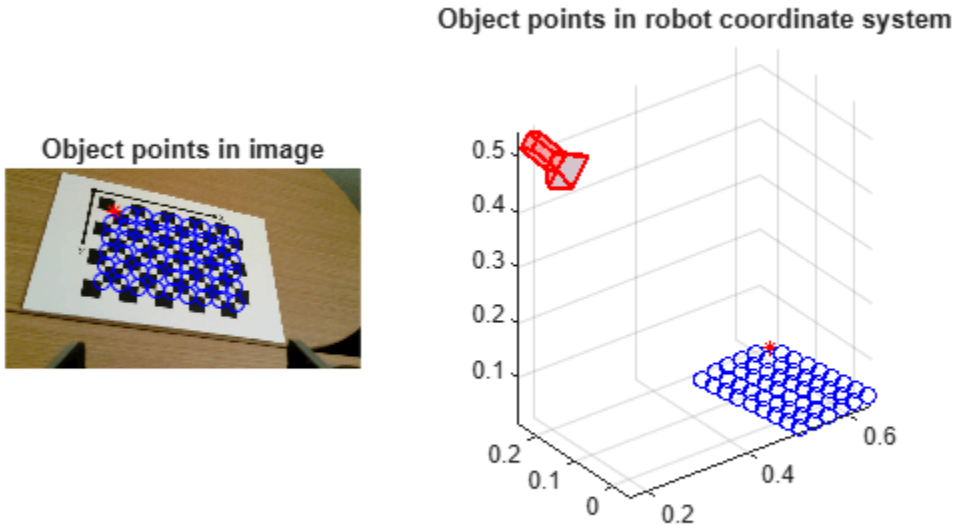
```
camToBasePose = rigidtform3d(ld2.gripperPose * camPose.A);
pointsToBasePose = rigidtform3d(camToBasePose.A * cameraExtrinsics.A);
objectPts = transformPointsForward(pointsToBasePose, objectWorldPts3D);
```

Visualize the object keypoints detected in the image and transformed to the robot coordinate system. Display the first checkerboard keypoint with a different marker and color.

```
tiledlayout(1,2)

nexttile
imshow(I)
hold on
plot(imagePts(1,1),imagePts(1,2),"r*");
plot(imagePts(2:end,1),imagePts(2:end,2),"bo");
title("Object points in image")

nexttile
plotCamera(AbsolutePose=camToBasePose,Size=.03)
hold on
plot3(objectPts(1,1),objectPts(1,2),objectPts(1,3),"r*")
plot3(objectPts(2:end,1),objectPts(2:end,2),objectPts(2:end,3),"bo")
axis equal
grid on
title("Object points in robot coordinate system")
```



Input Arguments

`cameraExtrinsics` — Camera extrinsic parameters

M-by-1 vector of `rigidtform3d` objects.

Camera extrinsic parameters, specified as an *M*-by-1 vector of `rigidtform3d` objects. The extrinsic parameters represent the transformation from the calibration object to the camera coordinate frame.

`gripperPose` — Pose of robot gripper relative to robot base

M-by-1 vector of `rigidtform3d` objects

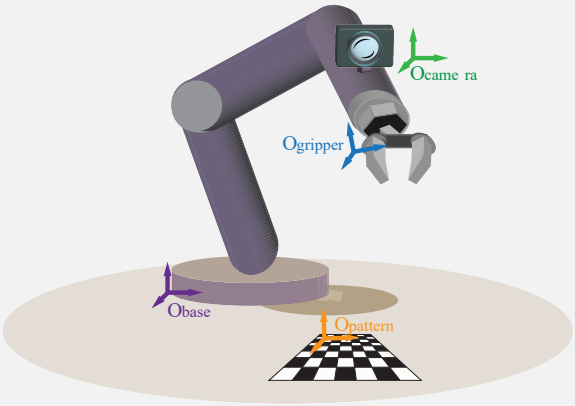
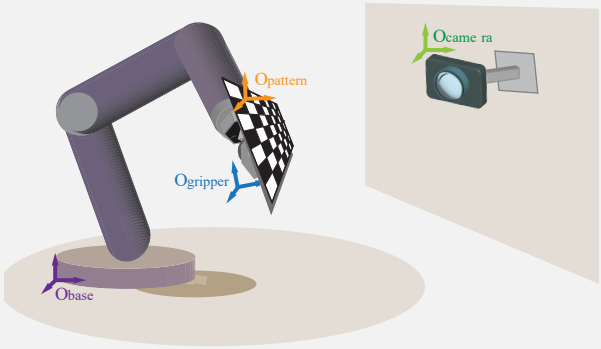
Pose of the robot gripper relative to the robot base, specified as an *M*-by-1 vector of `rigidtform3d` objects.

`configuration` — Robot camera configuration

`"moving-camera"` | `"stationary-camera"`

Robot camera configuration, specified as `"moving-camera"` or `"stationary-camera"`.

- `"moving-camera"` — Also referred to as "eye-in-hand," a setup where the camera is mounted on the robot manipulator, allowing it to move with the robot while the calibration target is fixed in the world coordinate frame.
- `"stationary-camera"` — Also referred to as "eye-to-hand," a setup where the camera is fixed in the world coordinate frame while the robot manipulator moves the calibration target within the camera field of view.

"moving-camera"	"stationary-camera"
	

Output Arguments

`relativePose` — Pose of camera relative to robot frame
`rigidtform3d` object

Pose of the camera relative to the robot frame, returned as a `rigidtform3d` object. In a `"moving-camera"` configuration, `relativePose` returns the transformation of position and orientation from the camera to the robot gripper. In a `"stationary-camera"` configuration, `relativePose` returns the transformation of position and orientation from the camera to robot base.

Algorithms

The hand-eye calibration method originated from a seminal paper by Tsai and Lenz [1 on page 3-44]. Their approach is based on solving the equation $AX = XB$, where:

- A is the transformation of the robot gripper from one position to the next.
- B is the transformation of the camera or the calibration target (whichever is moving with the robot gripper) from one position to the next.
- X is the unknown transformation.

The approach uses the least squares method to solve the equation $AX = XB$. To obtain the intermediate transformations A and B, you must have information about the robot gripper pose and the camera pose at two different locations. In general, you can calibrate any sensors that can establish the relationship $AX = XB$ using the hand-eye calibration method.

Version History

Introduced in R2025a

References

[1] Tsai, R.Y., and R.K. Lenz. "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration." *IEEE Transactions on Robotics and Automation* 5, no. 3 (1989): 345-58.

See Also

Apps
Camera Calibrator

Functions
`getTransform` | `estimateExtrinsics` | `extr2pose` | `estrelpose`

Topics
“What Is Robot Hand-Eye Calibration?”
“Estimate Pose of Moving Camera Mounted on a Robot”
“Estimate Pose of Fixed Camera Relative to Robot Base” (Robotics System Toolbox)

select

Select point or region features during code generation

Syntax

```
indexedPoints = select(points,ind)
```

Description

`indexedPoints = select(points,ind)` selects one or more sets of point or region features using the specified indices `ind`.

Examples

Select Point Features During Code Generation

Read an image into MATLAB® the workspace.

```
I = imread("cameraman.tif");
```

Define the `helperGetSubsetOfPoints` function which, detects and selects SURF points. You must save the helper function as a MATLAB script file with a `.m` extension.

```
function indexedLocation = helperGetSubsetOfPoints(I)
    points = detectSURFFeatures(I);
    indexedPoints = select(points,30:100);
    indexedLocation = indexedPoints.Location;
end
```

Create a configuration object for generating a MEX function.

```
cfg = coder.config("mex");
```

Generate the MEX function from the helper function `helperGetSubsetOfPoints` using the configuration `cfg`. The `helperGetSubsetOfPoints` function detects SURF features in the input image and selects a specified set of point features using the `select` object function of the `SURFPoints` object. The `helperGetSubsetOfPoints` function outputs the locations of the selected SURF point features in the image. You can use the `select` function only for code generation.

```
codegen -config cfg helperGetSubsetOfPoints -args {I}
```

```
Code generation successful.
```

Call the generated MEX function with the image `I` as input. Objects cannot be returned as outputs during code generation. Therefore, the `helperGetSubsetOfPoints` function outputs the locations of the selected SURF point features.

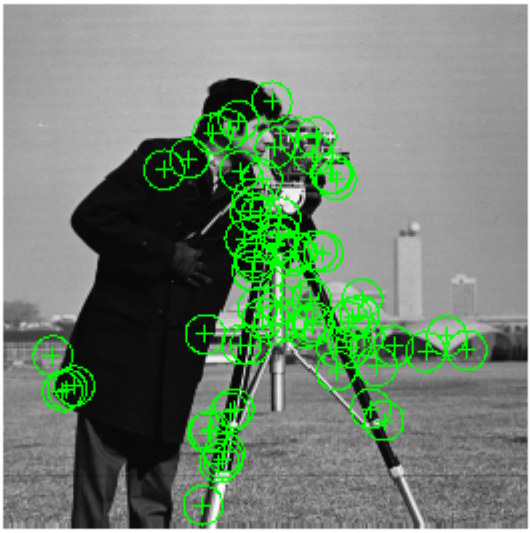
```
locations = helperGetSubsetOfPoints_mex(I);
```

Convert the output locations to `SURFPoints` objects.

```
points = SURFPoints(locations);
```

Display the selected SURF point features.

```
figure
imshow(I)
hold on
plot(points)
hold off
```



Input Arguments

`points` — Input features

`MSERRegions` object | `SURFPoints` object | `ORBPoints` object | `KAZEPoints` object | `BRISKPoints` object | `SIFTPoints` object | `cornerPoints` object

Input features, specified as an `MSERRegions`, `SURFPoints`, `ORBPoints`, `KAZEPoints`, `BRISKPoints`, `SIFTPoints`, or `cornerPoints` object.

`ind` — Indices of features to be selected

scalar | vector

Indices of the features to be selected from the input, specified as a scalar or vector.

Output Arguments

`indexedPoints` — Output features
`MSERRegions` object | `SURFPoints` object | `ORBPoints` object | `KAZEPoints` object | `BRISKPoints` object | `SIFTPoints` object | `cornerPoints` object

Output features, returned as an `MSERRegions`, `SURFPoints`, `ORBPoints`, `KAZEPoints`, `BRISKPoints`, `SIFTPoints`, or `cornerPoints` object. The `indexedPoints` output object type matches the type of object specified to the `points` input argument.

Version History

Introduced in R2024b

See Also

`detectBRISKFeatures` | `detectMSERFeatures` | `detectSURFFeatures` | `detectORBFeatures` | `detectHarrisFeatures` | `detectFASTFeatures` | `detectKAZEFeatures`

compareTrajectories

Compare estimated trajectory against ground truth

Syntax

```
metrics = compareTrajectories(estimatedPoses,groundTruthPose)
metrics = compareTrajectories( ___ ,Name=Value)
```

Description

`metrics = compareTrajectories(estimatedPoses,groundTruthPose)` returns metrics for the accuracy of an estimated trajectory, `estimatedPoses`, against a ground truth trajectory `groundTruthPose`, to assess the performance of an odometry or a Simultaneous Localization and Mapping (SLAM) system.

The function aligns the estimated trajectory to the ground truth trajectory by performing a least-square fitting between the translation components of poses in the two trajectories.

`metrics = compareTrajectories( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntaxes. For example, `AlignmentType="none"` sets the transformation type to `"none"` when aligning the estimated trajectory to the ground truth trajectory.

Input Arguments

`estimatedPoses` — Estimated poses

M-element array | `se3` object

Estimated poses, specified as an *M*-element array of `rigidtform3d` or `se3` objects. The estimated poses can be obtained from an odometry or a SLAM system.

`groundTruthPose` — Ground truth poses

M-element array | `se3` object

Ground truth poses, specified as an *M*-element array of `rigidtform3d` or `se3` objects. The ground truth poses are obtained from an external ground truth system.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `compareTrajectories(estimatedPoses,groundTruthPose,AlignmentType="none")` sets the transformation type to `"none"` when aligning the estimated trajectory to the ground truth trajectory.

`AlignmentType` — Transformation type

`"rigid"` (default) | `"similarity"` | `"none"`

Transformation type to align the estimated trajectory to the ground truth trajectory, specified as `"rigid"`, `"similarity"`, or `"none"`. To disable trajectory alignment, set `AlignmentType` to `"none"`.

`EvaluationInterval` — Fixed distance to compute relative trajectory error
1 (default) | scalar

Fixed distance to compute relative trajectory error for each pose within the estimated trajectory, specified as a scalar in world units.

Output Arguments

`metrics` — Error metrics object
`trajectoryErrorMetrics` object

Error metrics object, returned as an `trajectoryErrorMetrics` object. The object stores error metrics including the absolute trajectory error and the relative trajectory error.

Version History

Introduced in R2024b

References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. 2012. "A Benchmark for the Evaluation of RGB-D SLAM Systems." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 573-580.

[2] Zhang, Zichao, and Davide Scaramuzza. 2018. "A Tutorial on Quantitative Trajectory Evaluation for Visual (-Inertial) Odometry." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 7244-7251.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`trajectoryErrorMetrics` | `monovslam` | `stereovslam` | `rgbdvslam` | `se3` | `rigidtform3d`

generateCharucoBoard

Generate ChArUco board image

Syntax

```
I = generateCharucoBoard(imageSize,patternDims,markerFamily,checkerSize,
markerSize)
I = generateCharucoBoard( ___ ,Name=Value)
```

Description

`I = generateCharucoBoard(imageSize,patternDims,markerFamily,checkerSize, markerSize)` returns a grayscale ChArUco board image.

`I = generateCharucoBoard( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `MinMarkerID=3` sets the lowest marker ID in the board to 3.

Examples

Generate ChArUco Board Image for Single Camera Calibration

Specify properties to generate the ChArUco board. Image, checker, and marker size are in pixels.

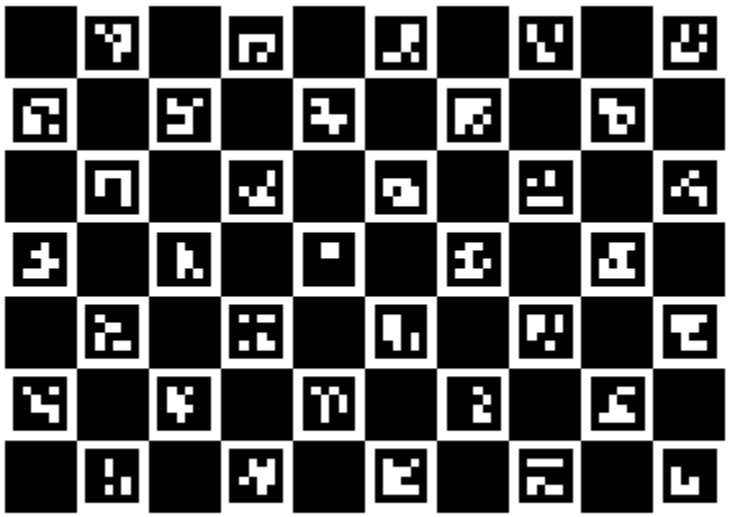
```
patternDims = [7 10];
family = "DICT_4X4_1000";
imageSize = [900 1200];
checkerSize = 100;
markerSize = 75;
```

Generate the ChArUco board based on specified properties.

```
I = generateCharucoBoard(imageSize,patternDims,family,checkerSize,markerSize);
```

Display the generated ChArUco board image.

```
imshow(I)
```



Write the ChArUco board image to a file.

```
imwrite(I,"ChArUco_7X10_4X4_100_75.png")
```

Next Steps: Print the ChArUco board image file. Use the printed board for capturing calibration images with the camera you intend to calibrate. To capture calibration images for an effective single camera calibration, see “Prepare Camera and Capture Images for Camera Calibration”.

Generate Multiple ChArUco Boards for Multi-Camera Calibration

Generate multiple ChArUco boards that are needed to calibrate a multi-camera system.

Specify properties to generate the ChArUco board. Image, checker, and marker size are in pixels.

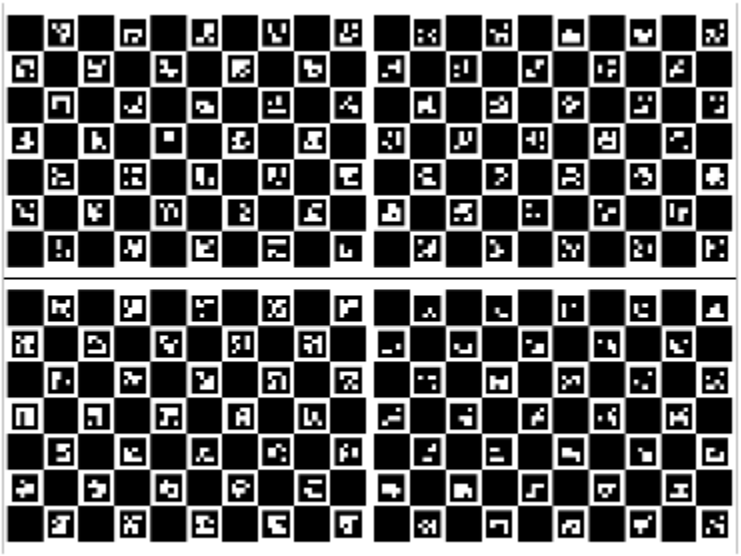
```
patternDims = [7 10];
family = "DICT_4X4_1000";
imageSize = [900 1200];
checkerSize = 100;
markerSize = 75;
```

Generate four ChArUco boards based on specified properties with unique ArUco markers. To ensure uniqueness of ArUco markers, separate MinMarkerID of each board by the number of markers in each board. For example, a 7-by-10 ChArUco board contains 35 ArUco markers. To generate four 7-by-10 ChArUco boards that can be used for multi-camera calibration, their `MinMarkerIDs` must be 0,35,70 and 105.

```
numMarkers = floor(prod(patternDims)/2);
I1 = generateCharucoBoard(imageSize,patternDims,family,checkerSize,markerSize,MinMarkerID=0);
I2 = generateCharucoBoard(imageSize,patternDims,family,checkerSize,markerSize,MinMarkerID=numMark
I3 = generateCharucoBoard(imageSize,patternDims,family,checkerSize,markerSize,MinMarkerID=2*numMa
I4 = generateCharucoBoard(imageSize,patternDims,family,checkerSize,markerSize,MinMarkerID=3*numMa
```

Display the generated ChArUco boards.

```
montage({I1,I2,I3,I4},BorderSize=[1 1])
```



Write the ChArUco board images to files.

```
imwrite(I1,"ChArUco_7X10_4X4_100_75_1.png")
imwrite(I2,"ChArUco_7X10_4X4_100_75_2.png")
imwrite(I3,"ChArUco_7X10_4X4_100_75_3.png")
imwrite(I4,"ChArUco_7X10_4X4_100_75_4.png")
```

Next Steps: Print the ChArUco board image files. Use the printed patterns for capturing calibration images with the cameras you intend to calibrate. To capture calibration images for an effective multi-camera calibration, see “Prepare Cameras and Capture Images for Multi-Camera Calibration”.

Input Arguments

`imageSize` — ChArUco board size

2-element vector (default)

ChArUco board size in pixels, specified as a 2-element vector of the form [*height*,*width*].

`patternDims` — Pattern dimensions

two-element vector

Pattern dimensions, specified as a two-element vector that represents the number of checker boxes in the *dim1* and *dim2* dimensions of the ChArUco board image, respectively. For more details on ChArUco board patterns, see “Calibration Patterns”.

`markerFamily` — ArUco marker family name

character vector | string scalar | vector of strings | cell array of character vectors

ArUco marker family name used in CharUco board, specified as a character vector, string scalar, vector of strings, or cell array of character vectors. Each element must be one of the valid ArUco marker families listed in the table. Select a marker family suitable for your application based on the total number of markers needed, the marker length, and the Hamming distance between the marker code words. The minimum Hamming distance between markers in the same family, determine the ability to detect them and correct errors. Marker families with less number of markers can be more robust against errors than a family with a large number of markers, which can increase marker detection difficulty due to a greater number of bits to extract. These general rules apply:

- The less number of bits to detect, the longer the range in distance a marker can be detected.
- For correct identification and decoding, ArUco markers require a white border, also known as the quiet zone.

Family	Marker Size (bits)	Number of Markers in Family	Minimum Hamming Distance Between Markers (bits)
`"DICT_ARUCO_ORIGINAL"`	5-by-5	1024	3
`"DICT_4X4_50"`	4-by-4	50	4
`"DICT_4X4_100"`		100	3
`"DICT_4X4_250"`		250	2
`"DICT_4X4_1000"`		1000	2
`"DICT_5X5_50"`	5-by-5	50	8
`"DICT_5X5_100"`		100	7
`"DICT_5X5_250"`		250	6
`"DICT_5X5_1000"`		1000	5
`"DICT_6X6_50"`	6-by-6	50	13
`"DICT_6X6_100"`		100	12
`"DICT_6X6_250"`		250	11
`"DICT_6X6_1000"`		1000	9
`"DICT_7X7_50"`	7-by-7	50	19
`"DICT_7X7_100"`		250	18
`"DICT_7X7_250"`		250	17
`"DICT_7X7_1000"`		1000	14

Data Types: `char` | `string` | `cell`

`checkerSize` — ChArUco board checker box side length
scalar

ChArUco board checker box side length, specified as a scalar in world units, such as millimeters or inches. The `checkerSize` and `markerSize` must be specified in the same world units and `checkerSize` must be a larger value than `markerSize`.

`markerSize` — ArUco marker side length
scalar

ArUco marker side length, specified as a scalar in world units, such as millimeters or inches. The `checkerSize` and `markerSize` must be specified in the same world units and `checkerSize` must be a larger value than `markerSize`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `generateCharucoBoard(____,MinMarkerID=3)` sets the lowest marker ID in the board to 3.

`MarginSize` — Number of white pixels to pad margin
`10` (default) | scalar

Number of white pixels to pad margin of output image, specified as a scalar.

`MinMarkerID` — Lowest value marker ID in pattern
`0` (default) | non-negative integer

Lowest value marker ID in the pattern, specified as a non-negative integer. The markers in the pattern must correspond to sequential IDs, each incremented by `1`.

`OriginCheckerColor` — Color of the checker box located in the top-left corner of board
`"black"` (default) | `"white"`

Color of the checker box located in the top-left corner of ChArUco board, specified as `"black"` or `"white"`. The top-left corner value represents the origin of the board. This value must not be set to `"white"` when the board contains an odd number of boxes in the *x* dimension, specified in `patternDims(1)`.

Output Arguments

`I` — ChArUco image
grayscale

ChArUco image, returned as a grayscale image.

Data Types: `uint8`

Version History

Introduced in R2024b

See Also

Functions
`patternWorldPoints` | `detectCharucoBoardPoints` | `estimateCameraParameters`

Objects
`cameraParameters` | `stereoParameters`

Topics
“Calibration Patterns”

patternWorldPoints

Generate world point locations of a camera calibration pattern

Syntax

```
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize)
worldPoints = patternWorldPoints("circle-grid-symmetric",patternDims,
centerDistance)
worldPoints = patternWorldPoints("circle-grid-asymmetric",patternDims,
centerDistance)
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize)
worldPoints = patternWorldPoints("aprilgrid",patternDims,tagSize,tagSpacing)
worldPoints = patternWorldPoints(patternName, ___ )
```

Description

`worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize)` generates keypoints for a checkerboard camera calibration pattern with dimensions specified by `patternDims` and `squareSize`.

`worldPoints = patternWorldPoints("circle-grid-symmetric",patternDims, centerDistance)` generates keypoints for a symmetric circle grid camera calibration pattern with dimensions specified by `patternDims` and `centerDistance`.

`worldPoints = patternWorldPoints("circle-grid-asymmetric",patternDims, centerDistance)` generates keypoints for an asymmetric circle grid camera calibration pattern with dimensions specified by `patternDims` and `centerDistance`.

`worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize)` generates keypoints for a ChArUco board camera calibration pattern with dimensions specified by `patternDims` and `checkerSize`.

`worldPoints = patternWorldPoints("aprilgrid",patternDims,tagSize,tagSpacing)` generates keypoints for an AprilGrid camera calibration pattern with dimensions specified by `patternDims`, `tagSize`, and `tagSpacing`.

`worldPoints = patternWorldPoints(patternName, ___ )` generates keypoints for the calibration pattern specified by `patternName`.

Examples

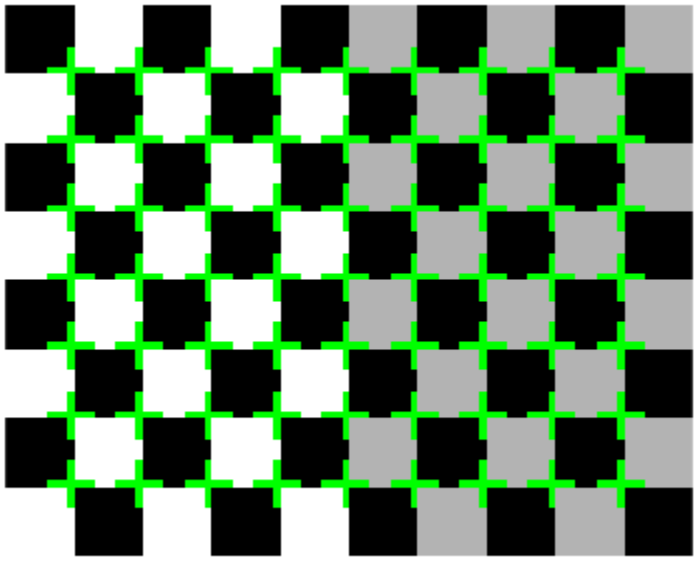
Generate and Plot Corners of an 8-by-10 Checkerboard

Generate the checkerboard and obtain world coordinates.

```
squareSize = 10;
patternDims = [8 10];
I = checkerboard(squareSize,patternDims(1)/2,patternDims(2)/2);
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Offset the points by placing the first point at the lower-right corner of the first square.

```
imshow(insertMarker(I,worldPoints + squareSize));
```



Generate Point Locations for AprilGrid

To generate the world point locations of keypoints in an AprilGrid, define the grid pattern dimensions (number of AprilTags along width and height), the size of each AprilTag, and the spacing between consecutive AprilTags. The tag size and spacing is specified in millimeters.

```
patternDims = [6,6];
tagSize = 25; % millimeters
tagSpacing = 10; % millimeters
worldPoints = patternWorldPoints("aprilgrid",patternDims,tagSize,tagSpacing);
```

Generate Point Locations for Asymmetric Circle Grid

Specify the center-to-center distance in millimeters (mm) between adjacent circles in the asymmetric circle grid pattern.

```
patternDims = [4,11];
centerDistance = 9; % millimeters
worldPoints = patternWorldPoints("circle-grid-asymmetric",patternDims,centerDistance);
```

Generate Point Locations for ChArUco board

Define the dimensions of the ChArUco board pattern and the size of each checker square to generate the world point locations of its keypoints. The checker square size is in millimeters.

```
patternDims = [7,10];
checkerSize = 27; % millimeters
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Generate Point Locations for Symmetric Circle Grid

Specify the center-to-center distance in millimeters (mm) between adjacent circles for the symmetric circle grid pattern.

```
patternDims = [8,11];
centerDistance = 19;  % millimeters
worldPoints = patternWorldPoints("circle-grid-symmetric",patternDims,centerDistance);
```

Input Arguments

`patternDims` — Pattern dimensions

two-element vector

Pattern dimensions, specified as a two-element vector. The first and second element represent the number of checker boxes or circles in the *x*- and *y*- dimensions, respectively. For a ChArUco board pattern, ensure that the board is oriented with the origin located at the top-left corner.

The pattern boards have these origin locations:

- Checkerboard — Lower-right corner of the top-left square of the board.
- Symmetric and asymmetric circle grids — Center of the top-left circle.
- ChArUco marker — Lower-right ChArUco corner, which corresponds to the minimum ArUco marker ID.
- AprilGrid pattern — Bottom-left corner, which corresponds to the minimum ID AprilTag.

For more details on calibration patterns, see “Calibration Patterns”.

`squareSize` — Side length of square

scalar

Side length of the square, specified as a scalar in world units such as millimeters or inches.

`centerDistance` — Center-to-center distance between two circles

scalar

Center-to-center distance between two circles, specified in world units such as millimeters or inches.

`checkerSize` — Side length of ChArUco checker board square

scalar

Side length of a ChArUco checker board square, specified in world units such as millimeters or inches.

`tagSize` — Size of AprilTag
scalar

Size of the AprilTag, specified as a scalar in world units, such as millimeters or inches.

`tagSpacing` — Space between two consecutive AprilTags
positive scalar

Space between two consecutive AprilTags, specified as a positive scalar.

`patternName` — Camera calibration pattern name
`"checkerboard"` | `"circle-grid-symmetric"` | `"circle-grid-asymmetric"` | `"charuco-board"` | `"aprilgrid"`

Camera calibration pattern name, specified as one of these options:

- `"checkerboard"`
- `"circle-grid-symmetric"`
- `"circle-grid-asymmetric"`
- `"charuco-board"`
- `"aprilgrid"`

Output Arguments

`worldPoints` — Keypoint locations
M-by-2 matrix

Keypoint locations, returned as an *M*-by-2 matrix in world coordinates. The matrix contains *M* points with coordinates in the form [*x y*]. The number of points the function returns is calculated based on the type of pattern:

- Checkerboard and ChArUco board — `(patternDims(1)-1)*(patternDims(2)-1)`
- Symmetric and asymmetric circle grid — `patternDims(1)*patternDims(2)`
- AprilGrid pattern — `patternDims(1)*patternDims(2)*4`

Version History
Introduced in R2024b

See Also

Functions
`detectCheckerboardPoints` | `detectCircleGridPoints` | `detectPatternPoints` | `generateCharucoBoard` | `detectCharucoBoardPoints` | `estimateCameraParameters` | `estimateMultiCameraParameters` | `detectAprilGridPoints`

Objects
`cameraParameters` | `stereoParameters`

Topics
“Calibration Patterns”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

detectCharucoBoardPoints

Detect ChArUco board pattern in images

Syntax

```
imagePoints = detectCharucoBoardPoints(I,patternDims,markerFamily,
checkerSize,markerSize)
imagePoints = detectCharucoBoardPoints( ___ ,intrinsics)
[imagePoints,imagesUsed] = detectCharucoBoardPoints(imageFileNames, ___ )
[ ___ ] = detectCharucoBoardPoints(images, ___ )

[imagePoints] = detectCharucoBoardPoints(I1,I2,patternDims,markerFamily,
checkerSize,markerSize)
[imagePoints] = detectCharucoBoardPoints(I1,I2,patternDims,markerFamily,
checkerSize,markerSize,intrinsics)
[imagePoints,pairsUsed] = detectCharucoBoardPoints(imageFileNames1,
imageFileNames2,patternDims,markerFamily,checkerSize,markerSize)
[ ___ ] = detectCharucoBoardPoints(I1,I2,intrinsics)

[ ___ ] = detectCharucoBoardPoints( ___ ,Name=Value)
```

Description

Single Image ChArUco Board Detection

`imagePoints = detectCharucoBoardPoints(I,patternDims,markerFamily, checkerSize,markerSize)` detects a ChArUco board in a 2-D truecolor or grayscale image. The function uses local homography to detect ChArUco corners.

`imagePoints = detectCharucoBoardPoints( ___ ,intrinsics)` detects ChArUco corners using approximated pose estimation. The approximation is based on the estimated extrinsic properties, given the camera `intrinsics`.

`[imagePoints,imagesUsed] = detectCharucoBoardPoints(imageFileNames, ___ )` detects a ChArUco board in the image files specified by `imageFileNames`, and additionally returns the list of images in which the board is detected `imagesUsed`.

`[ ___ ] = detectCharucoBoardPoints(images, ___ )` detects a ChArUco board in the specified images `images`.

Stereo Pair ChArUco Board Detection

`[imagePoints] = detectCharucoBoardPoints(I1,I2,patternDims,markerFamily, checkerSize,markerSize)` detects a ChArUco board in a set of 2-D truecolor or grayscale image.

`[imagePoints] = detectCharucoBoardPoints(I1,I2,patternDims,markerFamily, checkerSize,markerSize,intrinsics)` detects ChArUco corners in a set of 2-D truecolor or grayscale using approximated pose estimation. The approximation is based on the estimated extrinsic properties, given the camera `intrinsics`.

`[imagePoints,pairsUsed] = detectCharucoBoardPoints(imageFileNames1, imageFileNames2,patternDims,markerFamily,checkerSize,markerSize)` detects an

ChArUco pattern in the stereo pairs of image files specified by `imageFileNames1` and `imageFileNames2`. The function additionally returns a list of the pairs in which the board is detected `pairsUsed`.

`[ ___ ] = detectCharucoBoardPoints(I1,I2,intrinsics)` detects an ChArUco pattern in the stereo pairs of image files specified by `images1` and `images2`.

Optional Arguments

`[ ___ ] = detectCharucoBoardPoints( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `NumBorderBits=2` sets the number of border bits to detect a marker to 2.

Examples

Detect ChArUco Board in Images

Specify your calibration images.

```
imds = imageDatastore("charuco");
images = imds.Files;
```

Define the ChArUco board properties. Checker size and marker size are in millimeters.

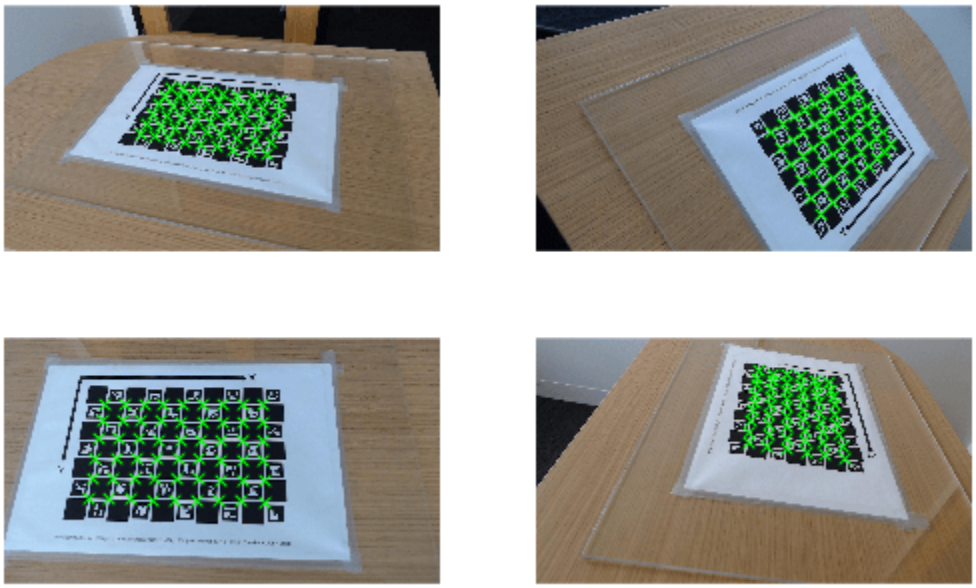
```
markerFamily = "DICT_4X4_1000";
patternDims = [7 10];
checkerSize = 29;
markerSize = 21;
```

Detect the board in the image.

```
imagePoints = detectCharucoBoardPoints(images,patternDims,markerFamily,checkerSize,markerSize);
```

Display the points from the first four images.

```
tiledlayout(2,2,TileSpacing="compact",Padding="compact");
for i = 1:4
  nexttile
  imshow(images{i})
  hold on
  plot(imagePoints(:,1,i,1),imagePoints(:,2,i,1),"gx");
end
```



Detect ChArUco Board in Stereo Image Pair

Load a stereo image pair containing a ChArUco board into the workspace.

```
I1 = imread("leftChArUco.jpg");
I2 = imread("rightChArUco.jpg");
```

Define the ChArUco board properties. The checker board and marker size are specified in millimeters.

```
markerFamily = "DICT_4X4_1000";
patternDims = [7 10];
checkerSize = 29;
markerSize = 21;
```

Detect the ChArUco board in the images.

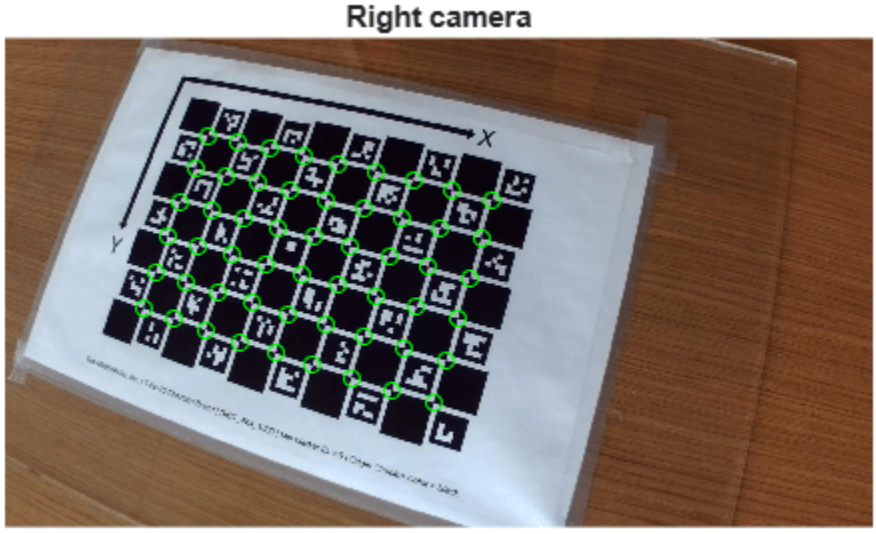
```
imagePoints = detectCharucoBoardPoints(I1,I2,patternDims,markerFamily,checkerSize,markerSize);
```

Display the detected points in the left and right images.

```
figure(1)
imshow(I1)
hold on
plot(imagePoints(:,1,1,1),imagePoints(:,2,1,1),"go");
title("Left camera")
```




```
figure(2)
imshow(I2)
hold on
plot(imagePoints(:,1,1,2),imagePoints(:,2,1,2),"go");
title("Right camera")
```

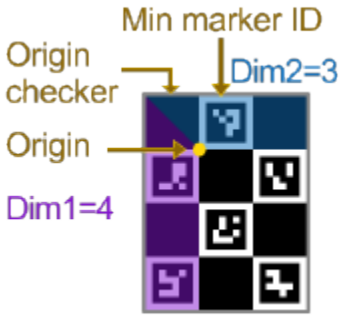


Input Arguments

`patternDims` — Pattern dimensions

two-element vector

Pattern dimensions, specified as a two-element vector that represents the number of checkers in the *dim1* and *dim2* dimensions of the ChArUco board, respectively.



For more details on ChArUco board patterns, see “Calibration Patterns”.

`markerFamily` — ArUco marker family used in ChArUco board

`"DICT_ARUCO_ORIGINAL"` | `"DICT_4X4_1000"` | `"DICT_5X5_1000"` | `"DICT_6X6_1000"` | `"DICT_7X7_1000"`

ArUco marker family used in ChArUco board, specified as `"DICT_ARUCO_ORIGINAL"`, `"DICT_4X4_1000"`, or `"DICT_5X5_1000"`, `"DICT_6X6_1000"`, or `"DICT_7X7_1000"`.

`checkerSize` — ChArUco board checker side length

scalar

ChArUco board checker side length, specified as a scalar in world units, such as millimeters or inches. The `checkerSize` and `markerSize` must be specified in the same world units and `checkerSize` must be a larger value than `markerSize`.

`markerSize` — ArUco marker side length

scalar

ArUco marker side length, specified as a scalar in world units, such as millimeters or inches. The `checkerSize` and `markerSize` must be specified in the same world units and `checkerSize` must be a larger value than `markerSize`.

`intrinsics` — Camera intrinsics

scalar `cameraIntrinsics` object | two-element vector

Camera intrinsics, specified as a scalar `cameraIntrinsics` object for a single camera, a two-element vector for a stereo camera, `cameraParameters` object, or a `stereoParameters` object.

To specify a `fisheyeIntrinsics` object, first undistort the image using the `undistortFisheyeImage` function. Use the undistorted image for the `detectChArUcoBoardPoints` image input and the `cameraIntrinsics` object that the `undistortFisheyeImage` function returns for the `intrinsics` argument.

`I` — Image
2-D truecolor image | grayscale image

Image, specified as a 2-D truecolor image or grayscale image.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images` — Images
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Images, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`I1` — Stereo pair images from camera one
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera one, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`I2` — Stereo pair images from camera two
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera two, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`imageFileNames` — Image file names
cell array of character vectors | array of strings

Image file names, specified as a cell array of character vectors or an array of strings.

`imageFileNames1` — Stereo image file names for camera one
cell array of character vectors | array of strings

Stereo image file names for camera one, specified as a cell array of character vectors or an array of strings.

`imageFileNames2` — Stereo image file names for camera two
cell array of character vectors | array of strings

Stereo image file names for camera two, specified as a cell array of character vectors or an array of strings.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectCharucoBoardPoints(___,NumBorderBits=2)` sets the number of marker border bits to detect a marker to 2.

Descriptive Properties

`MinMarkerID` — Lowest value marker ID in pattern

`0` (default) | non-negative integer

Lowest value marker ID in the pattern, specified as a non-negative integer. The markers in the pattern must correspond to sequential IDs, each incremented by `1`.

`OriginCheckerColor` — Color of the checker located in the top-left corner of board origin

`"black"` (default) | `"white"`

Color of the checker located in the top-left corner of ChArUco board origin, specified as `"black"` or `"white"`. The board origin is the keypoint closest to the ArUco marker with the lowest ID. This value must not be set to `"white"` when the board contains an odd number of boxes in the *x* dimension, specified in `patternDims(1)`.

Adaptive Thresholding Arguments

`WindowSizeRange` — Window size range

`[3 23]` (default) | two-element vector

Window size range, in pixels, specified as a two-element vector of the form `[`*`min max`*`]`. The elements set the minimum and maximum size, respectively, for the windows used for adaptive thresholding. Each value *n* in the range represents an *n*-by-*n* square window. Each element of this argument must be a positive integer greater than or equal to `3`. The function determines the local window threshold by calculating the average pixel value obtained from the sum of its surrounding pixels. Increase the *max* value when using ArUco markers that are larger than 400-by-400 pixels.

`WindowSizeStep` — Window step size

`10` (default) | positive integer

Window step size, in pixels, specified as a positive integer. The function uses the window step size for adaptive thresholding, but stays within the range set by the `WindowSizeRange` argument. The function starts at the minimum size of the range, and then increments the window size by the value of `WindowSizeStep`, until it reaches the maximum range value. If the window size is too small and a marker is large, the function might not detect that marker. This can occur when the window overlaps with the marker border. If the window size is large and a marker is small, the function might detect the marker, but the difference in size can negatively impact processing speed.

Contour Filtering Arguments

`MarkerSizeRange` — Range for marker boundary perimeter length

`[0.0075 1]` (default) | two-element vector

Range for marker boundary perimeter length, specified as a two-element vector of the form [*min max*]. The elements set the minimum and maximum size for the perimeter length of detected markers. Specify each element as a value normalized to the maximum dimension of the image. Reduce the range if a rejected marker completely encloses a valid marker.

`SquarenessTolerance` — Maximum polygonal approximation error

`0.03` (default) | positive scalar

Maximum polygonal approximation error, specified as a positive scalar. The function utilizes the Douglas-Peucker algorithm to apply a polygonal approximation to each marker candidate, accepting only those that closely resemble a square shape. The value of the `SquarenessTolerance` is relative to the candidate marker length (in pixels). Decrease this value to filter false corner detections in occluded markers. Increase this value to detect marker candidates in highly distorted images.

Bit Extraction Arguments

`NumBorderBits` — Number of border bits in marker
`1` (default) | positive integer

Number of border bits in the marker, specified as a positive integer. By default, the marker boundary contains one bit. If the function does not detect a marker candidate, first ensure that you have correctly set the `markerFamily`, then check the ChArUco manufacturer manual to get the correct value for the number of border bits to use.

`ResolutionPerBit` — Resolution of the decoder
`4` (default) | positive integer

Resolution of the decoder to use in the bit extraction step, in pixels, specified as a positive integer. This value sets the number of pixels, per bit, to use in the obtained image after correcting perspective distortion. Increase this value to improve the accuracy with which the function reads marker IDs at the expense of runtime speed.

Subpixel Corner Refinement Arguments

`RefineCorners` — Subpixel refinement process
`false` or `0` (default) | `true` or `1`

Subpixel refinement process, specified as a logical `1`(`true`) or `0`(`false`). Set this value to `true` to enable the subpixel refinement process for the detected marker corner locations. This step is computationally expensive. Enable this process only if the corner locations must be accurate for pose estimation or for camera calibration applications.

`RefinementWindowSize` — Window size of subpixel refinement process
`5` (default) | positive integer

Window size of the subpixel refinement process, in pixels, specified as a positive integer. Increasing this value can cause the window area to include a corner of the image. As a result of this proximity, marker corners might move to an incorrect locations.

`RefinementMaxIterations` — Maximum number of iterations for subpixel refinement process
`30` (default) | positive integer

Maximum number of iterations the subpixel refinement process, specified as a positive integer. The process stops after the specified number of iterations even if it has not achieved the termination tolerance `RefinementTolerance`.

`RefinementTolerance` — Termination tolerance
`0.1` (default) | positive scalar

Termination tolerance, specified as a positive scalar.

Output Arguments

`imagePoints` — Corner coordinates of detected ChArUco board

M-by-2 matrix | *M*-by-2-by- *numImages* | *M*-by-2-by- *numPairs*-by-2 array

Corner coordinates of detected ChArUco board, returned as an *M*-by-2 matrix, *M*-by-2-by- *numImages* array, or an *M*-by-2-by-*numPairs*-by-2 array. *M* is the number of ChArUco boards detected. Each row of the matrix or array specifies the *x*-*y* coordinates of the corner of a ChArUco pattern.

- For single images — This argument is an *M*-by-2 matrix or an *M*-by-2-by- *numImages* array, when detecting ChArUco board pattern in single images. *numImages* is the number of images.
- For stereo pairs — This argument is an *M*-by-2-by-*numPairs*-by-2 array when detecting ChArUco board patterns in stereo pairs of images. *numPairs* is the number of image pairs in which an ChArUco board patterns is detected. `imagePoints(:,:,:,1)` returns the points from the first set of images, and `imagePoints(:,:,:,2)` are the points from the second set.

When no ArUco marker was detected in the board image, the function returns `imagePoints` with `NaN` for its surrounding ChArUco corner locations.

`imagesUsed` — ChArco board pattern detection flag

logical vector

ChArUco board pattern detection flag, returned as a logical vector. A value of `true` in the `imagesUsed` vector indicates that the pattern has been detected in the corresponding image.

`pairsUsed` — Stereo pair ChArUco board pattern detection flag

logical vector

Stereo pair ChArUco board pattern detection flag, returned as a logical vector. A value of `true` in the `pairsUsed` vector indicates that the ChArUco board pattern has been detected in the corresponding image pair.

Tips

- When no ArUco marker was detected in the board image, the function returns `imagePoints` with `NaN` for its surrounding ChArUco corner locations.
- To debug any missed detections of ArUco markers, you can use the `readArucoMarker` function.
- To specify a `fisheyeIntrinsics` object, first undistort the image using the `undistortFisheyeImage` function. Use the undistorted image for the `detectChArUcoBoardPoints` image input and the `cameraIntrinsics` object that the `undistortFisheyeImage` function returns for the `intrinsics` argument.
- To detect multiple ChArUco boards in the same image, use this function with a unique for `MinMarkerID`, for each board.
- To use this function, the markers in the pattern must correspond to sequential IDs, each incremented by `1`.

Version History

Introduced in R2024b

R2025a: Generate C/C++ code using MATLAB Coder

The `detectCharucoBoardPoints` function now supports the generation of C/C++ code (requires MATLAB Coder).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates code only on host platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectCharucoBoardPoints` function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`generateCharucoBoard` | `estimateCameraParameters` | `detectCheckerboardPoints` | `patternWorldPoints`

Objects

`cameraParameters` | `stereoParameters`

Topics

“Calibration Patterns”

detectAprilGridPoints

Detect keypoints of AprilGrid pattern in images

Syntax

```
imagePoints = detectAprilGridPoints(I,patternDims,tagFamily)
[imagePoints,imagesUsed] = detectAprilGridPoints(imageFileNames, ___ )
[ ___ ] = detectAprilGridPoints(images, ___ )

imagePoints = detectAprilGridPoints(images1,images2,patternDims,tagFamily)
[imagePoints,pairsUsed] = detectAprilGridPoints(imageFileNames1,
imageFileNames2, ___ )
[ ___ ] = detectAprilGridPoints(images1,images2, ___ )

[ ___ ] = detectAprilGridPoints( ___ ,Name=Value)
```

Description

Single Image AprilGrid Detection

`imagePoints = detectAprilGridPoints(I,patternDims,tagFamily)` detects keypoints of an AprilGrid pattern in a 2-D truecolor or grayscale image `I`. `patternDims` specifies the number of tags in the pattern, and `tagFamily` specifies the AprilTag family.

`[imagePoints,imagesUsed] = detectAprilGridPoints(imageFileNames, ___ )` detects an AprilGrid pattern in the image files specified by name in `imageFileNames`, and additionally returns the list of images in which the pattern is detected `imagesUsed`.

`[ ___ ] = detectAprilGridPoints(images, ___ )` detects an AprilGrid in the specified images `images`.

Stereo Pair AprilGrid Detection

`imagePoints = detectAprilGridPoints(images1,images2,patternDims,tagFamily)` detects keypoints of an AprilGrid pattern in a 2-D truecolor or grayscale images `images1` and `images2`. `patternDims` specifies the number of tags in the pattern, and `tagFamily` specifies the AprilTag family.

`[imagePoints,pairsUsed] = detectAprilGridPoints(imageFileNames1, imageFileNames2, ___ )` detects an AprilGrid pattern in the stereo pairs of image files specified by the image file names in `imageFileNames1` and `imageFileNames2`. The function additionally returns a list of the pairs in which the pattern is detected `pairsUsed`.

`[ ___ ] = detectAprilGridPoints(images1,images2, ___ )` detects an AprilGrid pattern in the stereo pairs of images specified by `images1` and `images2`.

Optional Arguments

`[ ___ ] = detectAprilGridPoints( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `NumBorderBits=1` sets the number of tag border bits to generate an AprilTag to `1`.

Examples

Detect AprilGrid in Image

Load the calibration image containing an AprilGrid into the workspace.

```
I = imread("aprilgrid.jpg");
```

Define the AprilGrid properties.

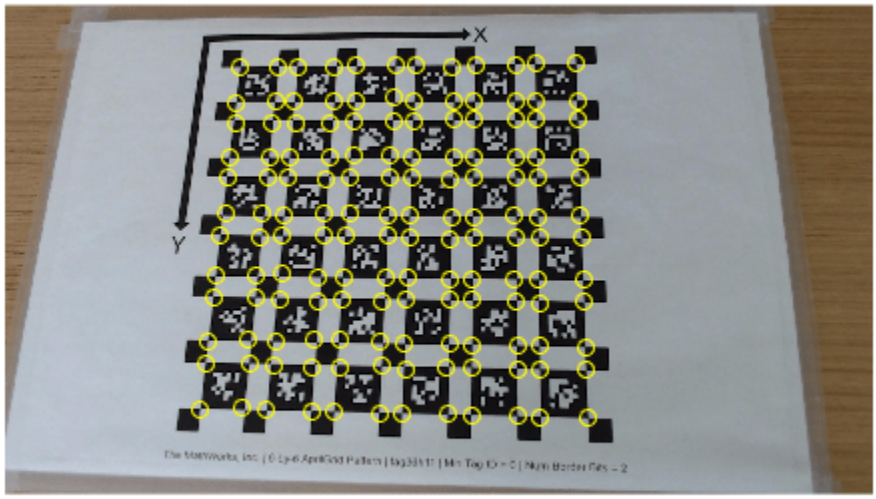
```
tagFamily = "tag36h11";
patternDims = [6 6];
```

Detect AprilGrid in the image.

```
imagePoints = detectAprilGridPoints(I,patternDims,tagFamily);
```

Display the detected keypoints of AprilGrid in the image.

```
imshow(I);
hold on;
plot(imagePoints(:,1),imagePoints(:,2),"yo");
```



Detect AprilGrid in Stereo Image Pair

Load a stereo image pair containing an AprilGrid pattern into the workspace.

```
I1 = imread("leftAprilGrid.jpg");
I2 = imread("rightAprilGrid.jpg");
```

Define the AprilGrid properties.

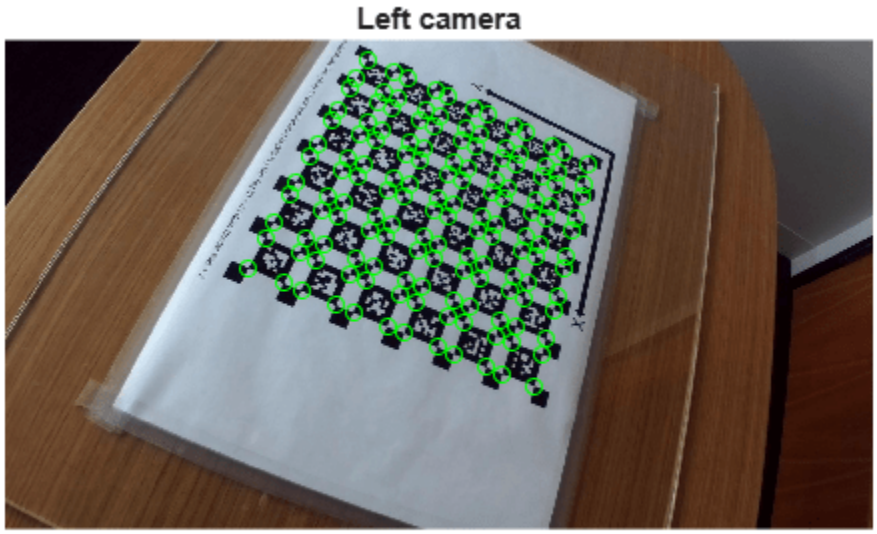
```
tagFamily = "tag36h11";
patternDims = [6 6];
```

Detect the AprilGrid keypoints.

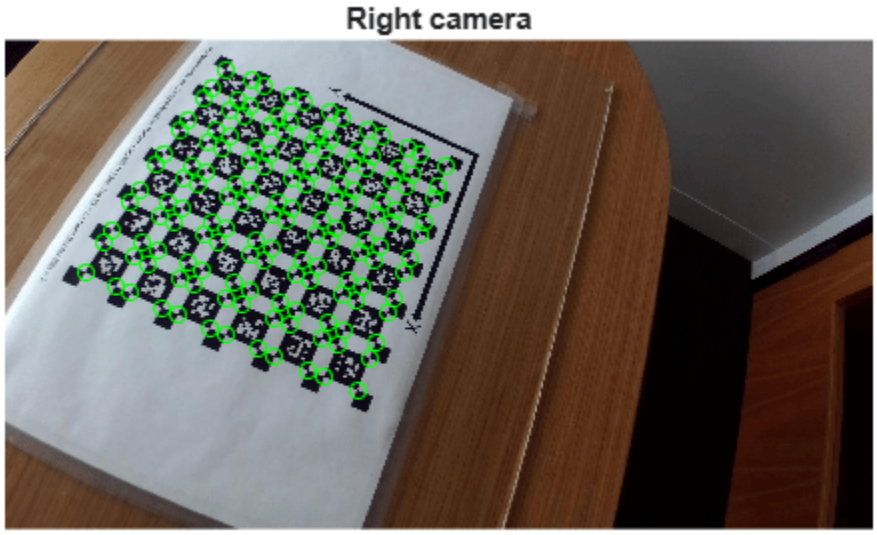
```
imagePoints = detectAprilGridPoints(I1,I2,patternDims,tagFamily);
```

Display the detected points on the image.

```
figure(1)
imshow(I1)
hold on
plot(imagePoints(:,1,1,1),imagePoints(:,2,1,1),"go");
title("Left camera")
```




```
figure(2)
imshow(I2)
hold on
plot(imagePoints(:,1,1,2),imagePoints(:,2,1,2),"go");
title("Right camera")
```

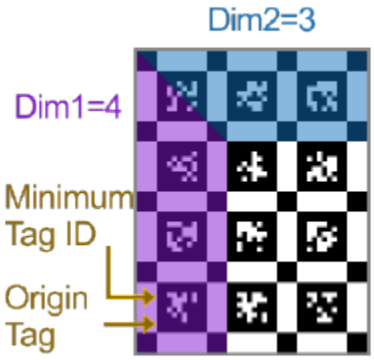


Input Arguments

`patternDims` — Pattern dimensions

two-element vector

Pattern dimensions, specified as a two-element vector that represents the number of enclosed squares in the vertical and horizontal dimensions of the AprilGrid image, respectively.



For more details on AprilTag patterns, see “Calibration Patterns”.

`tagFamily` — AprilTag marker family used in AprilGrid pattern

`"tag36h11"` | `"tag25h9"` | `"tag16h5"`

AprilTag marker family used in the AprilGrid pattern, specified as `"tag36h11"`, `"tag25h9"`, or `"tag16h5"`.

`I` — Image
2-D truecolor image | grayscale image

Image, specified as a 2-D truecolor image or grayscale image.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images` — Images
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Images, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images1` — Stereo pair images from camera one
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera one, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images2` — Stereo pair images from camera two
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera two, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`imageFileNames` — Image filenames
cell array of character vectors | array of strings

Image filenames, specified as a cell array of character vectors or an array of strings.

`imageFileNames1` — Stereo image filenames for camera one
cell array of character vectors | array of strings

Stereo image filenames for camera one, specified as a cell array of character vectors or an array of strings.

`imageFileNames2` — Stereo image filenames for camera two
cell array of character vectors | array of strings

Stereo image filenames for camera two, specified as a cell array of character vectors or an array of strings.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectAprilGridPoints(I,patternDims,NumBorderBits=1)` sets the number of tag border bits to generate an AprilTag to `1`.

`NumBorderBits` — Number of tag border bits
`2` (default) | `1`

Number of tag border bits, specified as `1` or `2`. The tag border bits are used to generate the AprilTags. The border bits relate to each generated tag image, but not to a tag family. Refer to the manufacturer manual for your AprilTag pattern, to set this value.

If the function does not detect a fully visible tag, check that the `tagFamily` is correct. Otherwise, the specified `NumborderBits` might be incorrect.

`MinTagID` — Lowest value tag ID in pattern
`0` (default) | nonnegative integer

Lowest value tag ID in the AprilGrid pattern, specified as a nonnegative integer. The tags in the pattern must have sequential IDs with an increment of `1`.

`GaussianSigma` — Standard deviation for 2-D Gaussian smoothing kernel
`0` (default) | nonnegative scalar

Standard deviation for 2-D Gaussian smoothing kernel, specified as a nonnegative scalar. The Gaussian kernel smooths the input image to decrease white Gaussian noise. Increase this value to increase the processing speed, but at the cost of reduced precision.

`DecimationFactor` — Decimation factor
`1` (default) | positive integer

Decimation factor, specified as a positive integer. The decimation factor controls the scale of the downsampling. Increase this value to increase the detection rate, while decreasing detection distance.

Output Arguments

`imagePoints` — Corner coordinates of detected AprilGrid pattern
M-by-2 matrix | *M*-by-2-by- *numImages* array | *M*-by-2-by-*numPairs*-by-2 array

Corner coordinates of the detected AprilGrid pattern, returned as an *M*-by-2 matrix, *M*-by-2-by-*numImages* array, or an *M*-by-2-by-*numPairs*-by-2 array. *M* is the number of AprilGrid patterns detected, calculated as the product of the number of Aprilgrid patterns detected in each dimension. Each row of the matrix or array specifies the *xy*- coordinates of the center an AprilGrid.

- For single image — This argument is an *M*-by-2 matrix.
- For mutliple images — This argument is an *M*-by-2-*numImages* array.
- For stereo pairs — This argument is an *M*-by-2-by-*numPairs*-by-2 array, where *numPairs* is the number of image pairs in which the function detects an AprilGrid. `imagePoints(:,:,:,1)` returns the points from the first set of images, and `imagePoints(:,:,:,2)` returns the points from the second set.

`imagesUsed` — Pattern detection flag
logical vector

Pattern detection flag, returned as a logical vector. A value of `true` in the `imagesUsed` vector indicates that the pattern has been detected in the corresponding image.

`pairsUsed` — Stereo pair pattern detection flag
logical vector

Stereo pair pattern detection flag, returned as a logical vector. A value of `true` in the `pairsUsed` vector indicates that the pattern has been detected in the corresponding image pair.

Version History

Introduced in R2024b

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectAprilGridPoints` function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`estimateCameraParameters` | `detectCheckerboardPoints` | `readAprilTag` | `patternWorldPoints`

Objects

`cameraParameters` | `stereoParameters`

Topics

“Calibration Patterns”

readArucoMarker

Detect and estimate pose for ArUco marker in image

Syntax

```
[id,loc] = readArucoMarker(I)
[id,loc] = readArucoMarker(I,markerFamily)
[id,loc,pose] = readArucoMarker( ___ ,intrinsics,markerSize)
[ ___ ,detectedFamily,rejections] = readArucoMarker( ___ )
[ ___ ] = readArucoMarker( ___ ,Name=Value)
```

Description

`[id,loc] = readArucoMarker(I)` detects ArUco markers in the input image `I`, and returns the locations and IDs associated with the markers.

`[id,loc] = readArucoMarker(I,markerFamily)` detects ArUco markers of only the specified families, `markerFamily`.

`[id,loc,pose] = readArucoMarker( ___ ,intrinsics,markerSize)` returns the poses of the markers with respect to the specified camera intrinsic parameters, `intrinsics`, and size of the markers, `markerSize`, in addition to any combination of input arguments from previous syntaxes.

`[ ___ ,detectedFamily,rejections] = readArucoMarker( ___ )` returns the recognized ArUco marker families, `detectedFamily`, and the *xy*-locations of rejected ArUco marker candidates, `rejections`, in addition to any combination of arguments from previous syntaxes.

`[ ___ ] = readArucoMarker( ___ ,Name=Value)` specifies options using one or more name-value arguments. For example, `WindowSizeStep=12` sets the number of pixels by which to increment the window size during adaptive thresholding to 12.

Examples

Read ArUco Markers in Image

Read an image into the workspace.

```
I = imread("ArUcoMarkersMulti.jpg");
```

Read the ArUco markers in the image to get their IDs, pixel locations, and marker families.

```
[ids,locs,detectedFamily] = readArucoMarker(I);
```

Overlay the marker edges, corners, and IDs on the image for visualization.

```
numMarkers = length(ids);
for i = 1:numMarkers
  loc = locs(:,:,i);

  % Display the marker ID and family
```

```
  disp("Detected marker ID, Family: " + ids(i) + ", " + detectedFamily(i))

  % Insert marker edges
  I = insertShape(I,"polygon",{loc},Opacity=1,ShapeColor="green",LineWidth=4);

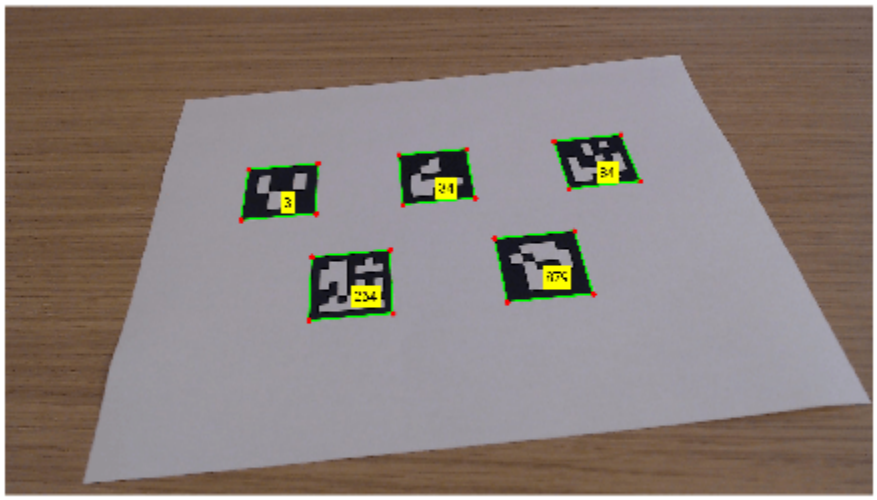
  % Insert marker corners
  markerRadius = 6;
  numCorners = size(loc,1);
  markerPosition = [loc,repmat(markerRadius,numCorners,1)];
  I = insertShape(I,"FilledCircle",markerPosition,ShapeColor="red",Opacity=1);

  % Insert marker IDs
  center = mean(loc);
  I = insertText(I,center,ids(i),FontSize=30,BoxOpacity=1);
end
```

```
Detected marker ID, Family: 3, DICT_4X4_1000
Detected marker ID, Family: 34, DICT_5X5_1000
Detected marker ID, Family: 84, DICT_6X6_1000
Detected marker ID, Family: 264, DICT_7X7_1000
Detected marker ID, Family: 879, DICT_ARUCO_ORIGINAL
```

Display the results.

```
imshow(I)
```



Estimate ArUco Poses in Image

Read an image that contains ArUco markers into the workspace.

```
I = imread("ArUcoBoard.jpg");
```

Load intrinsics of the camera that captured the image.

```
data = load("camIntrinsicsArucoBoard.mat");
intrinsics = data.intrinsics;
```

Undistort the image and display it.

```
[I,camIntrinsics] = undistortFisheyeImage(I,intrinsics);
imshow(I)
```



Measure the marker size in millimeters.

```
markerSize = 25.4; % millimeters
```

Specify the marker family for faster performance of `readArucoMarker` function.

```
markerFamily = "DICT_6X6_250";
```

Estimate marker poses.

```
[ids,locs,poses] = readArucoMarker(I,markerFamily,camIntrinsics,markerSize);
```

Overlay estimated poses on the image for visualization.

```
% Origin and axes vectors for the object coordinate system
worldPoints = [0 0 0; markerSize/2 0 0; 0 markerSize/2 0; 0 0 markerSize/2];

for i = 1:length(poses)
  % Get image coordinates for axes
  imagePoints = world2img(worldPoints,poses(i),camIntrinsics);

  axesPoints = [imagePoints(1,:) imagePoints(2,:);
                imagePoints(1,:) imagePoints(3,:);
                imagePoints(1,:) imagePoints(4,:)];
```

```
  % Draw colored axes
  I = insertShape(I, "Line", axesPoints, ...
      Color = ["red","green","blue"], LineWidth=10);
end
```

Display the results.

```
imshow(I)
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* grayscale image. The input image must be undistorted.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`markerFamily` — ArUco marker families

character vector | string scalar | vector of strings | cell array of character vectors

ArUco marker families, specified as a character vector, string scalar, vector of strings, or cell array of character vectors. Each element must be one of the valid ArUco marker families listed in the table. Specifying a family can reduce the run time of the function by restricting its search. Use this argument to improve the execution time when you know the marker family in the image. Be default, the function searches for:

`[ "DICT_ARUCO_ORIGINAL","DICT_4X4_1000","DICT_5X5_1000","DICT_6X6_1000", "DICT_7X7_1000"]`.

Family	Marker Size (bits)	Number of Markers in Family	Minimum Hamming Distance Between Markers (bits)
"DICT_ARUCO_ORIGINAL"	5-by-5	1024	3
"DICT_4X4_50"	4-by-4	50	4
"DICT_4X4_100"		100	3
"DICT_4X4_250"		250	2
"DICT_4X4_1000"		1000	2
"DICT_5X5_50"	5-by-5	50	8
"DICT_5X5_100"		100	7
"DICT_5X5_250"		250	6
"DICT_5X5_1000"		1000	5
"DICT_6X6_50"	6-by-6	50	13
"DICT_6X6_100"		100	12
"DICT_6X6_250"		250	11
"DICT_6X6_1000"		1000	9
"DICT_7X7_50"	7-by-7	50	19
"DICT_7X7_100"		250	18
"DICT_7X7_250"		250	17
"DICT_7X7_1000"		1000	14

Data Types: `char` | `string` | `cell`

`intrinsics` — Camera intrinsic parameters
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

`markerSize` — Size of marker
positive scalar

Size of the marker in world units (such as millimeters), specified as a positive scalar. The function defines the size of the markers as the length of one side of the square marker.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `readArucoMarker(I,WindowSizeStep=12)` sets the number of pixels by which to increment the window size during adaptive thresholding to `12`.

Adaptive Thresholding Arguments

`WindowSizeRange` — Window size range

`[3 23]` (default) | two-element vector

Window size range, in pixels, specified as a two-element vector of the form `[min max]`. The elements set the minimum and maximum size, respectively, for the windows used for adaptive thresholding. Each value *n* in the range represents an *n*-by-*n* square window. Each element of this argument must be a positive integer greater than or equal to `3`. The function determines the local window threshold by calculating the average pixel value obtained from the sum of its surrounding pixels. Increase the *max* value when using ArUco markers that are larger than 400-by-400 pixels.

`WindowSizeStep` — Window step size

`10` (default) | positive integer

Window step size, in pixels, specified as a positive integer. The function uses the window step size for adaptive thresholding, but stays within the range set by the `WindowSizeRange` argument. The function starts at the minimum size of the range, and then increments the window size by the value of `WindowSizeStep`, until it reaches the maximum range value. If the window size is too small and a marker is large, the function might not detect that marker. This can occur when the window overlaps with the marker border. If the window size is large and a marker is small, the function might detect the marker, but the difference in size can negatively impact processing speed.

Contour Filtering Arguments

`MarkerSizeRange` — Range for marker boundary perimeter length

`[0.0075 1]` (default) | two-element vector

Range for marker boundary perimeter length, specified as a two-element vector of the form [*min max*]. The elements set the minimum and maximum size for the perimeter length of detected markers. Specify each element as a value normalized to the maximum dimension of the image. Reduce the range if a rejected marker completely encloses a valid marker.

`SquarenessTolerance` — Maximum polygonal approximation error

`0.03` (default) | positive scalar

Maximum polygonal approximation error, specified as a positive scalar. The function utilizes the Douglas-Peucker algorithm to apply a polygonal approximation to each marker candidate, accepting only those that closely resemble a square shape. The value of the `SquarenessTolerance` is relative to the candidate marker length (in pixels). Decrease this value to filter false corner detections in occluded markers. Increase this value to detect marker candidates in highly distorted images.

Bit Extraction Arguments

`NumBorderBits` — Number of border bits in marker

`1` (default) | positive integer

Number of border bits in the marker, specified as a positive integer. By default, the marker boundary contains one bit. If the function does not detect a marker candidate, first ensure that you have correctly set the `markerFamily`, then follow one of these two steps:

- If you used the `generateArucoMarker` function to generate the marker image, then ensure this name-value argument value is set to the same value as `NumBorderBits` of the `generateArucoMarker` function.

- If you did not use the `generateArucoMarker` function, then check the ArUco marker manufacturer manual to get the correct value for the number of border bits to use.

`ResolutionPerBit` — Resolution of the decoder
`4` (default) | positive integer

Resolution of the decoder to use in the bit extraction step, in pixels, specified as a positive integer. This value sets the number of pixels, per bit, to use in the obtained image after correcting perspective distortion. Increase this value to improve the accuracy with which the function reads marker IDs at the expense of runtime speed. The maximum value for argument is `1000`.

Subpixel Corner Refinement Arguments

`RefineCorners` — Subpixel refinement process
`false` or `0` (default) | `true` or `1`

Subpixel refinement process, specified as a logical `1`(`true`) or `0`(`false`). Set this value to `true` to enable the subpixel refinement process for the detected marker corner locations. This step is computationally expensive. Enable this process only if the corner locations must be accurate for pose estimation or for camera calibration applications.

`RefinementWindowSize` — Window size of subpixel refinement process
`5` (default) | positive integer

Window size of the subpixel refinement process, in pixels, specified as a positive integer. Increasing this value can cause the window area to include a corner of the image. As a result of this proximity, marker corners might move to an incorrect locations.

`RefinementMaxIterations` — Maximum number of iterations for subpixel refinement process
`30` (default) | positive integer

Maximum number of iterations the subpixel refinement process, specified as a positive integer. The process stops after the specified number of iterations even if it has not achieved the termination tolerance `RefinementTolerance`.

`RefinementTolerance` — Termination tolerance
`0.1` (default) | positive scalar

Termination tolerance, specified as a positive scalar.

Output Arguments

`id` — Marker IDs
N-element vector

Marker IDs, returned as an *N*-element vector of positive integers.

`loc` — Locations of detected markers in image
4-by-2-by-*N* array

Locations of the detected markers in the image, returned as a 4-by-2-by-*N* array. The array contains the *xy*-locations of the four corners for each detected marker. *N* is the number of detected markers.

`pose` — Poses of markers
array of `rigidtform3d` objects

Poses of markers with respect to the camera, returned as an array of `rigidtform3d` objects. Each object encapsulates the 3-D rigid transformation of a marker in the same world units as the `markerSize` input argument. The origin of each marker frame is located at the center of the corresponding marker. The planar surface on which the ArUco marker sits defines the *XY*-plane of the marker frame with the *Z*-axis pointing out of the marker.

`detectedFamily` — Detected ArUco marker families
vector of strings

Detected ArUco marker families, returned as a vector of strings.

`rejections` — Locations of rejected ArUco marker candidates
4-by-2-by-*P* array

Locations of the rejected ArUco marker candidates, returned as a 4-by-2-by-*P* array. The array contains the *xy*-locations for the four corners of each rejected marker candidate. *P* is the number of rejected marker candidates.

Tips

- This function does not account for lens distortion during pose estimation. To estimate poses, the input image must be undistorted. To estimate poses with a camera that has been calibrated using the Scaramuzza model, you can use the virtual pinhole model returned by the `undistortFisheyeImage` function.
- Consider the number of markers your application requires, marker length, and Hamming distance when choosing a marker family. The minimum Hamming distance between markers in the same family determines how capable the function is of detecting them and correcting errors. Smaller marker families, in terms of the number of markers in the family, and marker families with more bits have a larger minimum Hamming distance, but the additional bits to extract can make marker detection more difficult. For example, if your application only requires 10 markers, you can use a family of 10 markers for increased error robustness over a family of 1000 markers. In summary:
  - The fewer bits used to encode the ID in the ArUco marker, the greater the distance from the camera that you can detect the marker. The marker family name specifies the number of bits (for example, `DICT_4X4_50` has a marker size of 4-by-4, totaling 16 bits).
  - Refer to the full list of ArUco family markers, specified in the `markerFamily` argument description.
- Use one or more of these debugging tips:
  - Include a white border (also called a quiet zone) around ArUco markers for proper identification and decoding.
  - The function does not detect markers if the images have been flipped vertically or horizontally.
  - The function does not detect markers if their square shape has been altered as a result of lens distortion. You can undistort images to improve marker detection.
  - If the function does not detect a marker image, and does not return it as a rejected marker candidate, you might need to modify the adaptive thresholding steps. You can do this by adjusting the `WindowSizeRange` and `WindowSizeStep` values.

- If the function does not detect a marker image, but returns it as a rejected candidate, verify or adjust the `markerFamily` and `NumBorderBits` values. Also, verify that the image has not been flipped vertically or horizontally.
- If a detected or rejected marker incorrectly encompasses any surrounding black borders, adjust the `MarkerSizeRange` value.
- If the function returns an incorrect marker ID, or if it detects the same marker using two different marker families, increase the `ResolutionPerBit` value.

Algorithms

When the detection process starts, the function binarizes the input image using adaptive thresholding at different neighborhood window sizes. The function extracts marker rejections on each of the binarized images.

The marker detection algorithm uses these steps:

1. Perform adaptive thresholding to binarize the image.
2. Extract and filter contours to detect square-shaped marker candidates.
3. Remove perspective distortion and extract marker bits to analyze the inner codification of the marker bits.
4. Perform error correction to identify valid markers.
5. Optionally, perform subpixel refinement of corner positions.

Version History

Introduced in R2024a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV (Version 4.7.0) libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- During code generation, `detectedFamily` output argument is returned as cell array unlike vector of strings in simulation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

During code generation, `detectedFamily` output argument is returned as cell array unlike vector of strings in simulation.

See Also

Functions

`generateArucoMarker` | `undistortImage`

Topics
“Calibration Patterns”

generateArucoMarker

Generate ArUco marker images

Syntax

```
imgs = generateArucoMarker(markerFamily,ids,markerSize)
imgs = generateArucoMarker( ___ ,NumBorderBits=numBits)
```

Description

`imgs = generateArucoMarker(markerFamily,ids,markerSize)` generates ArUco marker images `imgs` of the specified family `markerFamily` and size `markerSize` with the corresponding identifiers `ids`.

`imgs = generateArucoMarker( ___ ,NumBorderBits=numBits)` sets the number of marker border bits to add when generating the marker, in addition to all input arguments from the previous syntax. Specify `numBits` as a positive integer value. The default value is set to `1`.

Examples

Generate ArUco Marker Images

Generate 12 ArUco marker images, display them, and write them to PNG files.

Specify the number, size, family, and IDs of the markers to generate.

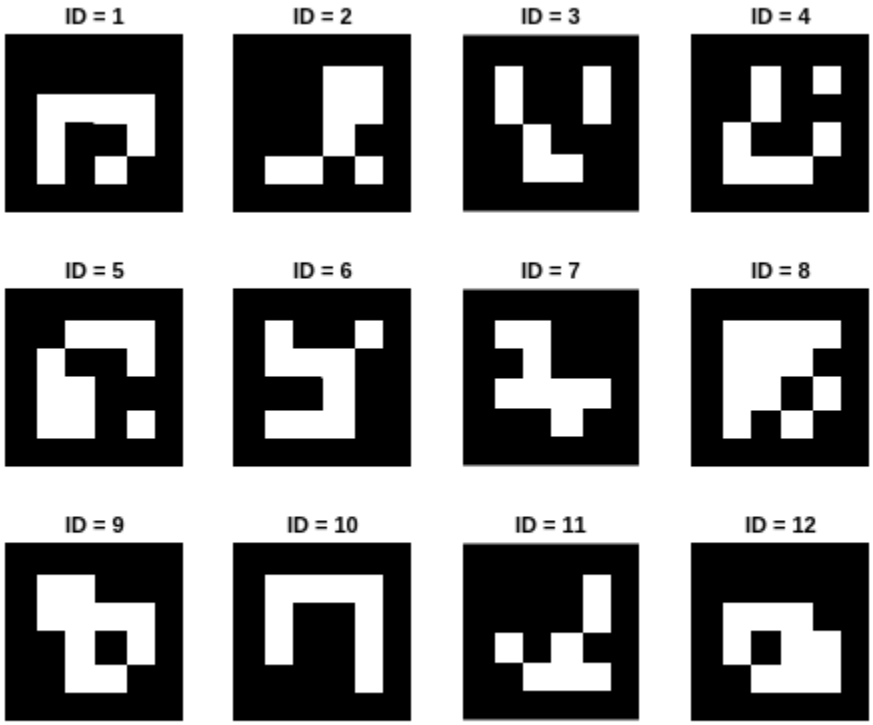
```
numMarkers = 12;
markerSize = 50; % in pixels
markerFamily = "DICT_4X4_250";
ids = 1:numMarkers;
```

Generate the ArUco markers.

```
imgs = generateArucoMarker(markerFamily,ids,markerSize);
```

Display the markers.

```
tiledlayout(3,4,TileSpacing="compact")
for i = 1:numMarkers
    nexttile
    imshow(imgs(:,:,i))
    title("ID = " + ids(i))
end
```



Write each marker image to a PNG file.

```
for i = 1:numMarkers
    filename = markerFamily + "_ID" + ids(i) + ".png";
    imwrite(imgs(:,:,i), filename)
end
```

Input Arguments

`markerFamily` — ArUco marker family name

character vector | string scalar | vector of strings | cell array of character vectors

ArUco marker families, specified as a character vector, string scalar, vector of strings, or cell array of character vectors. Each element must be one of the valid ArUco marker families listed in the table. Select a marker family suitable for your application based on the total number of markers needed, the marker length, and the Hamming distance between the marker code words. The minimum Hamming distance between markers in the same family, determine the ability to detect them and correct errors. Marker families with less number of markers can be more robust against errors than a family with a large number of markers, which can increase marker detection difficulty due to a greater number of bits to extract. These general rules apply:

- The less number of bits to detect, the longer the range in distance a marker can be detected.
- For correct identification and decoding, ArUco markers require a white border, also known as the quiet zone.

Family	Marker Size (bits)	Number of Markers in Family	Minimum Hamming Distance Between Markers (bits)
"DICT_ARUCO_ORIGINAL"	5-by-5	1024	3
"DICT_4X4_50"	4-by-4	50	4
"DICT_4X4_100"		100	3
"DICT_4X4_250"		250	2
"DICT_4X4_1000"		1000	2
"DICT_5X5_50"	5-by-5	50	8
"DICT_5X5_100"		100	7
"DICT_5X5_250"		250	6
"DICT_5X5_1000"		1000	5
"DICT_6X6_50"	6-by-6	50	13
"DICT_6X6_100"		100	12
"DICT_6X6_250"		250	11
"DICT_6X6_1000"		1000	9
"DICT_7X7_50"	7-by-7	50	19
"DICT_7X7_100"		250	18
"DICT_7X7_250"		250	17
"DICT_7X7_1000"		1000	14

Data Types: `char` | `string` | `cell`

`ids` — Marker IDs
N-element vector

Marker IDs, specified as an *N*-element vector of positive integers. Each element must be nonnegative and less than the family size, which is the total number of markers available in the marker family. For example, the family size of `"DICT_7X7_250"` is 250.

`markerSize` — Size of each marker
positive integer

Size of each marker, in pixels, specified as a positive integer. This value sets the length of one side of a square ArUco marker. The marker size must be large enough to store the number of bits required for the specified marker family. For example, you cannot generate an image of 5-by-5 pixels for a marker from the `"DICT_6X6_100"` family, which requires a 6-by-6 pixel size. For more information, see the table in the `markerFamily` input description.

Output Arguments

`imgs` — Marker images
M-by-*M*-by-*N* array

Marker images, returned as an *M*-by-*M*-by-*N* array.

Data Types: `uint8`

Version History

Introduced in R2024a

See Also

Functions

`readArucoMarker`

groundTruthToOpenLabel

Convert ground truth object to OpenLabel JSON file

Syntax

```
groundTruthToOpenLabel(gTruth,jsonFilename)
groundTruthToOpenLabel( ___ ,PrettyPrint=false)
```

Description

`groundTruthToOpenLabel(gTruth,jsonFilename)` converts ground truth data `gTruth`, to the ASAM OpenLABEL® format and writes the result to a JavaScript Object Notation (JSON), `jsonFilename`.

`groundTruthToOpenLabel( ___ ,PrettyPrint=false)` adds indentations to the output file. A value of `true` adds the indentation and a value of `false` will not add it. The default value is set to `false`, which produces a smaller size file.

Examples

Convert Ground Truth Object to ASAM OpenLABEL JSON File

Add the folder that contains images to the MATLAB® path.

```
imageDir = fullfile(matlabroot,"toolbox","vision","visiondata","vehicles");
addpath(imageDir);
```

Load ground truth object data related to vehicles from a previously saved MAT file.

```
data = load("vehicleTrainingGroundTruth.mat");
gTruth = data.vehicleTrainingGroundTruth;
```

Convert the ground truth object data to a JSON file in the ASAM OpenLABEL® format. Set `PrettyPrint` to `true` to insert indentations in the output file.

```
groundTruthToOpenLabel(gTruth,"vehicleGtruthOpenLABEL",PrettyPrint=true);
```

Inspect the output JSON file by using the `edit` command:

```
edit vehicleGtruthOpenLABEL.json
```

Input Arguments

`gTruth` — Ground truth object

`groundTruth` object (default)

Ground truth object, specified by a `groundTruth`. Only objects generated by the Image Labeler and Video Labeler apps that include labels can be converted using this function.

`jsonFilename` — JSON filename
character string (default)

JSON filename for the returned file, specified by a character string. For details regarding the JSON format, see ASAM OpenLABEL specification. Not all aspects of the ground truth object data can be represented in the OpenLABEL format. To compensate for the differences and ambiguities between the data that could not be represented and the ASAM specifications, the function makes these customizations to the JSON file:

- For an image collection used with the `Image Labeler`, each object in the JSON file's OpenLABEL *object_data* section, contains a *text* field that references the location of the images within the file system.
- Pixel label images are not embedded into the *image* OpenLABEL object. Instead, they are referenced in the *image* object using the *text* field in the *object_data_pointer* section of the JSON file.
- Projected cuboids from `groundTruth` objects are stored as eight points in an OpenLABEL `poly2d` object modified to accommodate it. These modified `poly2d` objects are tagged with the boolean field `matlab_projected_cuboid` set to `true`.
- Information stored in the `gTruth.LabelDefinitions` object such as, `Group`, `Description`, and `LabelColor` is placed in the *matlab_label_definitions* field of the custom *metadata* section in the OpenLABEL JSON file
- Pixel label values to class name correspondences are stored in the custom *matlab_label_definitions* field of the JSON file.
- Closed polygons contain a *hierarchy* field with the top-most layered polygon listed first, with a zero index. In the case of overlapping polygons, the top-layered polygons are marked with lower index numbers than the polygons they cover (below them).

Version History

Introduced in R2024a

See Also

Apps
Image Labeler

Objects
`groundTruth`

Functions
`jsonencode`

External Websites
ASAM OpenLABEL specification

groundTruthFromOpenLabel

Convert OpenLabel JSON file to ground truth object

Syntax

```
groundTruthFromOpenLabel(gTruth,jsonFile)
```

Description

`groundTruthFromOpenLabel(gTruth,jsonFile)` converts ASAM OpenLABEL® format JavaScript Object Notation (JSON) data `jsonFilename` to ground truth `gTruth`.

You can use this function with the JSON file produced by the `groundTruthToOpenLabel` function. However, using this function with any third-party tools might produce an invalid ground truth object. If you encounter this issue, you can contact the MathWorks® "Contact Technical Support" help system for assistance.

Examples

Convert ASAM OpenLABEL JSON file to Ground Truth Object

Add the folder that contains images, to the MATLAB® path. These data files must be available prior to converting the ASAM OpenLABEL JSON file.

```
imageDir = fullfile(matlabroot,"toolbox","vision","visiondata","vehicles");
addpath(imageDir);

data = load("vehicleTrainingGroundTruth.mat");
gTruthOriginal = data.vehicleTrainingGroundTruth;
```

Convert the ground truth object to JSON file in ASAM OpenLABEL format.

```
groundTruthToOpenLabel(gTruthOriginal,"vehicleGtruthOpenLABEL");
```

Convert JSON format file to a `groundTruth` object.

```
gTruth = groundTruthFromOpenLabel("vehicleGtruthOpenLABEL.json");
```

Inspect the data in the ground truth object.

```
imageLabeler(gTruth);
```

Input Arguments

`jsonFile` — JSON file

character string (default)

JSON file, specified by a character string. The file must contain ground truth stored in the ASAM OpenLabel format. For details regarding the JSON format, see ASAM OpenLABEL specification.

Output Arguments

`gTruth` — Ground truth object
`groundTruth` object (default)

Ground truth object, returned as a `groundTruth`.

Version History

Introduced in R2024b

See Also

Objects
`groundTruth`

Functions
`groundTruthToOpenLabel` | `jsonencode`

External Websites
ASAM OpenLABEL specification

folders2labels

Get list of labels from folder names

Syntax

```
lbls = folders2labels(loc)
lbls = folders2labels(loc,Name,Value)

lbls = folders2labels(ds)

[lbls,files] = folders2labels( ___ )
```

Description

Use this function when you are working on a machine or deep learning classification problem and your labeled data is stored in folders that have the corresponding label names.

`lbls = folders2labels(loc)` creates a list of labels based on the folder names specified by the location `loc`.

`lbls = folders2labels(loc,Name,Value)` specifies additional input arguments using name-value pairs. For example, `'FileExtensions','.mat'` includes only .mat files in the scan for labels.

`lbls = folders2labels(ds)` creates a list of labels based on the files contained in `ds`. `ds` can be a datastore, a `matlab.io.datastore.FileSet` object, or a `matlab.io.datastore.BlockedFileSet` object.

`[lbls,files] = folders2labels( ___ )` additionally returns a list of files. The `i`th element of `lbls` corresponds to the label of the `i`th file in `files`.

Examples

Labels from Folder Names

Create a folder called `Files` in the current folder containing three subfolders, `Files_1`, `Files_2`, and `Files_3`. Add to each subfolder a random number of files, each containing a random signal of random size.

```
mkdir Files

for kj = 1:3
    fname = "Files_" + kj;
    mkdir(fname)
    for jk = 1:randi(4)
        sname = "sig_" + kj + "_" + jk;
        sgn = randn(randi([30 50]),randi(2));
        save(sname,"sgn")
        movefile(sname + ".mat",fname)
    end
```

```
    movefile(fname,"Files")
end
```

List the contents of the folders.

```
dir("*/*/*")
```

```
Files Found in: Files\Files_1

.            ..           sig_1_1.mat  sig_1_2.mat  sig_1_3.mat  sig_1_4.mat

Files Found in: Files\Files_2

.            ..           sig_2_1.mat  sig_2_2.mat

Files Found in: Files\Files_3

.            ..           sig_3_1.mat  sig_3_2.mat  sig_3_3.mat
```

Create a list of labels based on the folder names.

```
lbls = folders2labels("Files")
```

```
lbls = 9×1 categorical
     Files_1
     Files_1
     Files_1
     Files_1
     Files_2
     Files_2
     Files_3
     Files_3
     Files_3
```

List the names of the files associated with the labels.

```
[~,files] = folders2labels("Files");
[~,fnames] = fileparts(files)
```

```
fnames = 9×1 string
    "sig_1_1"
    "sig_1_2"
    "sig_1_3"
    "sig_1_4"
    "sig_2_1"
    "sig_2_2"
    "sig_3_1"
    "sig_3_2"
    "sig_3_3"
```

Remove the `Files` directory you created at the beginning of the example.

```
rmdir Files s
```

Input Arguments

`loc` — Files or folders to scan for labels

character vector | cell array of character vectors | string scalar | string array

Files or folders to scan for labels, specified as a character vector, a cell array of character vectors, a string scalar, or a string array, containing the location of files or folders that are local or remote.

- Local files or folders — Specify `loc` as a local path to files or folders. If the files are not in the current folder, then the local path must specify full or relative paths. Files within subfolders of the specified folder are included by default. You can use the wildcard character (*) when specifying the local path. This character specifies that the file search include all matching files or all files in the matching folders.
- A remote location specified using an internationalized resource identifier (IRI).
- Remote files or folders — Specify `loc` to be the full paths of the files or folders as a uniform resource locator (URL) of the form `hdfs:///path_to_file`. For more information, see “Work with Remote Data”.

`folders2labels` looks for all file formats. To specify a custom list of file extensions to scan, use the `FileExtensions` argument.

Example: `'whale.mat'`

Example: `'../dir/data/signal.mat'`

Example: `"../dir/data/"`

Example: `{'dataFiles/Files_1/' 'dataFiles/Files_2/'}`

Example: `["dataFiles/Files_1/" "dataFiles/Files_2/"]`

Data Types: `char` | `string` | `cell`

`ds` — Data repository

datastore | `matlab.io.datastore.FileSet` object | `matlab.io.datastore.BlockedFileSet` object

Data repository, specified as a datastore, a `matlab.io.datastore.FileSet` object, or a `matlab.io.datastore.BlockedFileSet` object.

- If `ds` is a datastore, it must contain a `Files` property from which label names are parsed.
- If `ds` is a `matlab.io.datastore.FileSet` object, `folders2labels` obtains the label names from the file names listed in the `FileInfo` property of `ds`.
- If `ds` is a `matlab.io.datastore.BlockedFileSet` object, `folders2labels` obtains the label names from the file names listed in the `BlockInfo` property of `ds`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `folders2labels('C:\dir\signaldata','FileExtensions','.csv')` specifies a local path and includes only CSV files in the scan for labels.

`IncludeSubfolders` — Subfolder inclusion flag
`true` or `1` (default) | `false` or `0`

Subfolder inclusion flag, specified as `true` or `false`. Specify `true` to include all files and subfolders within each folder or `false` to include only the files within each folder.

Example: `'IncludeSubfolders',true`

Data Types: `logical` | `double`

`FileExtensions` — Signal file extensions
character vector | cell array of character vectors | string scalar | string array

Signal file extensions, specified as a string scalar, string array, character vector, or cell array of character vectors.

Example: `'FileExtensions','.csv'`

Data Types: `string` | `char` | `cell`

Output Arguments

`lbls` — List of labels
categorical vector

List of labels, returned as a categorical vector.

`files` — List of files
string vector

List of files, returned as a string vector. The `i`th element of `lbls` corresponds to a label for the `i`th file in `files`.

Version History

Introduced in R2021a

See Also

`labeledSignalSet` | `signalLabelDefinition` | `countlabels` | `filenames2labels` | `splitlabels`

splitlabels

Find indices to split labels according to specified proportions

Syntax

```
idxs = splitlabels(lblsrc,p)
idxs = splitlabels(lblsrc,p,'randomized')
idxs = splitlabels( ___ ,Name,Value)
```

Description

Use this function when you are working on a machine or deep learning classification problem and you want to split a dataset into training, testing, and validation sets that hold the same proportion of label values.

`idxs = splitlabels(lblsrc,p)` finds logical indices that split the labels in `lblsrc` based on the proportions or number of labels specified in `p`.

`idxs = splitlabels(lblsrc,p,'randomized')` randomly assigns the specified proportion of label values to each index set in `idxs`.

`idxs = splitlabels( ___ ,Name,Value)` specifies additional input arguments using name-value pairs. For example, `'UnderlyingDatastoreIndex',3` splits the labels only in the third underlying datastore of a combined datastore.

Examples

Split Vowels

Read William Shakespeare's sonnets with the `fileread` function. Extract all the vowels from the text and convert them to lowercase.

```
sonnets = fileread("sonnets.txt");
vowels = lower(sonnets(regexp(sonnets,"[AEIOUaeiou]")))';
```

Count the number of instances of each vowel.

```
cnts = countlabels(vowels)
```

```
cnts=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a      4940     18.368
      e      9028     33.569
      i      4895     18.201
      o      5710     21.232
      u      2321     8.6302
```

Split the vowels into a training set containing 500 instances of each vowel, a validation set containing 300, and a testing set with the rest. All vowels are represented with equal weights in the first two sets but not in the third.

```
spltn = splitlabels(vowels,[500 300]);

for kj = 1:length(spltn)
    cntsn{kj} = countlabels(vowels(spltn{kj}));
end
cntsn{:}
```

```
ans=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a        500       20
      e        500       20
      i        500       20
      o        500       20
      u        500       20


ans=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a        300       20
      e        300       20
      i        300       20
      o        300       20
      u        300       20


ans=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a       4140     18.083
      e       8228      35.94
      i       4095     17.887
      o       4910     21.447
      u       1521     6.6437
```

Split the vowels into a training set containing 50% of the instances, a validation set containing another 30%, and a testing set with the rest. All vowels are represented with the same weight across all three sets.

```
spltp = splitlabels(vowels,[0.5 0.3]);

for kj = 1:length(spltp)
    cntsp{kj} = countlabels(vowels(spltp{kj}));
end
cntsp{:}
```

```
ans=5×3 table
    Label    Count    Percent
    _____    _____    _______
```

```
      a      2470     18.367
      e      4514     33.566
      i      2448     18.203
      o      2855      21.23
      u      1161     8.6333


ans=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a      1482     18.371
      e      2708     33.569
      i      1468     18.198
      o      1713     21.235
      u       696     8.6277


ans=5×3 table
    Label    Count    Percent
    _____    _____    _______

      a       988     18.368
      e      1806     33.575
      i       979       18.2
      o      1142     21.231
      u       464     8.6261
```

Split Vowels and Consonants

Read William Shakespeare's sonnets with the `fileread` function. Remove all nonalphabetic characters from the text and convert to lowercase.

```
sonnets = fileread("sonnets.txt");
letters = lower(sonnets(regexp(sonnets,"[A-z]")))';
```

Classify the letters as consonants or vowels and create a table with the results. Show the first few rows of the table.

```
type = repmat("consonant",size(letters));
type(regexp(letters','[aeiou]")) = "vowel";

T = table(letters,type,'VariableNames',["Letter" "Type"]);
head(T)
```

```
    Letter       Type
    ______    __________

      t       "consonant"
      h       "consonant"
      e       "vowel"
      s       "consonant"
      o       "vowel"
      n       "consonant"
```

```
      n       "consonant"
      e       "vowel"
```

Display the number of instances of each category.

```
cnt = countlabels(T,'TableVariable',"Type")
```

```
cnt=2×3 table
      Type       Count    Percent
    _________    _____    _______

    consonant    46516    63.365
    vowel        26894    36.635
```

Split the table into two sets, one containing 60% of the consonants and vowels and the other containing 40%. Display the number of instances of each category.

```
splt = splitlabels(T,0.6,'TableVariable',"Type");

sixty = countlabels(T(splt{1},:),'TableVariable',"Type")
```

```
sixty=2×3 table
      Type       Count    Percent
    _________    _____    _______

    consonant    27910    63.366
    vowel        16136    36.634
```

```
forty = countlabels(T(splt{2},:),'TableVariable',"Type")
```

```
forty=2×3 table
      Type       Count    Percent
    _________    _____    _______

    consonant    18606    63.363
    vowel        10758    36.637
```

Split the table into two sets, one containing 60% of each particular letter and the other containing 40%. Exclude the letter *y*, which sometimes acts as a consonant and sometimes as a vowel. Display the number of instances of each category.

```
splt = splitlabels(T,0.6,'Exclude',"y");

sixti = countlabels(T(splt{1},:),'TableVariable',"Type")
```

```
sixti=2×3 table
      Type       Count    Percent
    _________    _____    _______

    consonant    26719    62.346
    vowel        16137    37.654
```

```
forti = countlabels(T(splt{2},:),'TableVariable',"Type")
```

```
forti=2×3 table
      Type       Count    Percent
```

```
    _________    _____    _______

    consonant    17813    62.349
    vowel        10757    37.651
```

Split the table into two sets of the same size. Include only the letters *e* and *s*. Randomize the sets.

```
halves = splitlabels(T,0.5,'randomized','Include',["e" "s"]);

cnt = countlabels(T(halves{1},:))
```

```
cnt=2×3 table
    Letter    Count    Percent
    ______    _____    _______

      e       4514     64.385
      s       2497     35.615
```

Split Data in Datastore

Create a dataset that consists of 100 Gaussian random numbers. Label 40 of the numbers as `A`, 30 as `B`, and 30 as `C`. Store the data in a combined datastore containing two datastores. The first datastore has the data and the second datastore contains the labels.

```
dsData = arrayDatastore(randn(100,1));
dsLabels = arrayDatastore([repmat("A",40,1); ...
            repmat("B",30,1); repmat("C",30,1)]);
dsDataset = combine(dsData,dsLabels);
cnt = countlabels(dsDataset,UnderlyingDatastoreIndex=2)
```

```
cnt=3×3 table
    Label    Count    Percent
    _____    _____    _______

      A       40        40
      B       30        30
      C       30        30
```

Split the data set into two sets, one containing 60% of the numbers and the other with the rest.

```
splitIndices = splitlabels(dsDataset,0.6,UnderlyingDatastoreIndex=2);

dsDataset1 = subset(dsDataset,splitIndices{1});
cnt1 = countlabels(dsDataset1,UnderlyingDatastoreIndex=2)
```

```
cnt1=3×3 table
    Label    Count    Percent
    _____    _____    _______

      A       24        40
      B       18        30
      C       18        30
```

```
dsDataset2 = subset(dsDataset,splitIndices{2});
cnt2 = countlabels(dsDataset2,UnderlyingDatastoreIndex=2)
```

```
cnt2=3×3 table
    Label    Count    Percent
    _____    _____    _______

      A       16        40
      B       12        30
      C       12        30
```

Input Arguments

`lblsrc` — Input label source

categorical vector | string vector | logical vector | numeric vector | cell array | table | datastore | `CombinedDatastore` object

Input label source, specified as one of these:

- A categorical vector.
- A string vector or a cell array of character vectors.
- A numeric vector or a cell array of numeric scalars.
- A logical vector or a cell array of logical scalars.
- A table with variables containing any of the previous data types.
- A datastore whose `readall` function returns any of the previous data types.
- A `CombinedDatastore` object containing an underlying datastore whose `readall` function returns any of the previous data types. In this case, you must specify the index of the underlying datastore that has the label values.

`lblsrc` must contain labels that can be converted to a vector with a discrete set of categories.

Example: `lblsrc = categorical(["B" "C" "A" "E" "B" "A" "A" "B" "C" "A"],["A" "B" "C" "D"])` creates the label source as a ten-sample categorical vector with four categories: `A`, `B`, `C`, and `D`.

Example: `lblsrc = [0 7 2 5 11 17 15 7 7 11]` creates the label source as a ten-sample numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `table` | `cell` | `categorical`

`p` — Proportions or numbers of labels

integer scalar | scalar in (0, 1) | vector of integers | vector of fractions

Proportions or numbers of labels, specified as an integer scalar, a scalar in the range (0, 1), a vector of integers, or a vector of fractions.

- If `p` is a scalar, `splitlabels` finds two splitting index sets and returns a two-element cell array in `idxs`.
  - If `p` is an integer, the first element of `idxs` contains a vector of indices pointing to the first `p` values of each label category. The second element of `idxs` contains indices pointing to the remaining values of each label category.

- If `p` is a value in the range (0, 1) and `lblsrc` has K_i elements in the ith category, the first element of `idxs` contains a vector of indices pointing to the first `p` × K_i values of each label category. The second element of `idxs` contains the indices of the remaining values of each label category.

- If `p` is a vector with N elements of the form p_1, p_2, ..., p_N, `splitlabels` finds N + 1 splitting index sets and returns an (N + 1)-element cell array in `idxs`.
  - If `p` is a vector of integers, the first element of `idxs` is a vector of indices pointing to the first p_1 values of each label category, the next element of `idxs` contains the next p_2 values of each label category, and so on. The last element in `idxs` contains the remaining indices of each label category.
  - If `p` is a vector of fractions and `lblsrc` has K_i elements of the ith category, the first element of `idxs` is a vector of indices concatenating the first p_1 × K_i values of each category, the next element of `idxs` contains the next p_2 × K_i values of each label category, and so on. The last element in `idxs` contains the remaining indices of each label category.

Note

- If `p` contains fractions, then the sum of its elements must not be greater than one.
- If `p` contains numbers of label values, then the sum of its elements must not be greater than the smallest number of labels available for any of the label categories.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'TableVariable',"AreaCode",'Exclude',["617" "508"]` specifies that the function split labels based on telephone area code and exclude numbers from Boston and Natick.

`Include` — Labels to include in index sets
vector of label categories | cell array of label categories

Labels to include in the index sets, specified as a vector or cell array of label categories. The categories specified with this argument must be of the same type as the labels in `lblsrc`. Each category in the vector or cell array must match one of the label categories in `lblsrc`.

`Exclude` — Labels to exclude from index sets
vector of label categories | cell array of label categories

Labels to exclude from the index sets, specified as a vector or cell array of label categories. The categories specified with this argument must be of the same type as the labels in `lblsrc`. Each category in the vector or cell array must match one of the label categories in `lblsrc`.

`TableVariable` — Table variable to read
first table variable (default) | character vector | string scalar

Table variable to read, specified as a character vector or string scalar. If this argument is not specified, then `splitlabels` uses the first table variable.

`UnderlyingDatastoreIndex` — Underlying datastore index
integer scalar

Underlying datastore index, specified as an integer scalar. This argument applies when `lblsrc` is a `CombinedDatastore` object. `splitlabels` counts the labels in the datastore obtained using the `UnderlyingDatastores` property of `lblsrc`.

Output Arguments

`idxs` — Splitting indices
cell array

Splitting indices, returned as a cell array.

Version History

Introduced in R2021a

See Also

`countlabels` | `filenames2labels` | `folders2labels`

cuboid2img

Project cuboids from 3-D world coordinates to 2-D image coordinates

Syntax

```
projectedCuboid = cuboid2img(cuboid,projectionMatrix)
[projectedCuboid,validIndex] = cuboid2img( ___ )
[projectedCuboid,validIndex,allFaceVertices] = cuboid2img( ___ )
[ ___ ] = cuboid2img( ___ ,Name=Value)
```

Description

`projectedCuboid = cuboid2img(cuboid,projectionMatrix)` projects the 3-D cuboids into an image using the image data and camera intrinsic parameters specified by the projection matrix. The function returns the image with projected cuboids.

`[projectedCuboid,validIndex] = cuboid2img( ___ )` returns a logical array that indicates which input cuboid projections are within the field of view of the camera, using the input arguments from the first syntax.

`[projectedCuboid,validIndex,allFaceVertices] = cuboid2img( ___ )` returns the face vertices of each projected cuboid to facilitate visualization of specific projected cuboid faces. You can use the `showShape` or `insertShape` function to visualize the projected cuboid faces.

`[ ___ ] = cuboid2img( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `cuboid2img(data.cuboid,projectionMatrix,OutputFormat="rectangles")` specifies to output each cuboid as two projected cuboid faces forming axis-aligned rectangles.

Examples

Project 3-D Cuboid to Image

Load an image into the workspace and resize it.

```
imOrig = imread(fullfile(matlabroot,"toolbox","vision","visiondata", ...
        "stopSignImages","image004.jpg"));
I = imresize(imOrig,[1080 1920]);
```

Load a MAT file that contains 3-D cuboid labels and the camera projection matrix of the sensor that captured the images.

```
data  = load("cuboidData.mat");
projectionMatrix = data.projectionMatrix;
cuboid = data.cuboid;
```

Convert the 3-D cuboids into 2-D projected cuboids.

```
vertices = cuboid2img(cuboid,projectionMatrix);
```

Insert the projected cuboids into the image, and then display the result.

```
Iout = insertShape(I,"projected-cuboid",vertices);
imshow(Iout)
```



Highlight One Face of -D Cuboid

Load an image into the MATLAB® workspace, and then resize it.

```
imOrig = imread(fullfile(matlabroot,"toolbox","vision","visiondata", ...
        "stopSignImages","image004.jpg"'));
I = imresize(imOrig,[1080 1920]);
```

Load MAT file previously created 3-D cuboid labels and the camera projection matrix of the sensor used to capture the images.

```
data  = load("cuboidData.mat");
projectionMatrix = data.projectionMatrix;
cuboid = data.cuboid;
```

Convert 3-D cuboids into 2-D projected cuboids, and obtain the face vertices of the projected cuboids.

```
[projectedCuboids,validIdx,allFaceVertices] = cuboid2img(cuboid,projectionMatrix);
```

Insert the projected cuboids into the image.

```
I = insertShape(I,"projected-cuboid",projectedCuboids);
```

Select the third face of each projected cuboid.

```
whichFace = 3;
selectedFaceVertices = allFaceVertices(:,:,whichFace);
```

Highlight the selected faces using a filled polygon and display the result.

```
J = insertShape(I,"filled-polygon",selectedFaceVertices);
imshow(J)
```



Input Arguments

`cuboid` — Cuboid positions, sizes, and orientations

M-by-9 matrix

Cuboid positions, sizes, and orientations, specified as an *M*-by-9 matrix. *M* is the number of cuboids. Each row specifies the position, size, and orientation of a cuboid in the form [*xctr yctr zctr xlen ylen zlen xrot yrot zrot*]:

- Position — *xctr*, *yctr*, and *zctr* are the coordinates of the center of the 3-D cuboid.
- Size — *xlen*, *ylen*, *zlen* are the lengths of the *x*,*y*, and *z* sides of the 3-cuboid.
- Orientation — *xrot yrot zrot* are the *x*,*y*, and *z* rotation angles of the 3-D cuboid in degrees.

Data Types: `single` | `double`

`projectionMatrix` — Camera projection matrix

3-by-4 matrix

Camera projection matrix, specified as a 3-by-4 matrix. The function uses the projection matrix to map the 3-D world points of each cuboid onto the 2-D image plane. You can create a projection matrix using the `cameraProjection` function.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `cuboid2img(data.cuboid,projectionMatrix,OutputFormat="rectangles")` specifies to output each cuboid as two projected cuboid faces forming axis-aligned rectangles.

`OutputFormat` — Output format

`"vertices"` (default) | `"rectangles"`

Output format in which to represent the projected cuboids, specified as `"vertices"` or `"rectangles"`.

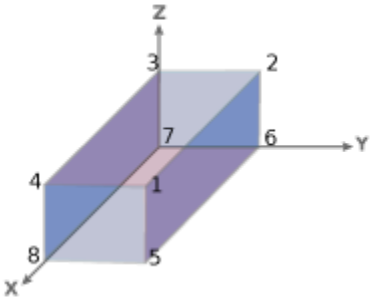
Output Format	Description
`"vertices"`	`projectedCuboid` is an 8-by-2-by-*P* array of vertices, where *P* is the number of projected cuboids. Each page of the array contains the [*x y*] locations for the four corners of the front face and four corners of the rear face of the projected cuboid.
`"rectangles"`	`projectedCuboid` is an *P*-by-8 matrix, where *P* is the number of rectangular projected cuboids. Each row of the matrix contains an eight-element vector of the form [*x1 y1 w1 h1 x2 y2 w2 h2*], where [*x1 y1*] and [*x2 y2*] are the upper-left coordinates of the front- and rear-facing rectangles, respectively. The function adjusts the projected cuboid vertices such that the front and rear cuboid faces form axis-aligned rectangles, and enclose all of the related projected vertices. Use the rectangular output when you need to create projected cuboid data compatible with the Image Labeler, Video Labeler, or Ground Truth Labeler app.

Output Arguments

`projectedCuboid` — Projected cuboids

8-by-2-by-*P* array | *P*-by-8 matrix

Projected cuboids, returned as an 8-by-2-by-*P* array or a *P*-by-8 matrix. *P* is the number of projected cuboids. *P* is less than or equal to the number of input 3-D cuboids *M*. Any cuboids specified in the input `cuboid` that are outside the camera field of view are not included in the output. Each page, *P* of the output contains the [*x y*] locations for the eight vertices of a projected cuboid. The order of the vertices is `[1 2 3 4 5 6 7 8]`, where `[1 4 8 5]` and `[2 3 7 6]` define the front-facing and rear-facing rectangles, respectively, counterclockwise from the upper-right vertex.



`validIndex` — Index of projected cuboids

M-by-1 logical vector

Index of projected cuboids, returned as an *M*-by-1 logical vector, where *M* is the number of input 3-D cuboids. A logical `1` value indicates a valid, corresponding projected cuboid in front of the camera, containing at least one vertex with positive coordinates. A logical `0` value indicates that the corresponding projected cuboid is invalid.

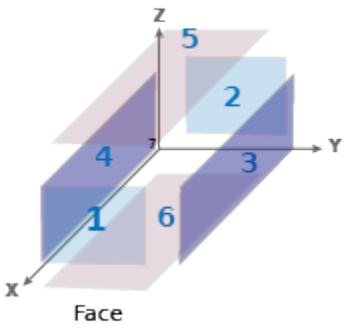
`allFaceVertices` — Face vertices

P-by-8-by-6 array

Face vertices, returned as a *P*-by-8-by-6 array, where *P* is the number of returned projected cuboids. Each page of the array contains the four vertices for the six faces of each of the returned projected cuboids. You can use this output to visualize specific faces of the cuboids. For example, to return the vertices of the fifth face of the third projected cuboid, specify:

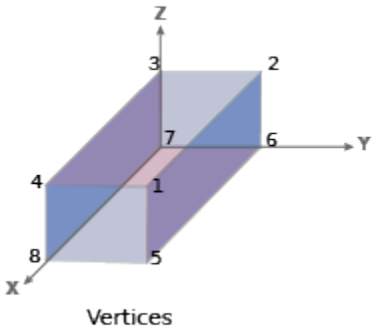
```
allFaceVertices(3,:,5)
```

The order of the cuboid faces in the array is front-facing, rear-facing, right-facing, left-facing, top-facing, and bottom-facing.



Face

Face	Face vertices
1	[1 4 8 5]
2	[2 6 7 3]
3	[1 5 6 2]
4	[4 3 7 8]
5	[1 2 3 4]
6	[5 8 7 6]



Vertices

You can use the `allFaceVertices` output with the `showShape` or `insertShape` function to visualize the faces of the projected cuboids. For an example, see “Highlight One Face of -D Cuboid” on page 3-109.

Version History

Introduced in R2022b

See Also

Objects
`cameraParameters`

Functions
`world2img` | `showShape` | `insertShape` | `insertObjectAnnotation`

generateCircleGridPoints

Generate circle grid point locations

Syntax

```
worldPoints = generateCircleGridPoints(patternDims,centerDistance)
worldPoints = generateCircleGridPoints( ___ ,Name=Value)
```

Description

`worldPoints = generateCircleGridPoints(patternDims,centerDistance)` generates circle grid point locations based on the specified number of circles in each dimension `patternDims` and the specified distance in world units between the centers of two adjacent circles `centerDistance`.

For more details on circle grid patterns, see “Circle Grid Patterns”.

`worldPoints = generateCircleGridPoints( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `PatternType="symmetric"` sets the pattern type to `"symmetric"`.

Examples

Generate Point Locations for Asymmetric Circle Grid

Specify the distance between the centers of adjacent circles, in millimeters (mm). By default, the function generates point locations for an asymmetric circle grid.

```
centerDistance = 9;
worldPoints = generateCircleGridPoints([4 11],centerDistance);
```

Generate Point Locations for Symmetric Circle Grid

Specify the distance between the centers, in millimeters (mm). Set the pattern type to `"symmetric"`.

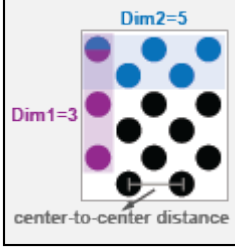
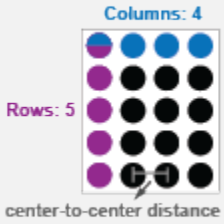
```
centerDistance = 19;
worldPoints = generateCircleGridPoints([8 11],centerDistance, ...
                                       PatternType="symmetric");
```

Input Arguments

`patternDims` — Pattern dimensions

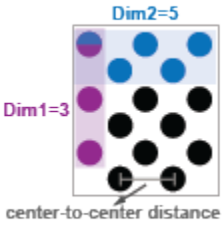
1-by-2 vector

Pattern dimensions, specified as a 1-by-2 vector that represents the number of circles in each dimension. For the asymmetric pattern, Dimension 1 corresponds to the number of circles in each column. Dimension 2 corresponds to the total number of circles in the first two adjacent rows. For more details on circle grid patterns, see “Circle Grid Patterns”.

Asymmetric	Symmetric
	

`centerDistance` — Distance between centers of two adjacent circles
scalar

Distance between centers of two adjacent circles in the same row or column, specified as a scalar in world units, such as millimeters.



Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `PatternType="symmetry"` sets the pattern type to `"symmetry"`

`PatternType` — Type of circle grid pattern
`"asymmetric"` (default) | `"symmetric"`

Type of circle grid pattern for which the function generates point locations, specified as `"asymmetric"` or `"symmetric"`.

Output Arguments

`worldPoints` — Circle grid center locations
M-by-2 matrix

Circle grid center locations, returned as an *M*-by-2 matrix. Each row of the matrix specifies the *xy*-coordinates of a circle center relative to the origin. The origin (0,0) is the center of the top-left circle in the grid. *M* is the total number of circles in the grid,calculated as the product of the circle count in each dimension: `prod(patternDims)`.

Version History

Introduced in R2021b

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`detectCircleGridPoints` | `estimateCameraParameters` | `generateCheckerboardPoints`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”

pcfromdepth

Convert depth image to point cloud

Syntax

```
ptCloud = pcfromdepth(depthImage,depthScaleFactor,intrinsics)
ptCloud = pcfromdepth( ___ ,Name=Value)
```

Description

`ptCloud = pcfromdepth(depthImage,depthScaleFactor,intrinsics)` converts a depth image using camera intrinsics, into a point cloud. The output point cloud is specified in 3-D world coordinates with its origin centered at the camera. The `depthScaleFactor`, normally provided by the RGB-D camera manufacturer specifies a division factor for the depth image.

This function applies only for RGB-D cameras.

`ptCloud = pcfromdepth( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `pcfromdepth(depthImage,depthScaleFactor,intrinsics,DepthRange=[0,Inf])` sets the depth range to the 2-element vector value of `[0,Inf]`.

Examples

Create Point Cloud Representation of Image Points Using Depth Image

Read and display the depth image taken from an RGB-D camera.

```
depthImage = imread("sampleDepth.png");
imshow(depthImage)
```



Read and display the color image that corresponds to the depth image.

```
colorImage = imread("sampleImage.png");
imshow("sampleImage.png")
```



Create a `cameraIntrinsics` object to store the camera intrinsics parameters. The focal length, principal point, and image size are in units of pixels. The maximum camera depth is in meters.

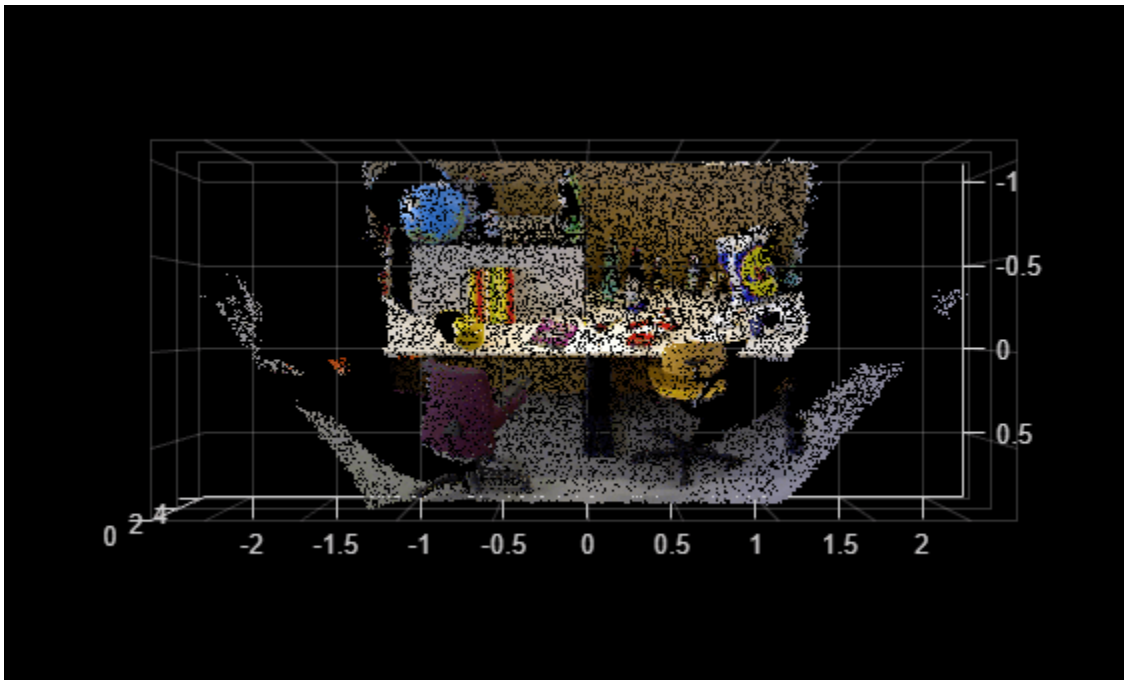
```
focalLength      = [535.4, 539.2];
principalPoint   = [320.1, 247.6];
imageSize        = size(depthImage,[1,2]);
intrinsics       = cameraIntrinsics(focalLength,principalPoint,imageSize);
depthScaleFactor = 5e3;
maxCameraDepth   = 5;
```

Convert depth map into world points, converting all points into point cloud.

```
ptCloud = pcfromdepth(depthImage,depthScaleFactor, intrinsics, ...
                      ColorImage=colorImage, ...
                      DepthRange=[0 maxCameraDepth]);
```

Display the resulting point cloud.

```
pcshow(ptCloud, VerticalAxis="Y", VerticalAxisDir="Up", ViewPlane="YX");
```



Detect and extract ORB features from the color image.

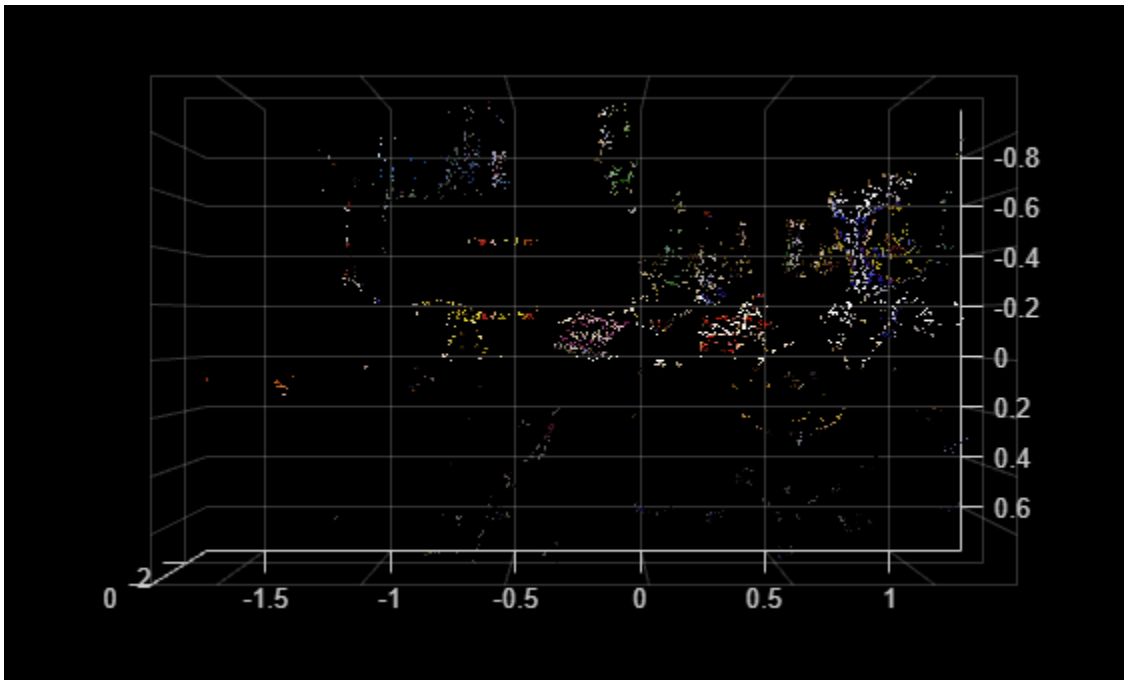
```
imagePoints = detectORBFeatures(im2gray(colorImage));
```

Convert depth map into world points, converting only feature points into point cloud.

```
ptCloud = pcfromdepth(depthImage,depthScaleFactor, intrinsics, ...
                      ColorImage=colorImage, ...
                      ImagePoints=imagePoints, ...
                      DepthRange=[0 maxCameraDepth]);
```

Display the resulting point cloud.

```
pcshow(ptCloud, VerticalAxis="Y", VerticalAxisDir="Up", ViewPlane="YX");
```



Input Arguments

`depthImage` — Depth image
M-by-*N* numeric matrix

Depth image, specified as an *M*-by-*N* numeric matrix. Input images must be undistorted.

`depthScaleFactor` — Depth correction factor
real scalar numeric

Depth correction factor, specified as a real scalar numeric type. This value is normally provided by the RGB-D camera manufacturer and specifies a division factor for the depth image input `depthImage`, which was taken with the camera. For example, for the world 3-D coordinates (*X*,*Y*,*Z*), where *Z* represents the depth, at any pixel coordinate (*u*,*v*), *Z* would be computed as:

```
Z = depthImage[u,v]/depthScaleFactor
```

`intrinsics` — Camera intrinsics
`cameraIntrinsics` object

Camera intrinsics, specified as a `cameraIntrinsics` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `pcfromdepth(depthImage,depthScaleFactor,intrinsics,DepthRange=[0,Inf])` sets the depth range to the 2-element vector value of `[0,Inf]`.

`ImagePoints` — Image points
`[]` (default) | *M*-by-2 matrix

Image points, specified as an *M*-by-2 matrix of [*x* *y*] coordinates, where *M* is the number of points, or by one of the feature point objects listed in “Point Feature Types”. Only points that are specified in the object are converted into a point cloud.

`ColorImage` — Color image
`[]` (default) | *M*-by-*N*-by-3 matrix

Color image, specified as an *M*-by-*N*-by-3 matrix of type numeric. The color image is associated with the depth image from the RGB-D camera.

`DepthRange` — Depth range of the RGB-D camera
`[0 inf]` (default) | 1-by-2 vector

Depth range of the RGB-D camera in world units, specified as a 1-by-2 vector. World units can be expressed as a measurement, such as meters. The range specifies the minimum and maximum depth range used to filter out invalid depth values after computing the point cloud.

`ROI` — Rectangular region of points to convert to a point cloud
`[1 1 size(depthImage,2) size(depthImage,1)]` (default) | [*x y width height*] vector

Rectangular region of points to convert to a point cloud, specified as [*x y width height*]. [*x y*] specifies the upper-left corner of the rectangular region of interest.

Output Arguments

`ptCloud` — Point cloud
`pointCloud` object

Point cloud created from depth image, returned as a `pointCloud` object. When you specify the `ImagePoints` name-value argument as an input, the function returns an unorganized point cloud. Otherwise, the function returns an organized point cloud.

Version History

Introduced in R2022b

References

[1] The TUM data set is provided by J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers under the CC-BY-4.0 license license. All warranties and representations are disclaimed; see the license for details. MathWorks modified the name of the RGB-D SLAM Dataset and Benchmark dataset to use in the example.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pcplayer` | `pointCloud` | `cameraIntrinsics`

Functions

`pcfromkinect` | `pcshow` | `detectORBFeatures`

Topics

“Point Feature Types”
“Visual SLAM with RGB-D Camera”

detectCircleGridPoints

Detect circle grid pattern in images

Syntax

```
imagePoints = detectCircleGridPoints(I,patternDims)
[imagePoints,imagesUsed] = detectCircleGridPoints(imageFileNames,patternDims)
[___] = detectCircleGridPoints(images,patternDims)

[imagePoints,pairsUsed] = detectCircleGridPoints(imageFileNames1,
imageFileNames2,patternDims)
[___] = detectCircleGridPoints(images1,images2,patternDims)

[___] = detectCircleGridPoints(___,Name=Value)
```

Description

Single Image Circle Grid Detection

`imagePoints = detectCircleGridPoints(I,patternDims)` detects a circle grid in a 2-D truecolor or grayscale image, `I`. For more details on circle grid patterns, see “Circle Grid Patterns”.

`[imagePoints,imagesUsed] = detectCircleGridPoints(imageFileNames,patternDims)` detects a circle grid in the image files specified by `imageFileNames`, and additionally returns the list of images in which the circle grid is detected `imagesUsed`.

`[___] = detectCircleGridPoints(images,patternDims)` detects a circle grid in the specified images `images`.

Stereo Pair Circle Grid Detection

`[imagePoints,pairsUsed] = detectCircleGridPoints(imageFileNames1, imageFileNames2,patternDims)` detects a circle grid in the stereo pairs of image files specified by `imageFileNames1` and `imageFileNames2`. The function additionally returns a list of the pairs in which the pattern is detected `pairsUsed`.

`[___] = detectCircleGridPoints(images1,images2,patternDims)` detects a circle grid in the stereo pairs of image files specified by `images1` and `images2`.

Optional Arguments

`[___] = detectCircleGridPoints(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `detectCircleGridPoints(I,patternDims,PatternType="symmetric")` detects a symmetric circle grid in the specified workspace.

Examples

Detect Symmetric Circle Grid in Image

Load an image containing a circle grid pattern into the workspace.

```
imageFileName = fullfile(toolboxdir("vision"),"visiondata", ...
                "calibration","circleGrid","mono","image08.jpg");
I = imread(imageFileName);
```

Define the dimensions for the circle grid pattern.

```
patternDims = [8 11];
```

Detect the circle grid points.

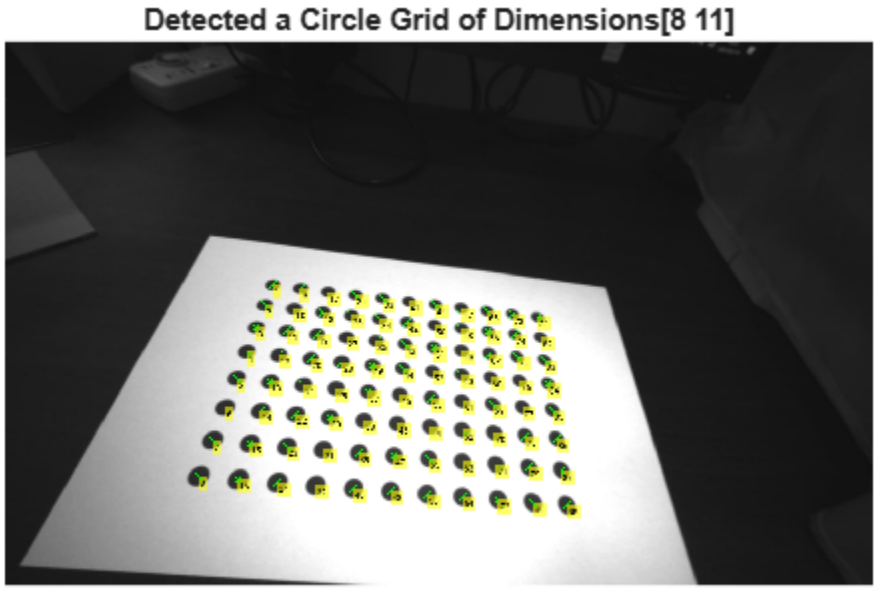
```
imagePoints = detectCircleGridPoints(I,patternDims,PatternType="symmetric")
```

```
imagePoints = 88×2

  395.1729  358.4451
  383.5573  388.4410
  370.8857  420.9698
  356.9887  456.5418
  341.6686  495.7388
  325.0617  538.9151
  306.5055  586.8763
  285.8323  640.1285
  436.4195  363.5016
  426.5806  393.4896
  415.9378  426.1730
  404.2823  461.8252
  391.4709  500.9889
  377.3850  543.9532
  361.7202  591.7051
      ⋮
```

Display the detected points on the image.

```
J = insertText(I,imagePoints,1:size(imagePoints,1));
J = insertMarker(J,imagePoints,"x",MarkerColor="green",Size=5);
imshow(J)
title("Detected a Circle Grid of Dimensions" + mat2str(patternDims))
```



Detect Asymmetric Circle Grid in Stereo Images

Specify your calibration images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata", ...
                            "calibration","circleGrid","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
images1 = leftImages.Files;
images2 = rightImages.Files;
```

Define the dimensions for the circle grid pattern.

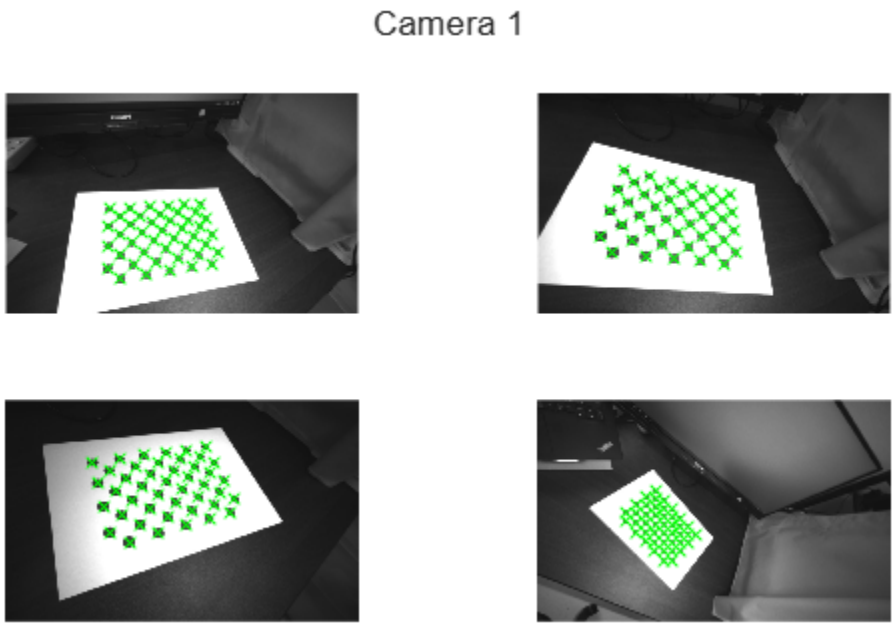
```
patternDims = [4 11];
```

Detect the circle grid points in the images.

```
[imagePoints,pairsUsed] = detectCircleGridPoints(images1,images2,patternDims);
```

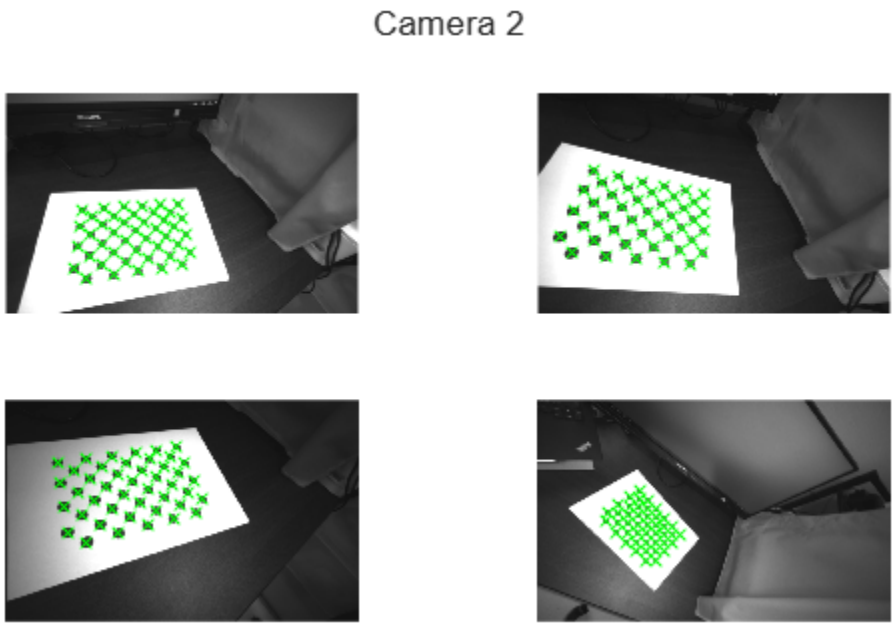
Display the points from the first four images of camera one.

```
figure
t1 = tiledlayout(2,2,TileSpacing="compact",Padding="compact");
for i = 1:4
  nexttile
  imshow(images1{i})
  hold on
  plot(imagePoints(:,1,i,1),imagePoints(:,2,i,1),"gx");
end
title(t1,"Camera 1")
```



Display points from the first four images from camera two.

```
images2 = images2(pairsUsed);
figure
t2 = tiledlayout(2,2,TileSpacing="compact",Padding="compact");
for i = 1:4
  nexttile
  imshow(images2{i})
  hold on
  plot(imagePoints(:,1,i,2),imagePoints(:,2,i,2),"gx");
end
title(t2,"Camera 2")
```

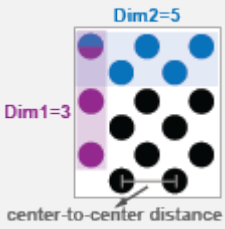
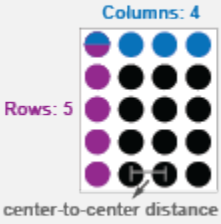


Input Arguments

`patternDims` — Pattern dimensions

two-element vector

Pattern dimensions, specified as a two-element vector [*nx ny*], where *nx* is the number of circles along the *x*-axis and *ny* is the number along the *y*-axis in the image.

Asymmetric	Symmetric
	

`I` — Image

2-D truecolor image | grayscale image

Image, specified as a 2-D truecolor image or grayscale image.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images` — Images

H-by-*W*-by-*numColorChannels*-by-*numImages* array

Images, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images1` — Stereo pair images from camera one
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera one, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`images2` — Stereo pair images from camera two
H-by-*W*-by-*numColorChannels*-by-*numImages* array

Stereo pair images from camera two, specified as an *H*-by-*W*-by-*numColorChannels*-by-*numImages* array, where *H* and *W* describe the height and width of each image, respectively. *numColorChannels* is the number of color channels in each image, and *numImages* is the number of images

Data Types: `uint8` | `int16` | `uint16` | `single` | `double`

`imageFileNames` — Image file names
cell array of character vectors | array of strings

Image file names, specified as a cell array of character vectors or an array of strings.

`imageFileNames1` — Stereo image file names for camera one
cell array of character vectors | array of strings

Stereo image file names for camera one, specified as a cell array of character vectors or an array of strings.

`imageFileNames2` — Stereo image file names for camera two
cell array of character vectors | array of strings

Stereo image file names for camera two, specified as a cell array of character vectors or an array of strings.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectCircleGridPoints(I,patternDims,PatternType="symmetric")` detects a symmetric circle grid in the specified workspace.

`PatternType` — Circle grid pattern type
`"asymmetric"` (default) | `"symmetric"`

Circle grid pattern type, specified as `"asymmetric"` or `"symmetric"`.

`circleColor` — Circle color
`"black"` (default) | `"white"`

Circle color, specified as `"black"` or `"white"`. Choose the color for the circle that has the strongest contrast with the background.

Output Arguments

`imagePoints` — Center coordinates of detected circle grid

M-by-2 matrix | *M*-by-2-by- *numPairs*-by-2 array

Center coordinates of detected circle grid, returned as an *M*-by-2 matrix or an *M*-by-2-by-*numPairs*-by-2 array. *M* is the number of circle grids detected, calculated as the product of the number of circle grids detected in each dimension. Each row of the matrix or array specifies the *x*-*y* coordinates of the center of a circle grid.

- For single images — This argument is an *M*-by-2 matrix when detecting circle grids in single images.
- For stereo pairs — This argument is an *M*-by-2-by-*numPairs*-by-2 array when detecting circle grids in stereo pairs of images. *numPairs* is the number of image pairs in which a circle grid is detected. `imagePoints(:,:,:,1)` returns the points from the first set of images, and `imagePoints(:,:,:,2)` are the points from the second set.

`imagesUsed` — Pattern detection flag

logical vector

Pattern detection flag, returned as a logical vector. A value of `true` in the `imagesUsed` vector indicates that the pattern has been detected in the corresponding image.

`pairsUsed` — Stereo pair pattern detection flag

logical vector

Stereo pair pattern detection flag, returned as a logical vector. A value of `true` in the `pairsUsed` vector indicates that the pattern has been detected in the corresponding image pair.

Algorithms

For a circle grid pattern to be detected, the pattern must:

- Contain at least 16 circles
- Be fully visible in the image of a single camera or in both images of a pair of stereo cameras
- Each circle in the pattern must have a minimum diameter of 50 pixels for successful detection.

For additional details on circle grid patterns, see “Circle Grid Patterns”.

Version History

Introduced in R2021b

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectCircleGridPoints` function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`generateCircleGridPoints` | `estimateCameraParameters` | `detectCheckerboardPoints` | `detectCharucoBoardPoints` | `detectAprilGridPoints` | `patternWorldPoints`

Objects
`cameraParameters` | `stereoParameters`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”

writeVideoScenes

Write video sequence to video file

Syntax

```
filenames = writeVideoScenes(gTruth,timeRanges,folderNames)
filenames = writeVideoScenes(gTruth,timeRanges,rootFolder,folderNames)
filenames = writeVideoScenes( ___ ,Name=Value)
```

Description

`filenames = writeVideoScenes(gTruth,timeRanges,folderNames)` writes sequences of ground truth data to separate files, `filenames`, that correspond to time ranges, `timeRanges`. Video files are written to the folders specified by the `folderNames` input.

The function writes video scenes using the image data and corresponding timestamp data contained in the specified `groundTruth` objects. All nonimage data, such as lidar, is ignored.

`filenames = writeVideoScenes(gTruth,timeRanges,rootFolder,folderNames)` also specifies the root folder name to prepend to each of the folder names.

`filenames = writeVideoScenes( ___ ,Name=Value)` specifies options using name-value arguments, in addition to any combination of input arguments from previous syntaxes. For example, `writeVideoScenes(gTruth,timeRanges,folderNames,Verbose=true)` displays progress information while writing video scenes to files.

Examples

Gather Scene Label Information and Write Video Scenes to Files

Load ground truth scene label definitions and label data into the workspace.

```
data = load("groundTruthSceneLabels.mat");
labelDefinitions = data.labelDefinitions;
labelData = data.labelData;
```

Create a ground truth data source using a video file.

```
gSource = groundTruthDataSource("viptrain.avi");
```

Create a ground truth object using the ground truth data source.

```
gTruth = groundTruth(gSource,labelDefinitions,labelData);
```

Gather all the scene time ranges and the scene labels.

```
[timeRanges,sceneLabels] = sceneTimeRanges(gTruth);
```

Select a folder in the temp directory to write the video scenes to.

```
rootFolder = fullfile(tempdir,"videoScenes");
```

Use the scene label names as folder names.

```
folderNames = sceneLabels;
```

Write the video scenes to the "`videoScenes`" folder. Specify the subfolder names for each duration as the scene label names.

```
filenames = writeVideoScenes(gTruth,timeRanges,rootFolder,folderNames);
```

```
[==================================================] 100%
Elapsed time: 00:00:02
Estimated time remaining: 00:00:00
```

Input Arguments

`gTruth` — Ground truth

vector of `groundTruth` objects | vector of `groundTruthMultisignal` objects

Ground truth, specified as a vector of `groundTruth` objects or `groundTruthMultisignal` objects.

`timeRanges` — Time ranges of ground truth source data

M-by-1 cell array | *M*-by-*N* cell array

Time ranges of ground truth source data, specified as an *M*-by-1 cell array for a `groundTruth` object or an *M*-by-*N* cell array for a `groundTruthMultisignal` object. *M* is the number of elements in `gTruth` and *N* represents the number of signals per element. Each cell of the array contains a *T*-by-2 duration matrix, where *T* is the number of time ranges in the corresponding element of `gTruth`. Each row of the matrix corresponds to a time range in the ground truth data for which a scene label has been applied, specified in the form [*rangeStart rangeEnd*].

`folderNames` — Folder names

M-1 cell array | *M*-by-*N* cell array

Folder names for the video scenes, specified as an *M*-by-1 cell array for a `groundTruth` object or an *M*-by-*N* cell array for a `groundTruthMultisignal` object. Each cell in the cell array corresponds to an element of `gTruth`, and contains a *T*-by-1 vector of strings or categorical vector that specifies the paths to the folders for the corresponding time ranges.

You must specify each folder name must be specified as a full path or as a relative path from the current folder. When you specify a folder name that does not exist, the function creates a new folder with the specified name. When writing video scene files, the function overwrites files with the same name that already exists in the specified folder. For example, if you call the `writeVideoScenes` function twice with the same input arguments, the files output by the second function call overwrite those from the first.

To select and write specific signals in a `groundTruthMultisignal` object to a specified location, use the `selectLabelsBySignalName` function before calling the `writeVideoScene` function.

`rootFolder` — Root folder name

character vector | string scalar

Root folder name to prepend to each of the folder names, specified as a character vector or a string scalar.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `writeVideoScenes(gTruth,timeRanges,folderNames,Verbose=true)` displays progress information while writing video scenes to files.

`NamePrefix` — File name prefix

array of strings | cell array of character vectors

File name prefix, specified as an array of strings or a cell array of character vectors. Default values depend on the `gTruth` input type:

- `groundTruth` object — The function uses the file names (without their extension) in the `DataSource` property of the `groundTruth` object.
- `groundTruthMultisignal` object — The function uses the input signal names.

`VideoProfile` — Video profile

`"Motion JPEG AVI"` (default) | `"Archival"` | `"Motion JPEG 2000"` | `"MPEG-4"` | `"Uncompressed AVI"` | `"Indexed AVI"` | `"Grayscale AVI"`

Video profile for writing video scenes, specified as one of the video profiles listed in the table. The `VideoProfile` argument usage is the same as the `profile` argument of the `VideoWriter` object.

Value of `VideoProfile`	Description
`"Archival"`	Motion JPEG 2000 file with lossless compression
`"Motion JPEG AVI"`	AVI file using Motion JPEG encoding
`"Motion JPEG 2000"`	Motion JPEG 2000 file
`'MPEG-4'`	MPEG-4 file with H.264 encoding (systems with Windows 7 or later, or macOS 10.7 and later)
`'Uncompressed AVI'`	Uncompressed AVI file with `RGB24` video
`'Indexed AVI'`	Uncompressed AVI file with indexed video
`'Grayscale AVI'`	Uncompressed AVI file with grayscale video

`Verbose` — Display progress on screen

`true` or `1` (default) | `false` or `0`

Display progress on screen, specified as a logical `1`(`true`) or `0`(`false`).

Output Arguments

`filenames` — Full path file names

M-by-1 cell array

Full path file names to the saved video scenes, returned as an *M*-by-1 cell array of *T*-by-1 vector of strings. *M* is the number of elements in the `gTruth` input argument. *T* is the number of time ranges in the corresponding element of `gTruth`. Each element of the vector of strings specifies the full path to the saved video scene for the corresponding time range.

The function sets the output file names as `NamePrefix_`*UID*`.`*EXT*, where *UID* is a unique integer index for each written scene and *EXT* is the video file extension determined by the `VideoProfile` argument.

Version History

Introduced in R2021b

See Also

Objects

`groundTruthMultisignal` | `groundTruthDataSource` | `groundTruth`

Functions

`selectLabelsBySignalName` | `gatherLabelData` | `objectDetectorTrainingData` | `pixelLabelTrainingData` | `sceneTimeRanges`

sceneTimeRanges

Time ranges of scene labels from ground truth data

Syntax

```
timeRanges = sceneTimeRanges(gTruth)
[timeRanges,sceneLabels] = sceneTimeRanges(gTruth)
```

Description

`timeRanges = sceneTimeRanges(gTruth)` returns the time ranges that define the start and end times for the scene labels applied to the ground truth data sources in `gTruth`.

`[timeRanges,sceneLabels] = sceneTimeRanges(gTruth)` also returns the scene labels that correspond to the time ranges.

Examples

Gather Scene Time Ranges and Scene Labels

Load ground truth scene label definitions and label data into the workspace.

```
data = load("groundTruthSceneLabels.mat");
labelDefinitions = data.labelDefinitions;
labelData = data.labelData;
```

Create a ground truth data source using a video file.

```
gSource = groundTruthDataSource("viptrain.avi");
```

Create a ground truth object using the ground truth data source.

```
gTruth = groundTruth(gSource,labelDefinitions,labelData);
```

Gather all the scene time ranges and the scene labels.

```
[timeRanges,sceneLabels] = sceneTimeRanges(gTruth);
```

Display the scene time ranges for the ground truth data.

```
timeRanges{1}
```

```
ans = 3×2 duration
        0 sec   8.0333 sec
      8.3 sec   10.033 sec
       12 sec   16.333 sec
```

Display the corresponding scene labels for the ground truth data.

```
sceneLabels{1}
```

```
ans = 3×1 categorical
     walking
     sitting
     trainMoving
```

Input Arguments

`gTruth` — Ground truth

vector of `groundTruth` objects | vector of `groundTruthMultisignal` objects

Ground truth, specified as a vector of `groundTruth` objects or `groundTruthMultisignal` objects.

Output Arguments

`timeRanges` — Time ranges of ground truth source data

M-by-1 cell array

Time ranges of the ground truth source data, returned as an *M*-by-1 cell array. *M* is the number of elements in `gTruth`. Each of the cell array contains a *T*-by-2 duration matrix. *T* is the number of time ranges in the corresponding element of `gTruth`. Each row of the matrices corresponds to a time range in the ground truth data for which a scene label has been applied, specified in the form [*rangeStart rangeEnd*].

To select and obtain scene time ranges for specific signals in a `groundTruthMultisignal` object, use the `selectLabelsBySignalName` function before calling the `sceneTimeRanges` function.

`sceneLabels` — Scene labels for each time range

M-by-1 cell array of categorical vectors

Scene labels for each time range, returned as an *M*-by-1 cell array of *T*-by-2 categorical vectors. *M* is the number of elements in `gTruth`. *T* is the number of time ranges in each corresponding ground truth data source.

Version History

Introduced in R2021b

See Also

Objects

`groundTruthMultisignal` | `groundTruthDataSource` | `groundTruth`

Functions

`selectLabelsBySignalName` | `gatherLabelData` | `objectDetectorTrainingData` | `pixelLabelTrainingData` | `writeVideoScenes`

stereoParametersFromOpenCV

Convert stereo camera parameters from OpenCV to MATLAB

Syntax

```
stereoParams = stereoParametersFromOpenCV(intrinsicMatrix1,
distortionCoefficients1,intrinsicMatrix2,distortionCoefficients2,
rotationOfCamera2,translationOfCamera2,imageSize)
stereoParams = stereoParametersFromOpenCV( ___ ,WorldUnits=worldUnits)
```

Description

`stereoParams = stereoParametersFromOpenCV(intrinsicMatrix1, distortionCoefficients1,intrinsicMatrix2,distortionCoefficients2, rotationOfCamera2,translationOfCamera2,imageSize)` converts the OpenCV stereo parameters, specified by the input arguments, into a MATLAB `stereoParameters` object `stereoParams`.

The OpenCV spatial coordinate system specifies the upper-left pixel center at (`0,0`), whereas the MATLAB spatial coordinate system specifies the pixel center at (`1,1`). The `stereoParametersFromOpenCV` function compensates for this difference by adding 1 to both of the *x* and *y*-values for the converted principal point.

OpenCV stereo parameters cannot be converted to a MATLAB `stereoParameters` object when an OpenCV fisheye model is used. In this case, you can recalibrate the stereo camera using the Stereo Camera Calibrator app.

`stereoParams = stereoParametersFromOpenCV( ___ ,WorldUnits=worldUnits)` specifies a string `worldUnits`, that describes the units of the world points in addition to the input arguments from the previous syntax. Specify worldUnits as a character vector or string scalar. For example, `stereoParametersFromOpenCV( ___ ,WorldUnits="mm")` describes all parameters in terms of the world units "mm".

Examples

Rectify Stereo Images in MATLAB Using Stereo Parameters from OpenCV

Load OpenCV stereo parameters from a MAT file into the workspace.

```
load openCVStereoParameters.mat
```

Convert the loaded stereo parameters from OpenCV to MATLAB format.

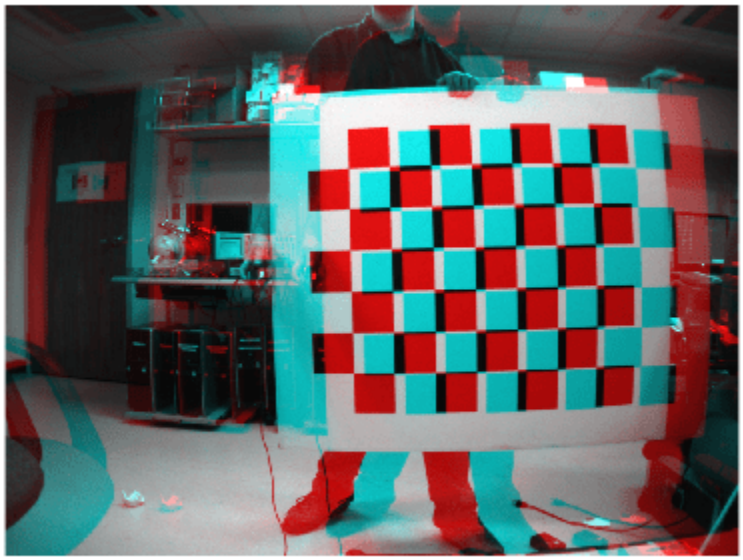
```
stereoParams = stereoParametersFromOpenCV(intrinsicMatrix1, ...
    distortionCoefficients1,intrinsicMatrix2,distortionCoefficients2, ...
    rotationOfCamera2,translationOfCamera2,imageSize);
```

Load the images to rectify.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','calibration','stereo');
I1 = imread(fullfile(imageDir,'left','left01.png'));
I2 = imread(fullfile(imageDir,'right','right01.png'));
```

Display the unrectified images.

```
imshow(stereoAnaglyph(I1,I2));
```

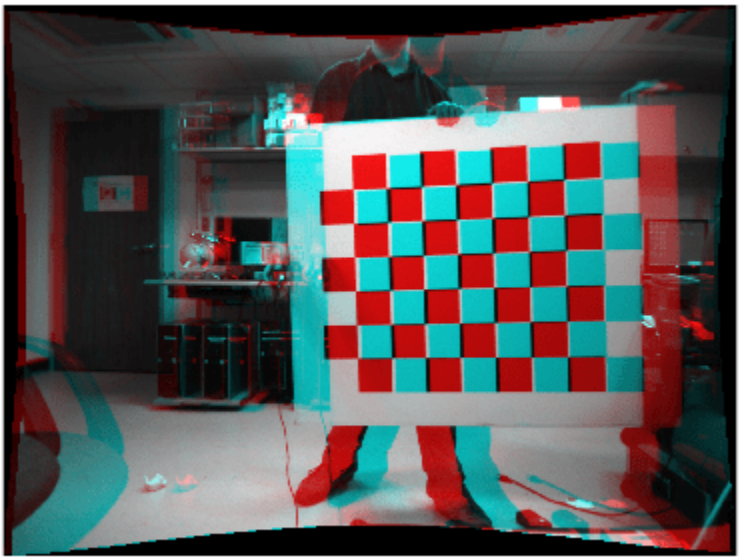


Rectify the images using the full output view.

```
[J1,J2] = rectifyStereoImages(I1,I2,stereoParams,OutputView='full');
```

Display the resulting rectified images.

```
imshow(stereoAnaglyph(J1,J2));
```



Import Stereo Camera Parameters from ROS

The ROS camera calibration package estimates stereo camera parameters using the OpenCV camera calibration tools [1]. After calibrating a stereo camera in ROS, you can export its camera parameters to an INI file using the camera calibration parser. To use the calibrated stereo camera with Computer Vision Toolbox™ functions, such as `rectifyStereoImages`, you must read the camera parameters from the INI file and convert them into a `stereoParameters` object using `stereoParametersFromOpenCV`.

Note: The `stereoParametersFromOpenCV` function supports importing stereo camera parameters for only those pinhole camera models that use the `ROS plumb-bob` distortion model.

Read Stereo Camera Parameters from ROS INI File

Read the stereo camera parameters stored in stereo`Params.ini` using the helper function `helperReadINI`.

```
stereoParamsINI = helperReadINI("stereoParams.ini");
```

Compute Baseline Parameters of Stereo Camera

The baseline parameters of a stereo camera describe the relative translation and rotation of the two cameras in the stereo camera pair. The relative rotation and translation of camera 2 with respect to camera 1 is required to create the `stereoParameters` object using `stereoParametersFromOpenCV`. You can compute these from the rectification and projection matrices read from the ROS INI file [2].

Extract the two camera parameters from the `stereoParams` structure.

```
cameraParams1 = stereoParamsINI.narrow_stereo_left;
cameraParams2 = stereoParamsINI.narrow_stereo_right;
```

Extract the translation of camera 2 relative to camera 1 from the last column of the projection matrix.

```
translationOfCamera2 = cameraParams2.projection(:,end);
```

The rotation of camera 2 relative to camera 1, R_{21}, is derived from the rectification matrices of the stereo pair R_1 and R_2. The rectification matrices are the rotation matrices that align the camera coordinate system to the ideal stereo image plane such that epipolar lines in both stereo images are parallel. Compute the rotation of camera 2 relative to camera 1 as $R_{21} = R_2 * R_1^T$.

```
rotationOfCamera2 = cameraParams2.rectification*cameraParams1.rectification';
```

Create `stereoParameters` Object Using `stereoParametersFromOpenCV`

Extract the intrinsic matrices and distortion coefficients of the two cameras from the `stereoParams` structure.

```
intrinsicMatrix1 = cameraParams1.camera_matrix;
intrinsicMatrix2 = cameraParams2.camera_matrix;

distortionCoefficients1 = cameraParams1.distortion;
distortionCoefficients2 = cameraParams2.distortion;
```

Obtain the image size from the `image` field of the `stereoParams` structure.

```
imageSize = [stereoParamsINI.image.height stereoParamsINI.image.width];
```

Use `stereoParametersFromOpenCV` to create a `stereoParameters` object from the ROS stereo camera parameters.

```
stereoParametersObj = stereoParametersFromOpenCV(intrinsicMatrix1, ...
    distortionCoefficients1,intrinsicMatrix2,distortionCoefficients2, ...
    rotationOfCamera2,translationOfCamera2,imageSize);
```

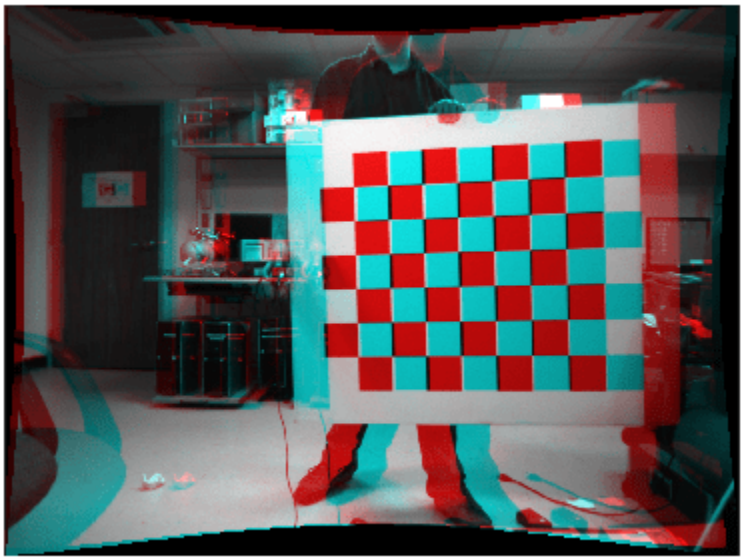
Rectify Pair of Stereo Images

Use the imported stereo parameters with `rectifyStereoImages` to rectify an image pair captured using the calibrated stereo camera.

```
% Load the image pair.
imageDir = fullfile(toolboxdir("vision"),"visiondata","calibration","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
I1 = readimage(leftImages,1);
I2 = readimage(rightImages,1);

% Rectify the image pair.
[J1,J2] = rectifyStereoImages(I1,I2,stereoParametersObj,OutputView="full");

% Display the results.
figure
J = stereoAnaglyph(J1,J2);
imshow(J)
```



Supporting Functions

helperReadINI

The `helperReadINI` function reads the camera parameters from its input INI file that has been exported from ROS.

```
function cameraParams = helperReadINI(filename)
% helperReadINI reads a ROS INI file, filename, and returns a structure with
% these fields: image, <camera_name1>, <camera_name2>. image is a
% structure describing the height and width of the image captured by the
% cameras of the stereo pair. The fields <camera_name1> and <camera_name2>
% are structures named after the camera names present in the INI file, and they contain
% these fields: camera_matrix, distortion, rectification_matrix,
% and projection_matrix. These fields are stored in the INI file with their
% values placed in a new line followed by their name.

    f = fopen(filename,"r");
    sectionName = '';

    while ~feof(f)
        % Read line from file.
        line = fgetl(f);

        % Trim leading and trailing whitespaces.
        line = strtrim(line);

        if isempty(line) || line(1)=='#'
            % Skip empty line and comments.
            continue
        elseif line(1) == '[' && line(end) == ']'
            % Identify section names and continue reading.
            sectionName = line(2:end-1);
```

```
            sectionName = strrep(sectionName,'/','_');
            continue
        end

        % Replace blankspaces with underscores to create valid MATLAB variable
        % name.
        name = line;
        name(name == ' ') = '_';

        % Read the value data in upcoming lines.
        value = [];
        while ~feof(f)
            line = fgetl(f);
            line = strtrim(line);

            if isempty(line)
                % A empty line indicates end of value data.
                break
            elseif line(1)=='#'
                % Skip comment lines.
                continue
            end
            line = str2num(line); %#ok
            value = [value; line]; %#ok
        end

        % Store post-processed value.
        if isempty(sectionName)
            cameraParams.(name) = value;
        else
            cameraParams.(sectionName).(name) = value;
        end
    end

    fclose(f);
end
```

References

[1] https://wiki.ros.org/camera_calibration

[2] https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/CameraInfo.html

Input Arguments

`intrinsicMatrix1` — Intrinsics matrix of camera 1

3-by-3 matrix

Intrinsics matrix of camera 1 from OpenCV, specified as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and *cx*,*cy*) is the principal point in specified in OpenCV.

`intrinsicMatrix2` — Intrinsics matrix of camera 2
3-by-3 matrix

Intrinsics matrix of camera 2 from OpenCV, specified as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and *cx*,*cy* is the principal point in specified in OpenCV.

`distortionCoefficients1` — Camera 1 radial and tangential distortion coefficients
5-element vector | 8-element vector

Camera 1 radial and tangential distortion coefficients, returned as a 5- or 8-element vector.

- 5-element vector — Of the form [*k1 k2 p1 p2 k3*]
- 8-element vector — Of the form [*k1 k2 p1 p2 k3 k4 k5 k6*].

  The values of *k1*, *k2*, ..., *k6* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.

`distortionCoefficients2` — Camera 2 radial and tangential distortion coefficients
5-element vector | 8-element vector

Camera 2 radial and tangential distortion coefficients, returned as a 5- or 8-element vector.

- 5-element vector — Of the form [*k1 k2 p1 p2 k3*]
- 8-element vector — Of the form [*k1 k2 p1 p2 k3 k4 k5 k6*].

  The values of *k1*, *k2*, ..., *k6* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.

`rotationOfCamera2` — Rotation of camera 2 relative to camera 1
3-by-3 matrix

Rotation of camera 2 relative to camera 1 from OpenCV, specified as a 3-by-3 matrix.

`translationOfCamera2` — Translation of camera 2 relative to camera 1
three-element vector

Translation of camera 2 relative to camera 1 from OpenCV, specified as a three-element vector.

`imageSize` — Image size
two-element vector

Image size, specified as a two-element vector in the form [*mrows*,*ncols*].

Output Arguments

`stereoParams` — Stereo parameters
`stereoParameters` object

Stereo parameters, returned as a `stereoParameters` object. The object contains OpenCV intrinsics for a pinhole camera model. In cases where the OpenCV pinhole model uses more than eight distortion coefficients, which cannot be converted to a `cameraIntrinsics` object, you can recalibrate your camera using the Stereo Camera Calibrator app as an alternative solution.

Version History

Introduced in R2021b

R2024b: Import OpenCV pinhole camera model with six radial distortion coefficients

The `stereoParametersFromOpenCV` has been updated to support OpenCV pinhole camera model with 6 radial distortion coefficients.

See Also

Apps
Stereo Camera Calibrator | Camera Calibrator

Objects
`cameraParameters` | `stereoParameters`

Functions
`undistortImage` | `stereoAnaglyph` | `stereoParametersToOpenCV` | `cameraIntrinsicsFromOpenCV` | `cameraIntrinsicsToOpenCV`

stereoParametersToOpenCV

Convert stereo camera parameters from MATLAB to OpenCV

Syntax

```
[intrinsicMatrix1,distortionCoefficients1,intrinsicMatrix2,
distortionCoefficients2,rotationOfCamera2,translationOfCamera2,imageSize] =
stereoParametersToOpenCV(stereoParams)
```

Description

`[intrinsicMatrix1,distortionCoefficients1,intrinsicMatrix2,
distortionCoefficients2,rotationOfCamera2,translationOfCamera2,imageSize] =
stereoParametersToOpenCV(stereoParams)` converts a MATLAB `stereoParameters` object `stereoParams` to OpenCV stereo parameters.

The OpenCV spatial coordinate system specifies the upper-left pixel center at `(0,0)`, whereas the MATLAB spatial coordinate system specifies the pixel center at `(1,1)`. The `stereoParametersToOpenCV` function compensates for this difference by subtracting 1 to both of the *x* and *y*-values for the converted principal point.

OpenCV stereo parameters do not include the skew of a pinhole camera model. Therefore, only the intrinsics estimated without the skew can be exported to OpenCV.

Examples

Calibrate Stereo Camera and Convert Parameters to OpenCV Format

Specify the calibration images.

```
leftImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
   "calibration","stereo","left"));
rightImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
   "calibration","stereo","right"));
```

Detect the checkerboards in the image pairs.

```
[imagePoints,patternDims] = detectCheckerboardPoints(leftImages.Files,rightImages.Files);
```

Specify the world coordinates of the checkerboard keypoints in millimeters.

```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
stereoParams = estimateCameraParameters(imagePoints,worldPoints, ...
   ImageSize=imageSize);
```

Convert the MATLAB® stereo parameters object to the OpenCV format.

```
[intrinsicMatrix1,distortionCoefficients1,intrinsicMatrix2, ...
   distortionCoefficients2,rotationOfCamera2,translationOfCamera2] =...
   stereoParametersToOpenCV(stereoParams);
```

Input Arguments

`stereoParams` — Stereo parameters

`stereoParameters` object

Stereo parameters, specified as a `stereoParameters` object.

Output Arguments

`intrinsicMatrix1` — Intrinsics matrix of camera 1

3-by-3 matrix

Intrinsics matrix of camera 1 formatted for OpenCV, returned as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and (*cx*,*cy*) is the principal point in specified in the OpenCV input.

`intrinsicMatrix2` — Intrinsics matrix of camera 2

3-by-3 matrix

Intrinsics matrix of camera 2 formatted for OpenCV, returned as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and (*cx*,*cy*) is the principal point in specified in the OpenCV input.

`distortionCoefficients1` — Camera 1 radial and tangential distortion coefficients

five-element vector

Camera 1 radial and tangential distortion coefficients, returned as a five-element vector in the form [*k1 k2 p1 p2 k3*]. The values of *k1*, *k2*, and *k3* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.

`distortionCoefficients2` — Camera 2 radial and tangential distortion coefficients

five-element vector

Camera 2 radial and tangential distortion coefficients, returned as a five-element vector in the form [*k1 k2 p1 p2 k3*]. The values of *k1*, *k2*, and *k3* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.

`rotationOfCamera2` — Rotation of camera 2 relative to camera 1
3-by-3 matrix

Rotation of camera 2 relative to camera 1 from OpenCV, specified as a 3-by-3 matrix.

`translationOfCamera2` — Translation of camera 2 relative to camera 1
three-element vector

Translation of camera 2 relative to camera 1 from OpenCV, specified as a three-element vector.

`imageSize` — Image size
two-element vector

Image size, specified as a two-element vector in the form [*mrows*,*ncols*].

Version History

Introduced in R2021b

See Also

Apps
Stereo Camera Calibrator | Camera Calibrator

Objects
`cameraParameters` | `stereoParameters`

Functions
`undistortImage` | `stereoAnaglyph` | `cameraIntrinsicsFromOpenCV` | `stereoParametersFromOpenCV`

cameraIntrinsicsFromOpenCV

Convert camera intrinsic parameters from OpenCV to MATLAB

Syntax

```
intrinsics = cameraIntrinsicsFromOpenCV(intrinsicMatrix,
distortionCoefficients,imageSize)
```

Description

`intrinsics = cameraIntrinsicsFromOpenCV(intrinsicMatrix, distortionCoefficients,imageSize)` converts the OpenCV intrinsics, specified by the input arguments, into a MATLAB object `intrinsics`, which can be `cameraIntrinsics` or `cameraIntrinsicsKB`.

The OpenCV spatial coordinate system specifies the upper-left pixel center at (`0,0`), whereas the MATLAB spatial coordinate system specifies the pixel center at (`1,1`). The `cameraIntrinsicsFromOpenCV` function compensates for this difference by adding 1 to both of the *x* and *y*-values for the converted principal point.

Examples

Undistort Image in MATLAB Using Camera Intrinsics from OpenCV

Define OpenCV camera intrinsic parameters in the workspace.

```
intrinsicMatrix = [729.4644    0       570.6455;
                   0       728.8196  346.0108;
                   0         0         1  ];
distortionCoefficients = [-0.4262  0.5460  0.0038  -0.0051  -0.6176];
```

Define the image size returned by the camera.

```
imageSize = [712 1072];
```

Convert the intrinsic parameters from OpenCV to MATLAB format.

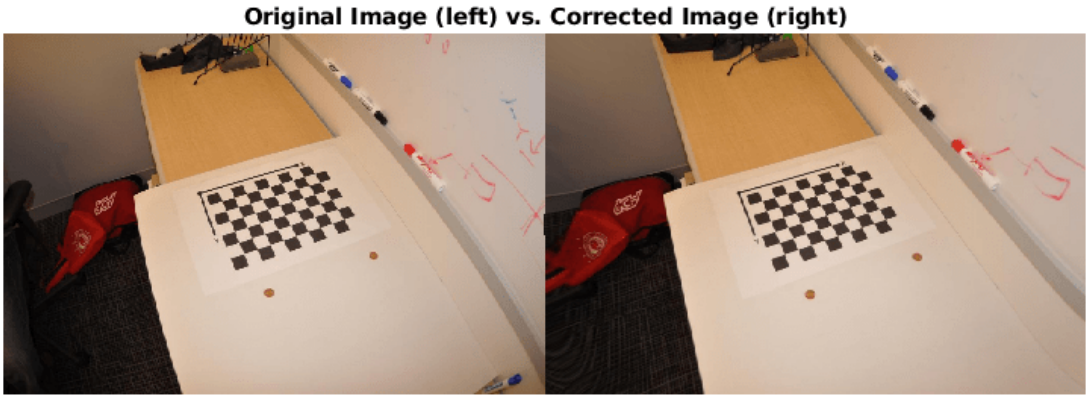
```
intrinsics = cameraIntrinsicsFromOpenCV(intrinsicMatrix, ...
                                        distortionCoefficients,imageSize);
```

Load the image to undistort.

```
filename = fullfile(toolboxdir("vision"),"visiondata","calibration", ...
                    "mono","image01.jpg");
I = imread(filename);
```

Undistort the image and display the results.

```
J = undistortImage(I,intrinsics);
imshowpair(I,J,"montage");
title("Original Image (left) vs. Corrected Image (right)");
```



Undistort Image Using Fisheye Camera Parameters from OpenCV

Define OpenCV fisheye parameters in the workspace.

```
intrinsicMatrix = [875.88  0.00   1005.62; ...
                   0.00    874.76 741.52; ...
                   0.00,   0.00,  1.00];
distortionCoeffs = [0.08, -0.16, 0.35, -0.26];
```

Define the image size returned by the camera.

```
imageSize =[1500 2000];
```

Import the intrinsic camera parameters for a fisheye lens from OpenCV.

```
intrinsicsKB = cameraIntrinsicsFromOpenCV(intrinsicMatrix,distortionCoeffs,imageSize)
```

```
intrinsicsKB =
  cameraIntrinsicsKB with properties:

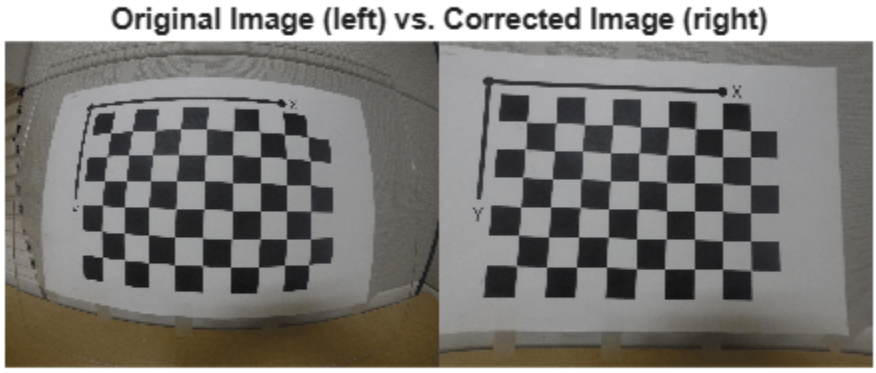
              FocalLength: [875.8800 874.7600]
            PrincipalPoint: [1.0066e+03 742.5200]
                 ImageSize: [1500 2000]
    DistortionCoefficients: [0.0800 -0.1600 0.3500 -0.2600]
                         K: [3×3 double]
```

Load an image to undistort.

```
filename = fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","gopro","gopro01.jpg");
I = imread(filename);
```

Undistort the image and display the results.

```
J = undistortImage(I,intrinsicsKB);
imshowpair(I,J,"montage");
title("Original Image (left) vs. Corrected Image (right)");
```



Import Camera Intrinsic Parameters from ROS

The ROS camera calibration package estimates camera intrinsic parameters using the OpenCV camera calibration tools [1]. After calibrating a camera in ROS, you can import its intrinsic parameters to a YAML file using the camera calibration parser in ROS. To use the calibrated camera with Computer Vision Toolbox™ functions, such as `undistortImage`, you must read the camera parameters from the YAML file and then convert them into a `cameraIntrinsics` object using `cameraIntrinsicsFromOpenCV`.

Note: The `cameraIntrinsicsFromOpenCV` function supports importing camera intrinsic parameters for only those pinhole camera models that use the `ROS plumb-bob` distortion model.

Read Camera Intrinsic Parameters from a ROS YAML File

Read the camera parameters stored in `cameraParams.yaml` using the helper function `helperReadYAML`.

```
intrinsicsParams = helperReadYAML("cameraParams.yaml");
```

Create `cameraIntrinsics` Object Using `cameraIntrinsicsFromOpenCV`

Use the `cameraIntrinsicsFromOpenCV` function to create a `cameraIntrinsics` object from the camera matrix and the distortion coefficients.

```
imageSize = [intrinsicsParams.image_height intrinsicsParams.image_width];
intrinsicMatrix = intrinsicsParams.camera_matrix;
distortionCoefficients = intrinsicsParams.distortion_coefficients;

intrinsicsObj = cameraIntrinsicsFromOpenCV(intrinsicMatrix,distortionCoefficients,imageSize);
```

Undistort Image

Use the imported camera intrinsics with `undistortImage` to undistort an image captured using the calibrated camera.

```
% Load the captured image.
imageName = fullfile(toolboxdir("vision"),"visiondata","calibration","stereo","left","left01.png
I = imread(imageName);

% Undistort the image.
J = undistortImage(I,intrinsicsObj,OutputView="full");

% Display the result.
figure
montage({I,J})
```



Supporting Functions

helperReadYAML

The `helperReadYAML` function reads the monocular camera parameters from the input YAML file that was exported from ROS.

```
function cameraParams = helperReadYAML(filename)
% helperReadYAML reads a ROS YAML file, filename, and returns a structure
% with these fields: image_width, image_height, camera_name,
% camera_matrix, distortion_model, distortion_coefficients,
% rectification_matrix, and projection_matrix. These fields are stored
% in the YAML file colon separated from their values in different lines.

    f = fopen(filename,'r');
    stringFields = {'camera_name','distortion_model'};
```

```
    while ~feof(f)

        [name,value,isEmptyLine] = helperReadYAMLLine(f);
        if isEmptyLine
            continue
        end

        if ~isempty(value)
            % Convert all values to numbers except for known string
            % fields.
            if ~any(contains(name, stringFields))
                value = str2num(value); %#ok
            end
        else
            % An empty value in ROS YAML files indicates a matrix in
            % upcoming lines. Read the matrix from the upcoming lines.
            value = helperReadYAMLMatrix(f);
        end

        % Store post-processed value.
        cameraParams.(name) = value;
    end

    fclose(f);
end
```

helperReadYAMLMatrix

The `helperReadYAMLMatrix` function reads the rows, columns and data fields of a matrix in the ROS YAML file.

```
function matrix = helperReadYAMLMatrix(f)
%   helperReadYAMLMatrix reads a matrix from the ROS YAML file. A matrix in
%   a ROS YAML file has three fields: rows, columns and data. rows and col
%   describe the matrix size. data is a contiguous array of the matrix
%   elements in row-major order. This helper function assumes the presence
%   of all three fields of a matrix to return the correct matrix.

    numRows = 0;
    numCols = 0;
    data = [];

    % Read numRows, numCols and matrix data.
    while ~feof(f)
        [name,value,isEmptyLine] = helperReadYAMLLine(f);

        if isEmptyLine
            continue
        end

        switch name
            case 'rows'
                numRows = str2num(value); %#ok
            case 'cols'
                numCols = str2num(value); %#ok
            case 'data'
                data    = str2num(value); %#ok
```

```
                % Terminate the while loop as data is the last
                % field of a matrix in the ROS YAML file.
                break
            otherwise
                % Terminate the while loop if any other field is
                % encountered.
                break
        end
    end

    if numel(data) == numRows*numCols
        % Reshape the matrix using row-major order.
        matrix = reshape(data,[numCols numRows])';
    end
end
```

helperReadYAMLLine

The `helperReadYAMLLine` function reads a line of a ROS YAML file.

```
function [name,value,isEmptyLine] = helperReadYAMLLine(f)

    % Read line from file.
    line = fgetl(f);

    % Trim leading and trailing white spaces.
    line = strtrim(line);

    if isempty(line) || line(1)=='#'
        % Empty line or comment.
        name = '';
        value = '';
        isEmptyLine = true;
    else
        % Split the line to get name and value.
        c = strsplit(line,':');
        assert(length(c)==2,'Unexpected file format')

        name = c{1};
        value = strtrim(c{2}); % Trim leading white space.
        isEmptyLine = false;
    end
end
```

References

[1] ROS.org Camera Calibration

Input Arguments

`intrinsicMatrix` — Camera intrinsic matrix

3-by-3 matrix

Camera intrinsic matrix from OpenCV, specified as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and (*cx*,*cy*) is the principal point in OpenCV.

`distortionCoefficients` — Camera distortion coefficients
4-element vector | 5-element vector | 8-element vector

Camera distortion coefficients from OpenCV, specified as a 4-, 5-, or 8-element vector. The function returns a `cameraIntrinsics` object when using extended pinhole model distortion coefficients and a `cameraIntrinsicsKB` object for fisheye distortion coefficients.

- Extended pinhole model — 5-element vector of the form [*k1 k2 p1 p2 k3*] or an 8-element vector of the form [*k1 k2 p1 p2 k3 k4 k5 k6*].

  In the Robot Operating System (ROS), the 5-element vector is referred to as the plumb_bob model, while the 8-element vector is known as the rational_polynomial model.

  The values of *k1*, *k2*, ..., *k6* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.
- Kannala-Brandt model [1] — 4-element vector of the form [*k1 k2 k3 k4*]. These values are the distortion coefficients from the OpenCV fisheye camera model.

`imageSize` — Image size
2-element vector

Image size, specified as a 2-element vector in the form [*mrows*,*ncols*].

Output Arguments

`intrinsics` — Camera intrinsic parameters
`cameraIntrinsics` object | `cameraIntrinsicsKB` object

Camera intrinsic parameters, returned as a `cameraIntrinsics` or `cameraIntrinsicsKB` object. The function returns a `cameraIntrinsics` object when using extended pinhole model distortion coefficients and a `cameraIntrinsicsKB` object for fisheye distortion coefficients.

- `cameraIntrinsics` object — Contains OpenCV intrinsics for a pinhole camera model. In cases where the OpenCV pinhole model uses more than eight distortion coefficients, which cannot be converted to a `cameraIntrinsics` object, you can recalibrate your camera using the Camera Calibrator app as an alternative solution.
- `cameraIntrinsicsKB` object — Contains camera intrinsic calibration parameters for a fisheye lens, using the Kannala-Brandt model.

Version History

Introduced in R2021b

R2024b: Import OpenCV pinhole camera model with six radial distortion coefficients

The `cameraIntrinsicsFromOpenCV` has been updated to support OpenCV pinhole camera model with 6 radial distortion coefficients.

References

[1] Juho Kannala and Sami Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28:1335–40, 09 2006.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps

Stereo Camera Calibrator | Camera Calibrator

Objects

`cameraIntrinsics` | `cameraIntrinsicsKB` | `cameraParameters` | `stereoParameters`

Functions

`undistortImage` | `stereoAnaglyph` | `stereoParametersFromOpenCV` | `stereoParametersToOpenCV` | `cameraIntrinsicsToOpenCV`

cameraIntrinsicsToOpenCV

Convert camera intrinsic parameters from MATLAB to OpenCV

Syntax

```
[intrinsicMatrix,distortionCoefficients] = cameraIntrinsicsToOpenCV(
intrinsics)
```

Description

`[intrinsicMatrix,distortionCoefficients] = cameraIntrinsicsToOpenCV(intrinsics)` converts a MATLAB `cameraIntrinsics` or `cameraParameters` object, specified by `intrinsics`, to OpenCV camera intrinsic parameters.

The OpenCV spatial coordinate system specifies the upper-left pixel center at (`0,0`), whereas the MATLAB spatial coordinate system specifies the pixel center at (`1,1`). The `cameraIntrinsicsToOpenCV` function compensates for this difference by subtracting 1 from both the *x* and *y*-values for the converted principal point.

OpenCV camera intrinsic parameters do not include the skew of a pinhole camera model. Therefore, only the intrinsics that were estimated without the skew can be exported to OpenCV.

Examples

Calibrate Camera in MATLAB and Convert Intrinsic Parameters to OpenCV

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
                        "calibration","mono"));
imageFileNames = images.Files;
```

Detect the checkerboard calibration pattern in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate the world coordinates of the corners of the squares. Square size is in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Convert the intrinsic parameters to OpenCV format

```
[intrinsicMatrix,distortionCoefficients] = cameraIntrinsicsToOpenCV(params);
```

Input Arguments

`intrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object | `cameraParameters` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` or a `cameraParameters` object.

Output Arguments

`intrinsicMatrix` — Camera intrinsic matrix

3-by-3 matrix

Camera intrinsics matrix formatted for OpenCV, returned as a 3-by-3 matrix of the form:

$$\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

where *fx* and *fy* are the focal lengths in the *x* and *y*-directions, and (*cx*,*cy*) is the principal point in OpenCV.

`distortionCoefficients` — Camera radial and tangential distortion coefficients

five-element vector

Camera radial and tangential distortion coefficients, returned as a five-element vector in the form [*k1 k2 p1 p2 k3*]. The values of *k1*, *k2*, and *k3* describe the radial distortion and *p1* and *p2* describe the tangential distortion, specified in OpenCV.

In the Robot Operating System (ROS), the 5-element vector is referred to as the plumb_bob model.

Version History

Introduced in R2021b

See Also

Apps

Stereo Camera Calibrator | Camera Calibrator

Objects

`cameraParameters` | `stereoParameters`

Functions

`undistortImage` | `stereoAnaglyph` | `stereoParametersToOpenCV` | `cameraIntrinsicsFromOpenCV` | `stereoParametersFromOpenCV` | `cameraIntrinsicsFromOpenCV`

detectSIFTFeatures

Detect scale invariant feature transform (SIFT) features

Syntax

```
points = detectSIFTFeatures(I)
points = detectSIFTFeatures(I,Name=Value)
```

Description

`points = detectSIFTFeatures(I)` detects SIFT features in the 2-D grayscale or binary input image `I` and returns a `SIFTPoints` object. The `detectSIFTFeatures` function implements the Scale-Invariant Feature Transform (SIFT) algorithm to find local features in an image.

`points = detectSIFTFeatures(I,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `detectSIFTFeatures(I,ContrastThreshold=0.0133)` detects SIFT features with a contrast of less than `0.0133`.

Examples

Detect Interest Points and Mark Locations

Load an image.

```
I = imread('cameraman.tif');
```

Detect SIFT features in the image.

```
points = detectSIFTFeatures(I);
```

Display the results.

```
imshow(I);
hold on;
plot(points.selectStrongest(10))
```



Input Arguments

`I` — Input image

M-by-*N* grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be a real, nonsparse value.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectSIFTFeatures(I,ContrastThreshold=0.0133)` detects SIFT features with a contrast of less than `0.0133`.

`ContrastThreshold` — Contrast threshold

`0.0133` (default) | non-negative scalar in range `[0,1]`

Contrast threshold for selecting the strongest features, specified as a non-negative scalar in the range `[0,1]`. The threshold is used to filter out weak features in low-contrast regions of the image. Increase the contrast threshold to decrease the number of returned features.

`EdgeThreshold` — Edge threshold

`10.0` (default) | non-negative scalar

Edge threshold, specified as a non-negative scalar greater than or equal to `1`. The threshold is used to filter out unstable edge-like features in the image that are susceptible to noise. Increase the edge threshold to decrease the number of features removed by filtering.

`NumLayersInOctave` — Number of layers in each octave
`3` (default) | integer scalar

Number of layers in each octave, specified as an integer scalar greater than or equal to `1`. The number of octaves is computed automatically from the image resolution. Increase the number of layers in each octave to detect larger features in the image.

`Sigma` — Sigma of the Gaussian
`1.6` (default) | scalar

Sigma of the Gaussian, specified as a scalar. The sigma of the Gaussian is applied to the input image at the initial octave. Sigma values are typically in the range `[1,2]`. Lower the sigma value if the image is blurry.

`ROI` — Region of interest
`[1 1 M N]` (default) | four-element vector

Region of interest for keypoint detection, specified as the comma-separated pair consisting of `'ROI'` and a vector of the format [*x y width height*]. The first two elements represent the location of the upper left corner of the region of interest. The last two elements represent the width and the height of the region of interest. The width and height of the region of interest must each be a value greater than or equal to 63.

Output Arguments

`points` — SIFT features
`SIFTPoints` object

SIFT features, returned as a `SIFTPoints` object. This object contains information about SIFT features detected in the 2-D grayscale or binary image.

Version History

Introduced in R2021b

References

[1] Lowe, David G. "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision* 60, no. 2 (November 2004): 91--110.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.

- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`SIFTPoints` | `SURFPoints` | `MSERRegions` | `BRISKPoints` | `cornerPoints` | `ORBPoints` | `KAZEPoints`

Functions
`detectBRISKFeatures` | `detectFASTFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectSURFFeatures` | `extractFeatures` | `matchFeatures`

Topics
“Create Panorama”

normalRotation

Compute transform for rotation of normal to plane

Syntax

```
tform = normalRotation(model,referenceVector)
```

Description

`tform = normalRotation(model,referenceVector)` returns the geometric transformation object for a normal vector of a plane to the reference vector, `referenceVector`. `model` is a `planeModel`.

Examples

Transform Point Cloud

Construct a velodyneFileReader object.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the first frame of lidar data.

```
frameNumber = 1;
ptCloud  = readFrame(veloReader,frameNumber);
```

Find the ground plane.

```
maxDistance = 0.4;
referenceVector = [0 0 1];

groundPlane = pcfitplane(ptCloud,maxDistance,referenceVector);
```

Transform the ground plane such that it is parallel to the X-Y plane.

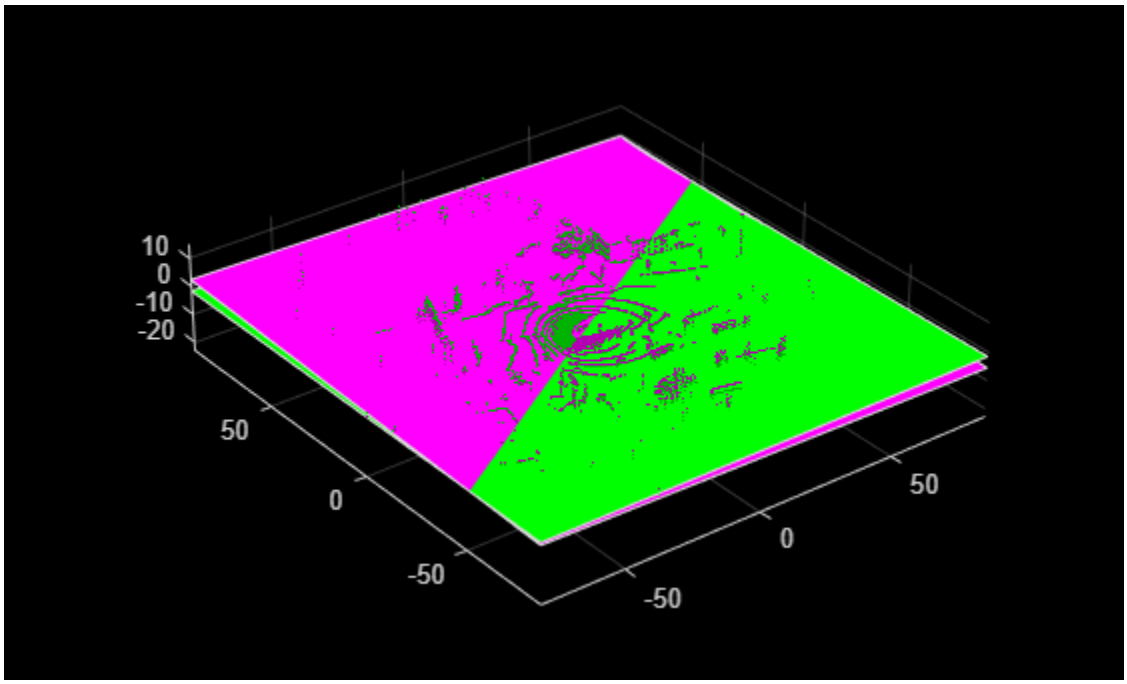
```
tform = normalRotation(groundPlane,referenceVector);
```

Transform the point cloud.

```
ptCloudOut = pctransform(ptCloud,tform);
```

Display the original and transformed point cloud.

```
planeParams = groundPlane.Parameters * tform.T;
transformedPlane = planeModel(planeParams);
figure;
pcshowpair(ptCloudOut,ptCloud);
hold on;
plot(groundPlane, 'Color', 'magenta');
plot(transformedPlane, 'Color', 'green');
```



Input Arguments

`model` — Parametric plane model
plane model

Parametric plane model returned by `planeModel`.

`referenceVector` — Reference vector
1-by-3 vector

Reference vector, specified as a 1-by-3 vector.

Output Arguments

`tform` — Geometric transformation
`rigidtform3d` object

Geometric transformation, returned as a `rigidtform3d` object. The `rigidtform3d` object describes the rigid 3-D transformation that aligns the normal vector of a plane to `referenceVector`.

Version History

Introduced in R2020b

R2022b: Returns `rigidtform3d` object
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `normalRotation` function now

returns `tform` as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `tform` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

See Also

Functions
`pcfitplane` | `pcregistercorr`

Objects
`pointCloud` | `rigidtform3d` | `planeModel`

showShape

Display shapes on image, video, or point cloud

Syntax

```
showShape(shape,position)
showShape( ___ ,Name=Value)
```

Description

`showShape(shape,position)` displays one or more instances of a shape `shape` in the current axes at the specified locations `position`.

`showShape( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments from the previous syntax. For example, `Color="yellow"` sets the color for the shapes to yellow.

Examples

Display Detected Objects in Image

Read an image into the workspace.

```
I = imread("visionteam1.jpg");
```

Create an aggregate channel features (ACF) people detector.

```
detector = peopleDetectorACF
```

```
detector =
  acfObjectDetector with properties:

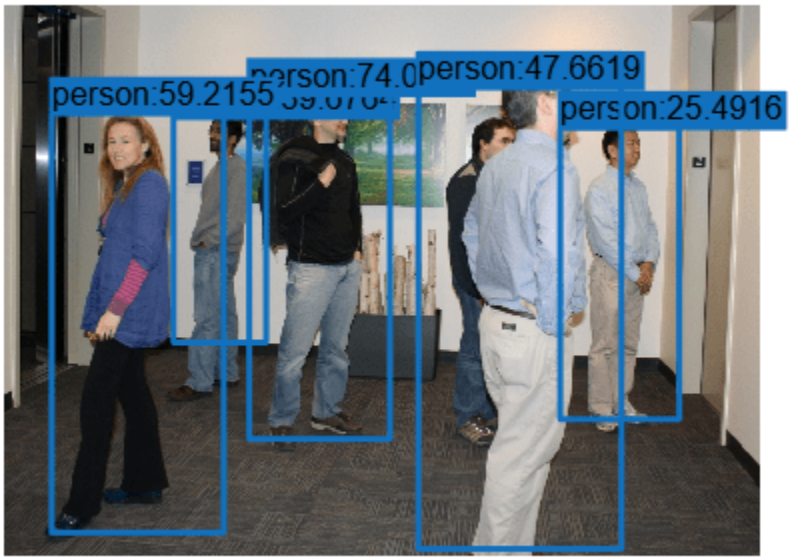
             ModelName: 'inria-100x41'
    ObjectTrainingSize: [100 41]
       NumWeakLearners: 2048
```

Detect people in the image.

```
[bboxes,scores] = detect(detector,I);
```

Display the image with a labeled rectangle around each detected person. Display the associated detection score around each detected person.

```
figure
imshow(I)
labels = "person:" + scores;
showShape("rectangle",bboxes,Label=labels)
```



Display Bounding Box Around Object in Point Cloud

Read point cloud data into the workspace.

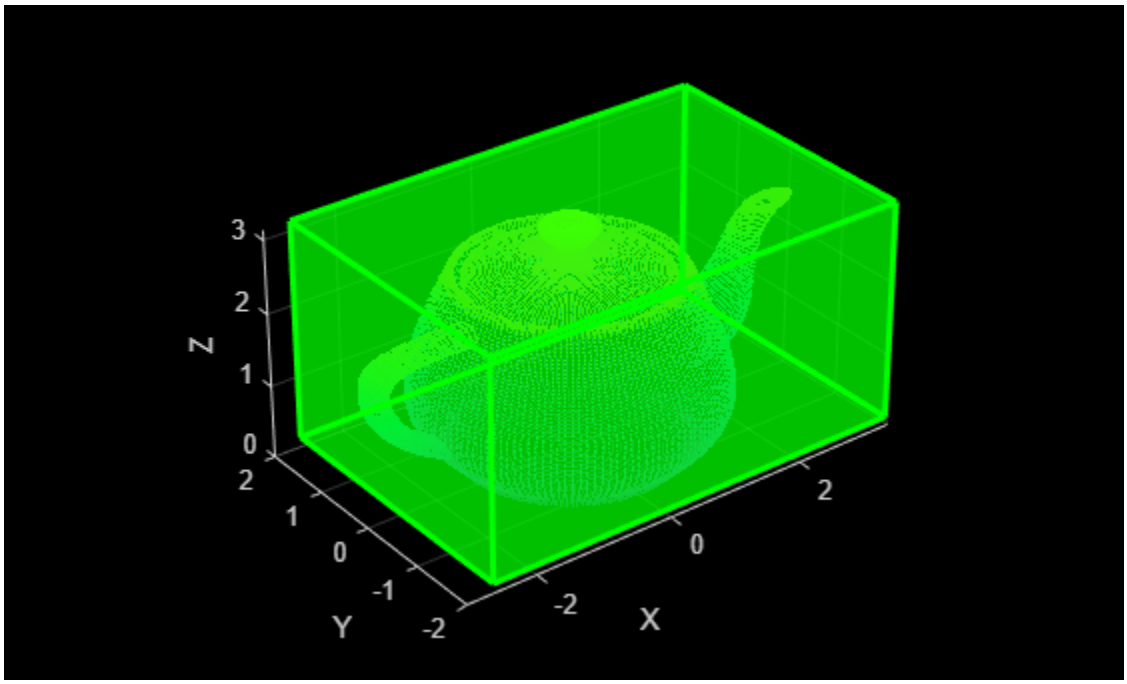
```
ptCloud = pcread("teapot.ply");
```

Display the point cloud data.

```
figure
pcshow(ptCloud)
xlabel("X")
ylabel("Y")
zlabel("Z")
```

Define a cuboid and display it in green with an opacity of `0.5`.

```
pos = [0.3753 0 1.65 6 4 3 0 0 0];
showShape("cuboid",pos,Color="green",Opacity=0.5)
```



Visualize Cuboid Around Moving Object in Point Cloud Stream

Read point cloud data into the workspace.

```
ptCloud = pcread("teapot.ply");
```

Define a rotation matrix and 3-D transform to rotate the point cloud, and an associated cuboid, by 5 degrees.

```
rot = 5;
R = [ cosd(rot) sind(rot) 0 0; ...
     -sind(rot) cosd(rot) 0 0; ...
      0       0           1 0; ...
      0       0           0 1];

tform = affine3d(R);
```

Compute the *x* and *y* rendered limits of the point cloud, to ensure that you do not clip the point cloud during rotation.

```
pcLimits = abs([ptCloud.XLimits ptCloud.YLimits]);
maxLimit = max(pcLimits);
```

Add a margin to the plot to prevent the rotation from clipping the cuboid.

```
margin = 1;
maxLimit = maxLimit + margin;

xlimits = [-maxLimit maxLimit];
ylimits = [-maxLimit maxLimit];
zlimits = ptCloud.ZLimits;
```

Create a player for visualizing the point cloud.

```
player = pcplayer(xlimits,ylimits,zlimits);
```

Customize the player axes labels.

```
xlabel(player.Axes,"X (m)");
ylabel(player.Axes,"Y (m)");
zlabel(player.Axes,"Z (m)");
```

Define a cuboid around the point cloud.

```
cuboidPosition = [0.3753 0 1.65 6 4 3 0 0 0];
```

Define the output view for a cuboid rotation. Use the same limits as the player so that the view does not clip the cuboid. Then, display the rotated point cloud and cuboid.

```
gridSize = [1 1 1];
ref = imref3d(gridSize,xlimits,ylimits,zlimits);

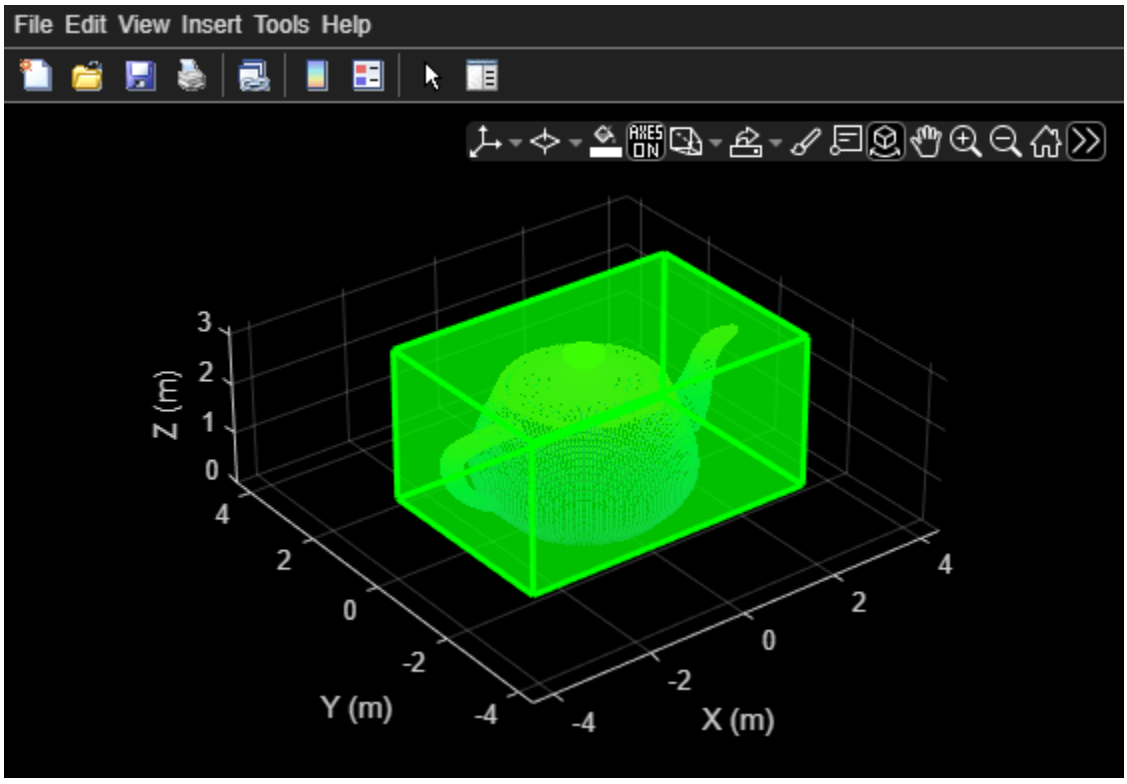
for i = 1:round((360/rot))
    % Rotate the point cloud.
    ptCloud = pctransform(ptCloud,tform);

    % Rotate the cuboid.
    cuboidPosition = bboxwarp(cuboidPosition,tform,ref);

    % Show the rotated point cloud data.
    view(player,ptCloud)

    % Show the rotated cuboid.
    showShape("cuboid",cuboidPosition, ...
        Parent=player.Axes, ...
        Color="green", ...
        Opacity=0.5)

    % Use drawnow to synchronize the point cloud and shape visualizations.
    drawnow
end
```



Input Arguments

`shape` — Type of shape

`"rectangle"` | `"line"` | `"polygon"` | `"circle"` | `"cuboid"` | `"projected-cuboid"` | `"ellipse"`

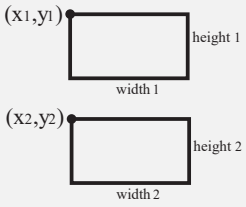
Type of shape, specified as `"rectangle"`, `"line"`, `"polygon"`, `"circle"`, `"cuboid"`, `"projected-cuboid"`, or `"ellipse"`.

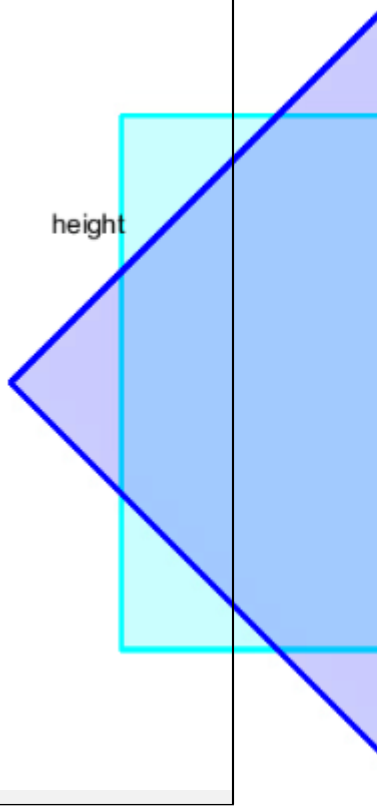
Data Types: `char`

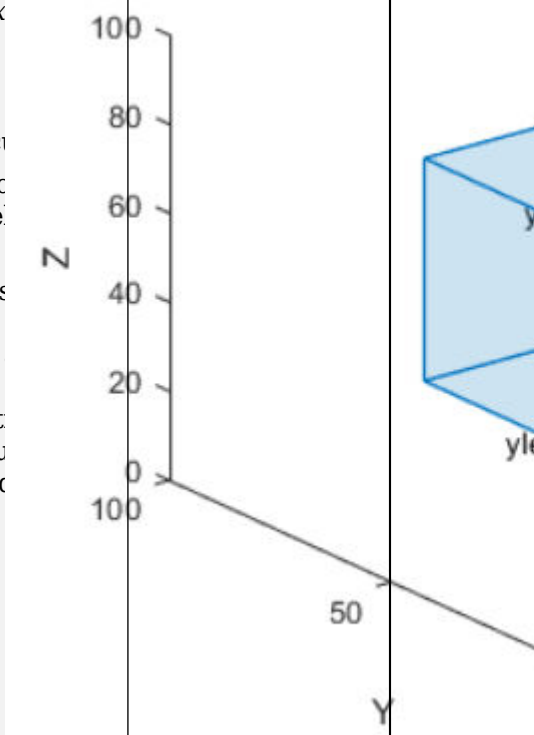
`position` — Positions and sizes of shapes

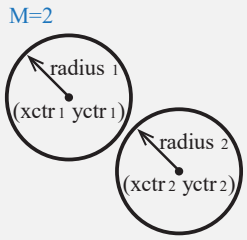
matrix | vector | cell array

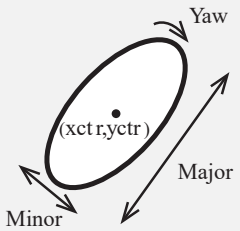
Position of shape, specified according to the type of shape, as described in the table.

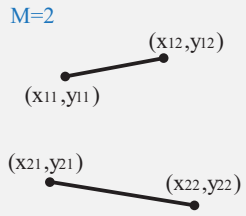
Shape	Position	Definition
`rectangle`	For one or more axis-aligned rectangles, specify as an M-by-4 numeric matrix, where each row specifies a rectangle of the form $[x\ y\ width\ height]$. • M is the number of axis-aligned rectangles. • x and y specify the upper-left corner of the rectangle. • w specifies the width of the rectangle, which is its length along the x-axis. • h specifies the height of the rectangle, which is its length along the y-axis. $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$	M=2 

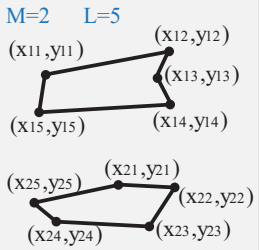
Shape	Position	Definition
	For one or more rotated rectangles, specify in spatial coordinates as an *M*-by-5 numeric matrix, where each row specifies a rotated rectangle of the form [*xctr yctr w h yaw*]. • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the rectangle.	

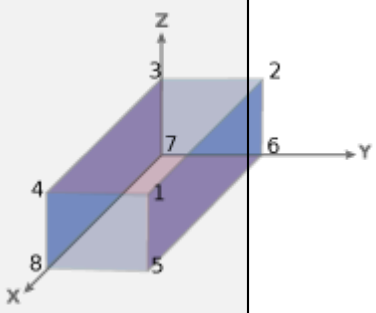
Shape	Position	Definition
`cuboid`	For one or more cuboids, specify in spatial coord an *M*-by-9 numeric matrix where each row speci cuboid of the form [*xctr yctr zctr xlen ylen zlen x* *zrot*], where • *M* is the number of cuboids. • *xctr*, *yctr*, and *zctr* specify the center of the c • *xlen*, *ylen*, and *zlen* specify the length of the c along the *x*-axis, *y*-axis, and *z*-axis, respective rotation. • *xrot*, *yrot*, and *zrot* specify the rotation angles cuboid around the *x*-axis, *y*-axis, and *z*-axis, respectively. The *xrot*, *yrot*, and *zrot* rotation are in degrees about the cuboid center. Each is clockwise-positive with respect to the posit direction of the associated spatial axis. The fu computes rotation matrices assuming ZYX orc angles [*xrot yrot zrot*].	

Shape	Position	Definition
`circle`	For one or more circles, specify spatial coordinates as an *M*-by-3 numeric matrix, where each row specifies a circle of the form [*xctr yctr radius*]. • *M* is the number of circles. • *xctr* and *yctr* specify the center of the circle. • *radius* specifies the radius of the circle. $\begin{bmatrix} xctr_1 & yctr_1 & radius_1 \\ xctr_2 & yctr_2 & radius_2 \\ \vdots & \vdots & \vdots \\ xctr_M & yctr_M & radius_M \end{bmatrix}$	

Shape	Position	Definition
`ellipse`	For one or more ellipses, specify in spatial coordinates as an *M*-by-5 matrix, where each row specifies an ellipse of the form [*xctr yctr major minor yaw*] • *M* is the number of ellipses. • *xctr* and *yctr* specify the center of the ellipse. • *major* and *minor* are the lengths of the major and minor ellipse axes, respectively. • *yaw* is the angle of rotation in degrees. The angle of rotation is clockwise positive around the center of the ellipse.	

Shape	Position	Definition
`line`	For one or more lines, specify spatial coordinates using one of these formats to specify a single line with connected endpoints, multiple lines with the same number of endpoints, or multiple lines with different numbers of endpoints. • Single lines with connected endpoints — Specify as a *P*-by-2 matrix, where each row specifies an endpoint in the form [*x y*], and the endpoint in one row connects directly to the endpoint in the next. *P* is the number of endpoints in the line. • Multiple lines with the same number of endpoints — Specify as an *M*-by-2*P* matrix, where each row specifies the consecutive endpoints of a single line in the form [*x1 y1 x2 y2 ... xp yp*]. Each [*x y*] pair is the coordinates of a single endpoint. *M* is the number of specified lines, and *P* is the number of endpoints per line. • Multiple lines with different numbers of endpoints — Specify as an *M*-by-1 cell array, where each cell contains the endpoints for a single line as a *P*-by-2 matrix or a 1-by-2*P* vector]. *M* is the number of specified lines, and *P* is the number of endpoints for each line	

Shape	Position	Definition
`polygon`	For one or more polygons, specify in spatial coordinates as an *M*-by-1 cell array, where each cell contains an *L*-by-2 matrix of [*x y*] vertex locations or a 1-by-2*L* vector of consecutive vertex locations of the form [x_1, y_1, x_2,y_2, … x_L,y_L]. • *M* is the number of polygons. • Each [*x y*] pair describes a vertex location or endpoint. • *L* is the number of vertices in a polygon or endpoints in a line. Each polygon or line can have a different number of vertices or endpoints.	

Shape	Position	Definition
`projected-cuboid`	For one or more projected cuboids, specify in spatial coordinates as an 8-by-2-by-*M* array or an *M*-by-8 matrix, where *M* is the number of projected cuboids. When specified as an 8-by-2-by-*M* array, each row must contain the [*x y*] location of a projected cuboid vertex. The vertices connect to form a cuboid with six faces. The order of the input vertices must match the order shown in the diagram. When specified as an *M*-by-8 matrix, each row specifies the dimensions of the front-facing and rear-facing sides of a projected cuboid in the form, [*x1 y1 w1 h1 x2 y2 w2 h2*], where [*x1 y1*] and [*x2 y2*] specify the upper-left coordinates of the front-facing and rear-facing sides, respectively, and [*w1 h1*] and [*w2 h2*] specify the corresponding widths and heights.	

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `showShape("rectangle",bboxes,Color="yellow")` sets the color for the shapes to yellow.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `showShape("rectangle",bboxes,"Color","yellow")` sets the color for the shapes to yellow.

`Label` — Shape label

`[]` (default) | scalar | *M*-element vector | *M*-element cell array of character vectors

Shape label, specified as a scalar, *M*-element vector, or an *M*-element cell array of character vectors, where *M* is the number of shapes.

If the input is a scalar value, then the function applies the label to each shape. If the input is an *M*-element vector or cell array, then the function applies the unique label to the corresponding shape, using the order in which you specify the shapes to the function.

`Color` — Shape color

`"green"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

Shape color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity

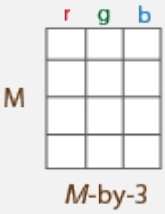
values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each shape or one color for all shapes. To specify one color for all markers, specify `Color` as a color name or an [*R G B*] vector.

Specification	Format	Example

Specify one color for all shapes (or markers)	Short color name or color name	`"r"`
		
		`"red"`

	RGB triplet	 `[1 0 0]`
Specify a color for each shape (or marker)	Vector of color names	`["red","yellow","blue"]`

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

This table lists the supported shape colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single` | `cell`

`Opacity` — Opacity of shape fill
`0` (default) | scalar in range [0 1] | *M*-element vector of values in range [0 1]

Opacity of the shape fill, specified as a scalar or *M*-element vector of numeric values in the range [0 1], where *M* is the number of shapes. To use the same opacity for all shapes, specify a scalar opacity value. For a fully opaque shape fill, specify `Opacity` as `1`.

`Parent` — Output axes
`gca` (default) | `Axes` graphics object

Output axes, specified as an `Axes` graphics object. For more information on Axes object properties, see Axes Properties.

`LineColor` — Shape border line color
`"auto"` (default) | one or more RGB triplets | one or more color names or short color names

Shape border line color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each shape or one color for all shapes. To specify one color for all markers, specify `LineColor` as a color name or an [*R G B*] vector.

`LineStripeColor` — Striped line color
`"none"` (default) | one or more RGB triplets | one or more color names or short color names

Striped line color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets. The striped line color alternates between the colors specified by `LineColor` and `LineStripeColor`.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each striped line or one color for all lines. To specify one color for all the lines, specify `LineStripeColor` as a color name or an [*R G B*] vector.

`LineWidth` — Border line width
`"auto"` (default) | positive scalar integer | *M*-element vector of positive integers

Border line width in pixels, specified as a positive integer or an *M*-element vector of positive integers, where *M* is the number of shapes. To use the same line width for all shapes, specify a scalar positive integer. Otherwise, specify an *M*-element vector of positive integers.

`LineOpacity` — Border line opacity of shape
`0` (default) | *M*-element vector of values in the range [0 1]

Border line opacity of shape, specified as a scalar or *M*-element vector of numeric values in the range [0 1], where *M* is the number of shapes. To use the same opacity for all shapes, specify a scalar opacity value. For a fully opaque shape fill, specify `Opacity` as `1`.

`LabelTextColor` — Label text color
`"black"` (default) | one or more RGB triplets | one or more color names or short color names

Label text color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each label or one color for all labels. To specify one color for all markers, specify `LabelTextColor` as a color name or an [*R G B*] vector.

`LabelOpacity` — Label opacity
`0` (default) | *M*-element vector of values in the range [0 1]

Label opacity of shape, specified as a scalar or *M*-element vector of numeric values in the range [0 1], where *M* is the number of shapes. To use the same opacity for all labels, specify a scalar opacity value. For a fully opaque shape fill, specify `Opacity` as `1`.

`LabelFont` — Label font
`"Helvetica"` (default) | `"FixedWidth"` | supported font

Label font, specified as `"FixedWidth"` or a font that your system supports. To display and print text properly, you must select a font that your system supports. If you select an unsupported font, the function returns an error. If you specify `"FixedWidth"`, the function uses the font specified by the `FixedWidthFont` property of the Root Properties object. The fixed-width font relies on the value of `get(0,"FixedWidthFontName")`.

`LabelFontSize` — Label font size
`12` (default) | positive scalar

Label font size in points, specified as a positive scalar.

`ShowOrientation` — Original orientation indicator
`false` or `0` (default) | `true` or `1`

Original orientation indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). A value of `1` (`true`) displays the original orientation of the rotated rectangle by using an arrow annotation on the bounding box. The original orientation is defined as a zero rotation angle. A value of `0` (`false`) does not show the arrow annotation.

This argument applies only when `Shape` describes a rotated rectangle or an ellipse.

Data Types: `logical` | `integer`

Version History

Introduced in R2020b

R2026a: Added `LineStripeColor` name-value argument

Added the `LineStripeColor` name-value argument to specify a color for a striped line.

R2024b: Support for ellipse shape

Added support for `"ellipse"` to visualize one or more ellipses on top of an image or on video data.

R2024b: Default value for orientation indicator changed from `true` to `false`
Behavior changed in R2024b

Starting in R2024b, the default value for the `ShowOrientation` name-value argument will be `false`.

R2022a: Bounding Box Coordinates: Data augmentation for object detection using spatial coordinates
Behavior changed in R2022a

The `bboxresize`, `bboxcrop`, `bboxwarp`, and `showShape` functions assume the input bounding box coordinates for axis-aligned rectangles are specified in spatial coordinates and return the transformed bounding boxes in spatial coordinates.

R2022b: Visualize projected cuboids

Added support to visualize projected cuboids.

R2023b: Rotated rectangle support for bounding box

You can specify a rotated rectangle bounding box using the form [*xctr yctr width height yaw*].

See Also

Functions
`insertObjectAnnotation` | `insertObjectMask` | `insertMarker` | `insertText` | `insertShape` | `pcplayer` | `pcshow` | `imshow` | `cuboid2img`

Topics
“Draw Markers on an Image” on page 3-874
“Insert Text into Image” on page 3-919
“Choose Function to Visualize Detected Objects”

pcregistercorr

Register two point clouds using phase correlation

Syntax

```
tform = pcregistercorr(moving,fixed,gridSize,gridStep)
[tform,rmse] = pcregistercorr( ___ )
[tform,rmse,peak] = pcregistercorr( ___ )
[ ___ ] = pcregistercorr( ___ ,Name=Value)
```

Description

`tform = pcregistercorr(moving,fixed,gridSize,gridStep)` computes the rigid transformation that registers the moving point cloud `moving`, to the fixed point cloud `fixed`, using an image-based phase correlation algorithm.

The function performs registration by first converting both point clouds to a 2-D occupancy grid in the *X*-*Y* plane with center at the origin (0,0,0). The occupancy of each grid cell is determined using the *Z*-coordinate values of points within the grid.

`[tform,rmse] = pcregistercorr( ___ )` additionally returns the root mean square error of the Euclidean distance between the aligned point clouds.

`[tform,rmse,peak] = pcregistercorr( ___ )` additionally returns the peak correlation value of the phase difference between the two occupancy grids.

`[ ___ ] = pcregistercorr( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Window=false` sets the `Window` name-value argument to `false` to suppress using windowing.

Examples

Register Lidar Moving Point Cloud to Fixed Point Cloud

Read data from a Velodyne® packet capture (PCAP) file into the workspace.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read a fixed and a moving point cloud from frames of the lidar data.

```
frameNumber = 1;
skipFrame   = 5;
fixed  = readFrame(veloReader,frameNumber);
moving = readFrame(veloReader,frameNumber + skipFrame);
```

Find the ground planes for both the moving and the fixed point clouds. Set the maximum distance in meters.

```
maxDistance = 0.4;
referenceVector = [0 0 1];
```

```
groundMoving = pcfitplane(moving,maxDistance,referenceVector);
groundFixed = pcfitplane(fixed,maxDistance,referenceVector);
```

Transform the point clouds so that their ground planes are parallel to the *X*-*Y* plane.

```
tformMoving = normalRotation(groundMoving,referenceVector);
tformFixed = normalRotation(groundFixed,referenceVector);

movingCorrected = pctransform(moving,tformMoving);
fixedCorrected = pctransform(fixed,tformFixed);
```

Register the moving point cloud against the fixed point cloud. Set the occupancy grid size to 100-by-100 meters and the size of each grid cell to 0.5-by-0.5 meters.

```
gridSize = 100;
gridStep = 0.5;

tform = pcregistercorr(movingCorrected,fixedCorrected,gridSize,gridStep);
```

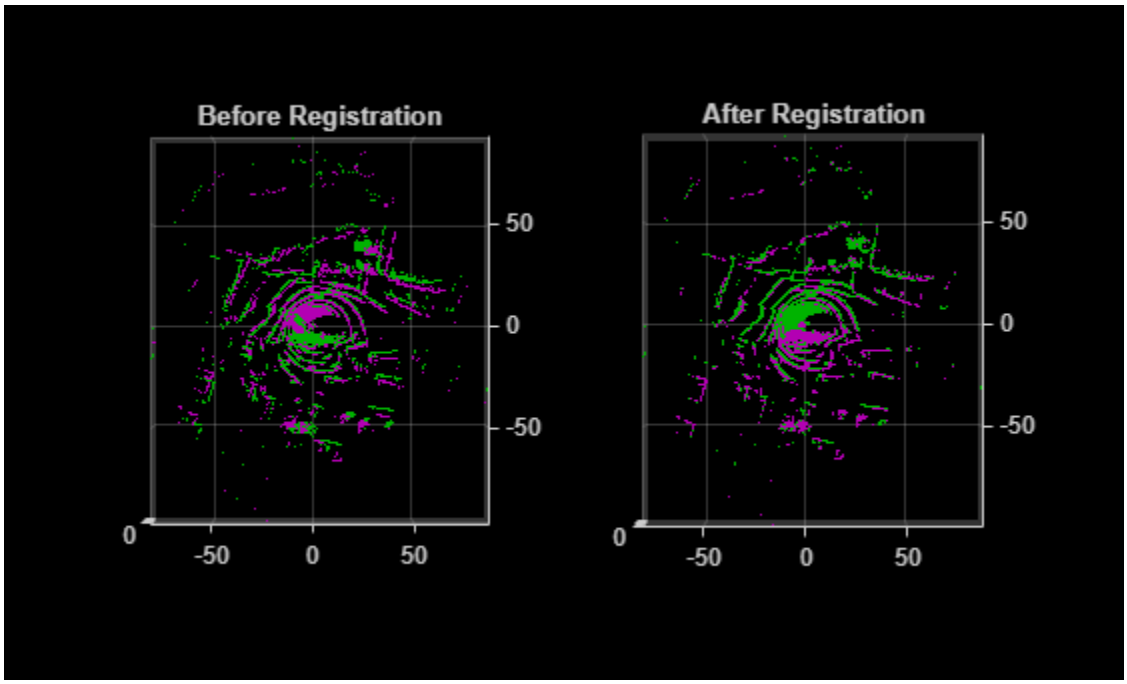
Transform the moving point cloud using the estimated rigid transformation.

```
combinedTform = rigidtform3d(tformFixed.A * tformMoving.A * tform.A);
movingReg = pctransform(moving,combinedTform);
```

Visualize the registration.

```
figure
subplot(121)
pcshowpair(moving,fixed)
title('Before Registration')
view(2)

subplot(122)
pcshowpair(movingReg,fixed)
title('After Registration')
view(2)
```



Input Arguments

`moving` — Moving point cloud
`pointCloud` object

Moving point cloud, specified as a `pointCloud` object.

`fixed` — Fixed point cloud
`pointCloud` object

Fixed point cloud, specified as a `pointCloud` object.

`gridSize` — Size of square occupancy grid
scalar

Size of square occupancy grid, specified as a scalar value in world units. The occupancy grid has both width and height equal to this value. The occupancy grid is centered at the origin (0, 0, 0) and the spatial extent of the occupancy grid is [-`gridSize`/2, `gridSize`/2].

`gridStep` — Size of each grid cell
scalar

Size of each grid cell, specified as a scalar value in world units.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Zlimit=[0 3]` sets the *Z*-axis lower limit to `0` and the upper limit to `3`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Zlimit` — *Z*-axis limits
`[0 3]` (default) | vector of form [*zmin zmax*]

Z-axis limits to compute the occupancy of a grid cell, specified as a vector of the form [*zmin zmax*], where *zmin* and *zmax* are numeric scalars. The function scales points with a *Z*-axis value from *zmin* to *zmax* to probabilities in the range [0, 1]. Values less than *zmin* are assigned an occupancy value of `0`. Values greater than *zmax* are assigned an occupancy value of `1`.

`Window` — Logical to use windowing
`1` (`true`) (default) | `0` (`false`)

Logical to use windowing to suppress spectral leakage effects in the frequency domain, specified as a numeric or logical `0` (`false`) or `1` (`true`). When you set `Window` to `true`, the function uses a Blackman window to increase the stability of registration results. If the common features that will be aligned in the occupancy grids are oriented along the edges, then setting `Window` to `false` could provide superior registration results.

Output Arguments

`tform` — Rigid transformation
`rigidtform3d` object

Rigid transformation, returned as a `rigidtform3d` object. The `rigidtform3d` object describes the rigid 3-D transformation that registers the moving point cloud to the fixed point cloud.

`rmse` — Root mean square error
positive scalar

Root mean square error, returned as a positive numeric scalar that represents the Euclidean distance between the aligned points. The aligned points is a result of the moving point cloud transformed by the `tform` output and the fixed point cloud. For each point in the fixed point cloud, the algorithm finds the closest point in the transformed moving point cloud, then computes the Euclidean distance between the points, and then computes the `rmse`. A low `rmse` value indicates a more accurate registration. The `rmse` calculation is:

```
rmse = sqrt(sum(distance.^2,'all')/numel(distance));
```

peak — Peak correlation value
scalar value

Peak correlation value of the phase difference between the two occupancy grids, returned as a scalar value. A peak value less than `0.03` indicates a poor registration result.

Tips

- The phase correlation method is best used to register point clouds when the transformation can be described by a translation in the *X*-*Y* plane and a rotation around the *Z*-axis. For example, a ground vehicle with a horizontally mounted lidar moving on a flat surface.
- The phase correlation algorithm expects motion to be exclusively along the *X*-*Y* plane, as with the ground plane. If motion is not exactly in the *X*-*Y* plane, you can use the `normalRotation` function

to transform the point clouds. For example, in vehicular motion, you can reduce the effects of vehicle suspension or surface features such as potholes and speed bumps by using the `normalRotation` function.

- Increasing the size of the occupancy grid increases the computational demands of this function. You can control the size of the occupancy grid by modifying the `gridSize` and `gridStep` arguments.
- If you obtain poor registration results and the `peak` correlation value is less than `0.03`, try setting the `Window` argument to `false`.

Version History

Introduced in R2020b

R2022b: Returns `rigidtform3d` object

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `pcregistercorr` function now returns `tform` as a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `tform` as a `rigid3d` object, which uses the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2023b: `pcregistercorr` supports GPU code generation

The `pcregistercorr` function supports CUDA code generation for NVIDIA GPUs. You must have MATLAB Coder and GPU Coder to generate CUDA code.

References

[1] Dimitrievski, Martin, David Van Hamme, Peter Veelaert, and Wilfried Philips. “Robust Matching of Occupancy Maps for Odometry in Autonomous Vehicles.” In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 626–633. Rome, Italy: SCITEPRESS - Science and Technology Publications, 2016.

[2] Reddy, B.S., and B.N. Chatterji. “An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration.” *IEEE Transactions on Image Processing* 5, no. 8 (August 1996): 1266–71. https://doi.org/10.1109/83.506761.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`pcregisterndt` | `pcregistericp` | `pcregistercpd` | `pctransform` | `pcshow` | `pcshowpair` | `pcdownsample` | `pcfitplane` | `imregcorr` | `normalRotation` | `pcdenoise` | `pcmerge`

Objects
`pointCloud` | `rigidtform3d`

Topics
“3-D Point Cloud Registration and Stitching”
“Implement Point Cloud SLAM in MATLAB”

scanContextDistance

Distance between scan context descriptors

Syntax

```
dist = scanContextDistance(descriptor1,descriptor2)
```

Description

`dist = scanContextDistance(descriptor1,descriptor2)` computes the normalized distance between scan context descriptors. Determine loop closure candidates using the computed distance.

A scan context descriptor is a 2-D global feature descriptor for a point cloud that can be used to detect loop closures. To extract a scan context descriptor from a point cloud, use the `scanContextDescriptor` function.

Examples

Extract and Compare Scan Context Descriptors of Point Clouds

Create a Velodyne® packet capture (PCAP) file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the 1st, 2nd, and 30th scan into the workspace.

```
ptCloud1 = readFrame(veloReader,1);
ptCloud2 = readFrame(veloReader,2);
ptCloud30 = readFrame(veloReader,30);
```

Extract scan context descriptors from each of the point clouds.

```
descriptor1 = scanContextDescriptor(ptCloud1);
descriptor2 = scanContextDescriptor(ptCloud2);
descriptor30 = scanContextDescriptor(ptCloud30);
```

Compute the descriptor distance between the 1st and 2nd scan context descriptors, and between the 1st and 30th scan context descriptors.

```
dist1to2 = scanContextDistance(descriptor1,descriptor2);
dist1to30 = scanContextDistance(descriptor1,descriptor30);
```

Display the scan context descriptor distances.

```
disp("Descriptor distance from frame 1 to 2:  " + num2str(dist1to2))
```

```
Descriptor distance from frame 1 to 2:  0.087647
```

```
disp("Descriptor distance from frame 1 to 30: " + num2str(dist1to30))
```

```
Descriptor distance from frame 1 to 30: 0.31551
```

Input Arguments

`descriptor1` — Scan context descriptor of point cloud

M-by-*N* matrix

Scan context descriptor of point cloud, specified as an *M*-by-*N* matrix. To ensure an accurate distance measurement, use the same radial range when extracting this descriptor as for `descriptor2`.

Data Types: `single` | `double`

`descriptor2` — Scan context descriptor of point cloud

M-by-*N* matrix

Scan context descriptor of point cloud, specified as an *M*-by-*N* matrix. To ensure an accurate distance measurement, use the same radial range when extracting this descriptor as for `descriptor1`.

Data Types: `single` | `double`

Output Arguments

`dist` — Distance between scan context descriptors

scalar in the range [0,1)

Distance between the scan context descriptors, returned as a scalar in the range [0,1). A small value indicates that the scan contexts are likely to belong to the same location, and are loop closure candidates. The function returns `dist` as a `single` data type when either `descriptor1` or `descriptor2` inputs is of `single` data type. Otherwise, the function returns `dist` as a `double` data type.

Data Types: `single` | `double`

Algorithms

- The function computes the distance between scan context descriptors using a modified cosine distance and normalizes it to the range [`0` to `1`).

Version History

Introduced in R2020b

References

[1] G. Kim and A. Kim, "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4802-4809, doi: 10.1109/IROS.2018.8593953.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pointCloud`

Functions
`scanContextDescriptor`

Topics
“Implement Point Cloud SLAM in MATLAB”

scanContextDescriptor

Extract scan context descriptor from point cloud

Syntax

```
descriptor = scanContextDescriptor(ptCloud)
descriptor = scanContextDescriptor(ptCloud,Name,Value)
```

Description

`descriptor = scanContextDescriptor(ptCloud)` extracts a scan context descriptor from a point cloud, `ptCloud`.

A scan context descriptor is a 2-D global feature descriptor of a point cloud that can be used to detect loop closures. The function computes the descriptor by first binning points from a 3-D point cloud scan into concentric radial and azimuthal bins, and then selecting the maximum *z*-height of points in each bin.

`descriptor = scanContextDescriptor(ptCloud,Name,Value)` specifies options using one or more name-value pair arguments.

Examples

Extract and Compare Scan Context Descriptors of Point Clouds

Create a Velodyne® packet capture (PCAP) file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read the 1st, 2nd, and 30th scan into the workspace.

```
ptCloud1 = readFrame(veloReader,1);
ptCloud2 = readFrame(veloReader,2);
ptCloud30 = readFrame(veloReader,30);
```

Extract scan context descriptors from each of the point clouds.

```
descriptor1 = scanContextDescriptor(ptCloud1);
descriptor2 = scanContextDescriptor(ptCloud2);
descriptor30 = scanContextDescriptor(ptCloud30);
```

Compute the descriptor distance between the 1st and 2nd scan context descriptors, and between the 1st and 30th scan context descriptors.

```
dist1to2 = scanContextDistance(descriptor1,descriptor2);
dist1to30 = scanContextDistance(descriptor1,descriptor30);
```

Display the scan context descriptor distances.

```
disp("Descriptor distance from frame 1 to 2:  " + num2str(dist1to2))
```

```
Descriptor distance from frame 1 to 2:  0.087647

disp("Descriptor distance from frame 1 to 30: " + num2str(dist1to30))

Descriptor distance from frame 1 to 30: 0.31551
```

Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as `pointCloud` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumBins',[20 60]` sets the number of radial bins to `20`, and the number of azimuthal bins to `60`.

`NumBins` — Number of concentric radial and angular bins

`[20 60]` (default) | vector of form [*numRadialBins numAzimuthalBins*]

Number of concentric radial and angular bins, specified as the comma-separated pair consisting of `'NumBins'` and a vector of the form [*numRadialBins numAzimuthalBins*], where *numRadialBins* is a positive integer that specifies the number of concentric radial bins and *numAzimuthalBins* specifies the number of concentric angular bins. To extract more compact descriptors, you can decrease the number of bins, but this reduces the discriminative power.

`MinPointsPerBin` — Minimum number of points for a bin to be included in descriptor

`5` (default) | scalar

Minimum number of points for a bin to be included in descriptor, specified as the comma-separated pair consisting of `'MinPointsPerBin'` and a positive integer. Bins that do not contain the minimum number of points are not included in the descriptor. For these bins, the function returns `NaN`. To reduce the chances for noise to be included as a bin, increase this value.

`SensorOrigin` — Sensor origin

`[0 0]` (default) | two-element vector

Sensor origin, specified as the comma-separated pair consisting of `'SensorOrigin'` and a two-element vector. The elements set the *x*- and *y*-axis coordinate points, respectively, of the sensor origin in world units. The function centers the bins around the origin of the scan.

`RadialRange` — Radial bounds of descriptor

full spatial extent of point cloud (default) | two-element vector

Radial bounds of the descriptor used when binning points, specified as the comma-separated pair consisting of `'RadialRange'` and a two-element vector in the form [*rmin*,*rmax*]. Use this name-value argument to restrict the spatial bounds covered by the descriptor.

Output Arguments

`descriptor` — Scan context descriptor

M-by-*N* matrix

Scan context descriptor, returned as an *M*-by-*N* matrix, where *M* and *N* are the number of concentric radial and angular bins, respectively, in the descriptor. These are specified by the `NumBins` property.

The function returns inherits the data type of the `Location` property of the input point cloud.

Data Types: `single` | `double`

Tips

- The scan context descriptor function assumes that the sensor is mounted roughly horizontally and that the input point cloud is in the sensor coordinate system.
- To determine loop closure candidates, compare the distance between two scan context descriptors using the `scanContextDistance` function.

Version History

Introduced in R2020b

References

[1] G. Kim and A. Kim, "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4802-4809, doi: 10.1109/IROS.2018.8593953.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pointCloud`

Functions

`scanContextDistance`

Topics

“Implement Point Cloud SLAM in MATLAB”

pcalign

Align array of point clouds

Syntax

```
ptCloudOut = pcalign(ptClouds,tforms)
ptCloudOut = pcalign(ptClouds,tforms,gridStep)
ptCloudOut = pcalign( ___ ,GridFilter=gridFilterMethod)
```

Description

`ptCloudOut = pcalign(ptClouds,tforms)` aligns an array of point clouds, `ptClouds`, into one point cloud by using the specified transformations `tforms`.

`ptCloudOut = pcalign(ptClouds,tforms,gridStep)` applies a box grid filter with 3-D boxes of size specified as `gridStep`, to the aligned point cloud. The grid filter returns a single point per 3-D box.

`ptCloudOut = pcalign( ___ ,GridFilter=gridFilterMethod)` sets the method to downsample the aligned point cloud.

Examples

Align Point Clouds from View Set

Load a point cloud view set into the workspace.

```
data = load('vSetPointClouds.mat');
vSet = data.vSet;
```

Extract the point clouds and their absolute poses from the view set.

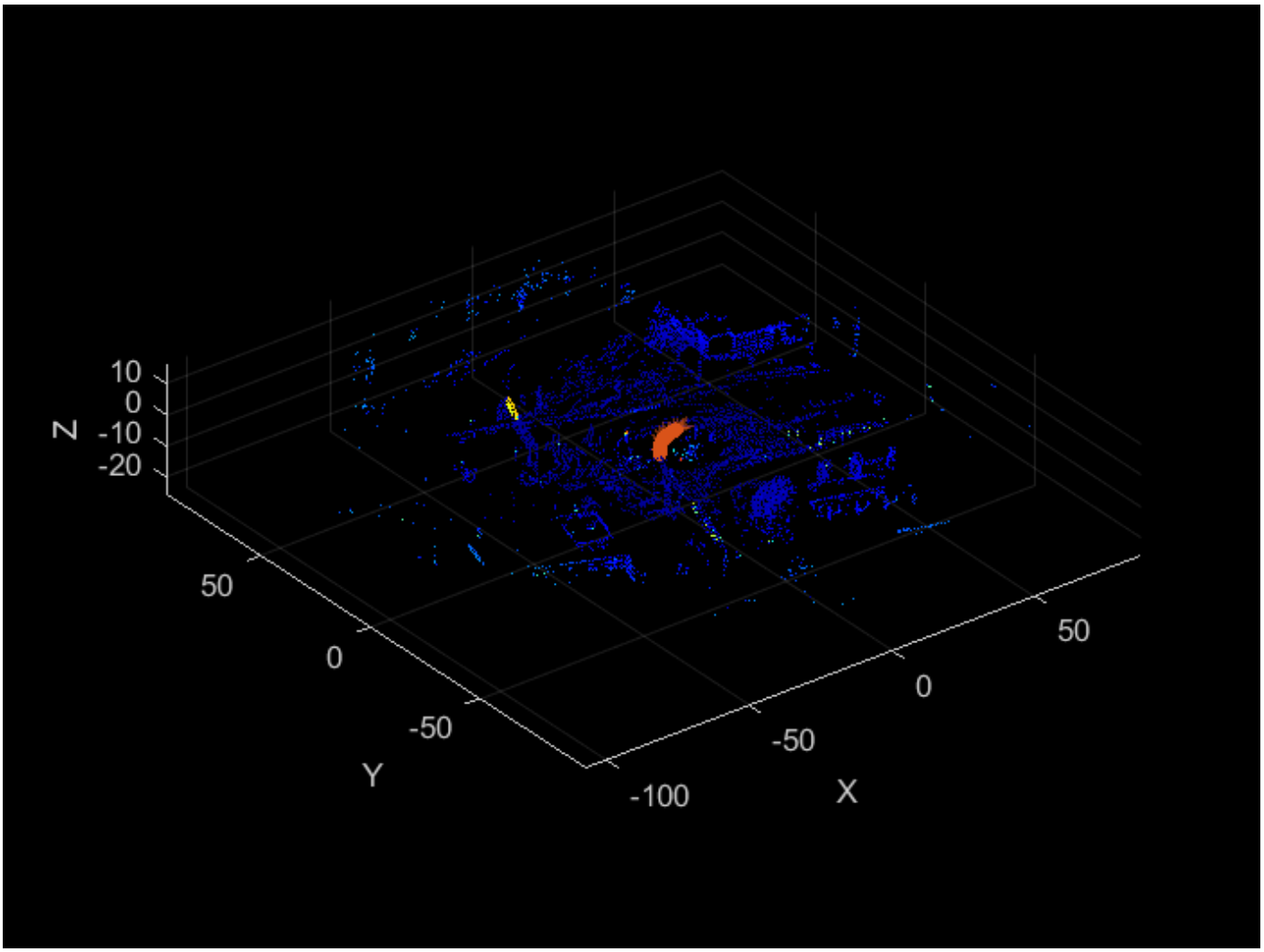
```
ptClouds = vSet.Views.PointCloud;
tforms   = vSet.Views.AbsolutePose;
```

Align the point clouds using their absolute poses, and apply a 3-D box filter of size 1 to the aligned point cloud.

```
gridStep = 1;
ptCloudMap = pcalign(ptClouds,tforms,gridStep);
```

Display the aligned point cloud map.

```
pcshow(ptCloudMap)
hold on
plot(vSet)
```



Input Arguments

`ptClouds` — Point clouds
`pointCloud` array

Point clouds, specified as a `pointCloud` array.

`tforms` — 3-D rigid or affine geometric transformations
M-by-1 `rigidtform3d` array | *M*-by-1 `affinetform3d` array

3-D rigid or affine geometric transformations, specified as an *M*-by-1 array of `rigidtform3d` objects or an *M*-by-1 array of `affinetform3d` objects. The `tforms` argument must contain the same number of elements as the `ptClouds` argument.

`gridStep` — Size of 3-D box for grid filter
positive scalar

Size of the 3-D box for the grid filter, specified as a positive scalar. Increasing the value of `gridStep` can reduce memory usage by merging more points, but it can also reduce the amount of detail captured. The grid filter returns a single point per 3-D box.

Data Types: `single` | `double`

`gridFilterMethod` — Method to downsample aligned point cloud
`"average"` (default) | `"nearest"`

Method to downsample the aligned point cloud, specified as one of:

- `"average"` (default) — Selects a point in the voxel by averaging the location, color, intensity, and normal of all points within the voxel.
- `"nearest"` — Selects the voxel's location, color, intensity, and normal based on the point nearest to the centroid of the voxel.

Output Arguments

`ptCloudOut` — Aligned point cloud
`pointCloud` object

Aligned point cloud, returned as a `pointCloud` object.

Tips

- If the input point clouds do not all have an assigned value for a property, the function does not assign a value for that property in the returned point cloud. For example, if some of the input point clouds have values for the `Color` property but another one does not, then the function does not return a value for the `Color` property of `ptCloudOut`.
- The function filters out points with `NaN` or `Inf` values.

Version History

Introduced in R2020b

R2022b: Supports new geometric transformation objects

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify `tforms` as an array of `rigidtform3d` or `affinetform3d` objects, which use the premultiply convention.

Although you can still specify `tforms` as an array of `rigid3d` or `affine3d` objects, these objects are not recommended because they use the postmultiply convention. You can streamline your geometric transformation workflows by switching to the new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`pointCloud` | `planeModel` | `rigidtform3d` | `affinetform3d` | `pcviewset`

Functions
`pcplayer` | `pcshow` | `pcwrite` | `pcread` | `pcmerge` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp` | `pccat`

Topics
“3-D Point Cloud Registration and Stitching”

pccat

Concatenate 3-D point cloud array

Syntax

```
ptCloudOut = pccat(ptClouds)
```

Description

`ptCloudOut = pccat(ptClouds)` concatenates a 3-D point cloud array, `ptClouds`. `ptCloudOut` contains the `Location`, `Color`, `Normal`, and `Intensity` property values of all the concatenated point clouds.

Examples

Combine Sequence of Point Clouds

Read a sequence of aligned point clouds.

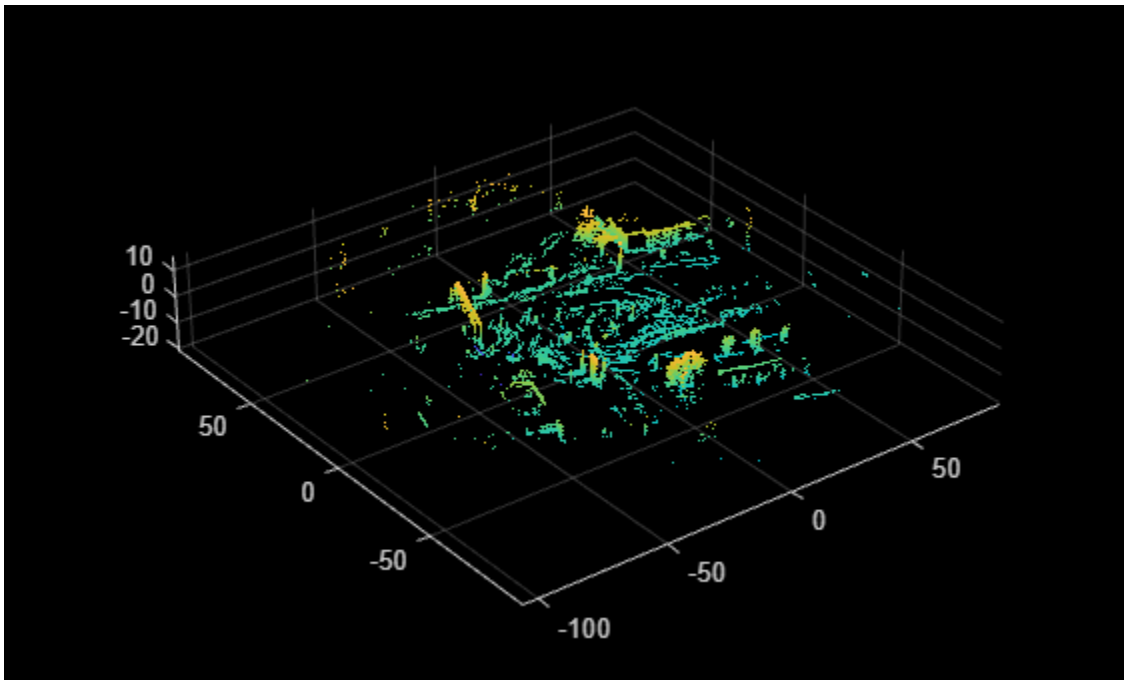
```
filePath = fullfile(toolboxdir('vision'),'visiondata','pcdmapseq');
fileDS = fileDatastore(filePath,'ReadFcn',@pcread,'UniformRead',true);
ptClouds = readall(fileDS);
```

Concatenate point clouds into a single point cloud.

```
ptCloudOut = pccat(ptClouds);
```

Display concatenated point cloud.

```
pcshow(ptCloudOut)
```



Input Arguments

`ptClouds` — Point clouds

M-by-1 `pointCloud` array

Point cloud objects, specified as an *M*-by-1 `pointCloud` array.

Output Arguments

`ptCloudOut` — Concatenated point cloud

`pointCloud` object

Concatenated point cloud, returned as a `pointCloud` object. `ptCloudOut` contains the `Location`, `Color`, `Normal`, and `Intensity` property values of all the concatenated point clouds.

Tips

- If the input point clouds do not all have an assigned value for a property, the function does not assign a value for that property in the returned point cloud. For example, if some of the input point clouds have values for the `Color` property but another one does not, then the function does not return a value for the `Color` property of `ptCloudOut`.
- The function filters out points with `NaN` or `Inf` values.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- All point clouds in the input point cloud array must be either organized or unorganized point clouds.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pointCloud`

Functions

`pcalign` | `pcmerge`

Topics

“3-D Point Cloud Registration and Stitching”

readAprilTag

Detect and estimate pose for AprilTag in image

Syntax

```
[id,loc] = readAprilTag(I)
[id,loc] = readAprilTag(I,tagFamily)
[___,pose] = readAprilTag(___,intrinsics,tagSize)
[___,detectedFamily] = readAprilTag(___)
[___] = readAprilTag(___,Name=Value)
```

Description

`[id,loc] = readAprilTag(I)` detects AprilTags in the input image `I` and returns the locations and IDs associated with the tags.

`[id,loc] = readAprilTag(I,tagFamily)` detects AprilTags of only the specified families, `tagFamily`.

`[___,pose] = readAprilTag(___,intrinsics,tagSize)` returns the poses of the tags with respect to the specified camera intrinsic parameters, `intrinsics` and size of the tags `tagSize` in addition to any combination of arguments in previous syntaxes.

`[___,detectedFamily] = readAprilTag(___)` returns the recognized AprilTag families, `detectedFamily`.

`[___] = readAprilTag(___,Name=Value)` specifies options using one or more name-value arguments. For example, `GaussianSigma=2`, sets the standard deviation to 2 for the 2-D Gaussian smoothing kernel.

Examples

Detect AprilTags in Image

Read an image into the workspace.

```
I = imread("aprilTagsMulti.jpg");
```

Specify the AprilTag formats to search in the image.

```
tagFamily = ["tag36h11","tagCircle21h7","tagCircle49h12","tagCustom48h12","tagStandard41h12"];
```

Detect tags and tag locations in the image, and read the tag IDs.

```
[id,loc,detectedFamily] = readAprilTag(I,tagFamily);

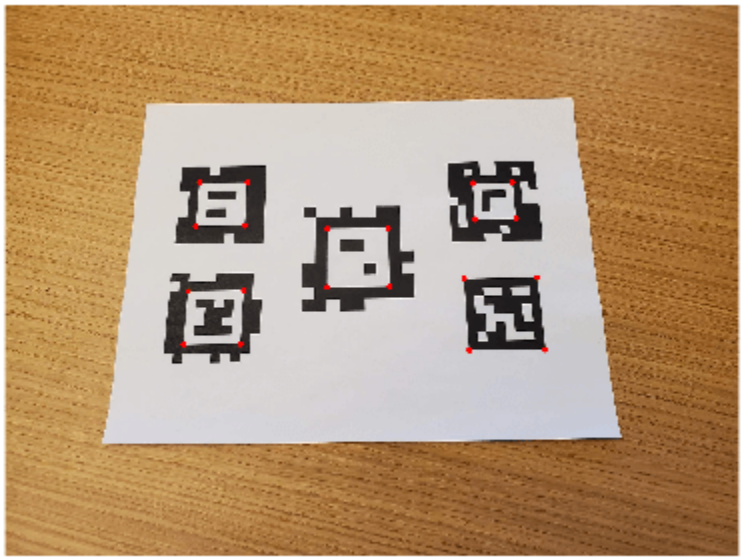
for idx = 1:length(id)
        % Display the ID and tag family
        disp("Detected Tag ID, Family: " + id(idx) + ", " ...
            + detectedFamily(idx));
```

```
        % Insert markers to indicate the locations
        markerRadius = 8;
        numCorners = size(loc,1);
        markerPosition = [loc(:,:,idx),repmat(markerRadius,numCorners,1)];
        I = insertShape(I,"FilledCircle",markerPosition,ShapeColor="red",Opacity=1);
end
```

```
Detected Tag ID, Family: 30, tag36h11
Detected Tag ID, Family: 32, tagCircle21h7
Detected Tag ID, Family: 98, tagStandard41h12
Detected Tag ID, Family: 195, tagCustom48h12
Detected Tag ID, Family: 884, tagCircle49h12
```

Display the image with markers to indicate the corner locations of the detected tags.

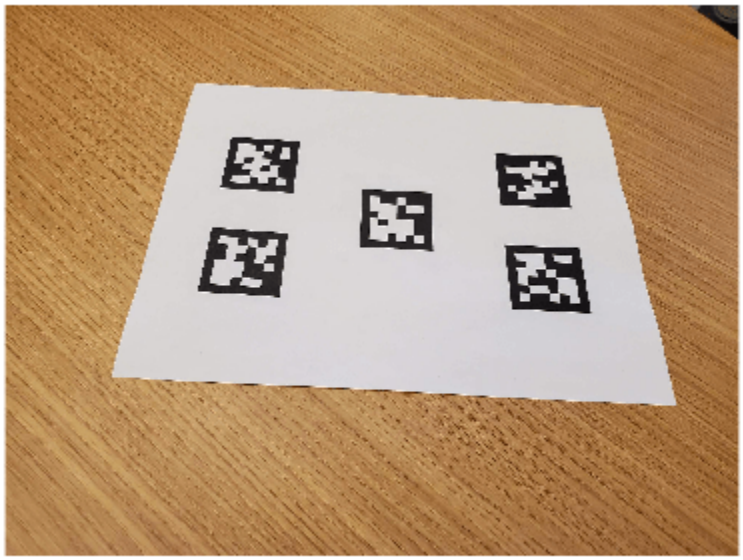
```
imshow(I)
```



Estimate AprilTag Poses in Image

Read an image that contains AprilTags into the workspace. Display the image.

```
I = imread("aprilTag36h11.jpg");
imshow(I)
```



Load the camera intrinsic parameters.

```
data = load("camIntrinsicsAprilTag.mat");
intrinsics = data.intrinsics;
```

Specify the tag size in meters.

```
tagSize = 0.04;
```

Undistort the input image using the camera intrinsic parameters.

```
I = undistortImage(I,intrinsics,OutputView="same");
```

Detect a specific family of AprilTags and estimate the tag poses.

```
[id,loc,pose] = readAprilTag(I,"tag36h11",intrinsics,tagSize);
```

Set the origin for the axes vectors and for the tag frames.

```
worldPoints = [0 0 0; tagSize/2 0 0; 0 tagSize/2 0; 0 0 tagSize/2];
```

Add the tag frames and IDs to the image.

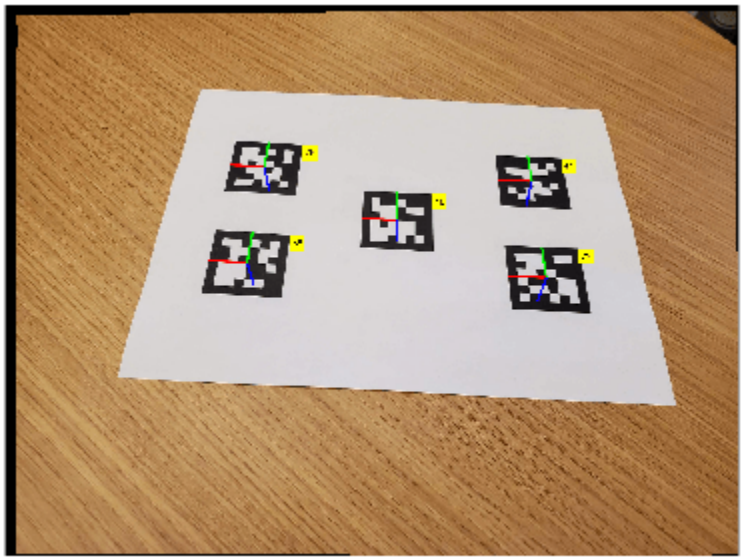
```
for i = 1:length(pose)
    % Get image coordinates for axes.
    imagePoints = world2img(worldPoints,pose(i),intrinsics);

    % Draw colored axes.
    I = insertShape(I,Line=[imagePoints(1,:) imagePoints(2,:); ...
        imagePoints(1,:) imagePoints(3,:); imagePoints(1,:) imagePoints(4,:)], ...
        ShapeColor=["red","green","blue"],LineWidth=7);

    I = insertText(I,loc(1,:,i),id(i),BoxOpacity=1,FontSize=25);
end
```

Display the annotated image.

```
imshow(I)
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* grayscale image.

`tagFamily` — AprilTag families

one or more valid AprilTag families

AprilTag families, specified as one or more of the valid AprilTag families listed in this table. Specifying a family can reduce the run time of the function by restricting the search.

AprilTag Family
`"tag16h5"`
`"tag25h9"`
`"tag36h11"`
`"tagCircle21h7"`
`"tagCircle49h12"`
`"tagCustom48h12"`
`"tagStandard41h12"`
`"tagStandard52h13"`

Data Types: `char` | `string` | `cell`

`intrinsics` — Camera intrinsic parameters
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

`tagSize` — Size of tag
positive scalar

Size of the tags in world units (such as millimeters), specified as a positive scalar. The function defines the size of the tags as the length between two adjacent corner outer black edges.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `readAprilTag(I,GaussianSigma=2)` sets the standard deviation to 2 for the 2-D Gaussian smoothing kernel.

`GaussianSigma` — Standard deviation for 2-D Gaussian smoothing kernel
`0` (default) | non-negative scalar

Standard deviation for 2-D Gaussian smoothing kernel, specified as a non-negative scalar. The Gaussian kernel smooths the input image to decrease white Gaussian noise. Increase this value to increase the speed of processing, but at the cost of precision.

`DecimationFactor` — Decimation factor
2 (default) | positive integer

Decimation factor, specified as a positive integer. The decimation factor controls the scale of the downsampling. Increase this value to increase the detection rate, while decreasing detection distance.

Output Arguments

`id` — Tag IDs
vector of positive integers

Tag IDs, returned as a vector of positive integers.

`loc` — Locations of tags in image
4-by-2-by-*N* array

Locations of tags in image, returned as a 4-by-2-by-*N* array. The array contains the (*x*,*y*) locations for each of the four corners for *N* tags.

`pose` — Poses of tags
array of `rigidtform3d` objects

Poses of tags with respect to the camera, returned as an array of `rigidtform3d` objects. Each object encapsulates the 3-D rigid transformation of a tag in the same world units as the `tagSize` input

argument. The origin of each tag frame is located at the center of the corresponding tag. The *X*-*Y* plane of the tag frame is defined by the planar surface on which the AprilTag sits with the *Z*-axis pointing into the tag.

`detectedFamily` — Detected tag families
vector of strings

Detected tag families, returned as a vector of strings.

Tips

- For applications that require real-time performance, while also minimizing false-positive detections, consider using the `"tag36h11"` family tag.
- For applications that require a faster detection time as opposed to the number of supported IDs, use the `"tagStandard41h12"` family tag.
- For pregenerated tags for all supported tag families, see Pregenerated AprilTag Images on GitHub.
- The `readAprilTag` function cannot read AprilTags containing 2 border bits.

Version History

Introduced in R2020b

R2022b: Returns poses as `rigidtform3d` objects
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `readAprilTag` function now returns `pose` as an array of `rigidtform3d` objects, which use the premultiply convention. Before, the function returned `pose` as an array of `rigid3d` objects, which use the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

- Generates code only on host platforms.
- You must specify the input argument `tagFamily` as a character vector or an array of character vectors. String inputs are not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectAprilGridPoints` | `readBarcode` | `patternWorldPoints` | `rigidtform3d` | `cameraIntrinsics` | `world2img` | `insertText` | `insertShape` | `insertMarker`

Topics
“Camera Calibration Using Custom Planar Calibration Patterns”
“Landmark SLAM Using AprilTag Markers” (Navigation Toolbox)
“Calibration Patterns”

evaluateDetectionAOS

(To be removed) Evaluate average orientation similarity metric for object detection

Note `evaluateDetectionAOS` will be removed in a future release. Use `evaluateObjectDetection` instead.

Syntax

```
metrics = evaluateDetectionAOS(detectionResults,groundTruthData)
metrics = evaluateDetectionAOS(detectionResults,groundTruthData,threshold)
```

Description

`metrics = evaluateDetectionAOS(detectionResults,groundTruthData)` computes the average orientation similarity (AOS) metric. The metric can be used to measure the detection results `detectionResults` against ground truth data `groundTruthData`. The AOS is a metric for measuring detector performance on rotated rectangle detections.

`metrics = evaluateDetectionAOS(detectionResults,groundTruthData,threshold)` additionally specifies the overlap threshold for assigning a detection to a ground truth bounding box.

Input Arguments

`detectionResults` — Detection results

three-column table

Detection results, specified as a three-column table. The columns contain rotated rectangle bounding boxes, scores, and labels.

Bounding Box	Format	Description
Rotated rectangle	[x_{center}, y_{center}, *width*, *height*, *yaw*]	This type of bounding box is defined in spatial coordinates as an *M*-by-5 matrix representing *M* bounding boxes. The x_{center} and y_{center} coordinates represent the center of the bounding box. The *width* and *height* elements represent the length of the box along the *x* and *y* axes, respectively. The *yaw* represents the rotation angle in degrees. The amount of rotation about the center of the bounding box is measured in the clockwise direction.

`groundTruthData` — Labeled ground truth images

datastore | table

Labeled ground truth images, specified as a datastore or a table.

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with two or three columns. When the output contains two columns, the first column must contain bounding boxes, and the second column must contain labels, {*boxes*,*labels*}. When the output contains three columns, the second column must contain the bounding boxes, and the third column must contain the labels. In this case, the first column can contain any type of data. For example, the first column can contain images or point cloud data, such as:

data	***boxes***	***labels***
The first column must be images.	Bounding boxes	The third column must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

- If you use a table, the table must have two or more columns.

data	***boxes***	***...***
The first column can contain data, such as point cloud data or images.	Each of the remaining columns must be a cell vector that contains *M*-by-5 matrices representing rotated rectangle bounding boxes. Each rotated rectangle must be of the form[x_{center}, y_{center}, *width*, *height*, *yaw*]. The vectors represent the location and size of bounding boxes for the objects in each image.	

`threshold` — Overlap threshold

nonnegative scalar

Overlap threshold, specified as a nonnegative scalar. The overlap ratio is defined as the intersection over union.

Output Arguments

`metrics` — AOS metrics

five-column table

AOS metrics, returned as a five-column table. Each row in the table contains the evaluation metrics for a class which is defined in the ground truth data contained in the `groundTruthData` input. To get the object class names:

`metrics.Properties.RowNames`

This table describes the five columns in the `metrics` table.

Column	Description
`AOS`	Average orientation similarity value
`AP`	Average precision over all the detection results, returned as a numeric scalar. Precision is a ratio of true positive instances to all positive instances of objects in the detector, based on the ground truth.
`OrientationSimilarity`	Orientation similarity values for each detection, returned as an *M*-element numeric column. *M* is one more than the number of detections assigned to a class. The first value of `OrientationSimilarity` is `1`. Orientation similarity is a normalized variant of the cosine similarity that measures the similarity between the predicted rotation angle and the ground truth rotation angle.
`Precision`	Precision values from each detection, returned as an *M*-element numeric column vector. *M* is one more than the number of detections assigned to a class. For example, if your detection results contain 4 detections with class label 'car', then `Precision` contains 5 elements. The first value of `Precision` is `1`. Precision is a ratio of true positive instances to all positive instances of objects in the detector, based on the ground truth.
`Recall`	Recall values from each detection, returned as an *M*-element numeric column vector. *M* is one more than the number of detections assigned to a class. For example, if your detection results contain 4 detections with class label 'car', then `Recall` contains 5 elements. The first value of `Recall` is `0`. Recall is a ratio of true positive instances to the sum of true positives and false negatives in the detector, based on the ground truth.

Version History

Introduced in R2020a

R2023b: `evaluateDetectionAOS` will be removed in a future release
Not recommended starting in R2023b

The `evaluateDetectionAOS` function will be removed in a future release. Use the `evaluateObjectDetection` function to evaluate object detection results with metrics such as the AOS, instead.

References

[1] Geiger, A., P. Lenz., and R. Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite." *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012.

See Also

Functions
`evaluateObjectDetection` | `evaluateDetectionPrecision` | `evaluateDetectionMissRate` | `bboxOverlapRatio`

Objects
`boxLabelDatastore`

evaluateObjectDetection

Evaluate object detection data set against ground truth

Syntax

```
metrics = evaluateObjectDetection(detectionResults,groundTruthData)
metrics = evaluateObjectDetection(detectionResults,groundTruthData,threshold)
metrics = evaluateObjectDetection( ___ ,Name=Value)
```

Description

`metrics = evaluateObjectDetection(detectionResults,groundTruthData)` evaluates the quality of the object detection results `detectionResults` against the labeled ground truth `groundTruthData` and returns various metrics.

`metrics = evaluateObjectDetection(detectionResults,groundTruthData,threshold)` specifies the overlap threshold for assigning an output bounding box to a ground truth bounding box.

`metrics = evaluateObjectDetection( ___ ,Name=Value)` specifies one or more name-value arguments, in addition to any combination of input arguments from previous syntaxes, to configure the object detection results evaluation. For example, `AdditionalMetrics="AOS"` includes average orientation similarity metrics in the output.

Examples

Plot Precision-Recall Curve for Object Detection

Load a table containing images and ground truth bounding box labels. The first column contains the images, and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Set the value of the `dataDir` variable as the location where the `vehicleTrainingData.mat` file is located. Load the test data into a local vehicle data folder.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Display the first 5 rows of the training data table.

```
disp(trainingData(1:5,:))
```

imageFilename	vehicle

```
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00001.jpg"    {[126 78 20 16]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00002.jpg"    {[100 72 35 26]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00003.jpg"    {[ 62 69 26 19]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00004.jpg"    {[ 71 64 22 21]}
    "B:\runnable\matlab\toolbox\vision\visiondata\vehicles\image_00005.jpg"    {[147 71 25 18]}
```

Create a `boxLabelDatastore` using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load Pretrained Object Detector

Load a pretrained YOLO v2 object detector trained to detect vehicles into the workspace.

```
vehicleDetector = load("yolov2VehicleDetector.mat");
detector = vehicleDetector.detector;
```

Evaluate and Plot Object Detection Metrics

Run the detector on the test images. Set the detection threshold to a low value to detect as many objects as possible. This helps you evaluate the detector precision across the full range of recall values.

```
results = detect(detector,imds,Threshold=0.01);
```

Use `evaluateObjectDetection` to compute metrics for evaluating the performance of an object detector.

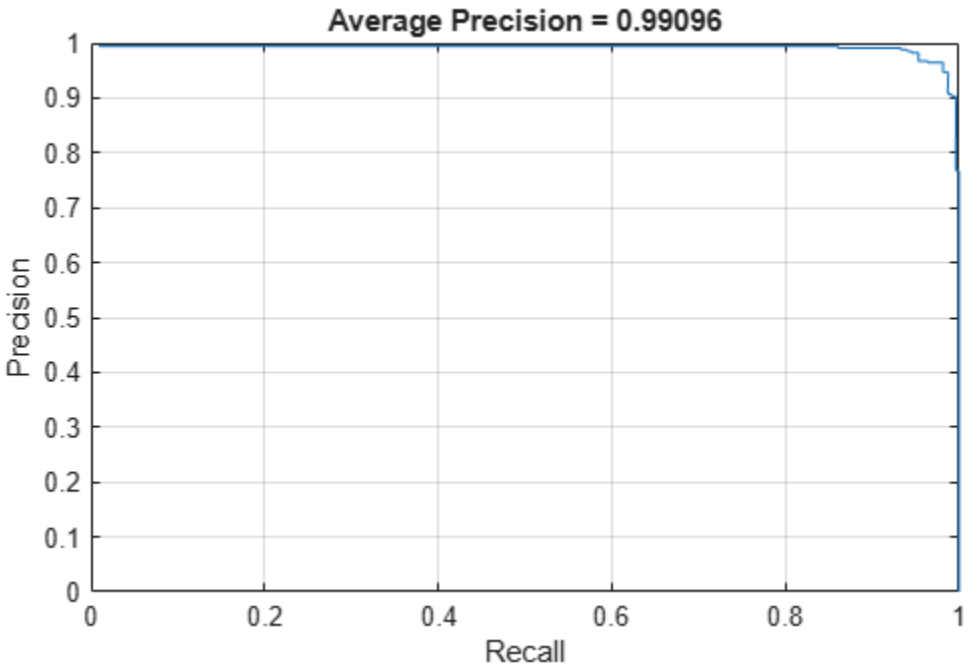
```
metrics = evaluateObjectDetection(results,blds);
```

Return the precision and recall metrics for the vehicle class using the `precisionRecall` object function.

```
[recall,precision,scores] = precisionRecall(metrics);
ap = averagePrecision(metrics);
```

Plot the precision-recall curve for the vehicle class, the only class in the data set. Compute the average precision (AP) using the `averagePrecision` object function.

```
figure
plot(recall{1},precision{1})
grid on
title("Average Precision = " + ap);
xlabel("Recall");
ylabel("Precision");
```



Compute the summary of the object detection metrics for the data set using the `summarize` object function.

```
[summaryDataset,summaryClass] = summarize(metrics);
disp(summaryDataset)
```

```
    NumObjects    mAPOverlapAvg    mAP0.5
    __________    _____________    _______

       336           0.99096       0.99096
```

Evaluate Trained ACF Object Detector on Rail Road Signs Data

This example shows how to evaluate the predictions from a trained ACF object detector on a dataset of rail road signs.

The test dataset images are stored as a zipped file attached to this example. Unzip this into the current working directory.

```
unzip("RRSignsTest.zip")
```

Load the ground truth for the railroad signs test dataset stored as a `groundTruth` object.

```
load("RRSignsGroundTruth.mat", "gTruthTest")
```

Load the pretrained ACF object detector.

```
load("RRSignsACFDetector.mat", "acfDetector")
```

Extract images and bounding boxes from the `groundTruth` object by using the `objectDetectorTrainingData` function. The function creates and stores data in the form an image datastore and a labeled bounding box datastore.

```
[imdsTest, bldsTest] = objectDetectorTrainingData(gTruthTest);
imshow(preview(imdsTest))
```



Combine the test images and ground truth labels into a single datastore.

```
testData = combine(imdsTest, bldsTest);
```

Run the detector on the test images.

```
testDetections = detect(acfDetector,testData);
```

Arrange the detector results into a format suitable for object detection evaluation. The ACF object detector does not explicitly predict the class label of an object. This label information needs to be added to the `testDetections` table.

```
numImages = height(testDetections);
labels = cell(numImages,1);
className = string(gTruthTest.LabelDefinitions.Name{1});
for i = 1:numImages
    numDetections = numel(testDetections.Scores{i});
    labels{i} = categorical(repmat(className, [numDetections 1]), className);
end

testDetections.Labels = labels;
```

Evaluate the detector results.

```
metrics = evaluateObjectDetection(testDetections, testData);
```

Display the average precision metric, which summarizes the detector's performance over the test dataset. The ACF detector performs well on this dataset with an AP of 0.8.

```
disp(summarize(metrics, MetricName="AP"))
```

```
    NumObjects    mAPOverlapAvg    mAP0.5
    __________    _____________    ______

        10             0.8           0.8
```

Display the detector performance on individual images.

```
imageSummary = imageMetrics(metrics);
disp(imageSummary)
```

```
    ImageIndex    NumPredictedObjects    NumGroundTruthObjects    TP    FP    FN    Precision
    __________    ___________________    _____________________    __    __    __    _________

         1                1                       1              1     0     0          1
         2                1                       1              1     0     0          1
         3                0                       1              0     0     1        NaN
         4                1                       1              1     0     0          1
         5                1                       1              1     0     0          1
         6                1                       1              1     0     0          1
         7                1                       1              1     0     0          1
         8                1                       1              1     0     0          1
         9                1                       1              1     0     0          1
        10                0                       1              0     0     1        NaN
```

The detector makes a mistake on only 2 out of 10 images in the test set, shown on rows 3 and 10.

Evaluate Miss Rate of a YOLO v2 Object Detector

This example shows how to evaluate miss rate of a pretrained YOLO v2 object detector.

Load the Vehicle Ground Truth Data

Load a table containing the vehicle training data. The first column contains the training images, the remaining columns contain the labeled bounding boxes.

```
data = load('vehicleTrainingData.mat');
trainingData = data.vehicleTrainingData(1:100,:);
```

Add fullpath to the local vehicle data folder.

```
dataDir = fullfile(toolboxdir('vision'), 'visiondata');
trainingData.imageFilename = fullfile(dataDir, trainingData.imageFilename);
```

Create an imageDatastore using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a boxLabelDatastore using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Load YOLOv2 Detector for Detection

Load the detector containing the layerGraph for trainining.

```
vehicleDetector = load('yolov2VehicleDetector.mat');
detector = vehicleDetector.detector;
```

Evaluate and Plot the Miss Rate Results

Run the detector with imageDatastore.

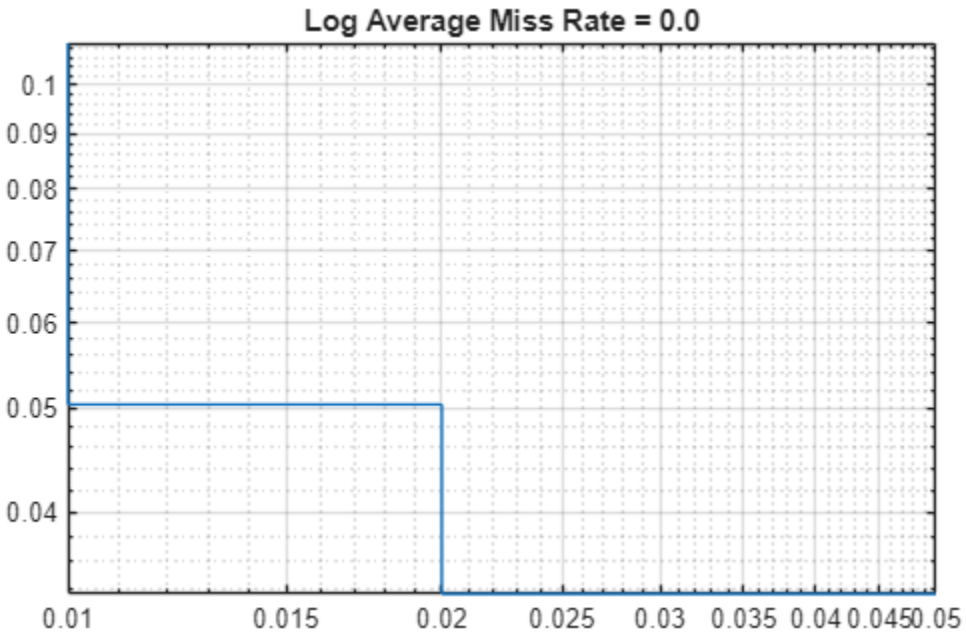
```
results = detect(detector,imds);
```

Evaluate the results against the ground truth data using the `evaluateObjectDetection` function. Specify the `AdditionalMetrics="LAMR"` name-value argument to include miss rate and false positives per image (FPPI) in the returned metrics.

```
metrics = evaluateObjectDetection(results,blds,AdditionalMetrics="LAMR");
```

Plot the log-miss-rate/FPPI curve.

```
figure;
loglog(metrics.ClassMetrics.FPPI{1}, metrics.ClassMetrics.MR{1});
grid on
title(sprintf('Log Average Miss Rate = %.1f', metrics.ClassMetrics.LAMR{1}))
```



Create Ground Truth and Use AOS Metric To Evaluate Rotated Rectangle Detections

Define ground truth bounding boxes in two images corresponding to a vehicle class. Each row defines a rotated bounding box of the form [*xcenter*, *ycenter*, *width*, *height*, *yaw*].

```
gtbbox_img1 = [
    2 2 10 20 45
    80 80 30 40 15
    ];

gtbbox_img2 = [
    4 4 20 40 90
```

```
    160 160 60 80 30
    ];

gtlabel = ["vehicle"
           "vehicle"
           ];
gtlabel = categorical(gtlabel);
```

Create a table to hold the ground truth data. Each row in the table shows ground truth boxes and labels for one image.

```
groundTruthData = table({gtbbox_img1;gtbbox_img2},{gtlabel;gtlabel},'VariableNames',{'Boxes','Lal
```

```
groundTruthData=2×2 table
       Boxes             Lables
    ___________    _________________

    {2×5 double}    {2×1 categorical}
    {2×5 double}    {2×1 categorical}
```

For each image, define detection results for rotated bounding boxes, scores, and labels.

```
bbox_img1 = [
    4 4 10 20 20
    50 50 30 10 30
    90 90 40 50 10 ];

bbox_img2 = [
    8 8 20 40 40
    100 100 60 20 60
    180 180 80 100 20 ];

scores = [0.9 0.7 0.8]';

labels = [
    "vehicle"
    "vehicle"
    "vehicle"
    ];
labels = categorical(labels,"vehicle");
```

Create a table to hold the detection results. Each row in the table shows detected boxes,scores and labels for one image

```
detectionResults = table({bbox_img1;bbox_img2},...
    {labels;labels},...
    {scores;scores},'VariableNames',{'Boxes','Labels','Scores'})
```

```
detectionResults=2×3 table
       Boxes             Labels              Scores
    ___________    _________________    ____________

    {3×5 double}    {3×1 categorical}    {3×1 double}
    {3×5 double}    {3×1 categorical}    {3×1 double}
```

Evaluate the results against the ground truth data using the `evaluateObjectDetection` function. Specify the `AdditionalMetrics="AOS"` name-value argument to include average orientation similarity (AOS) in the returned metrics.

```
metrics = evaluateObjectDetection(detectionResults,groundTruthData,AdditionalMetrics="AOS");
```

Display average orientation similarity (AOS) for each class averaged over all the specified overlap thresholds.

```
meanAOSOverAllOverlaps = metrics.ClassMetrics.AOSOverlapAvg
```

```
meanAOSOverAllOverlaps =
0.2600
```

Display orientation similarity vector for each class.

```
orientationSimilarity = metrics.ClassMetrics.OrientationSimilarity
```

```
orientationSimilarity = 1×1 cell array
    {[1 0.9532 0.4766 0.3177 0.2383 0.1906 0.1589]}
```

Input Arguments

`detectionResults` — Predicted object detection results

table

Predicted object detection results, specified as a three-column table with the same number of rows as the number of images in the data set. The table must contain the bounding box, predicted label, and score for each detected object in this format:

Bounding Boxes	Labels	Scores
Predicted 2-D bounding boxes for M objects, specified as an M-by-4 or M-by-5 numeric array. Each 2-D bounding box must be in the format [*x y width height*] if axis-aligned, or in the format [x_{center} y_{center} *width height yaw*] if rotated.	Predicted object labels, specified as an M-by-1 categorical vector.	Predicted scores, specified as an M-by-1 numeric vector.

The order of the elements does not matter. When the datastore returns a cell array with more than three elements, the `evaluateObjectDetection` function assumes that the first element with an M-by-4 or M-by-5 numeric array contains the bounding boxes, the first element with categorical data contains the label data, and the first element with M-by-1 numeric data contains the scores.

You can create the detection results table using the `detect` function associated with your object detector.

`groundTruthData` — Labeled ground truth

datastore | table

Labeled ground truth, specified as a datastore with two outputs or a table with two columns. If you are specifying a table, the number of rows must match the number of images in the data set with these columns:

boxes	*labels*
Ground truth 2-D bounding boxes for *M* objects, specified as an *M*-by-4 or *M*-by-5 numeric array. Each 2-D bounding box must be in the format [*x y width height*] if axis-aligned, or in the format [x_{center} y_{center} *width height yaw*] if rotated.	Ground truth object labels, specified as an *M*-by-1 categorical vector.

The order of the elements does not matter. When the datastore returns a cell array with more than three elements, the `evaluateObjectDetection` function assumes that the first element with an *M*-by-4 or *M*-by-5 numeric array contains the bounding boxes, the first element with categorical data contains the label data, and the first element with *M*-by-1 numeric data contains the scores.

`threshold` — Overlap threshold
`0.5` (default) | numeric scalar | numeric vector

Overlap threshold for assigning a detection to a ground truth box, specified as a numeric scalar in the range [0, 1] or a numeric vector. The overlap ratio is the intersection over union (IoU) ratio of two bounding boxes, or the ratio of the bounding box overlap area to the combined area of the predicted boxes and ground truth (minus the overlap).

IoU = Area of Overlap / (Ground Truth Area + Predicted Box Area – Intersection Area)

When you specify a numeric vector, the `evaluateObjectDetection` function calculates metrics for each threshold.

Tip To reduce the number of false negatives, you can lower the IoU threshold at the possible expense of increasing the number of false positives. To learn about the role of IoU in detection evaluation, see “Overlap Threshold (Intersection Over Union)” on page 3-222.

Tip To optimize model performance for your application, you can evaluate the object detection metrics across a range of overlap thresholds. To learn how to fine-tune object detector performance using multiple overlap thresholds, see “Tips” on page 3-222.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `evaluateObjectDetection(__,AdditionalMetrics=["LAMR","AOS"])` additionally returns LAMR and AOS.

`Verbose` — Evaluation progress display toggle
`true` or `1` (default) | `false` or `0`

Evaluation progress display toggle, specified as a numeric or logical `1` (`true`) or `0` (`false`). If you specify `Verbose` as `true`, the function displays progress information in the Command Window. The

displayed information includes a progress bar, elapsed time, estimated time remaining, and data set metrics.

`AdditionalMetrics` — Additional metrics
`[]` (default) | `"LAMR"` | `"AOS"` | string array

Additional metrics, specified as `"LAMR"`, `"AOS"`, or a string array. The function returns the specified metrics as additional columns in the `ClassMetrics` table of the `objectDetectionMetrics` object, as shown in this table below.

`AdditionalMetrics`	`ClassMetrics`
`"LAMR"`	**`LAMROverlapAverage`:** Log-average miss rate for each class, averaged over all specified thresholds, as a numeric scalar. **`LAMR`:** Log-average miss rate for each class, computed at each threshold, as a *numThresh*-by-1 numeric vector. **`MR`:** Miss rate vector for each class as a *numThresh*-by-*numPredictions* matrix, where *numPredictions* is the number of predicted objects. **`FPPI`:** False positives per image vector for each class as a *numThresh*-by-*numPredictions* matrix.
`"AOS"`	**`AOSOverlapAvg`:** Average orientation similarity (AOS) for each class averaged over all the specified overlap thresholds, as a numeric scalar. **`AOS`:** Average orientation (AOS) similarity for each class computed at each overlap threshold as a *numThresh*-by-1 numeric vector. **`OrientationSimilarity`:** Orientation similarity vector for each class as a *numThresh*-by-(*numPredictions*+1) matrix.

To specify `"AOS"` as an additional metric, your input data must contain rotated bounding boxes.

Output Arguments

`metrics` — Object detection metrics
`objectDetectionMetrics` object

Object detection metrics, returned as an `objectDetectionMetrics` object. The `“OverlapThreshold” on page 2-0` property of the object corresponds to the value of the `threshold` argument.

More About

Overlap Threshold (Intersection Over Union)

Use overlap (or IoU) threshold to determine whether a detection is a true positive or false positive.

A detection is a true positive (TP) when both of these conditions are met:

- The predicted class matches the class of a ground truth (indicated by the confidence score exceeding the confidence threshold).
- The predicted bounding box has an IoU greater than or equal to the specified overlap threshold.

A detection is a false positive (FP) when at least one of these conditions is met:

- The predicted class does not match the class of a ground truth.
- The predicted bounding box has an IoU less than the specified overlap threshold.

A missed detection is a false negative (FN) when one or more of these conditions is met:

- The confidence score of the detection is lower than the confidence score threshold, resulting in a missed detection.
- The detection has an IoU less than the specified threshold, resulting in the predicted bounding box does not sufficiently overlapping with the ground truth, leading to the detection being discarded.

Quantify the accuracy of an object detector on a particular dataset by evaluating the overlap between the predicted bounding boxes and the ground truth bounding boxes.

Tips

To get started with evaluating object detector performance using various metrics such as precision-recall, average precision, and confusion matrix, see “Evaluate Object Detector Performance”.

To evaluate model performance across different levels of localization accuracy and fine-tune an object detector, compute metrics at multiple overlap (IoU) thresholds. Overlap thresholds vary typically from 0.5 to 0.95 during evaluation. For example, set the overlap threshold value to 0.5 to accept moderate overlap accuracy, and set the value to 0.95 to ensure very high localization accuracy. There are many ways to use the overlap threshold to tune object detector performance.

- By evaluating common metrics, such as precision and recall, at different overlap thresholds, you can evaluate the tradeoff between detecting more objects (higher recall) and ensuring those detections are accurate (higher precision).
- By evaluating model performance at a lower overlap threshold, such as 0.5, and at higher thresholds, such as 0.95, you can determine how good the detector is at identifying objects (detection) at the expense of precise localization (bounding box accuracy).
- By computing metrics such as the mean average precision (mAP) over multiple overlap thresholds, you can produce a single performance figure that accounts for both detection and localization accuracy, enabling you to compare different detection models.

To learn how to compute metrics at a range of overlap thresholds to tune detector performance, see the "Precision and Recall for a Single Class" section of the “Multiclass Object Detection Using YOLO v2 Deep Learning” example.

Version History

Introduced in R2023b

R2026a: Evaluate per-image object detection metrics

To evaluate the per-image object detection metrics for all images or a subset of images in the data set, use the `imageMetrics` object function.

To update your code to compute per-image metrics, replace instances of the `ImageMetrics` property with the `imageMetrics` object function. For an example, see the“Multiclass Object Detection Using YOLO v2 Deep Learning” example.

R2023b: Recommended over `evaluateDetectionPrecision`, `evaluateDetectionMissRate`, `evaluateDetectionAOS`

The `evaluateObjectDetection` function enables you to simultaneously calculate the average precision, miss rate, and AOS (if applicable) to evaluate detector performance. Therefore, `evaluateObjectDetection` is recommended over `evaluateDetectionPrecision`, `evaluateDetectionMissRate`, and `evaluateDetectionAOS`, which will be removed in a future release. To update your code, ensure that your ground truth data is in the correct format. See the `groundTruthData` argument for more details.

`evaluateObjectDetection` improves on `evaluateDetectionPrecision` by enabling you to compute the object detection performance metrics for the entire data set over all classes and at specified overlap thresholds, use rotated bounding boxes, and calculate the confusion matrix. The corresponding `objectDetectionMetrics` object also enables you to evaluate object detection metrics across various object size ranges using the `metricsByArea` function.

See Also

Apps
Object Detector Analyzer

Functions
`objectDetectionMetrics` | `yoloxObjectDetector` | `yolov4ObjectDetector` | `yolov3ObjectDetector` | `yolov2ObjectDetector` | `ssdObjectDetector` | `boxLabelDatastore`

Topics
Get Started with Object Detector Analyzer
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Detect Small Objects Using Tiled Training of YOLOX Network”
“Evaluate Object Detector Performance”
“Calibrate Object Detection Confidence Scores”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”

External Websites
https://cocodataset.org/#detection-eval

evaluateReidentificationNetwork

Evaluate re-identification network using cumulative matching characteristic (CMC) and mean average precision (mAP) metrics

Syntax

```
[metrics,queries,gallerySets] = evaluateReidentificationNetwork(features,
labels)
[metrics,queries,gallerySets] = evaluateReidentificationNetwork( ___ ,
Name=Value)
```

Description

`[metrics,queries,gallerySets] = evaluateReidentificationNetwork(features, labels)` evaluates the performance of a re-identification (ReID) network and returns the cumulative matching characteristic (CMC) and mean average precision (mAP) metrics.

`[metrics,queries,gallerySets] = evaluateReidentificationNetwork( ___ , Name=Value)` configures the evaluation metrics using one or more name-value arguments, in addition to the input arguments from the previous syntax. For example, `DistanceMetric="euclidean-squared"` specifies `"euclidean-squared"` as the distance metrics method for measuring the distance between feature vectors.

Input Arguments

`features` — Features extracted by re-identification network

M-by-*N* matrix

Features extracted by the re-identification network using the `extractReidentificationFeatures` object function, specified as an *M*-by-*N* matrix. *N* is the number of images in the input datastore `ds`.

`labels` — Labels

1-by-*N* string array

Labels corresponding to the class name of each `features` column vector output by the `extractReidentificationFeatures` object function, specified as a 1-by-*N* string array. *N* is the number of images in the input datastore, `ds`.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `evaluateReidentificationNetwork(features,labels,DistanceMetric="euclidean-squared")` specifies `"euclidean-squared"` as the distance metrics method for measuring the distance between feature vectors.

`DistanceMetric` — Distance metrics method
`"cosine"` (default) | `"euclidean-squared"`

Distance metrics method to use in query evaluations to measure the distance between feature vectors, specified as `"cosine"` or `"euclidean-squared"`. For most cross-entropy training scenarios, use the `"euclidean-squared"` method.

`RandomizeQueries` — Order of queries
`true` or `1` (default) | `false` or `2`

Order of the queries, specified as a logical `true` or `false`. The function randomizes the order of the queries by default. If you specify `RandomizeQueries` as `false`, the function orders the queries in the same order as the columns in `features` and `labels`.

`Rank` — Ranks
`1` (default) | numeric scalar | numeric vector | `"all"`

Ranks to store in the `reidentificationMetrics` object for the CMC metric, specified as a non-repeating and increasing numeric scalar or numeric vector in the range [1, *C*], or `"all"`. *C* is the number of unique object classes in `labels`. If you specify `Rank` as `"all"`, the `reidentificationMetrics` stores all valid ranks of the CMC metric.

`Verbose` — Evaluation progress display toggle
`false` or `0` (default) | `true` or `1`

Evaluation progress display toggle, specified as a numeric or logical `1` (`true`) or `0` (`false`). If you specify `Verbose` as `true`, the function displays progress information in the Command Window. The displayed information includes a progress bar, elapsed time, estimated time remaining, and data set metrics.

Output Arguments

`metrics` — Re-identification metrics
`reidentificationMetrics` object

Re-identification metrics, returned as a `reidentificationMetrics` object.

`queries` — Queries for object retrieval
1-by-*N* vector of integers

Queries for object retrieval, returned as 1-by-*N* vector of integers. *N* is the number of columns in `features`. Each element in the vector represents the column index of the features used to query a gallery.

`gallerySets` — Gallery sets of images
C-by-*N* matrix of integers

Gallery sets of images for querying, returned as a *C*-by-*N* matrix of integers. *N* is the number of columns in `features`, and *C* is the number of unique classes in `labels`.

Version History

Introduced in R2024a

See Also

`reidentificationMetrics` | `trainReidentificationNetwork` | `reidentificationNetwork` | `extractReidentificationFeatures`

Topics

“Reidentify People Throughout a Video Sequence Using ReID Network”

ssdLayers

(Removed) SSD multibox object detection network

Note `ssdLayers` function has been removed. Use the `ssdObjectDetector` function to create a SSD object detection network, instead.

Syntax

```
lgraph = ssdLayers(imageSize,numClasses,networkName)
lgraph = ssdLayers(imageSize,numClasses,network,anchorBoxes,
predictorLayerNames)
```

Description

`lgraph = ssdLayers(imageSize,numClasses,networkName)` creates a single shot detector (SSD) multibox object detection network based on the `networkName`, input image size, and the number of classes the network should be configured to classify. The network is returned as an `LayerGraph` object.

The SSD is a convolutional neural network-based object detector that predicts bounding box coordinates, classification scores, and corresponding class labels.

`lgraph = ssdLayers(imageSize,numClasses,network,anchorBoxes, predictorLayerNames)` returns an SSD that contains custom anchor boxes specified by `anchorBoxes` that are connected to the network layers at locations specified by `predictorLayerNames`. Specify these arguments in addition to the input argument from the previous syntax.

Examples

Create SSD Network Using VGG-16 Network

Specify the base network.

```
networkName = 'vgg16';
```

Specify the image size.

```
imageSize = [300 300 3];
```

Specify the classes to detect.

```
numClasses = 2;
```

Create the SSD object detection network.

```
lgraph = ssdLayers(imageSize,numClasses,networkName);
```

Visualize the network using the network analyzer.

```
analyzeNetwork(lgraph)
```

Input Arguments

`imageSize` — Size of input image

two-element vector | three-element vector

Size of input image, specified as one of these values.

- Two-element vector of the form [*H W*] for a grayscale image of size *H*-by-*W*
- Three-element vector of the form [*H W* 3] for an RGB color image of size *H*-by-*W*

When you set the `baseNetwork` input to `'vgg16'`, `'resnet50'`, or `'resnet101'`, the `imageSize` input must be of the form [*H W* 3].

`numClasses` — Number of classes for network to classify

positive scalar

Number of classes for the network to classify, specified as a positive scalar.

`network` — Pretrained convolutional neural network

`LayerGraph` object | `DAGNetwork` object | `SeriesNetwork` object

Pretrained convolutional neural network, specified as a `LayerGraph`, `DAGNetwork`, or `SeriesNetwork` object.

The pretrained convolutional neural network is used as the base for the SSD multibox object detection network. For details on pretrained networks in MATLAB, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

`networkName` — Pretrained convolutional neural network

`'vgg16'` | `'resnet50'` | `'resnet101'`

Pretrained convolutional neural network, specified as a string or character vector for the `'vgg16'`, `'resnet50'`, or `'resnet101'` network. To specify one of these names, you must download and install the network support packages for the corresponding valid network names.

- `vgg16`
- `resnet50`
- `resnet101`

The pretrained convolutional neural network is used as the base for the SSD multibox object detection network. For details on pretrained networks in MATLAB, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

`anchorBoxes` — Anchor boxes

1-by-*M* cell array

Anchor boxes, specified as a 1-by-*M* cell array for *M* number of predictor layers in the SSD network. Each predictor layer contains a *K*-by-2 matrix that defines *K* anchor boxes of the form [*height width*]. The number of anchor boxes in each element can vary.

The size of each anchor box is determined based on the scale and aspect ratio of different object classes present in input training data. The size of each anchor box must be smaller than or equal to

the size of the input image. You can use the clustering approach for estimating anchor boxes from the training data. For more information, see “Estimate Anchor Boxes from Training Data”.

`predictorLayerNames` — Names of layers in input
M-element vector of strings | 1-by-*M* cell array of character vectors

Names of layers in input, specified as an *M*-element vector of strings or a 1-by-*M* cell array of character vectors. The SSD detection subnetworks are attached to the predictor layers specified by this input.

Output Arguments

`lgraph` — SSD multibox object detection network
`LayerGraph` object

SSD multibox object detection network, returned as a `LayerGraph` object.

Note The default value for the `Normalization` property of the image input layer in the returned `lgraph` object is set to the `Normalization` property of the base network specified in `baseNetwork`.

Algorithms

The `ssdLayers` function creates an SSD network and returns `lgraph`, an object that represents the network architecture for an SSD object detector.

The `trainSSDObjectDetector` function trains and returns an SSD object detector, `ssdObjectDetector`. Use the `detect` object function for the `ssdObjectDetector` object to detect objects using the detector trained with the SSD network architecture.

```
bbox = detect(detector,I)
```

The `ssdLayers` function uses a pretrained neural network as the base network, to which it adds a detection subnetwork required for creating an SSD object detection network. Given a base network, `ssdLayers` removes all the layers succeeding the feature layer in the base network and adds the detection subnetwork. The detection subnetwork is comprised of groups of serially connected convolution, rectified linear unit (ReLU), and batch normalization layers. The SSD merge layer, a box regression layer, and a focal loss classification layer are added to the detection subnetwork.

Version History

Introduced in R2020a

R2024b: `ssdLayers` function has been removed
Errors starting in R2024b

The `ssdLayers` function has been removed. You can use the `ssdObjectDetector` function to create a SSD object detection network, instead.

R2023b: `ssdLayers` function will be removed
Warns starting in R2023b

The `ssdLayers` function will be removed in a future release. You can use the `ssdObjectDetector` function to create a SSD object detection network, instead.

R2022a: `ssdLayers` function is not recommended
Not recommended starting in R2022a

The `ssdLayers` function is not recommended. You can use the `ssdObjectDetector` function to create a SSD object detection network, instead.

References

[1] Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector." In *Computer Vision – ECCV 2016*, edited by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, 9905:21-37. Cham: Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-46448-0_2.

[2] Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, et al. "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3296-97. Honolulu, HI:IEEE, 2017. https://doi.org/10.1109/CVPR.2017.351.

See Also

Objects
`ssdObjectDetector`

Functions
`trainSSDObjectDetector`

Topics
“Object Detection Using SSD Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using Single Shot Multibox Detector”
“Getting Started with SSD Multibox Detection”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

pcbin

Spatially bin point cloud points

Syntax

```
bins = pcbin(ptCloud,numBins)
bins = pcbin(ptCloud,numBins,spatialLimits)
[bins,binLocations] = pcbin( ___ )
[ ___ ] = pcbin( ___ ,BinOutput=true)
```

Description

`bins = pcbin(ptCloud,numBins)` spatially grids points in the point cloud, `ptCloud`. The output `bins` stores spatial bin locations, which are the indices for the points. The function finds the bin locations by sampling the point cloud in each dimension by the number of bins specified by `numBins`.

`bins = pcbin(ptCloud,numBins,spatialLimits)` specifies the spatial limits to use when the function places points into bins.

`[bins,binLocations] = pcbin( ___ )` also returns `binLocations`, the spatial bounds of each bin in `bins`. `binLocations` is the same size as `bins`.

`[ ___ ] = pcbin( ___ ,BinOutput=true)` also specifies the format of the `bins` and `binLocations` output.

When you set `BinOutput` to `true`, `bins` and `binLocations` are returned as cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*].

When you set `BinOutput` to `false` and `ptCloud` are unorganized point clouds, `bins` is returned as an *M*-by-1 numeric vector. `binLocations` is returned as an *M*-by-6 matrix.

When you set `BinOutput` to `false` and `ptCloud` are organized point clouds, `bins` is returned an *M*-by-*N* matrix. `binLocations` is returned as an *M*-by-*N*-by-6 matrix.

Examples

Build Occupancy Grid from Point Cloud

Read point cloud data into the workspace.

```
ptCloud = pcread("teapot.ply");
```

Spatially bin the point cloud into a 32-by-32-by-32 grid.

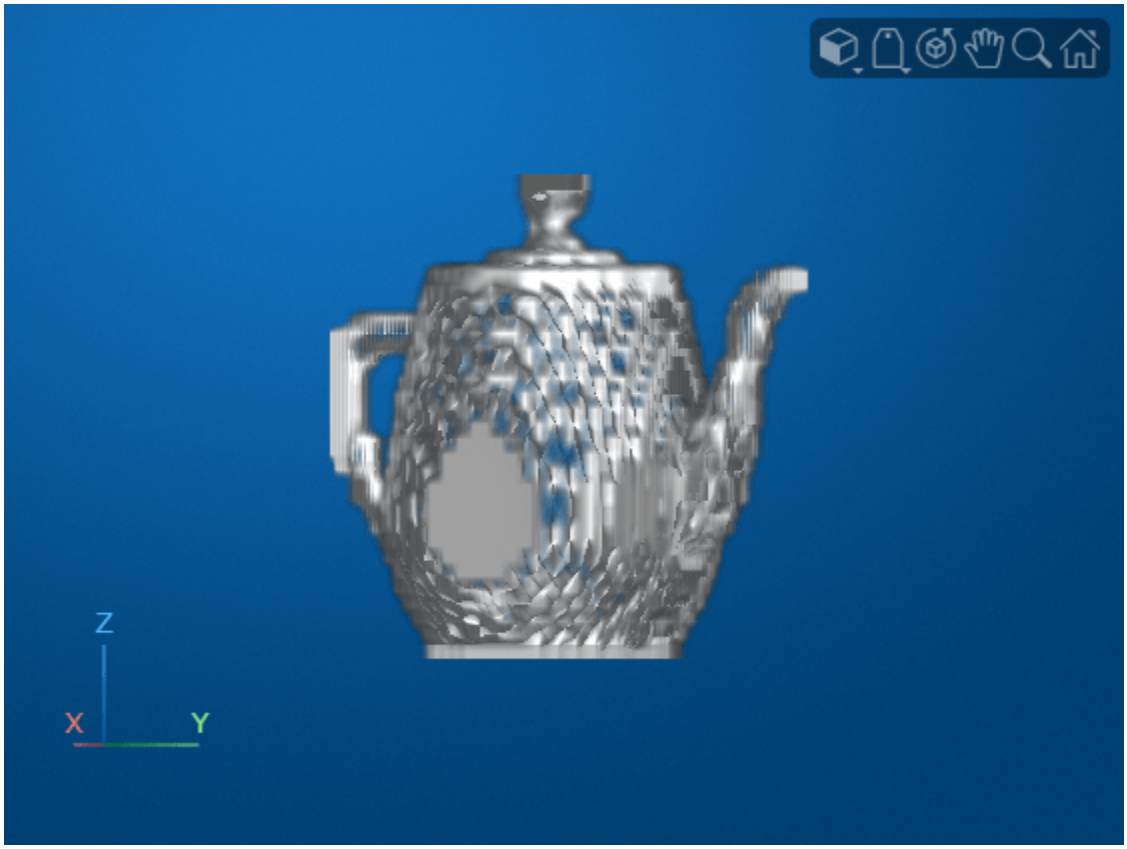
```
indices = pcbin(ptCloud,[32 32 32]);
```

Build an occupancy grid.

```
occupancyGrid = cellfun(@(c) ~isempty(c), indices);
```

Display the occupancy grid, and change the camera perspective for a clearer view of the grid.

```
v = volshow(occupancyGrid);
viewer = v.Parent;
viewer.CameraPosition = [44 25 16];
viewer.CameraTarget = [16 16 16];
```





Build Bird's Eye View Density Grid from Point Cloud

Read point cloud data into the workspace.

```
ptCloud = pcread('teapot.ply');
```

Spatially bin the point cloud into a 32-by-32-by-1 grid.

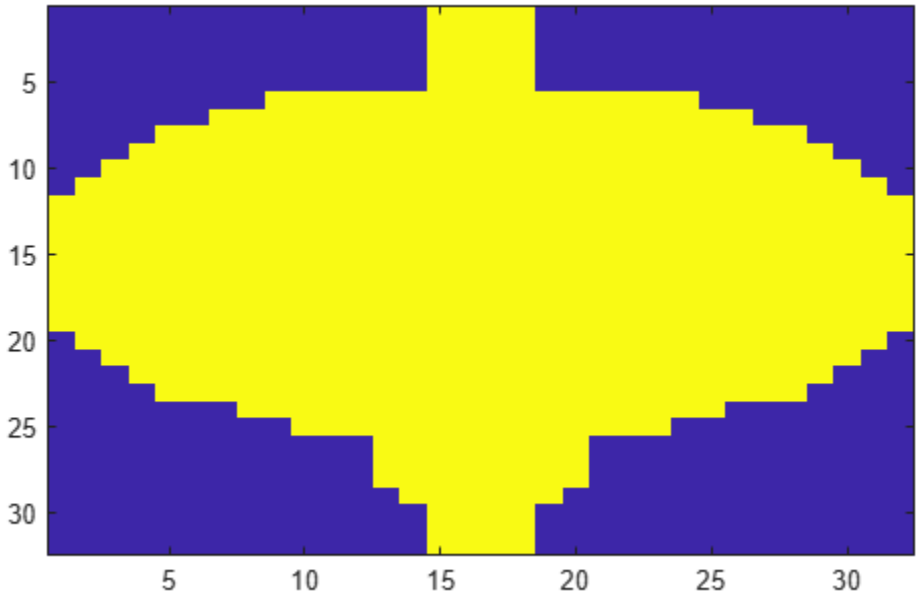
```
indices = pcbin(ptCloud,[32 32 1]);
```

Build a density grid.

```
densityGrid = cellfun(@(c) ~isempty(c),indices);
```

Display the density grid.

```
figure;
imagesc(densityGrid);
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object. Points containing `Inf` or `NaN` values are invalid points and are not included in the output.

`numBins` — Number of bins

three-element numeric vector

Number of bins, specified as a three-element vector of the form [*numBinsX*,*numBinsY*, *numBinsZ*]. The vector elements indicate the number of bins to use in each dimension, respectively.

`spatialLimits` — Spatial limits

3-by-2 numeric matrix

Spatial limits, specified as a 3-by-2 numeric matrix of the form [*Xmin*, *Xmax*; *Ymin*, *Ymax*; *Zmin*, *Zmax*]. The spatial limits define the bounds used when the function bins points. Points that are outside of the specified limits are not included in the output.

Output Arguments

`bins` — Spatial bin locations

cell array | cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*] | *M*-by-1 numeric vector | *M*-by-*N* matrix

Spatial bin locations, returned as a cell array of the same size as input `numBins`, an *M*-by-1 numeric vector, or an *M*-by-*N* matrix. The function returns a cell array when you do not use the `BinOutput`

argument. Each cell in the array contains indices to locations of points in the input point cloud that define the bins. Each bin includes the left bin edge, except for the last bin which includes both bin edges.

The table describes the size of `bins` based how you specify the `BinOutput` argument, and whether `ptCloud` is unorganized or organized.

BinOutput	**bins**
`true`	cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*]
`false`	Unorganized point clouds *M*-by-1 numeric vector, where *M* represents the number of points.
	Organized point clouds *M*-by-*N* matrix, where *M* and *N* represents the total number of points set up as an organized point cloud.

The function processes invalid points differently, depending on how you specify the `BinOutput` argument. Invalid points are points with coordinates that contain `Inf` or `NaN`. When you set `BinOutput` to `true`, the function ignores invalid points and does not include them in the output. When you set `BinOutput` to `false`, the function returns `NaN` values for invalid points.

`binLocations` — Bin spatial bounds

cell array | cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*] | *M*-by-6 matrix | *M*-by-*N*-by-6 matrix

Bin spatial bounds, returned as a cell array the same size as the output `bins`, cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*], an *M*-by-6 matrix, or an *M*-by-*N*-by-6 matrix. The function returns a cell array when you do not use the `BinOutput` argument. Each cell in the array contains the spatial bounds of the bins in `bins`. The bounds for a bin are of the form [*Xmin*, *Xmax*; *Ymin*, *Ymax*; *Zmin*, *Zmax*].

The table describes the size of `binLocations` based how you specify the `BinOutput` argument, and whether `ptCloud` is unorganized or organized.

BinOutput	**binLocations**
`true`	cell arrays of size [*numBinsX*,*numBinsY*,*numBinsZ*]
`false`	Unorganized point clouds: *M*-by-6 matrix, where *M* represents the number of points located in the bin specified by the [*Xmin*, *Xmax*; *Ymin*, *Ymax*; *Zmin*, *Zmax*] spatial bound.

BinOutput	binLocations
	Organized point clouds: *M*-by-*N*-by-6 matrix, where *M* and *N* represents the number of points in an organized point cloud, with each point located in the bin specified by the [*Xmin*, *Xmax*; *Ymin*, *Ymax*; *Zmin*, *Zmax*] spatial bound.

Version History

Introduced in R2020a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`pctransform`

Objects

`pointCloud`

readBarcode

Detect and decode 1-D or 2-D barcode in image

Syntax

```
msg = readBarcode(I)
msg = readBarcode(I,roi)
msg = readBarcode( ___ ,format)
[msg,detectedFormat,loc] = readBarcode( ___ )
```

Description

`msg = readBarcode(I)` detects and decodes a 1-D or 2-D barcode in the input image and returns the message associated with that barcode.

If the input image contains multiple barcodes, the `readBarcode` function decodes only the first barcode detected.

`msg = readBarcode(I,roi)` specifies a rectangular region of interest (ROI) to reduce the area in which the function searches for a barcode. The ROI must be fully contained in the input image.

`msg = readBarcode( ___ ,format)` specifies barcode formats to reduce the types of barcodes for which the function searches. Specify this argument in addition to any of the input argument combinations in previous syntaxes.

`[msg,detectedFormat,loc] = readBarcode( ___ )` additionally returns the format, `detectedFormat`, and location, `loc`, of the detected barcode.

Examples

Detect and Decode 1-D Barcode

Read an image containing a barcode into the workspace.

```
I = imread("barcode1D.jpg");
```

Detect the barcode and decode its message.

```
msg = readBarcode(I);
```

Display the decoded barcode message.

```
disp("Decoded barcode message: " + msg)
```

```
Decoded barcode message: 1234567890128
```

Search Image for 1-D Barcode

Read an image containing a barcode into the workspace.

```
I = imread("barcode1D.jpg");
```

Search the image for a 1-D barcode, returning its message, format, and location.

```
[msg,detectedFormat,loc] = readBarcode(I,'1D');
```

Display the detected barcode format.

```
disp("Barcode format: " + detectedFormat)
```

```
Barcode format: EAN-13
```

Annotate the image with the decoded barcode message.

```
xyBegin = loc(1,:);
Imsg = insertText(I,xyBegin,msg,'BoxOpacity',1,'FontSize',30);
```

Insert a line to show the scan row.

```
imSize = size(Imsg);
Imsg = insertShape(Imsg,'Line',[1 xyBegin(2) imSize(2) xyBegin(2)],'LineWidth',5);
```

Display the image.

```
imshow(Imsg)
```



Read a QR Barcode

Read an image containing a barcode.

```
I = imread("barcodeQR.jpg");
```

Define the ROI in the image that contains the barcode.

```
roi = [470, 300, 720, 620];
```

Search for a QR barcode within the ROI.

```
[msg,~,loc] = readBarcode(I,roi,"QR-CODE");
```

Annotate the image with the decoded message from the detected barcode.

```
xyText =  loc(2,:)/4;
Imsg = insertText(I,xyText,msg,"BoxOpacity",1,"FontSize",35);
```

Insert red circles onto the image to indicate the finder pattern locations.

```
Imsg = insertShape(Imsg, "FilledCircle", [loc, ...
     repmat(10, length(loc), 1)],"ShapeColor","red","Opacity",1);
```

Display the image.

```
imshow(Imsg)
```



Input Arguments

I — Input image

truecolor image | grayscale image

Input image, specified as a truecolor or grayscale image.

`roi` — Region of interest
four-element vector

Region of interest, specified as a four-element row vector of the form [*x*, *y*, *width*, *height*]. The rectangular ROI must be fully contained in the input image. [*x*,*y*] specifies the starting point for the ROI relative to the upper-left corner of the image.

If an image contains multiple barcodes, specifying an ROI can help the function detect a particular barcode. For more information, see “Read Barcodes in Image”.

`format` — Barcode format
`'1D'` | `'2D'` | one or more valid barcode formats

Barcode format, specified as one of these options. The table lists valid barcode formats.

- `'all'` — Use this option to specify all valid barcode formats. If you do not specify a format, the function uses this option.
- `'1D'` — Use this option to specify all valid 1-D barcode formats.
- `'2D'` — Use this option to specify all valid 2-D barcode formats.
- A character vector or string scalar of a valid format — Use this option to specify one barcode format.
- A cell array of character vectors or vector of strings of valid formats — Use this option to specify multiple barcode formats. The function prioritizes its search for specific barcode formats based on the order of the elements in this array.

Specifying a format can reduce the run time of the function by restricting the barcode search.

1-D Formats	2-D Formats
UPC-A	QR-CODE
UPC-E	DATA-MATRIX
EAN-8	AZTEC
EAN-13	PDF-417
CODE-39	
CODE-93	
CODE-128	
CODABAR	
ITF	
RSS-14	
RSS-EXPANDED	

Output Arguments

`msg` — Barcode message
string scalar

Barcode message, returned as a string scalar.

`detectedFormat` — Detected barcode format
string scalar

Detected barcode format, returned as a string scalar of one of the formats in this table.

1-D Formats	2-D Formats
`UPC-A`	`QR-CODE`
`UPC-E`	`DATA-MATRIX`
`EAN-8`	`AZTEC`
`EAN-13`	`PDF-417`
`CODE-39`	
`CODE-93`	
`CODE-128`	
`CODABAR`	
`ITF`	
`RSS-14`	
`RSS-EXPANDED`	

`loc` — Location of barcode
M-by-2 matrix | 2-by-2 matrix

Location of barcode, returned as an *M*-by-2 matrix for 2-D barcodes or a 2-by-2 matrix for 1-D barcodes. The matrix elements represent finder pattern locations.

Barcode	`loc` Value	Finder Pattern Locations
2-D	*M*-by-2 matrix. *M* represents the number of [*x*, *y*] locations of finder patterns.	 
1-D	2-by-2 matrix of the form [*x1*, *y1*;*x2*, *y2*], where each row represents the location of a finder pattern.	

Tips

- The function detects only clearly visible barcodes.
- Specifying a format can reduce the run time of the function by restricting the barcode search.
- For noisy images with unclear barcodes, use image preprocessing functions, such as `imsharpen`.

- The function detects only horizontally or vertically aligned barcodes. Use `imrotate` to correct poorly aligned barcodes.

Version History

Introduced in R2020a

R2024b: Generate optimized C/C++ code using Embedded Coder

Starting in R2024b, you can generate optimized C/C++ code using Embedded Coder.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Input argument `format` must be specified as a string scalar, character vector, or cell array of character vectors. String arrays are not supported.

You can also generate optimized C/C++ code using Embedded Coder.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`imrotate` | `imsharpen` | `ocr` | `insertText` | `insertShape` | `insertMarker` | `readAprilTag`

Topics

“Read Barcodes in Image”

balanceBoxLabels

Balance bounding box labels for object detection

Syntax

```
locationSet = balanceBoxLabels(boxLabels,blockedImages,blockSize,
numObservations)
[locationSet,bldsNew] = balanceBoxLabels(boxLabels,blockedImages,blockSize,
numObservations)
[___] = balanceBoxLabels(boxLabels,blockedImages,blockSize,numObservations,
Name=Value)
```

Description

`locationSet = balanceBoxLabels(boxLabels,blockedImages,blockSize, numObservations)` balances bounding box labels, `boxLabels`, by oversampling blocks of images containing less frequent classes, contained in the collection of blocked image objects `blockedImages`. `numObservations` is the required number of block locations, and `blockSize` specifies the block size.

`[locationSet,bldsNew] = balanceBoxLabels(boxLabels,blockedImages,blockSize, numObservations)` additionally returns bounding box label data corresponding to the balanced box labels in `locationSet`.

`[___] = balanceBoxLabels(boxLabels,blockedImages,blockSize,numObservations, Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `OverlapThreshold=0.5` specifies the overlap threshold between a bounding box and a cropping window to before boxes are clipped or discarded.

Examples

Sample Block Sets to Use in Blocked Image Object Detection

Load box labels data that contains boxes and labels for one image. The height and width of each box is 20-by-20 pixels.

```
d = load("balanceBoxLabelsData.mat");
boxLabels = d.BoxLabels;
```

Create a blocked image of size 500-by-500 pixels.

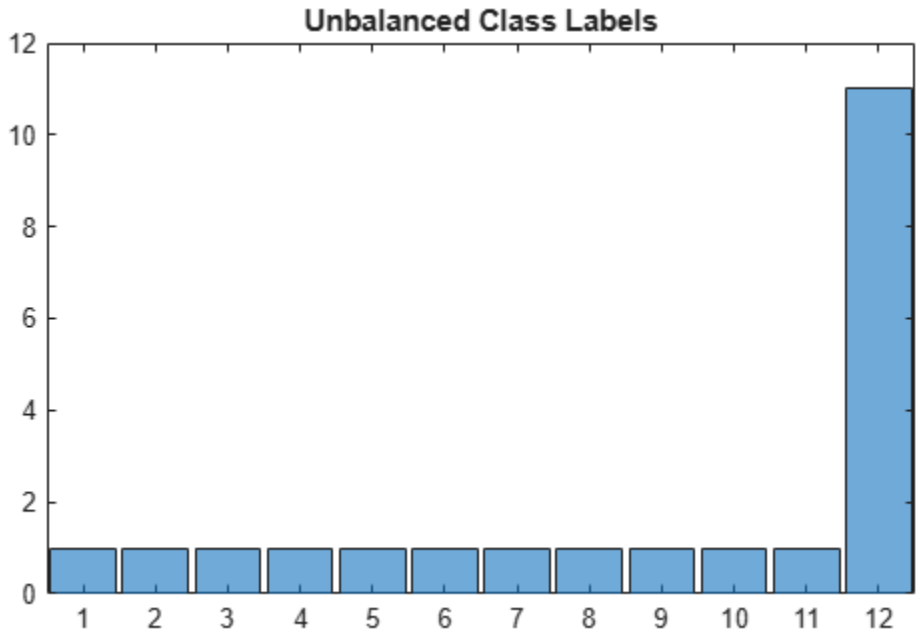
```
blockedImages = blockedImage(zeros([500 500]));
```

Choose the images size of each observation.

```
blockSize = [50 50];
```

Visualize using a histogram to identify any class imbalance in the box labels.

```
blds = boxLabelDatastore(boxLabels);
datasetCount = countEachLabel(blds);
figure
unbalancedLabels = datasetCount.Label;
unbalancedCount  = datasetCount.Count;
h1 = histogram(Categories=unbalancedLabels,BinCounts=unbalancedCount);
title("Unbalanced Class Labels")
```



Measure the distribution of box labels. If the coefficient of variation is more than 1, then there is class imbalance.

```
cvBefore = std(datasetCount.Count)/mean(datasetCount.Count)
```

```
cvBefore =
1.5746
```

Choose a heuristic value for number of observations by finding the mean of the counts of each class, multiplied by the number of classes.

```
numClasses = height(datasetCount);
numObservations = mean(datasetCount.Count) * numClasses;
```

Control the amount a box can be cut using `OverlapThreshold`. Using a lower threshold value will cut objects more at the border of a block. Increase this value to reduce the amount an object can be clipped at the border, at the expense of a less balanced box labels.

```
ThresholdValue = 0.5;
```

Balance `boxLabels` using the `balanceBoxLabels` function.

```
locationSet = balanceBoxLabels(boxLabels,blockedImages,blockSize, ...
        numObservations,OverlapThreshold=ThresholdValue);
```

```
[==================================================] 100%
Elaps[==================================================] 100%
```

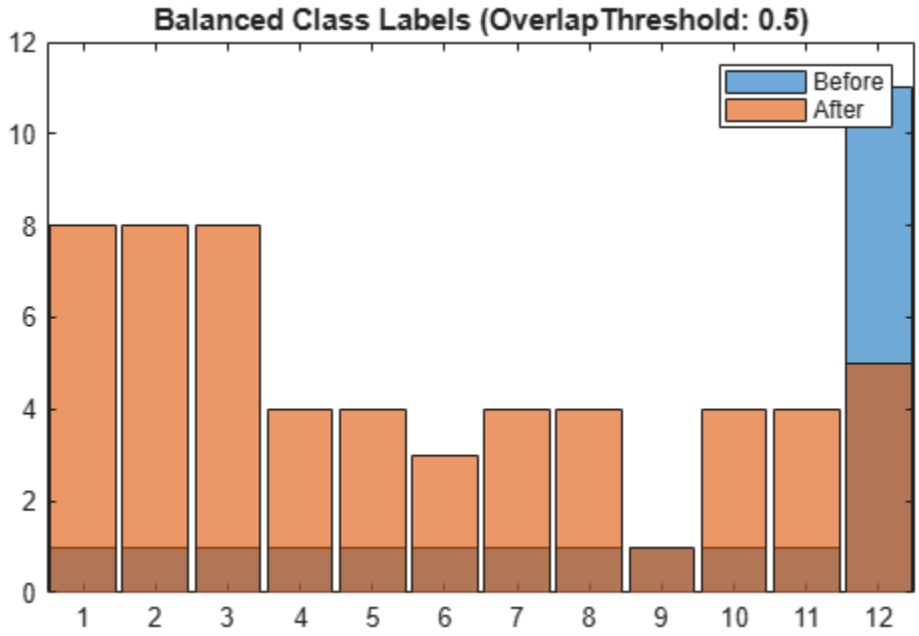
```
Elapsed time: 00:00:00
Estimated time remaining: 00:00:00
Balancing box labels complete.
```

Count the labels that are contained within the image blocks.

```
bldsBalanced = boxLabelDatastore(boxLabels,locationSet);
balancedDatasetCount = countEachLabel(bldsBalanced);
```

Overlay another histogram against the original label count to see if the box labels are balanced. If the labels appear to be not balanced by looking at the histograms, increase the value for `numObservations`.

```
hold on
balancedLabels = balancedDatasetCount.Label;
balancedCount  = balancedDatasetCount.Count;
h2 = histogram(Categories=balancedLabels,BinCounts=balancedCount);
title(h2.Parent,"Balanced Class Labels (OverlapThreshold: " + ThresholdValue + ")" )
legend(h2.Parent,["Before" "After"])
```



Measure the distribution of the new balanced box labels.

```
cvAfter = std(balancedCount)/mean(balancedCount)
```

```
cvAfter =
0.4588
```

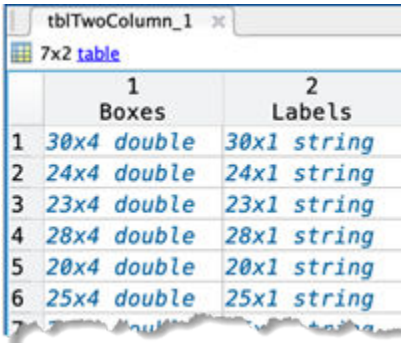
Input Arguments

`boxLabels` — Labeled bounding box data

table with two columns

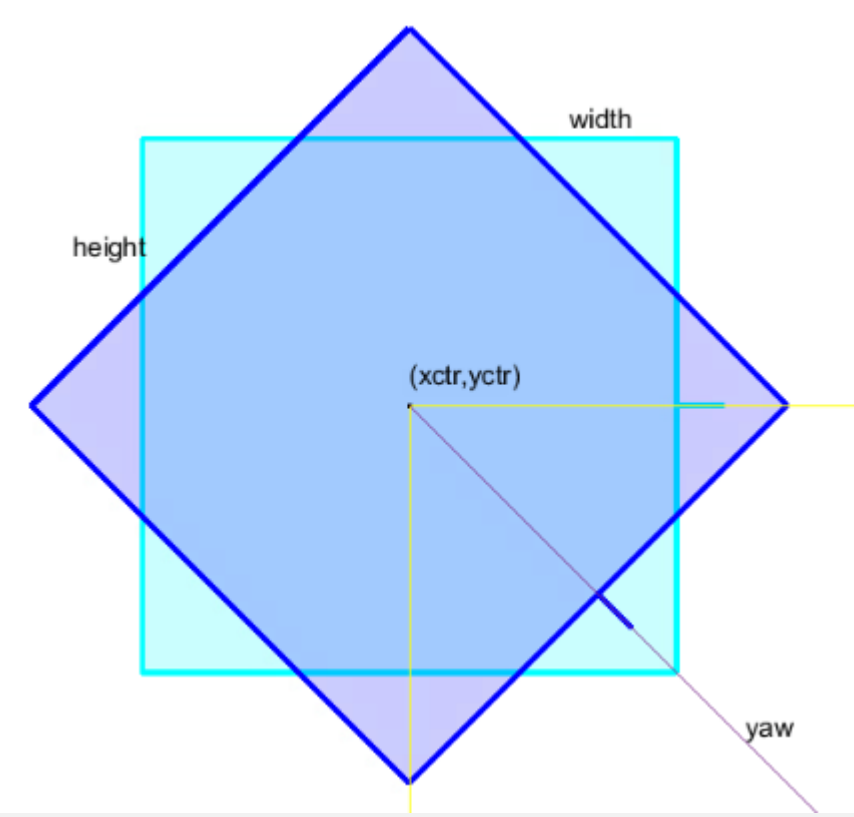
Labeled bounding box data, specified as a table with two columns.

- The first column contains either all rectangle or all rotated rectangle bounding boxes.
- The second column must be a cell vector that contains the label names corresponding to each bounding box. Each element in the cell vector must be an *M*-by-1 categorical or string vector.



The table describes the format of the bounding boxes:

Bounding Box	Description
`rectangle`	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
`rotated-rectangle`	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

To create a box label table from ground truth data,

1. Use the Image Labeler or Video Labeler app to label your ground truth. Export the labeled ground truth data to your workspace.
2. Create a bounding box label datastore using the `objectDetectorTrainingData` function.
3. You can obtain the `boxLabels` from the `LabelData` property of the box label datastore returned by `objectDetectorTrainingData`, (`blds.LabelData`).

`blockedImages` — Labeled blocked images
array of `blockedImage` objects

Labeled blocked images, specified as an array of `blockedImage` objects containing pixel label images.

`blockSize` — Block size
two-element row vector of positive integers

Block size of read data, specified as a two-element row vector of positive integers, [*numrows*,*numcols*]. The first element specifies the number of rows in the block. The second element specifies the number of columns.

`numObservations` — Number of block locations
positive integer

Number of block locations to return, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `(OverlapThreshold=0.5)` specifies the overlap threshold between a bounding box and a cropping window to before boxes are clipped or discarded.

`Levels` — Resolution level of each image
1 (default) | positive integer scalar | *B*-by-1 vector of positive integers

Resolution level of each image in the array of `blockedImage` objects, specified as a positive integer scalar or a *B*-by-1 vector of positive integers, where *B* is the length of the array of `blockedImage` objects.

`OverlapThreshold` — Overlap threshold
1 (default) | scalar in the range [0,1]

Overlap threshold, specified as a positive scalar in the range [0,1]. When the overlap between a bounding box and a cropping window is greater than the threshold, boxes in the `boxLabels` input are clipped to the image block window border. When the overlap is less than the threshold, the boxes are discarded. When you lower the threshold, part of an object can get discarded. To reduce the amount an object can be clipped at the border, increase the threshold. Increasing the threshold can also cause less-balanced box labels.

The amount of overlap between the bounding box and a cropping window is defined as.

$$area(bboxA \cap window)/area(bboxA)$$

`Verbose` — Display progress information
`true` or `1` (default) | `false` or `0`

Display progress information, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set this property to `true` to display information.

Output Arguments

`locationSet` — Balanced box labels

`blockLocationSet` object

Balanced box labels, returned as a `blockLocationSet` object. The object contains `numObservations` number of locations of balanced blocks, each of size `blockSize`.

`bldsNew` — Bounding box label data

`boxLabelDatastore` object

Bounding box label data corresponding to the balanced box labels in `locationSet`, returned as a `boxLabelDatastore` object.

Algorithms

Balancing Box Labels

To balance box labels, the function over samples classes that are less represented in the blocked image or big image. The box labels are counted across the dataset and sorted based on each class count. Each image size is split into several quadrants, based on the `blockSize` input value. The algorithm randomly picks several blocks within each quadrant with less-represented classes. The blocks without any objects are discarded. The balancing stops once the specified number of blocks are selected.

Checking for Balance

You can check the success of balancing by comparing the histograms of label count before and after balancing. You can also check the coefficient of variation value. For best results, the value should be less than the original value. For more information, see the National Institute of Standards and Technology (NIST) website, see Coefficient of Variation for more information.

Version History

Introduced in R2020a

R2025a: Option to return datastore for balanced bounding box label data

Specify the `bldsNew` output argument to return a `boxLabelDatastore` object containing bounding box label data that corresponds to the balanced box labels.

R2021a: `bigLabeledImages` argument is not recommended

Not recommended starting in R2021a

The `bigLabeledImages` argument, which supports `bigimage` objects, is not recommended. Use the `blockedImages` argument instead, which supports `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigLabeledImages` argument at this time, switch to the `blockedImages` argument to take advantage of the additional capabilities and flexibility.

To update your code, follow these steps:

- Replace `bigimage` object input with `blockedImage` object input for the second argument of this function.
- If you want to select blocks of any of the blocked images at a resolution level other than 1, then specify the '`Levels`' name-value argument. You can omit this argument when you want to select blocks from all blocked images at resolution level 1.

The table gives an example of how to update your code.

Discouraged Usage	Recommended Replacement

This example selects blocks at resolution level 1 from a `bigimage` object.
 	
`boxLabels = load('balanceBoxLabelsData.mat').boxLabels;`
`bim = bigimage(zeros([500,500]));`
`blockSize = [50 50];`
`numObservations = 20;`
`locationSet = balanceBoxLabels(boxLabels,bim, ...`
`    blockSize,numObservations);`
	Here is equivalent code, replacing the input `bigimage` object with a `blockedImage` object.
	
	`boxLabels = load('balanceBoxLabelsData.mat').boxLabel`
	`bim = blockedImage(zeros([500,500]));`
	`blockSize = [50 50];`
	`numObservations = 20;`
	`locationSet = balanceBoxLabels(boxLabels,bim, ...`
	`    blockSize,numObservations);`

See Also

Objects

`blockLocationSet` | `blockLocationsWithROI` | `blockedImage` | `blockedImageDatastore` | `boxLabelDatastore`

blockLocationsWithROI

Select image block locations that contain bounding box ROIs

Syntax

```
[locationSet,bldsNew] = blockLocationsWithROI(blset,blds)
[locationSet,bldsNew] = blockLocationsWithROI(blset,blds,Name=Value)
```

Description

`[locationSet,bldsNew] = blockLocationsWithROI(blset,blds)` selects block locations that contain bounding box ROIs specified by `blds`. The function returns the block locations as a `blockLocationSet` object, `locationSet`, and corresponding bounding boxes as a `boxLabelDatastore` object, `bldsNew`.

`[locationSet,bldsNew] = blockLocationsWithROI(blset,blds,Name=Value)` specifies options using one or more name-value arguments. For example, `OverlapThreshold=0.5` specifies the overlap threshold between a bounding box and an image block as `0.5`.

Input Arguments

`blset` — Block locations

`blockLocationSet` object

Block locations, specified as a `blockLocationSet` object. Select blocks from blocked images using the `selectBlockLocations` function.

`blds` — Bounding box label datastore

`boxLabelDatastore` object

Bounding box label datastore of blocked images, specified as a `boxLabelDatastore` object. The `boxLabelDatastore` object must contain only axis-aligned rectangle ROIs of size *M*-by-4 or rotated rectangle ROIs of size *M*-by-4-by-5. *M* is the number of axis-aligned or rotated rectangles.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `OverlapThreshold=0.5` specifies the overlap threshold between a bounding box and an image block as `0.5`.

`Classes` — Object classes

`""` (default) | cell array of character vectors | string vector | categorical vector

Object classes, specified as a cell array of character vectors, a string vector, or a categorical vector.

`OverlapThreshold` — Overlap threshold between bounding box and image block

`1` (default) | scalar in range [0,1]

Overlap threshold between a bounding box and image block, specified as a scalar in the range [0,1]. When the overlap between a bounding box ROI and an image block is greater than or equal to this threshold, the `blockLocationsWithROI` function consider the block location, specified by the `blset` argument, to contain an object. For example, when the `OverlapThreshold` is 0.5, the `blockLocationsWithROI` function, by default, selects image blocks with at least 50% overlap with an object ROI. Meanwhile, any image block that has less than a 50% overlap with an object ROI is a background image block, and is not selected.

The amount of overlap between a bounding box ROI and an image block is defined as the area of intersection between `bboxA` and an image block, divided by the minimum area between `bboxA` and the image block.

$$\frac{area(bboxA \cap block)}{\min(area(bboxA), area(block))}$$

`SelectBackgroundImageBlocks` — Background image block selection
`false` or `0` (default) | `true` or `1`

Background image block selection, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set this value to `true` or `1` to select block locations that do not contain any ROIs.

`DiscardROIsBelowThreshold` — Discard bounding box ROIs below overlap threshold
`false` or `0` (default) | `true` or `1`

Discard bounding box ROIs below the overlap threshold, specified as a numeric or logical `1` (`true`) or `0` (`false`). Set this argument as `true` or `1` to automatically remove all bounding box ROIs with an overlap threshold below the `OverlapThreshold` value from the bounding box datastore returned by the `bldsNew` argument.

Output Arguments

`locationSet` — Block locations containing object ROIs
`blockLocationSet` object

Block locations containing object ROIs, returned as a `blockLocationSet` object. By default, the `blockLocationsWithROI` function selects block locations that contain bounding box ROIs which have an overlap threshold higher than or equal to the value specified by the `OverlapThreshold` name-value argument.

Tip Specify the `SelectBackgroundImageBlocks` name-value argument as `true` to create `blockLocationSet` objects which contain only the background image blocks.

`bldsNew` — Bounding box ROIs that overlap with block locations
`boxLabelDatastore` object

Bounding box ROIs that overlap with block locations, returned as a `boxLabelDatastore` object.

Note If you specify the `SelectBackgroundImageBlocks` name-value argument as `true`, `bldsNew` returns an empty array for each background image block.

Tip To discard any bounding box ROIs with an overlap threshold below the value specified by the `OverlapThreshold` name-value argument, specify the `DiscardROIsBelowThreshold` name-value argument as `true` or `1`.

Version History

Introduced in R2025a

See Also

Objects

`blockLocationSet` | `selectBlockLocations` | `balanceBoxLabels` | `blockedImage` | `blockedImageDatastore` | `boxLabelDatastore`

Topics

“Detect Small Objects Using Tiled Training of YOLOX Network”
“Object Detection in Large Satellite Imagery Using Deep Learning”

balancePixelLabels

Balance pixel labels by oversampling block locations in large images

Syntax

```
blockLocations = balancePixelLabels(blockedImages,blockSize,numObservations)
blockLocations = balancePixelLabels(blockedImages,blockSize,numObservations,
Name,Value)
```

Description

`blockLocations = balancePixelLabels(blockedImages,blockSize,numObservations)` creates a list of block locations in the large labeled images, `blockedImages`, that result in a class balanced data set by oversampling image regions that contain less-common labels. `numObservations` is the required number of block locations, and `blockSize` specifies the block size.

A balanced dataset can produce better results when used for training workflows such as semantic segmentation in deep learning.

`blockLocations = balancePixelLabels(blockedImages,blockSize,numObservations, Name,Value)` specifies additional aspects of the selected blocks using name-value arguments.

Examples

Balance Pixel Labels in Unbalanced Data Set

Specify the location of a labeled image data set.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
labelDir = fullfile(dataDir,"buildingPixelLabels");
fileSet = matlab.io.datastore.FileSet(labelDir,FileExtensions=".png");
```

Create an array of labeled images from the data set.

```
blockedImages = blockedImage(fileSet);
```

Set the block size of the images. Assume the finest resolution level.

```
blockSize = [20 15];
```

Create a `blockedImageDatastore` from the image array.

```
blabelds = blockedImageDatastore(blockedImages,BlockSize=blockSize);
```

Count pixel label occurrences of each class. The classes in the pixel label images are not balanced.

```
pixelLabelID = [1 2 3 4];
classNames = ["sky" "grass" "building" "sidewalk"];
labelCounts = countEachLabel(blabelds, ...
    Classes=classNames,PixelLabelIDs=pixelLabelID);
```

Specify the number of block locations to sample from the data set.

```
numObservations = 2000;
```

Select the block locations from the labeled images to achieve class balancing.

```
locationSet = balancePixelLabels(blockedImages,blockSize,numObservations, ...
    Classes=classNames,PixelLabelIDs=pixelLabelID);
```

Create a `blockedImageDatastore` using the block locations after balancing.

```
blabeldsBalanced = blockedImageDatastore(blockedImages,BlockLocationSet=locationSet);
```

Recalculate the pixel label occurrences for the balanced data set.

```
labelCountsBalanced = countEachLabel(blabeldsBalanced, ...
    Classes=classNames,PixelLabelIDs=pixelLabelID);
```

Compare the original unbalanced labels and labels after label balancing.

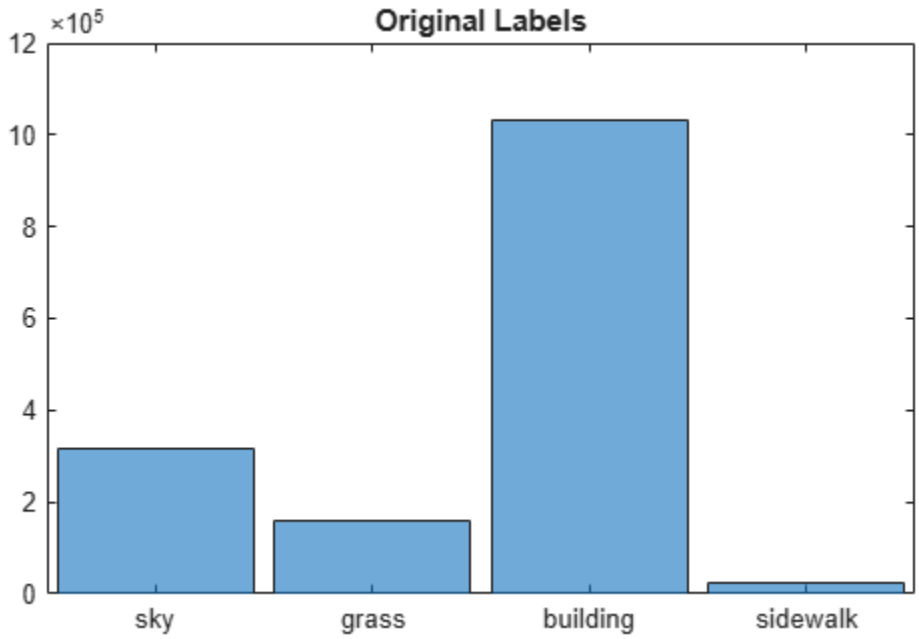
```
figure
h1 = histogram("Categories",labelCounts.Name,...
    BinCounts=labelCounts.PixelCount)
```

```
h1 =
  Histogram with properties:

              Data: [0×0 categorical]
            Values: [314849 159787 1031235 25313]
    NumDisplayBins: 4
        Categories: {'sky'  'grass'  'building'  'sidewalk'}
      DisplayOrder: 'manual'
     Normalization: 'count'
      DisplayStyle: 'bar'
         FaceColor: 'auto'
         EdgeColor: [0.1294 0.1294 0.1294]

  Show all properties
```

```
title(h1.Parent,"Original Labels")
```



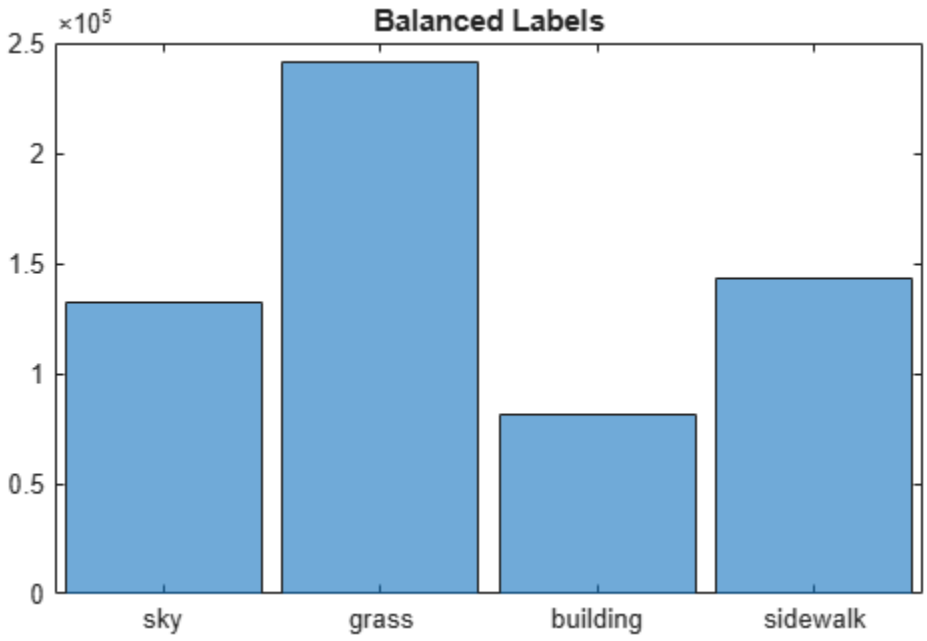

```
figure
h2 = histogram("Categories",labelCountsBalanced.Name,...
    BinCounts=labelCountsBalanced.PixelCount)
```

```
h2 =
  Histogram with properties:

              Data: [0×0 categorical]
            Values: [131906 241546 81006 143167]
    NumDisplayBins: 4
        Categories: {'sky'  'grass'  'building'  'sidewalk'}
      DisplayOrder: 'manual'
     Normalization: 'count'
      DisplayStyle: 'bar'
         FaceColor: 'auto'
         EdgeColor: [0.1294 0.1294 0.1294]

  Show all properties
```

```
title(h2.Parent,"Balanced Labels")
```



Input Arguments

`blockedImages` — Labeled blocked images

`blockedImage` object | vector of `blockedImage` objects

Labeled blocked images, specified as a `blockedImage` object or a vector of `blockedImage` objects containing pixel label images.

`blockSize` — Block size

two-element row vector of positive integers

Block size of read data, specified as a two-element row vector of positive integers, [*numrows*,*numcols*]. The first element specifies the number of rows in the block. The second element specifies the number of columns.

`numObservations` — Number of block locations

positive integer

Number of block locations to return, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Classes',classNames,'PixelLabelIDs',pixelLabelID`

`Levels` — Image resolution levels
`1` (default) | numeric scalar | integer-valued vector

Image resolution levels, specified as a numeric scalar or an integer-valued vector of the same length as the vector of `blockedImages`. If you specify a scalar value, then all blocked images supply blocks at the specified resolution level.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Classes` — Set of class names
`{}` (default) | `string` vector | cell array of `char` vectors

Set of class names, specified as a string vector or a cell array of character vectors.

You must specify this argument when `blockedImages` yields numeric data, such as when pixel label data is stored as an RGB image. Do not specify this argument when `blockedImages` yields categorical data.

Data Types: `char` | `string`

`PixelLabelIDs` — Numeric IDs that map labels to class names
`[]` (default) | numeric vector | *M*-by-3 numeric matrix

Numeric IDs that map labels to class names, specified as a vector of numeric IDs for each label or an *M*-by-3 matrix where *M* is the number of class names. The length of the vector must equal the number of class names. Each row is a three-element vector representing the RGB pixel value associated with each class name.

You must specify this argument when `blockedImages` yields numeric data, such as when pixel label data is stored as an RGB image. Do not specify this argument when `blockedImages` yields categorical data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`UseParallel` — Use new or existing pool
`false` or `0` (default) | `true` or `1`

Use a new or existing parallel pool, specified as a numeric or logical `1` (`true`) or `0` (`false`). If no parallel pool is active, then a new pool is opened based on the default parallel settings. The `DataSource` property of all input `blockedImage` objects should be valid paths on each of the parallel workers.

This syntax requires the Parallel Computing Toolbox.

Data Types: `logical`

Output Arguments

`blockLocations` — Block locations
`blockLocationSet` object

Block locations, returned as a `blockLocationSet` object. The object contains `numObservations` number of locations of balanced blocks, each of size `blockSize`.

Algorithms

To balance pixel labels, the function oversamples the minority classes in the input images. The minority class is determined by calculating the overall pixel label counts for the complete dataset. The algorithm follows these steps.

1. The images in the input image array are divided into macro blocks, which is a multiple of the `blockSize` input value.
2. The function counts pixel labels for all classes in each macro block. Then, it selects the macro block with the greatest occurrences of minority classes using weighted random selection.
3. The algorithm uses a random block location within the selected macro block to perform oversampling. The origin of the block location must always be fully within the limits of the macro block.
4. The function updates the overall label counts based on the pixel label counts of the classes found for the selected macro block.
5. The function includes the new (oversampled) classes to compute new minority class.
6. This process repeats until the number of block locations processed equals the value specified by the `numObservations` input value.

Version History

Introduced in R2020a

R2021a: `bigLabeledImages` argument is not recommended

Not recommended starting in R2021a

The `bigLabeledImages` argument, which supports `bigimage` objects, is not recommended. Use the `blockedImages` argument instead, which supports `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigLabeledImages` argument at this time, switch to the `blockedImages` argument to take advantage of the additional capabilities and flexibility.

To update your code, follow these steps:

- Replace `bigimage` object input with `blockedImage` object input for the first argument of this function.
- If you want to select blocks of any of the blocked images at a resolution level other than 1, then specify the '`Levels`' name-value argument. You can omit this argument when you want to select blocks from all blocked images at resolution level 1.
- If the blocked images yield numeric data, such as representing pixel label data as an RGB image, then specify the `'Classes'` and `'PixelLabelsIDs'` name-value arguments.

The table gives an example of how to update your code.

Discouraged Usage	Recommended Replacement

This example selects blocks at resolution level 1 from a labeled `bigimage` object.
 	
`pixelLabelIDs = [1 2 3 4];`
`classNames = ["sky" "grass" "building" "sidewalk"];`
`labelDir = fullfile(toolboxdir('vision'), ...`
`    'visiondata','buildingPixelLabels');`
`filename = fullfile(labelDir,'Label_1.png');`
`bim = bigimage(imread(filename), ...`
`    'Classes',classNames,'PixelLabelIDs',pixelLabelIDs);`
`blockSize = [20 15];`
`numObservations = 2000;`
`locationSet = balancePixelLabels(bim,1, ...`
`    blockSize,numObservations);`
	Here is equivalent code, replacing the input `bigimage` object with a `blockedImage` object.
	
	`pixelLabelIDs = [1 2 3 4];`
	`classNames = ["sky" "grass" "building" "sidewalk"];`
	`labelDir = fullfile(toolboxdir('vision'), ...`
	`    'visiondata','buildingPixelLabels');`
	`filename = fullfile(labelDir,'Label_1.png');`
	`bim = blockedImage(filename);`
	`blockSize = [20 15];`
	`numObservations = 2000;`
	`locationSet = balancePixelLabels(bim, ...`
	`    blockSize,numObservations, ...`
	`    'Classes',classNames,'PixelLabelIDs',pixelLabelIDs);`

See Also

Objects

`blockLocationSet` | `blockedImage` | `blockedImageDatastore` | `countEachLabel`

showvipblockdatatypetable

Simulink Block Data Type Support Table

Syntax

```
showvipblockdatatypetable
```

Description

`showvipblockdatatypetable` returns a table that shows characteristics for the Simulink blocks in Computer Vision Toolbox. The table provides details regarding capabilities and limitations pertaining to code generation, variable-sizing, and supported data types for all Computer Vision Toolbox blocks. An "X" in a cell indicates a given block supports the capability indicated by the column heading. Numbered footnotes included in cells, "(#)", are described below the table.

Examples

Show Block Characteristics For Computer Vision Toolbox

You can use the `showvipblockdatatypetable` function to return a table of block characteristics for the Computer Vision Toolbox.

To return the table of block characteristics, at the MATLAB® command line type:

```
showvipblockdatatypetable
```

```
Loading Computer Vision Toolbox Library.
```

The table opens in a separate window.

Version History

Introduced in R2008b

read

Read data from label datastore

Syntax

```
C = read(ds)
[C,info] = read(ds)
```

Description

`C = read(ds)` returns data from a pixel label datastore or box label datastore. Subsequent calls to the `read` function continue reading from the endpoint of the previous call.

`[C,info] = read(ds)` also returns information about the extracted data in `info`, including metadata.

Input Arguments

ds — Input datastore

`PixelLabelDatastore` object | `boxLabelDatastore` object

Input datastore, specified as a `PixelLabelDatastore` or a `boxLabelDatastore` object.

Output Arguments

C — Output data

cell array of categorical matrices | *M*-by-2 cell array | table

Output data, returned as an *M*-by-2 cell array, cell array of categorical matrices, or a table.

Datastore	Output Description
`PixelLabelDatastore`	Table with `MiniBatchSize` number of rows. For the last batch of data in the datastore, `numObservations` must be divisible by `MiniBatchSize` or `read` returns a partial batch containing all the remaining observations in the datastore.

Datastore	Output Description
`boxLabelDatastore`	*N*-by-2 or *N*-by-3 cell matrix. *N* must be less than or equal to `ReadSize(ds)`. The first column can contain data, such as point cloud data for point cloud detectors, or images for object detectors. The second column must be a cell vector that contains *M*-by-5 matrices of bounding boxes in the format[*xcenter*,*ycenter*,*width*,*height*,*yaw*]. The third column must be a cell vector that contains the label names corresponding to each bounding box. Label names are represented as an *M*-by-1 categorical vector.

You can use the `combine` function to create a datastore to use for training.

- `combine(imds,blds)` — Combine images in an `imageDatastore` with bounding boxes and labels in a `boxLabelDatastore`.
- `combine(pxds,blds)` — Combine pixel data in a `PixelLabelDatastore` with bounding boxes and labels in a `boxLabelDatastore`.

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

`info` — Information about read data
structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Datastore	Field Name	Description
`PixelLabelDatastore`	`Filename`	Fully resolved path containing the path string, name of the file, and file extension. For `PixelLabelDatastore` objects whose `ReadSize` property is greater than 1, `Filename` is a cell array of file names corresponding to each image.
	`FileSize`	Total file size, in bytes. For MAT-files, `FileSize` is the total number of key-value pairs in the file. For `PixelLabelDatastore` objects whose `ReadSize` property is greater than 1, `FileSize` is a vector of file sizes corresponding to each image.
`boxLabelDatastore`	`CurrentIndex`	Starting position of each `read` operation of the label data.

Datastore	Field Name	Description
	`ReadSize`	`ReadSize` property of the datastore.

Tips

- `read(ds)` returns an error if there is no more data in the input datastore, `ds`. Use `hasdata(ds)` with `read(ds)` to avoid the error.

Version History

Introduced in R2017b

See Also

`pixelLabelDatastore` | `boxLabelDatastore` | `readByIndex` | `readall` | `combine`

Topics

“Datastores for Deep Learning” (Deep Learning Toolbox)

shuffle

Return shuffled version of label datastore

Syntax

```
shuffle(ds)
```

Description

`shuffle(ds)` returns a randomly shuffled version of the input datastore.

Examples

Create Randomly Shuffled Datastore

Set the location of the image and pixel label data.

```
dataDir = fullfile(toolboxdir('vision'), 'visiondata');
pxDir = fullfile(dataDir, 'buildingPixelLabels');
```

Create a pixel label datastore.

```
classNames = ["sky" "grass" "building" "sidewalk"];
pixelLabelID = [1 2 3 4];
pxds = pixelLabelDatastore(pxDir, classNames, pixelLabelID);
```

Create a new randomly shuffled datastore.

shpxds = shuffle(pxds);

Input Arguments

ds — Datastore with labeled data

`pixelLabelDatastore` | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or a `boxLabelDatastore` object.

Version History

Introduced in R2020a

See Also

Functions

`objectDetectorTrainingData` | `trainnet`

Objects
`boxLabelDatastore` | `pixelLabelDatastore`

preview

Read first row of data in datastore

Syntax

```
preview(ds)
```

Description

`preview(ds)` returns the first row of data in the input datastore.

Examples

Preview Data from Datastore

Load training data that contains bounding boxes with labels for a vehicles class.

```
data = load('vehicleTrainingData.mat');
trainingData = data.vehicleTrainingData;
```

Add the fullpath to the local vehicle data folder.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
blds = boxLabelDatastore(trainingData(:,2:end));
```

Preview the data from the box label datastore.

```
preview(blds)
```

```
ans=1×2 cell array
    {[126 78 20 16]}    {[vehicle]}
```

Input Arguments

ds — Datastore with labeled data

`boxLabelDatastore` object | `pixelLabelDatastore`

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `boxLabelDatastore` or a `pixelLabelDatastore` object.

Version History

Introduced in R2020a

See Also

Functions
`objectDetectorTrainingData`

Objects
`datastore` | `boxLabelDatastore` | `pixelLabelDatastore`

progress

Percentage of data read from a datastore

Syntax

```
progress(ds)
```

Description

`progress(ds)` returns a fraction between 0.0 and 1.0 to indicate the percentage of data read from the datastore.

Examples

Read Datastore and Check Progress

Load training data that contains bounding boxes with labels for vehicles.

```
data = load('vehicleTrainingData.mat');
trainingData = data.vehicleTrainingData;
```

Add fullpath to the local vehicle data folder.

```
dataDir = fullfile(toolboxdir('vision'), 'visiondata');
trainingData.imageFilename = fullfile(dataDir,...
                                      trainingData.imageFilename);
blds = boxLabelDatastore(trainingData(:,2:end));
```

Read a couple of times, before looking at the progress.

```
read(blds);
read(blds);
```

See the progress made by the boxLabelDatastore.

```
preview(blds)
```

```
ans=1×2 cell array
    {[126 78 20 16]}    {[vehicle]}
```

Input Arguments

ds — Datastore with labeled data

`boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `boxLabelDatastore` object.

Version History

Introduced in R2019b

See Also

Functions

`objectDetectorTrainingData`

Objects

`datastore` | `boxLabelDatastore`

estimateAnchorBoxes

Estimate anchor boxes for deep learning object detectors

Syntax

```
anchorBoxes = estimateAnchorBoxes(trainingData,numAnchors)
[anchorBoxes,meanIoU] = estimateAnchorBoxes(trainingData,numAnchors)
```

Description

`anchorBoxes = estimateAnchorBoxes(trainingData,numAnchors)` estimates the specified number of anchor boxes using the training data.

`[anchorBoxes,meanIoU] = estimateAnchorBoxes(trainingData,numAnchors)` additionally returns the mean intersection-over-union (IoU) value of the anchor boxes in each cluster.

Examples

Estimate Anchor Boxes for YOLO v2 Object Detection Network

This example shows how to estimate anchor boxes using a table containing the training data. The first column contains the training images and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Create a `boxLabelDatastore` object using the labeled bounding boxes from the training data.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Specify the class names using the labels from the training data.

```
classes = trainingData.Properties.VariableNames(2:end);
```

Estimate the anchor boxes using the `boxLabelDatastore` object.

```
numAnchors = 5;
anchorBoxes = estimateAnchorBoxes(blds,numAnchors);
```

Specify the image size.

```
inputImageSize = [128 228 3];
```

Use a pretrained ResNet-50 network as a base network for the YOLO v2 network.

```
baseNet = imagePretrainedNetwork("resnet50");
```

Specify the network layer to use for feature extraction. You can use the `analyzeNetwork` function to see all the layer names in a network.

```
featureLayer = "activation_49_relu";
```

Create the YOLO v2 object detection network.

```
detector = yolov2ObjectDetector(baseNet,classes,anchorBoxes, ...
    DetectionNetworkSource=featureLayer)
```

```
detector =
  yolov2ObjectDetector with properties:

                  Network: [1×1 dlnetwork]
                InputSize: [224 224 3]
        TrainingImageSize: [224 224]
              AnchorBoxes: [5×2 double]
               ClassNames: vehicle
    ReorganizeLayerSource: ''
              LossFactors: [5 1 1 1]
                ModelName: ''
```

Visualize the network using the network analyzer.

```
analyzeNetwork(detector.Network)
```

Estimate Anchor Boxes from Training Data

Anchor boxes are important parameters of deep learning object detectors such as Faster R-CNN and YOLO v2. The shape, scale, and number of anchor boxes impact the efficiency and accuracy of the detectors.

For more information, see “Anchor Boxes for Object Detection”.

Load Training Data

Load the vehicle dataset, which contains 295 images and associated box labels.

```
data = load('vehicleTrainingData.mat');
vehicleDataset = data.vehicleTrainingData;
```

Add the full path to the local vehicle data folder.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
vehicleDataset.imageFilename = fullfile(dataDir,vehicleDataset.imageFilename);
```

Display the data set summary.

```
summary(vehicleDataset)
```

```
Variables:

    imageFilename: 295×1 cell array of character vectors

    vehicle: 295×1 cell
```

Visualize Ground Truth Box Distribution

Visualize the labeled boxes to better understand the range of object sizes present in the data set.

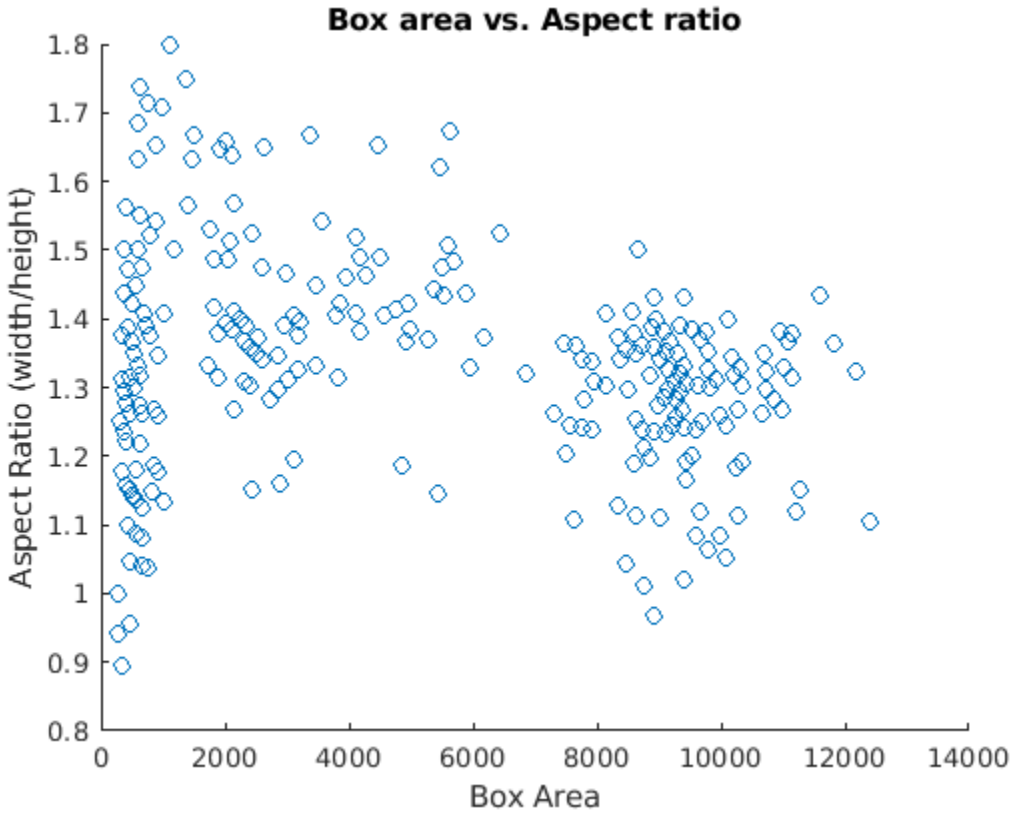
Combine all the ground truth boxes into one array.

```
allBoxes = vertcat(vehicleDataset.vehicle{:});
```

Plot the box area versus the box aspect ratio.

```
aspectRatio = allBoxes(:,3) ./ allBoxes(:,4);
area = prod(allBoxes(:,3:4),2);

figure
scatter(area,aspectRatio)
xlabel("Box Area")
ylabel("Aspect Ratio (width/height)");
title("Box Area vs. Aspect Ratio")
```



The plot shows a few groups of objects that are of similar size and shape, However, because the groups are spread out, manually choosing anchor boxes is difficult. A better way to estimate anchor boxes is to use a clustering algorithm that can group similar boxes together using a meaningful metric.

Estimate Anchor Boxes

Estimate anchor boxes from training data using the `estimateAnchorBoxes` function, which uses the intersection-over-union (IoU) distance metric.

A distance metric based on IoU is invariant to the size of boxes, unlike the Euclidean distance metric, which produces larger errors as the box sizes increase [1]. In addition, using an IoU distance metric

leads to boxes of similar aspect ratios and sizes being clustered together, which results in anchor box estimates that fit the data.

Create a `boxLabelDatastore` using the ground truth boxes in the vehicle data set. If the preprocessing step for training an object detector involves resizing of the images, use `transform` and `bboxresize` to resize the bounding boxes in the `boxLabelDatastore` before estimating the anchor boxes.

```
trainingData = boxLabelDatastore(vehicleDataset(:,2:end));
```

Select the number of anchors and estimate the anchor boxes using `estimateAnchorBoxes` function.

```
numAnchors = 5;
[anchorBoxes,meanIoU] = estimateAnchorBoxes(trainingData,numAnchors);
anchorBoxes
```

```
anchorBoxes = 5×2

    21    27
```

Choosing the number of anchors is another training hyperparameter that requires careful selection using empirical analysis. One quality measure for judging the estimated anchor boxes is the mean IoU of the boxes in each cluster. The `estimateAnchorBoxes` function uses a *k*-means clustering algorithm with the IoU distance metric to calculate the overlap using the equation, `1 - bboxOverlapRatio(allBoxes,boxInCluster)`.

```
meanIoU
```

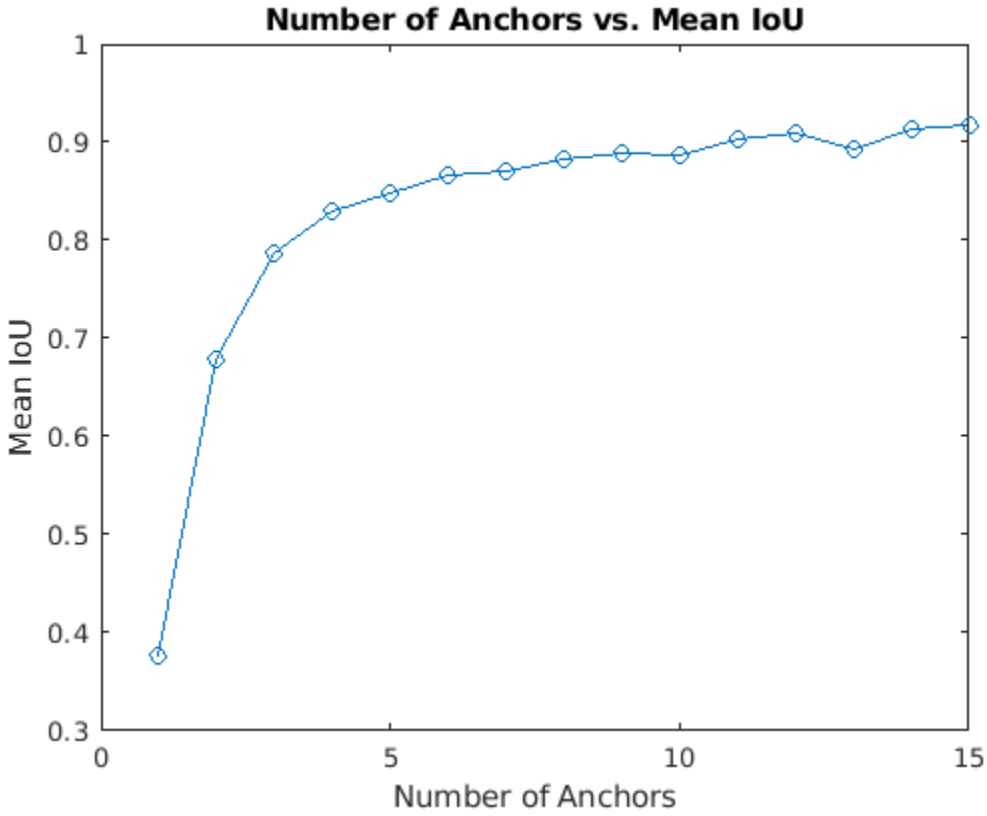
```
meanIoU =
0.8411
```

The mean IoU value greater than 0.5 ensures that the anchor boxes overlap well with the boxes in the training data. Increasing the number of anchors can improve the mean IoU measure. However, using more anchor boxes in an object detector can also increase the computation cost and lead to overfitting, which results in poor detector performance.

Sweep over a range of values and plot the mean IoU versus number of anchor boxes to measure the tradeoff between number of anchors and mean IoU.

```
maxNumAnchors = 15;
meanIoU = zeros([maxNumAnchors,1]);
anchorBoxes = cell(maxNumAnchors, 1);
for k = 1:maxNumAnchors
    % Estimate anchors and mean IoU.
    [anchorBoxes{k},meanIoU(k)] = estimateAnchorBoxes(trainingData,k);
end

figure
plot(1:maxNumAnchors,meanIoU,'-o')
ylabel("Mean IoU")
xlabel("Number of Anchors")
title("Number of Anchors vs. Mean IoU")
```



Using two anchor boxes results in a mean IoU value greater than 0.65, and using more than 7 anchor boxes yields only marginal improvement in mean IoU value. Given these results, the next step is to train and evaluate multiple object detectors using values between 2 and 6. This empirical analysis helps determine the number of anchor boxes required to satisfy application performance requirements, such as detection speed, or accuracy.

Input Arguments

`trainingData` — Labeled ground truth images

datastore | table

Labeled ground truth images, specified as a datastore or a table.

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with two or three columns.

  When the output contains two columns, the first column must contain bounding boxes, and the second column must contain labels, {*boxes*,*labels*}.

  When the output contains three columns, the second column must contain the bounding boxes, and the third column must contain the labels, {*data*,*boxes*,*labels*}. In this case, the first column can contain any type of data. For example, the first column can contain images or point cloud data. The last column, *labels*, must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that specifies a single object class such as *vehicle*, *flower*, or *stop sign* in the form of an axis-aligned or a rotated rectangle bounding box. The columns must be only axis-aligned or rotated rectangle bounding boxes. For example, column 4 cannot specify axis-aligned boxes if column 3 specifies rotated rectangles. All the columns must be of the same bounding box type.

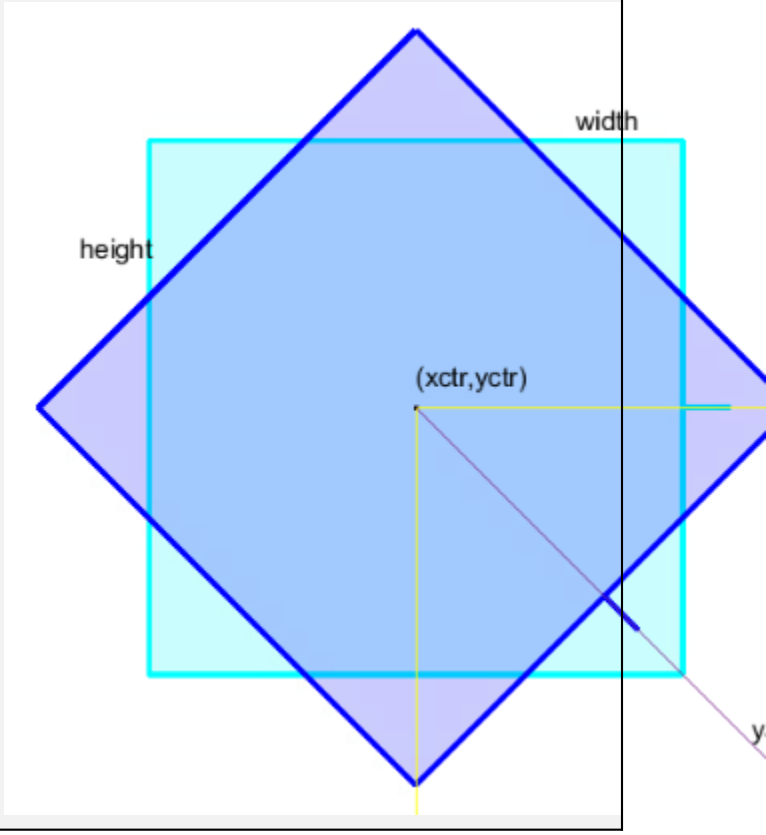
  Note When the training data is specified using a table, the `estimateAnchorBoxes` function checks these conditions

  - Axis-aligned bounding box values must be integers. Otherwise, the function automatically rounds each noninteger values to its nearest integer.
  - The bounding box must not be empty and must be within the image region. While training the network, the function ignores empty bounding boxes and bounding boxes that lie partially or fully outside the image region.

- When bounding boxes are rotated rectangles, the `estimateAnchorBoxes` function uses only the location, width, and height values of the bounding box to form similar axis-aligned anchor boxes.

The table describes the format of bounding boxes.

Bounding Box	***Description***
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	*Description*
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

`numAnchors` — Number of anchor boxes
integer

Number of anchor boxes for the function to return, specified as an integer.

Output Arguments

`anchorBoxes` — Anchor boxes
N-by-2 matrix

Anchor boxes, returned as an *N*-by-2 matrix, where *N* is the number of anchor boxes and each entry has the format [*height*, *width*]. Use `numAnchors` to specify the number of anchor boxes.

`meanIoU` — Distance metric
scalar

Distance metric, returned as a scalar value. The distance metric provides the mean intersection-over-union (IoU) value of the anchor boxes in each cluster. To ensure anchor boxes overlap well with the boxes in the training data, the `meanIoU` value must be greater than 0.5. The *k*-means clustering algorithm uses the IoU distance metric to calculate the overlap using the equation `1-bboxOverlapRatio(box1,box2)`.

Version History

Introduced in R2019b

R2023b: Rotated rectangle support for bounding box

Input can be a rotated rectangle of the form [*xctr*, *yctr*, *width*, *height*, *yaw*].

See Also

Objects
`boxLabelDatastore` | `yolov2ObjectDetector` | `yolov3ObjectDetector` | `yolov4ObjectDetector`

Topics
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Training Data for Object Detection and Semantic Segmentation”

caliper

Measure distance in image data using caliper tool

Syntax

```
measurementData = caliper(I,linePosition)
measurementData = caliper(I,linePosition,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`measurementData = caliper(I,linePosition)` measures distance in a grayscale or RGB input image, `I`, using a caliper tool by positioning a profile scan line at the position `linePosition`.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`measurementData = caliper(I,linePosition,Name=Value)` specifies options using one or more name-value arguments. For example, `Width=10` specifies the width of the profile scan as 10 pixels.

Examples

Measure Distance Between Object Edges Using Caliper

Load an image into the workspace.

```
I = imread("coins.png");
```

Display the image using the `imageshow` function.

```
imageshow(I)
```



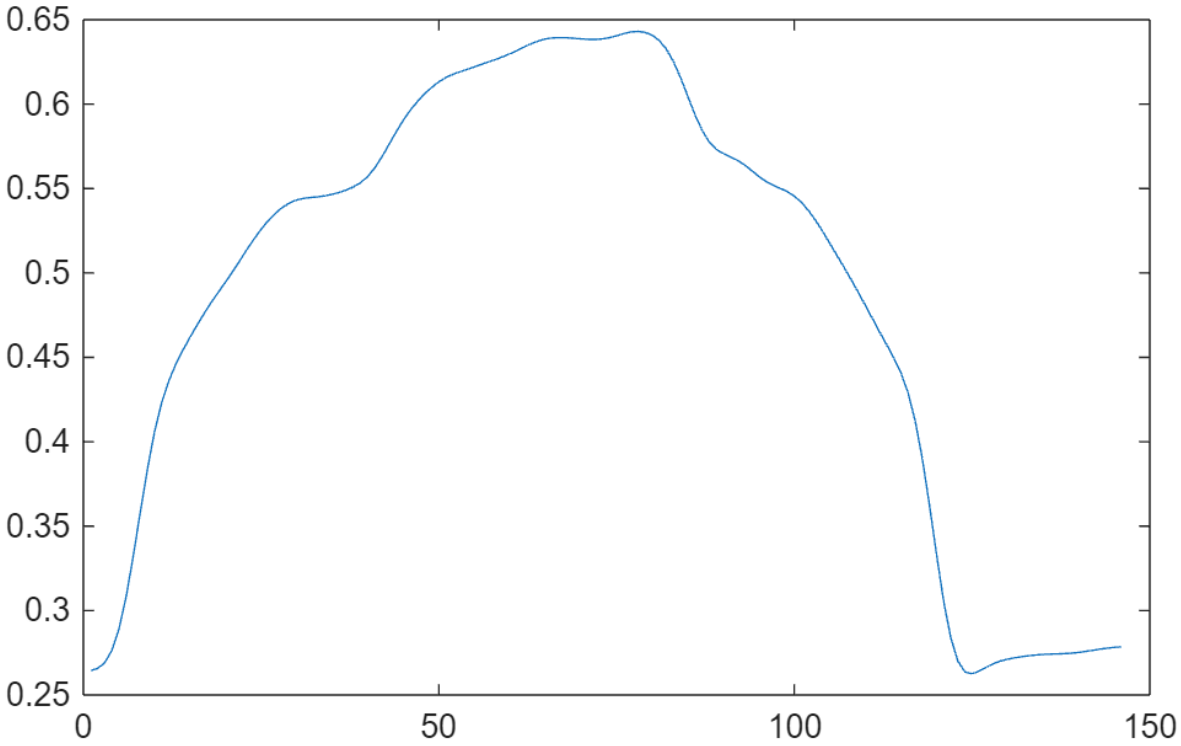
Measure the distance between the edges of the darkest nickel coin using the `caliper` function. Specify the start and end point of the profile scan line as `[171 89; 218 131]`. To detect the coin edges, specify a gradient threshold lower than the maximum intensity gradient across the scan length using the `GradientThreshold` name-value argument. Since the coin contains two edge pairs, the function returns the distance between them, or the coin width, as `IntraEdgeDistance` value.

```
measureOut = caliper(I,[158 89; 218 131],GradientThreshold=0.04)
```

```
measureOut = struct with fields:
    InterEdgeDistance: [1×0 double]
    IntraEdgeDistance: 56.4544
        GradientValue: [0.0520 -0.0579]
             Distance: [3.4186 59.8729]
          ProfileData: [146×1 double]
```

Display the intensity gradient value along the profile scan line.

```
figure
plot(measureOut.ProfileData)
```



Input Arguments

`I` — Input image

H-by-*W* numeric matrix | *H*-by-*W*-by-3 numeric array

Input image for which to perform the caliper measurement, specified as one of these values:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
RGB	*H*-by-*W*-by-3 numeric array

`linePosition` — Profile line position

2-by-2 numeric matrix

Profile line position, in pixels, specified as a 2-by-2 numeric matrix of the form $[x_1\ y_1; x_2\ y_2]$, where the $[x_1\ ,\ y_1]$ and $[x_2\ ,y_2]$ are coordinates of the start point and end point of the line, respectively.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `caliper(I,linePosition,Width=10)` specifies the width of the profile scan as 10 pixels.

`Width` — Profile scan width
`50` (default) | positive scalar

Profile scan width, specified as a positive scalar. The width is the area within which the `caliper` function computes profile scan lines measurements. The function scans in lines orthogonal to the profile centerline and using these measurements, computes a mean 1-D profile for gradient detection.

`EdgeTransition` — Edge transition type
`"both"` (default) | `"rising"` | `"falling"`

Edge transition type used in edge detection, specified as one of these options:

- `"rising"` — Detect darker-to-brighter edge transitions.
- `"falling"` — Detect brighter-to-darker edge transitions.
- `"both"` — Detect both brighter-to-darker and darker-to brighter-edge transitions.

The edge transition is relative to the direction of the scan line, from start point to end point.

`GradientThreshold` — Gradient threshold selection
`0.1` (default) | positive scalar in range `[0, 1]`

Gradient threshold for a gradient to be considered an edge, specified as a positive scalar in range `[0, 1]`. The gradient at each scan point is the normalized average gradient across the profile line width, `Width`. The `caliper` function evaluates this value at each point along the scan line. Increase the gradient threshold value to detect fewer false positive edges.

`MeasurementMode` — Measurement mode
`"edge-pair"` (default) | `"single-edge"`

Measurement mode used to compute edge data, specified as `"edge-pair"` or `"single-edge"`. This argument determines the edge type to detect, and how the `measurementData` argument outputs distance.

Value	Edge Detection Mode	`measurementData` Distance Output
`"edge-pair"`	Detect edge pairs of the type specified by the `EdgeTransition` name-value argument. When the profile scan line spans multiple edges, the caliper tool pairs each starting edge with the spatially closest ending edge to form a pair.	`measurementData` populates the `InterEdgeDistance` and `IntraEdgeDistance` fields with distance information. • `InterEdgeDistance` is the distance between the end of an edge pair and the start of the next edge pair. • `IntraEdgeDistance` is the distance between the start and end of each edge pair.

Value	Edge Detection Mode	measurementData Distance Output
"single-edge"	Detect single edges of the type specified by the EdgeTransition name-value argument.	measurementData populates the Distance field with distance information. Distance is the distance from the start point of the profile line to the position where the function detects the gradients.

Sigma — Standard deviation for Gaussian smoothing of 1-D profile data
1.0 (default) | positive scalar

Standard deviation for Gaussian smoothing of the 1-D profile data, specified as a positive scalar. Increase this value to increase the amount of Gaussian smoothing and reduce measurement noise at the possible expense of lower accuracy in detecting edge positions.

Transformation — Geometric transformation for input image data
affinetform2d object (default) | imref2d object | rigidtform2d object | simtform2d object | transltform2d object

Geometric transformation for the input image data that defines a 2-D world coordinate system, specified as an affinetform2d object, an imref2d object, a rigidtform2d object, a simtform2d object, or a transltform2d object.

Output Arguments

measurementData — Caliper measurement data
structure

Caliper measurement data, returned as a structure with these fields:

MeasurementData Field	Description
Distance	Distance between the first point of the profile line to the position of each detected edge, stored as a numeric array.
IntraEdgeDistance	Distance between the start and end of each edge pair, stored as a numeric array.
InterEdgeDistance	Distance between the end of an edge pair and the start of the next edge pair, stored as a numeric array.
ProfileData	1-D profile data that determines detected edges, stored as an *M*-element numeric vector. *M* is the integer length of the profile scan, in pixels, and each element of the vector is the mean gradient value at each scan point.

MeasurementData Field	Description
`GradientValue`	Gradient values corresponding to each edge detection, stored as a row vector or *N*-by-2 numeric matrix, depending on the value of the `measurementMode` name-value argument. • `"single-edge"` – `GradientValue` is a row vector containing gradient values associated with each detected edge. • `"edge-pair"` – `GradientValue` is an *N*-by-2 numeric matrix, where each row contains the gradient values of the start point and the end point of the corresponding edge pair. *N* is the number of detected edge pairs.

Version History

Introduced in R2025a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `EdgeTransition` and `MeasurementMode` name-value arguments must be compile-time constants. Code generation does not support cell arrays for these name-value arguments.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `EdgeTransition` and `MeasurementMode` name-value arguments must be compile-time constants. Code generation does not support cell arrays for these name-value arguments.

See Also

`uicaliper` | `affinetform2d` | `imref2d` | `rigidtform2d` | `simtform2d` | `transltform2d`

deeplabv3plusLayers

(Removed) Create DeepLab v3+ convolutional neural network for semantic image segmentation

Note `deeplabv3plusLayers` has been removed. Use the `deeplabv3plus` function instead. For more information, see “Version History”.

Syntax

```
layerGraph = deeplabv3plusLayers(imageSize,numClasses,network)
layerGraph = deeplabv3plusLayers( ___ ,"DownsamplingFactor",value)
```

Description

`layerGraph = deeplabv3plusLayers(imageSize,numClasses,network)` returns a DeepLab v3+ layer with the specified base network, number of classes, and image size.

`layerGraph = deeplabv3plusLayers( ___ ,"DownsamplingFactor",value)` additionally sets the downsampling factor (output stride) [1] to either 8 or 16. The downsampling factor sets the amount the encoder section of DeepLab v3+ downsamples the input image.

Examples

Create DeepLab v3+ Network Based on ResNet-18

Create a DeepLab v3+ network based on ResNet-18.

```
imageSize = [480 640 3];
numClasses = 5;
network = "resnet18";
lgraph = deeplabv3plusLayers(imageSize,numClasses,network, ...
             "DownsamplingFactor",16);
```

Display the network.

```
analyzeNetwork(lgraph)
```

Input Arguments

`imageSize` — Network input image size

2-element vector | 3-element vector

Network input image size, specified as a:

- 2-element vector in the format [*height*, *width*].
- 3-element vector in the format [*height*, *width*, 3]. The third element, 3, corresponds to RGB.

`numClasses` — Number of classes

integer greater than 1

Number of classes for network to classify, specified as an integer greater than 1.

`network` — Base network
`'resnet18'` | `'resnet50'` | `'mobilenetv2'` | `'xception'` | `'inceptionresnetv2'`

Base network, specified as `resnet18`, `resnet50`, `mobilenetv2`, `xception`, or `inceptionresnetv2`. You must install the corresponding network add-on.

Output Arguments

`layerGraph` — DeepLab v3+ network
`layerGraph` object

DeepLab v3+ network, returned as a `layerGraph` object for semantic image segmentation. The network uses encoder-decoder architecture, dilated convolutions, and skip connections to segment images. You must use the `trainNetwork` function (requires Deep Learning Toolbox) to train the network before you can use the network for semantic segmentation.

Algorithms

- When you use either the `xception` or `mobilenetv2` base networks to create a DeepLab v3+ network, depth separable convolutions are used in the atrous spatial pyramid pooling (ASPP) and decoder subnetworks. For all other base networks, convolution layers are used.
- This implementation of DeepLab v3+ does not include a global average pooling layer in the ASPP.

Version History

Introduced in R2019b

R2026a: Removed
Errors starting in R2026a

The `deeplabv3plusLayers` function has been removed. Use the `deeplabv3plus` function instead.

R2024b: Warns
Warns starting in R2024b

The `deeplabv3plusLayers` function issues a warning that it will be removed in a future release. Use the `deeplabv3plus` function instead. The `deeplabv3plus` function returns a `dlnetwork` object, which has these advantages over `layerGraph` objects:

- `dlnetwork` objects support a wider range of network architectures which you can then easily train using the `trainnet` function or import from external platforms.
- `dlnetwork` objects provide more flexibility. They have wider support with current and upcoming Deep Learning Toolbox functionality.
- `dlnetwork` objects provide a unified data type that supports network building, prediction, built-in training, compression, and custom training loops.
- `dlnetwork` training and prediction is typically faster than `DAGNetwork` and `SeriesNetwork` training and prediction.

To update your code, replace instances of the `deeplabv3plusLayers` with the `deeplabv3plus` function. You do not need to change the input arguments.

Discouraged Usage	Recommended Replacement

This example uses the `deeplabv3plusLayers` function to create a DeepLab v3+ network, returned as a `LayerGraph` object.
 	
`imageSize = [480 640 3];`
`numClasses = 5;`
`network = "resnet18";`
`deepLabNetwork = deeplabv3plusLayers(imageSize,numClasses,network, ...`
`    DownsamplingFactor=16);`
	Here is equivalent code that instead uses the `deeplabv3plus` function to create a DeepLab v3+ network, returned as a `dlnetwork` object.
	
	`imageSize = [480 640 3];`
	`numClasses = 5;`
	`network = "resnet18";`
	`deepLabNetwork = deeplabv3plus(imageSize,numClasses,n`
	`    DownsamplingFactor=16);`

R2024a: To be removed
Not recommended starting in R2024a

The `deeplabv3plusLayers` function will be removed in a future release.

References

[1] Chen, L., Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation." *Computer Vision — ECCV 2018*, 833-851. Munic, Germany: ECCV, 2018.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation, you must first create a DeepLab v3+ network by using the `deeplabv3plusLayers` function. Then, use the `trainNetwork` function on the resulting `lgraph` object to train the network for segmentation. Once the network is trained and evaluated, you can generate code for the deep learning network object using GPU Coder.

See Also

`deeplabv3plus` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

fasterRCNNLayers

(Not recommended) Create a faster R-CNN object detection network

Note `fasterRCNNLayers` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector For more information, see "Version History".

Syntax

```
lgraph = fasterRCNNLayers(inputImageSize,numClasses,anchorBoxes,network)
lgraph = fasterRCNNLayers(inputImageSize,numClasses,anchorBoxes,network,
featureLayer)
lgraph = fasterRCNNLayers( ___ ,Name=Value)
```

Description

`lgraph = fasterRCNNLayers(inputImageSize,numClasses,anchorBoxes,network)` returns a Faster R-CNN network as a `layerGraph` object. A Faster R-CNN network is a convolutional neural network based object detector. The detector predicts the coordinates of bounding boxes, objectness scores, and classification scores for a set of anchor boxes. To train the created network, use the `trainFasterRCNNObjectDetector` function. For more information, see "Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN".

`lgraph = fasterRCNNLayers(inputImageSize,numClasses,anchorBoxes,network, featureLayer)` returns the object detection network based on the specified `featureLayer` of the network. Use this syntax when you specify the network as a `SeriesNetwork`, `DAGNetwork`, or `layerGraph`. object.

`lgraph = fasterRCNNLayers( ___ ,Name=Value)` returns the object detection network with optional input properties specified by one or more name-value arguments.

Using this function requires Deep Learning Toolbox.

Examples

Create a Faster R-CNN Object Detection Network

Specify the image size.

```
inputImageSize = [224 224 3];
```

Specify the number of objects to detect.

```
numClasses = 1;
```

Use a pretrained ResNet-50 network as the base network for the Faster R-CNN network. You must download the `resnet50` (Deep Learning Toolbox) support package.

```
network = "resnet50";
```

Specify the network layer to use for feature extraction. You can use the `analyzeNetwork` (Deep Learning Toolbox) function to see all the layer names in a network.

```
featureLayer = "activation_40_relu";
```

Specify the anchor boxes. You can also use the `estimateAnchorBoxes` function to estimate anchor boxes from your training data.

```
anchorBoxes = [64,64; 128,128; 192,192];
```

Create the Faster R-CNN object detection network.

```
lgraph = fasterRCNNLayers(inputImageSize,numClasses,anchorBoxes, ...
                          network,featureLayer)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'input_1'}
    OutputNames: {'rcnnClassification'  'boxDeltas'  'rpnBoxDeltas'  'rpnClassification'}
         Layers: [188×1 nnet.cnn.layer.Layer]
    Connections: [205×2 table]
```

Visualize the network using the network analyzer.

```
analyzeNetwork(lgraph)
```

Input Arguments

`inputImageSize` — Network input image size

3-element vector

Network input image size, specified as a 3-element vector in the format [*height*, *width*, *depth*]. *depth* is the number of image channels. Set *depth* to `3` for RGB images, to `1` for grayscale images, or to the number of channels for multispectral and hyperspectral images.

`numClasses` — Number of classes

integer greater than 0

Number of classes for the network to classify, specified as an integer greater than 0.

`anchorBoxes` — Anchor boxes

M-by-2 matrix

Anchor boxes, specified as an *M*-by-2 matrix of *M* anchor boxes in the format [*height*, *width*]. Anchor boxes are determined based on the scale and aspect ratio of objects in the training data set. For example, if an object is localized by a square window, then you can set the size of the anchor boxes to `[64 64;128 128]`.

`network` — Pretrained classification network

`SeriesNetwork` object | `DAGNetwork` object | `LayerGraph` object | `"lexnet"` | `"vgg16"` | `"vgg19"` | `"resnet18"` | `"resnet50"` | `"resnet101"` | `"inceptionv3"` | `"googlenet"` | `"inceptionresnetv2"` | `"mobilenetv2"` | `"squeezenet"`

Pretrained classification network, specified as a `SeriesNetwork`, `DAGNetwork`, or `layerGraph`, or as on of the following:

- `'alexnet'`
- `'vgg16'`
- `'vgg19'`
- `'resnet18'`
- `'resnet50'`
- `'resnet101'`
- `'inceptionv3'`
- `'googlenet'`
- `'inceptionresnetv2'`
- `'squeezenet'`
- `'mobilenetv2'`

When you specify the network as a `SeriesNetwork` object, a `DAGNetwork` object, or by name, the function transforms the network into a Faster R-CNN network. It transforms the network by adding a region proposal network (RPN), and ROI max pooling layer, and new classification and regression layers to support object detection.

`featureLayer` — Feature extraction layer
character vector | string scalar

Feature extraction layer, specified as a character vector or a string scalar. Use one of the deeper layers in the network you specify. You can use the `analyzeNetwork` function to view the names of the layers in the input network.

Note You can specify any network layer except the fully connected layer as the feature layer.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: Set `ROIMaxPoolingLayer="auto"` to specify the function to inserts a new ROI max pooling layer after the feature extraction layer when the layer next to the feature extraction layer is not a max pooling layer.

`ROIMaxPoolingLayer` — ROI max pooling layer
`"auto"` (default) | `"insert"` | `"replace"`

ROI max pooling layer, specified as a `"auto"`, `"insert"`, or `"replace"`. You can specify whether a `roiMaxPooling2dLayer` replaces the pooling layer or follows the feature extraction layer.

If you select `"auto"`, the function:

- Inserts a new ROI max pooling layer after the feature extraction layer when the layer next to the feature extraction layer is not a max pooling layer.

- Replaces the current pooling layer after the feature extraction layer with an ROI max pooling layer.

`ROIOutputSize` — ROI max pooling layer output size
`"auto"` (default) | 2-element vector

ROI max pooling layer output size, specified as `"auto"` or a 2-element vector of positive integers. When you set the value to `"auto"`, the function determines the output size based on the `ROIMaxPoolingLayer` property. It uses the output size of the feature extraction layer or the pooling layer following the feature extraction layer.

Note If the input image dimensions specified by the `inputImageSize` argument are not in a 1-to-1 aspect ratio, you must set the value of `ROIOutputSize` using dimensions which are compatible with the feature layer output size.

Output Arguments

`lgraph` — Object detection network
`layerGraph` object

Object detection network, returned as a `layerGraph` object. The output and base network `imageInputLayer` normalization values are equal.

Version History

Introduced in R2019b

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps
Deep Network Designer

Functions
`trainFasterRCNNObjectDetector` | `fasterRCNNObjectDetector` | `estimateAnchorBoxes` | `analyzeNetwork`

Topics
“Estimate Anchor Boxes from Training Data”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”

yolov2Layers

(To be removed) Create YOLO v2 object detection network

Note `yolov2Layers` will be removed in a future release. Use the `yolov2ObjectDetector` function instead. For more information, see “Version History”.

Syntax

```
lgraph = yolov2Layers(imageSize,numClasses,anchorBoxes,network,featureLayer)
lgraph = yolov2Layers(___,"ReorgLayerSource",reorgLayer)
```

Description

`lgraph = yolov2Layers(imageSize,numClasses,anchorBoxes,network,featureLayer)` creates a YOLO v2 object detection network and returns it as a `LayerGraph` object.

`lgraph = yolov2Layers(___,"ReorgLayerSource",reorgLayer)` adds a reorganization layer to the YOLO v2 network architecture following the layer `reorgLayer`,

Examples

Create YOLO v2 Network for Object Detection

Specify the size of the input image for training the network.

```
imageSize = [224 224 3];
```

Specify the number of object classes the network has to detect.

```
numClasses = 1;
```

Define the anchor boxes.

```
anchorBoxes = [1 1;4 6;5 3;9 6];
```

Specify the pretrained ResNet-50 network as the base network for YOLO v2. To use this pretrained network, you need to install the "*Deep Learning Toolbox Model for ResNet-50 Network*" support package.

```
network = resnet50;
```

Analyze the network architecture to view all the network layers.

```
analyzeNetwork(network)
```

Specify the network layer to be used for feature extraction. You can choose any layer except the fully connected layer as feature layer.

```
featureLayer = 'activation_49_relu';
```

Create the YOLO v2 object detection network. The network is returned as a `LayerGraph` object.

```
lgraph = yolov2Layers(imageSize,numClasses,anchorBoxes,network,featureLayer);
```

Analyze the YOLO v2 network architecture. The layers succeeding the feature layer are removed. A series of convolution, ReLU, and batch normalization layers along with the YOLO v2 transform and YOLO v2 output layers are added to the feature layer of the base network.

```
analyzeNetwork(lgraph)
```

Input Arguments

`imageSize` — Size of input image

two-element vector | three-element vector

Size of an input image, specified as one of these values:

- Two-element vector of form [*H W*] - For a grayscale image of size *H*-by-*W*
- Three-element vector of form [*H W* 3] - For an RGB color image of size *H*-by-*W*

`numClasses` — Number of object classes

positive integer

Number of object classes, specified as a positive integer.

`anchorBoxes` — Anchor boxes

M-by-2 matrix

Anchor boxes, specified as an *M*-by-2 matrix defining the size and the number of anchor boxes. Each row in the *M*-by-2 matrix denotes the size of the anchor box in the form of [*height width*]. *M* denotes the number of anchor boxes. This input sets the `AnchorBoxes` property of the output layer.

The size of each anchor box is determined based on the scale and aspect ratio of different object classes present in input training data. Also, the size of each anchor box must be smaller than or equal to the size of the input image. You can use the clustering approach for estimating anchor boxes from the training data. For more information, see “Estimate Anchor Boxes from Training Data”.

`network` — Pretrained convolutional neural network

`LayerGraph` object | `DAGNetwork` object | `SeriesNetwork` object

Pretrained convolutional neural network, specified as an `LayerGraph`, `DAGNetwork`, or `SeriesNetwork` object. This pretrained convolutional neural network is used as the base for the YOLO v2 object detection network. For details on pretrained networks in MATLAB, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

`featureLayer` — Name of feature layer

character vector | string scalar

Name of feature layer, specified as a character vector or a string scalar. The name of one of the deeper layers in the `network` to be used for feature extraction. The features extracted from this layer are given as input to the YOLO v2 object detection subnetwork. You can use the `analyzeNetwork` function to view the names of the layers in the input network.

Note You can specify any network layer except the fully connected layer as the feature layer.

`reorgLayer` — Name of reorganization layer
character vector | string scalar

Name of reorganization layer, specified as a character vector or a string scalar. The name of one of the deeper layers in the `network` to be used as input to the reorganization layer. You can use the `analyzeNetwork` function to view the names of the layers in the input `network`. The reorganization layer is the pass-through layer that reorganizes the dimension of low layer features to facilitate concatenation with high layer features.

Note The input to the reorganization layer must be from any one of the network layers that lie above the feature layer.

Output Arguments

`lgraph` — YOLO v2 object detection network
`LayerGraph` object

YOLO v2 object detection network, returned as a `LayerGraph` object.

Note The default value for the `Normalization` property of the image input layer in the returned `lgraph` object is set to the `Normalization` property of the base network specified in `network`.

Algorithms

The `yolov2Layers` function creates a YOLO v2 network, which represents the network architecture for YOLO v2 object detector. Use the `trainYOLOv2ObjectDetector` function to train the YOLO v2 network for object detection. The function returns an object that generates the network architecture for YOLO v2 object detection network presented in [1] and [2].

`yolov2Layers` uses a pretrained neural network as the base network to which it adds a detection subnetwork required for creating a YOLO v2 object detection network. Given a base network, `yolov2Layers` removes all the layers succeeding the feature layer in the base network and adds the detection subnetwork. The detection subnetwork comprises of groups of serially connected convolution, ReLU, and batch normalization layers. The YOLO v2 transform layer and YOLO v2 output layer are added to the detection subnetwork. If you specify the name-value argument `"ReorgLayerSource"`, the YOLO v2 network concatenates the output of reorganization layer with the output of feature layer.

For information on creating a custom YOLO v2 network layer-by-layer, see “Create Custom YOLO v2 Object Detection Network”.

Version History

Introduced in R2019a

R2025a: `yolov2Layers` will be removed
Warns starting in R2025a

The `yolov2Layers` function will be removed in a future release. When you call the `yolov2Layers` function, it issues a warning that it will be removed. Create a YOLO v2 object detection network by using the `yolov2ObjectDetector` object instead.

R2024b: `yolov2Layers` will be removed
Not recommended starting in R2024b

The `yolov2Layers` function will be removed in a future release. Use the `yolov2ObjectDetector` function instead. The `yolov2ObjectDetector` function creates a pretrained or untrained YOLO v2 object detector, and stores the network as a `dlnetwork` object.

To update your code:

- Define your network as a `dlnetwork` object. You can load a pretrained feature extraction network by using the `imagePretrainedNetwork` function.
- Select a feature extraction layer.
- Create a `yolov2ObjectDetector` object, specifying the base network and the name of the feature extraction layer. Also, specify the names of the classes and the anchor boxes as inputs for training the network.

For more information, see “Getting Started with YOLO v2”.

References

[1] Joseph. R, S. K. Divvala, R. B. Girshick, and F. Ali. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788. Las Vegas, NV: CVPR, 2016.

[2] Joseph. R and F. Ali. "YOLO 9000: Better, Faster, Stronger." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525. Honolulu, HI: CVPR, 2017.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation, you must first create a YOLO v2 network by using the `yolov2Layers` function. Then, use the `yolov2ObjectDetector` function on the resulting `lgraph` object to train the network for object detection. Once the network is trained and evaluated, you can generate code for the `yolov2ObjectDetector` object using GPU Coder.

See Also

`yolov2ObjectDetector` | `trainYOLOv2ObjectDetector` | `spaceToDepthLayer` | `yolov2TransformLayer` | `analyzeNetwork` | `imagePretrainedNetwork`

Topics

“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Estimate Anchor Boxes from Training Data”

“Code Generation for Object Detection by Using YOLO v2”
“Getting Started with YOLO v2”
“Anchor Boxes for Object Detection”

gatherLabelData

Gather label data from ground truth

Syntax

```
labelData = gatherLabelData(gTruth,typeOfLabel)
[labelData,timestamps] = gatherLabelData( ___ )
[ ___ ] = gatherLabelData( ___ ,Name=Value)
```

Description

`labelData = gatherLabelData(gTruth,typeOfLabel)` returns label data gathered from ground truth data, `gTruth`. The function returns label data specified by `typeOfLabel`.

`[labelData,timestamps] = gatherLabelData( ___ )` additionally returns the image timestamps associated with the gathered label data, using the arguments from the previous syntax.

`[ ___ ] = gatherLabelData( ___ ,Name=Value)` uses name-value arguments to specify how to gather data. For example, `labelData = gatherLabelData(gTruth,typeofLabel,SampleFactor=5)` specifies to drop every 5th frame.

Examples

Gather Polygon Labels

Gather all of the polygon labels from the groundtruth image.

```
data = load('groundtruthVisionTeam.mat');
gtruth = data.groundtruthVisionTeam;
```

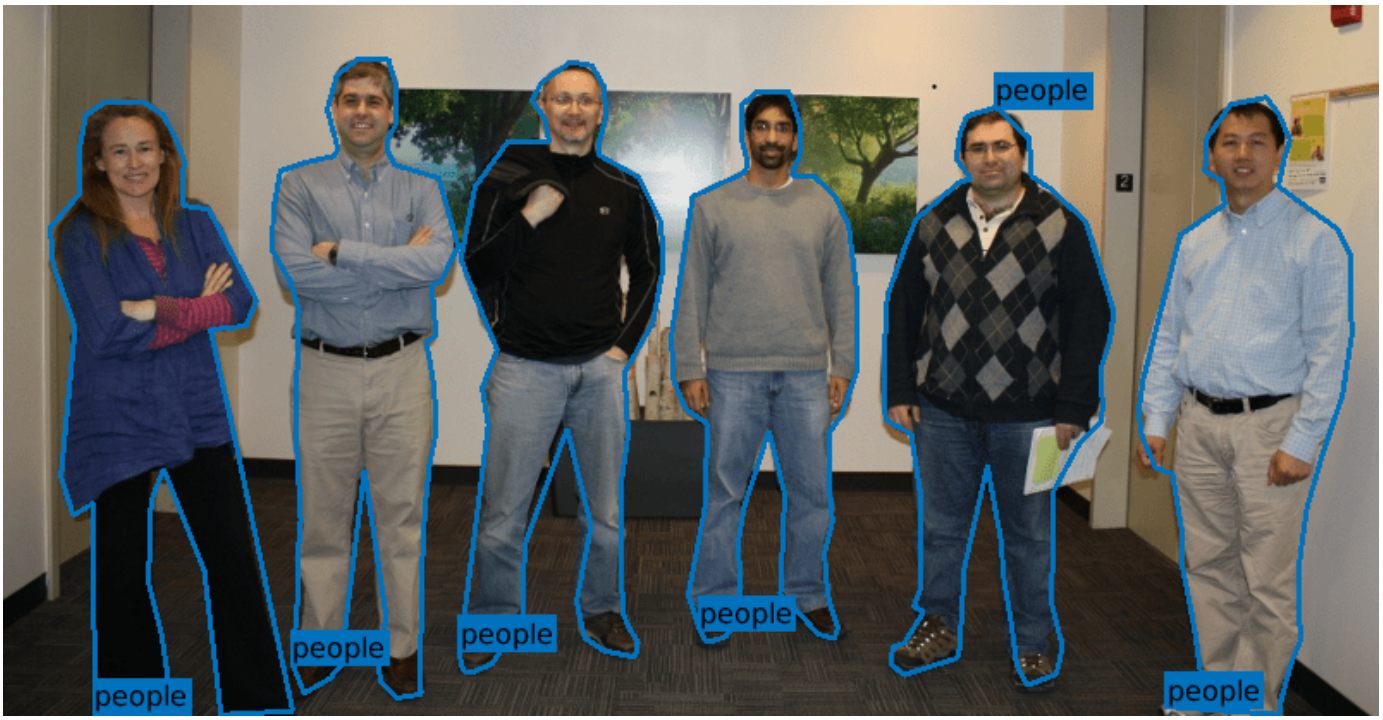
Gather all of the polygon objects.

```
labelData = gatherLabelData(gtruth, labelType('Polygon'), 'GroupLabelData', 'LabelType');

polygons = labelData{1}.PolygonData{1}(:,1);
polygonLabels = labelData{1}.PolygonData{1}(:,2);
```

Visualize the polygon labels.

```
im = imread('visionteam.jpg');
imshow(im);
showShape('polygon', polygons, 'Label', polygonLabels);
```



Input Arguments

`gTruth` — Ground truth data

`groundTruth` object | vector of `groundTruth` objects

Ground truth data, specified as a `groundTruth` object or vector of `groundTruth` objects.

`typeOfLabel` — Label types

vector of `labelType` objects

Label types from which to gather label data, specified as a vector of `labelType` objects. The `gatherLabelData` function gathers label data from each `groundTruth` object specified by input `gTruth` The label data is grouped in columns either by label name or by label type, specified by the `GroupLabelData` name-value argument.

Valid Enumeration Types

You can specify one or more of these enumeration types.

- `labelType.Rectangle` — Rectangle ROI labels
- `labelType.RotatedRectangle` — Rotated Rectangle ROI labels
- `labelType.Cuboid` — Cuboid ROI labels (point clouds)
- `labelType.ProjectedCuboid` — Projected cuboid ROI labels (images and video data)
- `labelType.Line` — Line ROI labels
- `labelType.PixelLabel` — Pixel ROI labels
- `labelType.Polygon` — Pixel ROI labels

- `labelType.Scene` — Scene labels

To gather label data for scenes, you must specify `labelTypes` as the `labelType.Scene` enumeration scalar. You cannot specify any other label types with `labelType.Scene`.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `labelData = gatherLabelData(gTruth,typeofLabel,SampleFactor=5)` specifies to drop every 5th frame.

`SampleFactor` — Sample factor

1 (default) | positive integer

Sample factor to subsample label data, specified as a positive integer. A sample factor of *K* includes every *K*th frame. Increase the sample factor to drop redundant frames from signals with high sample rates, such as video.

`GroupLabelData` — Group columns from label data

`'LabelName'` (default) | `'LabelType'`

Group columns from label data, specified as `'LabelName'` or `'LabelType'`.

- `'LabelName'` — Groups the label data by label definitions.
- `'LabelType'` — Groups the label data by label type. Use this option to gather label data with the region-of-interest (ROI) stacking order retained by the label type.

Group label data with the scene label type (`labelType.Scene`) by label definitions only.

Output Arguments

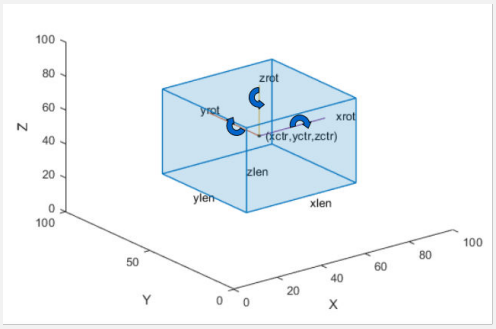
`labelData` — Label data

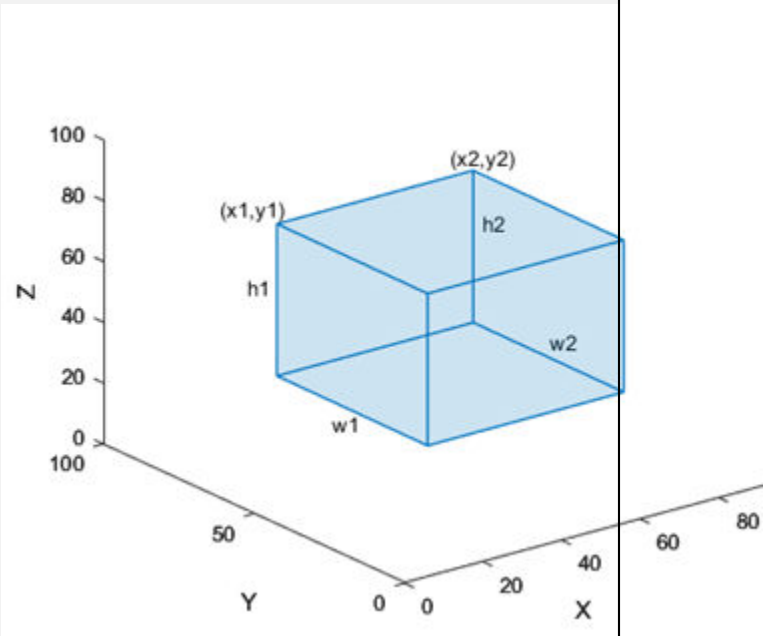
cell array of tables

Label data, returned as an `M`-by-`1` cell array of tables, where `M` is the number of `groundTruth` objects in `gTruth`. The columns of the tables represent label data grouped by either label name or label type, specified by the `GroupLabelData` name-value argument.

For each cell in the table, the format of the returned label data depends on the type of label.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.Rectangle`	`M`-by-4 numeric matrix of the form `[x, y, w, h]`, where: • `M` is the number of labels in the frame. • `x` and `y` specify the upper-left corner of the rectangle. • `w` specifies the width of the rectangle, which is its length along the *x*-axis. • `h` specifies the height of the rectangle, which is its length along the *y*-axis.
`labelType.RotatedRectangle`	For one or more rotated rectangles, specify in spatial coordinates as an *M*-by-5 numeric matrix, where each row specifies a rotated rectangle of the form [*xctr yctr w h yaw*]. • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the rectangle.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.Cuboid`	M-by-9 numeric matrix with rows of the form `[xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot]`, where: • `M` is the number of labels in the frame. • `xctr`, `yctr`, and `zctr` specify the center of the cuboid. • `xlen`, `ylen`, and `zlen` specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation has been applied. • `xrot`, `yrot`, and `zrot` specify the rotation angles for the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively. These angles are clockwise-positive when looking in the forward direction of their corresponding axes. The figure shows how these values determine the position of a cuboid. 

Label Type	Storage Format for Labels at Each Timestamp
`labelType.ProjectedCuboid`	M-by-8 vector of the form `[x1, y1, w1, h1, x2, y2, w2, h2]`, where: • `M` is the number of labels in the frame. • `x1, y1` specifies the *x*,*y* coordinates for the upper-left location of the front-face of the projected cuboid • `w1` specifies the width for the front-face of the projected cuboid. • `h1` specifies the height for the front-face of the projected cuboid. • `x2, y2` specifies the *x*,*y* coordinates for the upper-left location of the back-face of the projected cuboid. • `w2` specifies the width for the back-face of the projected cuboid. • `h2` specifies the height for the back-face of the projected cuboid. The figure shows how these values determine the position of a cuboid. 
`labelType.Line`	M-by-1 vector of cell arrays, where `M` is the number of labels in the frame. Each cell array contains an `N`-by-2 numeric matrix of the form `[x1 y1; x2 y2; ... ; xN yN]` for `N` points in the polyline.

Label Type	Storage Format for Labels at Each Timestamp
`labelType.PixelLabel`	Label data for all pixel label definitions is stored in a single *M*-by-1 `PixelLabelData` column for *M* images or frames. Each element contains a filename for a pixel label image. A pixel label image describes the label or labels contained in the corresponding image. The labels can be described as a 1- or 3- channel label matrix. To use `PixelLabelData` with any of the labeler apps, you must use a single-channel label matrix, where the values are of type `uint8`. You can convert a 3-channel pixel label data matrix to a single-channel label matrix programmatically to use with the labeler apps.
`labelType.Polygon`	`M`-by-1 vector of cell arrays, where `M` is the number of labels. Each cell array contains an `N`-by-2 numeric matrix of the form `[x1 y1; x2 y2; ... ; xN yN]` for `N` points in the polygon.
`labelType.Scene`	Logical `1` (`true`) if the scene label is applied, otherwise logical `0` (`false`)

`timestamps` — Timestamps
cell array of `duration` vectors

Timestamps, returned as an `M`-by-`1` cell array of `duration` vectors, where:

- `M` is the number of `groundTruth` objects in `gTruth`.
- `labelData{m}` contains the timestamps that is in the `m`th `groundTruth` object of `gTruth`.

Limitations

- The `gatherLabelData` function does not gather label data for sublabels or attributes. If a label contains sublabels or attributes, in the `labelData` output, the function returns the position of the parent label only.

Version History

Introduced in R2021a

R2022b: Project cuboids from 3-D world coordinates to 2-D image coordinates

Updated to support 3-D projected cuboid labels.

See Also

`groundTruth` | `boxLabelDatastore`

Topics
“Label Objects Using Polygons for Instance Segmentation”

selectLabelsByGroup

Select ground truth labels by label group

Syntax

```
gtLabel = selectLabelsByGroup(gTruth,labelGroups)
```

Description

`gtLabel = selectLabelsByGroup(gTruth,labelGroups)` selects labels belonging to the groups specified by `labelGroups` from a `groundTruth` object, `gTruth`. The function returns a corresponding `groundTruth` object, `gtLabel`, that contains only the selected labels. If `gTruth` is a vector of `groundTruth` objects, then the function returns a vector of corresponding `groundTruth` objects that contain only the selected labels.

Examples

Select Ground Truth Data By Group

Load data to create a ground truth object. Add the image folder to the path.

```
data = load('stopSignsAndCars.mat');
imageFilenames = data.stopSignsAndCars.imageFilename(1:2)
```

```
imageFilenames = 2×1 cell
    {'stopSignImages/image001.jpg'}
    {'stopSignImages/image002.jpg'}
```

```
imageFilenames = fullfile(toolboxdir('vision'),'visiondata',imageFilenames);
dataSource = groundTruthDataSource(imageFilenames);
```

Define labels for identifying ground truth data.

```
names = {'stopSign';'carRear'};
types = [
    labelType('Rectangle')
    labelType('Rectangle')
    ];
groups = {'TrafficSigns';'Vehicles'};

labelDefs = table(names,types,groups,'VariableNames', {'Name','Type','Group'})
```

```
labelDefs=2×3 table
        Name          Type            Group
    ____________    _________    _______________

    {'stopSign'}    Rectangle    {'TrafficSigns'}
    {'carRear' }    Rectangle    {'Vehicles'    }
```

Initialize label data for rectangle ROIs.

```
numRows = numel(imageFilenames);
stopSignTruth = {[856   318    39    41]; [445   523  52    54]};
carRearTruth = {[398   378   315   210]; [332   633   691   287]};
```

Construct a table containing label data.

```
labelData = table(stopSignTruth,carRearTruth,'VariableNames',names)
```

```
labelData=2×2 table
        stopSign               carRear
    ___________________    ___________________

    {[856 318 39 41]}    {[398 378 315 210]}
    {[445 523 52 54]}    {[332 633 691 287]}
```

Create a `groundTruth` object.

```
gTruth = groundTruth(dataSource,labelDefs,labelData)
```

```
gTruth =
  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [2×3 table]
           LabelData: [2×2 table]
```

Select labels by group.

```
vehicleGroundTruth = selectLabelsByGroup(gTruth, 'Vehicles')
```

```
vehicleGroundTruth =
  groundTruth with properties:

          DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [1×3 table]
           LabelData: [2×1 table]
```

Input Arguments

`gTruth` — Ground truth

`groundTruth` object | vector of `groundTruth` objects

Ground truth, specified as a `groundTruth` object or vector of `groundTruth` objects.

`labelGroups` — Label groups

character vector | string scalar | cell array of character vectors | string vector

Label groups, specified as a character vector, string scalar, cell array of character vectors, or string vector.

To view all label groups in a `groundTruth` object, `gTruth`, enter this command at the MATLAB command prompt.

```
unique(gTruth.LabelDefinitions.Group)
```

Example: `'Vehicles'`

Example: `"Vehicles"`

Example: `{'Vehicles','Signs'}`

Example: `["Vehicles" "Signs"]`

Output Arguments

`gtLabel` — Ground truth with only selected labels
`groundTruth` object | vector of `groundTruth` objects

Ground truth with only the selected labels, returned as a `groundTruth` object or vector of `groundTruth` objects.

Each `groundTruth` object in `gtLabel` corresponds to a `groundTruth` object in the `gTruth` input. The returned objects contain only the labels belonging to the groups specified by the `labelGroups` input.

Version History

Introduced in R2019a

See Also

Objects
`groundTruth`

Functions
`selectLabelsByType` | `selectLabelsByName`

selectLabelsByType

Select ground truth labels by label type

Syntax

```
gtLabel = selectLabelsByType(gTruth,labelTypes)
```

Description

`gtLabel = selectLabelsByType(gTruth,labelTypes)` selects labels of the types specified by `labelTypes` from a `groundTruth` object, `gTruth`. The function returns a corresponding `groundTruth` object, `gtLabel`, that contains only the selected labels. If `gTruth` is a vector of `groundTruth` objects, then the function returns a vector of corresponding `groundTruth` objects that contain only the selected labels.

Examples

Select Ground Truth Data by Types

Add the image directory to the MATLAB path.

```
imageDir = fullfile(matlabroot,'toolbox','vision','visiondata','stopSignImages');
    addpath(imageDir);
```

Load the `groundTruth` object.

```
load('stopSignsAndCarsGroundTruth.mat');
```

View the label definitions.

```
stopSignsAndCarsGroundTruth.LabelDefinitions
```

```
ans=3×3 table
        Name          Type        Group
    ______________    _________    ________

    {'stopSign'}    Rectangle    {'None'}
    {'carRear' }    Rectangle    {'None'}
    {'carFront'}    Rectangle    {'None'}
```

Obtain the ground truth data for `Rectangle` `labelType`.

```
rectGroundTruth = selectLabelsByType(stopSignsAndCarsGroundTruth,labelType.Rectangle);
```

Remove the image directory from the path.

```
rmpath(imageDir);
```

Input Arguments

gTruth — Ground truth
groundTruth object | vector of groundTruth objects

Ground truth, specified as a groundTruth object or vector of groundTruth objects.

labelTypes — Label types
labelType enumeration | vector of labelType enumerations

Label types, specified as a labelType enumeration or vector of labelType enumerations.

To view all label types in a groundTruth object, gTruth, enter this command at the MATLAB command prompt.

```
unique(gTruth.LabelDefinitions.Type)
```

Example: labelType.Rectangle

Example: [labelType.Rectangle labelType.Line]

Output Arguments

gtLabel — Ground truth with only selected labels
groundTruth object | vector of groundTruth objects

Ground truth with only the selected labels, returned as a groundTruth object or vector of groundTruth objects.

Each groundTruth object in gtLabel corresponds to a groundTruth object in the gTruth input. The returned objects contain only the labels that are of the types specified by the labelTypes input.

Limitations

- Selecting sublabels by type is not supported.

Version History

Introduced in R2019a

See Also

Objects
groundTruth

Functions
selectLabelsByName | selectLabelsByGroup

selectLabelsByName

Select ground truth labels by label name

Syntax

```
gtLabel = selectLabelsByName(gTruth,labelNames)
```

Description

`gtLabel = selectLabelsByName(gTruth,labelNames)` selects labels specified by `labelNames` from a `groundTruth` object, `gTruth`. The function returns a corresponding `groundTruth` object, `gtLabel`, that contains only the selected labels. If `gTruth` is a vector of `groundTruth` objects, then the function returns a vector of corresponding `groundTruth` objects that contain only the selected labels.

Examples

Select Ground Truth Data By Name

Add the image directory to the MATLAB path.

```
imageDir = fullfile(matlabroot,'toolbox','vision','visiondata','stopSignImages');
addpath(imageDir);
```

Load the `groundTruth` object.

```
load('stopSignsAndCarsGroundTruth.mat');
```

View the label definitions.

```
stopSignsAndCarsGroundTruth.LabelDefinitions
```

```
ans=3×3 table
        Name           Type         Group
    _____________    _________    ________

    {'stopSign'}    Rectangle    {'None'}
    {'carRear' }    Rectangle    {'None'}
    {'carFront'}    Rectangle    {'None'}
```

Obtain the ground truth data for `StopSign` label name.

```
stopSignGroundTruth = selectLabelsByName(stopSignsAndCarsGroundTruth, ...
                                         'stopSign');
```

Obtain ground truth data for `carRear` and `carFront`.

```
carGroundTruth = selectLabels(stopSignsAndCarsGroundTruth, ...
                                      {'carRear','carFront'});
```

Remove the image directory from the path.

```
rmpath(imageDir);
```

Input Arguments

gTruth — Ground truth

groundTruth object | vector of groundTruth objects

Ground truth, specified as a groundTruth object or vector of groundTruth objects.

labelNames — Label names

character vector | string scalar | cell array of character vectors | string vector

Label names, specified as a character vector, string scalar, cell array of character vectors, or string vector.

To view all label names in a groundTruth object, gTruth, enter this command at the MATLAB command prompt.

```
gTruth.LabelDefinitions.Name
```

Example: 'car'

Example: "car"

Example: {'car','lane'}

Example: ["car" "lane"]

Output Arguments

gtLabel — Ground truth with only selected labels

groundTruth object | vector of groundTruth objects

Ground truth with only the selected labels, returned as a groundTruth object or vector of groundTruth objects.

Each groundTruth object in gtLabel corresponds to a groundTruth object in the gTruth input. The returned objects contain only the labels with names specified by the labelNames input.

Limitations

- Selecting pixel labels by name is not supported. However, you can select all labels of type pixel. Use the selectLabelsByType function, specifying the label type as a labelType.PixelLabel enumeration.
- Selecting sublabels by name is not supported.

Version History

Introduced in R2019a

See Also

Objects
`groundTruth`

Functions
`selectLabelsByGroup` | `selectLabelsByType`

correct

Namespace: `vision`

Correction of measurement, state, and state estimation error covariance

Syntax

```
[z_corr,x_corr,P_corr] = correct(kalmanFilter,z)
```

Description

`[z_corr,x_corr,P_corr] = correct(kalmanFilter,z)` returns the correction of measurement, state, and state estimation error covariance. The correction is based on the current measurement *z*. The object overwrites the internal state and covariance of the Kalman filter with corrected values.

Examples

Tracking Objects

Use the `predict` and `correct` functions based on detection results.

When the tracked object is detected, use the `predict` and `correct` functions with the Kalman filter object and the detection measurement. Call the functions in the following order:

```
[...] = predict(kalmanFilter);
[...] = correct(kalmanFilter,measurement);
```

When the tracked object is not detected, call the `predict` function, but not the `correct` method. When the tracked object is missing or occluded, no measurement is available. Set the functions up with the following logic:

```
[...] = predict(kalmanFilter);
If measurement exists
    [...] = correct(kalmanFilter,measurement);
end
```

If the tracked object becomes available after missing for the past *t*-1 contiguous time steps, you can call the `predict` function *t* times. This syntax is particularly useful to process asynchronous video.. For example,

```
for i = 1:k
  [...] = predict(kalmanFilter);
end
[...] = correct(kalmanFilter,measurement)
```

Input Arguments

`kalmanFilter` — Kalman filter object

object

Kalman filter object.

z — Current measurement
N-element vector

Current measurement, specified as an *N*-element vector.

Version History

Introduced in R2012b

See Also

`configureKalmanFilter` | `assignDetectionsToTracks`

distance

Namespace: `vision`

Confidence value of measurement

Syntax

```
d = distance(kalmanFilter,zmatrix)
```

Description

`d = distance(kalmanFilter,zmatrix)` computes a distance between the location of a detected object and the predicted location by the Kalman filter object. This distance computation takes into account the covariance of the predicted state and the process noise. The `distance` function can only be called after the `predict` function.

Use the `distance` function to find the best matches. The computed distance values describe how a set of measurements matches the Kalman filter. You can thus select a measurement that best fits the filter. This strategy can be used for matching object detections against object tracks in a multiobject tracking problem. This distance computation takes into account the covariance of the predicted state and the process noise.

Examples

Track Location of An Object

Track the location of a physical object moving in one direction.

Generate synthetic data which mimics the 1-D location of a physical object moving at a constant speed.

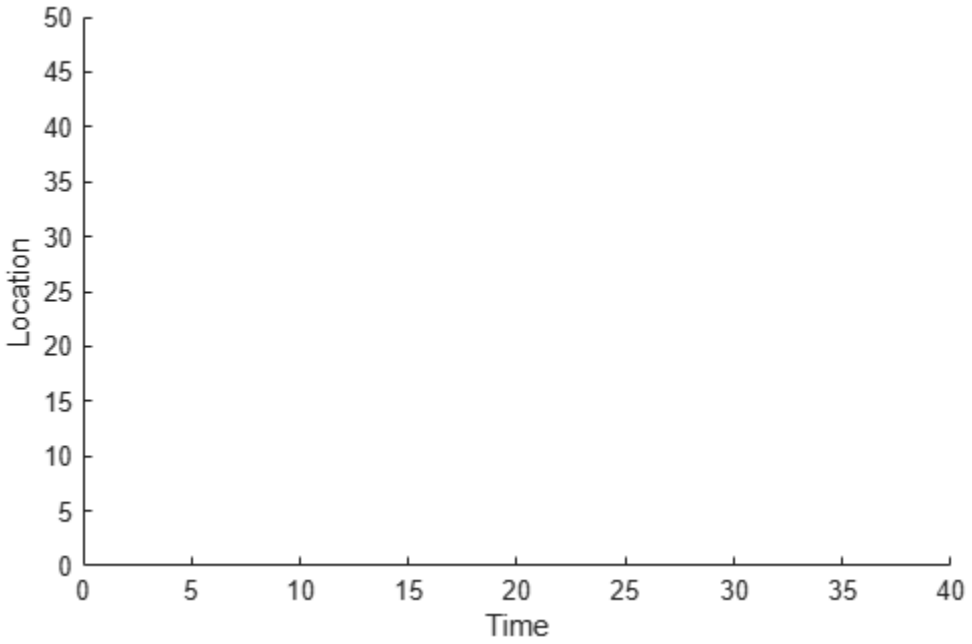
```
detectedLocations = num2cell(2*randn(1,40) + (1:40));
```

Simulate missing detections by setting some elements to empty.

```
detectedLocations{1} = [];
  for idx = 16: 25
      detectedLocations{idx} = [];
  end
```

Create a figure to show the location of detections and the results of using the Kalman filter for tracking.

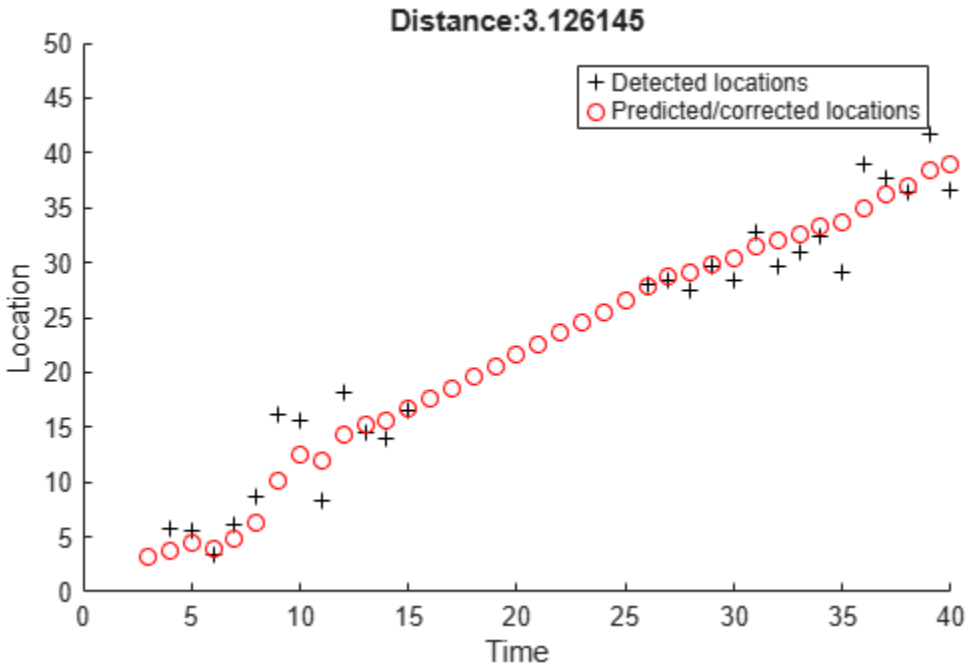
```
figure;
hold on;
ylabel('Location');
ylim([0,50]);
xlabel('Time');
xlim([0,length(detectedLocations)]);
```



Create a 1-D, constant speed Kalman filter when the physical object is first detected. Predict the location of the object based on previous states. If the object is detected at the current time step, use its location to correct the states.

```
kalman = [];
for idx = 1: length(detectedLocations)
   location = detectedLocations{idx};
   if isempty(kalman)
     if ~isempty(location)

       stateModel = [1 1;0 1];
       measurementModel = [1 0];
       kalman = vision.KalmanFilter(stateModel,measurementModel,'ProcessNoise',1e-4,'MeasurementN
      kalman.State = [location, 0];
     end
   else
     trackedLocation = predict(kalman);
     if ~isempty(location)
       plot(idx, location,'k+');
      d = distance(kalman,location);
       title(sprintf('Distance:%f', d));
       trackedLocation = correct(kalman,location);
     else
       title('Missing detection');
     end
     pause(0.2);
     plot(idx,trackedLocation,'ro');
   end
 end
legend('Detected locations','Predicted/corrected locations');
```



Remove Noise from a Signal

Use Kalman filter to remove noise from a random signal corrupted by a zero-mean Gaussian noise.

Synthesize a random signal that has value of 1 and is corrupted by a zero-mean Gaussian noise with standard deviation of 0.1.

```
x = 1;
len = 100;
z = x + 0.1 * randn(1,len);
```

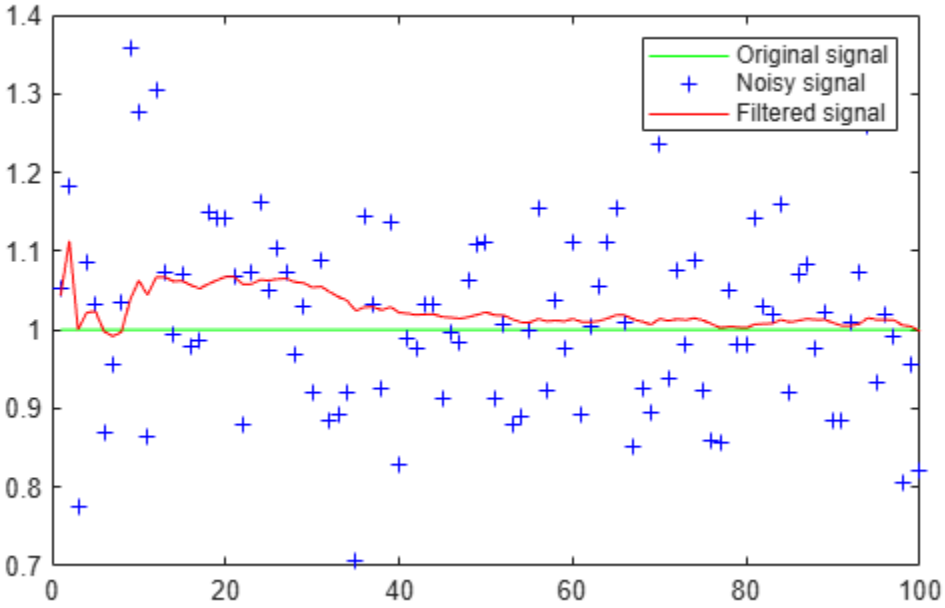
Remove noise from the signal by using a Kalman filter. The state is expected to be constant, and the measurement is the same as state.

```
stateTransitionModel = 1;
measurementModel = 1;
obj = vision.KalmanFilter(stateTransitionModel,measurementModel,'StateCovariance',1,'ProcessNoise

z_corr = zeros(1,len);
for idx = 1: len
 predict(obj);
 z_corr(idx) = correct(obj,z(idx));
end
```

Plot results.

```
figure, plot(x * ones(1,len),'g-');
hold on;
plot(1:len,z,'b+',1:len,z_corr,'r-');
legend('Original signal','Noisy signal','Filtered signal');
```



Input Arguments

`kalmanFilter` — Kalman filter object

object

Kalman filter object.

`zmatrix` — Location of a detected object

N-column matrix

Location of a detected object, specified as an *N*-column matrix. Each row matrix contains a measurement vector. The `distance` function returns a row vector where each distance element corresponds to the measurement input.

More About

Distance Equation

$$d(z) = (z - Hx)^T \sum^{-1} (z - Hx) + \ln\left|\sum\right|$$

Where $\Sigma = HPH^T + R$ and $|\Sigma|$ is the determinant of Σ. You can then find the best matches by examining the returned distance values.

Version History

Introduced in R2012b

See Also

`configureKalmanFilter` | `assignDetectionsToTracks`

predict

Namespace: `vision`

Prediction of measurement

Syntax

```
[z_pred,x_pred,P_pred] = predict(kalmanFilter)
[z_pred,x_pred,P_pred] = predict(kalmanFilter,u)
```

Description

`[z_pred,x_pred,P_pred] = predict(kalmanFilter)` returns the prediction of measurement, state, and state estimation error covariance at the next time step (e.g., the next video frame). The object overwrites the internal state and covariance of the Kalman filter with the prediction results.

`[z_pred,x_pred,P_pred] = predict(kalmanFilter,u)` additionally lets you specify the control input, *u*. This syntax applies when you set the control model, *B*.

Examples

Tracking Objects

Use the `predict` and `correct` functions based on detection results.

When the tracked object is detected, use the `predict` and `correct` functions with the Kalman filter object and the detection measurement. Call the functions in the following order:

```
[...] = predict(kalmanFilter);
[...] = correct(kalmanFilter,measurement);
```

When the tracked object is not detected, call the `predict` function, but not the `correct` method. When the tracked object is missing or occluded, no measurement is available. Set the functions up with the following logic:

```
[...] = predict(kalmanFilter);
If measurement exists
    [...] = correct(kalmanFilter,measurement);
end
```

If the tracked object becomes available after missing for the past *t*-1 contiguous time steps, you can call the `predict` function *t* times. This syntax is particularly useful to process asynchronous video.. For example,

```
for i = 1:k
  [...] = predict(kalmanFilter);
```

```
end
[...] = correct(kalmanFilter,measurement)
```

Input Arguments

`kalmanFilter` — Kalman filter object

object

Kalman filter object.

`u` — Control input

L-element vector

Control input, specified as an *L*-element vector.

Version History

Introduced in R2012b

See Also

`configureKalmanFilter` | `assignDetectionsToTracks`

pcregistercpd

Register two point clouds using CPD algorithm

Syntax

```
tform = pcregistercpd(moving,fixed)
[tform,movingReg] = pcregistercpd( ___ )
[tform,movingReg,rmse] = pcregistercpd( ___ )
[ ___ ] = pcregistercpd( ___ ,Name=Value)
```

Description

`tform = pcregistercpd(moving,fixed)` returns a transformation that registers a moving point cloud with a fixed point cloud using the CPD algorithm. The coherent point drift (CPD) algorithm [1] supports non-rigid transformations.

Note The CPD algorithm is robust to noise, outlier and missing points, at the expense of speed. Consider downsampling point clouds using `pcdownsample` before using `pcregistercpd` to improve the efficiency of registration.

Alternatively, you can speed up the algorithm by using the `extractFPFHFeatures` function to extract features before performing registration. For more details, see the “Improve CPD Performance Using FPFH Descriptors” example.

`[tform,movingReg] = pcregistercpd( ___ )` also returns the transformed point cloud that aligns with the fixed point cloud.

`[tform,movingReg,rmse] = pcregistercpd( ___ )` also returns the root mean square error of the Euclidean distance between the aligned point clouds.

`[ ___ ] = pcregistercpd( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MaxIterations=20` stops the CPD algorithm after 20 iterations.

Examples

Align Two Point Clouds Using CPD Algorithm

Load point cloud data into the workspace. Extract the moving and the fixed point clouds from the point cloud data in workspace.

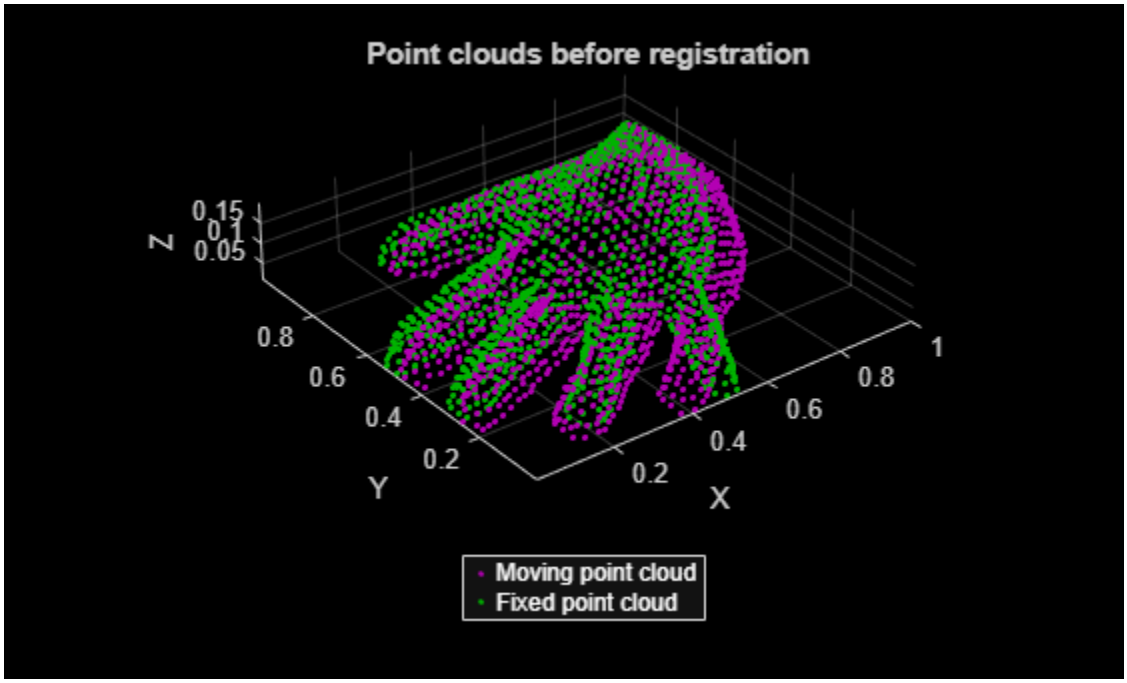
```
handData = load('hand3d.mat');
moving = handData.moving;
fixed = handData.fixed;
```

To improve the efficiency and accuracy of the CPD registration algorithm, downsample the moving and the fixed point clouds.

```
movingDownsampled = pcdownsample(moving,'gridAverage',0.03);
fixedDownsampled = pcdownsample(fixed,'gridAverage',0.03);
```

Display the downsampled point clouds before registration.

```
figure
pcshowpair(movingDownsampled,fixedDownsampled,'MarkerSize',50)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Point clouds before registration')
legend({'Moving point cloud','Fixed point cloud'},'TextColor','w')
legend('Location','southoutside')
```

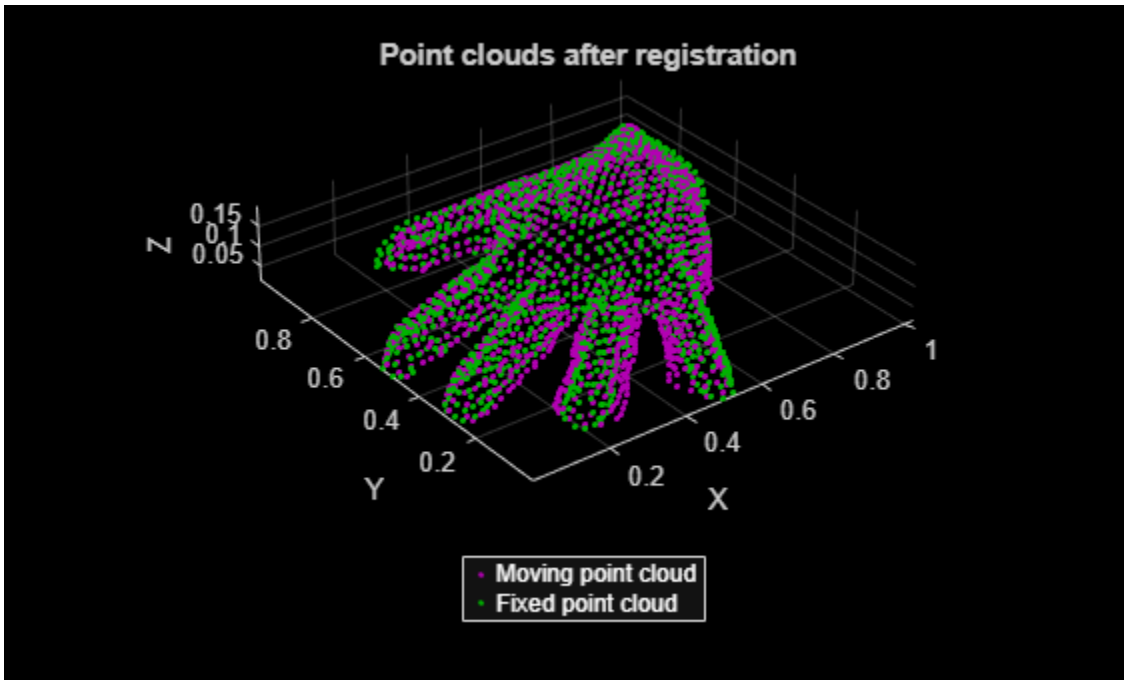


Perform non-rigid registration using the CPD algorithm.

```
[tform,movingReg] = pcregistercpd(movingDownsampled,fixedDownsampled);
```

Display the downsampled point clouds after registration.

```
figure
pcshowpair(movingReg,fixedDownsampled,'MarkerSize',50)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Point clouds after registration')
legend({'Moving point cloud','Fixed point cloud'},'TextColor','w')
legend('Location','southoutside')
```



Input Arguments

`moving` — Moving point cloud
`pointCloud` object

Moving point cloud, specified as a `pointCloud` object.

`fixed` — Fixed point cloud
`pointCloud` object

Fixed point cloud, specified as a `pointCloud` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxIteration=20` stops the CPD algorithm after 20 iterations.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Transform` — Type of transformation
`"Nonrigid"` (default) | `"Rigid"` | `"Affine"`

Type of transformation, specified as `"Nonrigid"`, `"Rigid"`, or `"Affine"`.

Data Types: `char` | `string`

`OutlierRatio` — Expected percentage of outliers
`0.1` (default) | scalar in the range [0, 1)

Expected percentage of outliers with respect to a normal distribution, specified as a scalar in the range [0, 1). Increasing this value reduces the influence of outliers and noise.

Data Types: `single` | `double`

`MaxIterations` — Maximum number of iterations
`20` (default) | positive integer

Maximum number of iterations before CPD stops, specified as a positive integer.

`MaxIterations` and `Tolerance` are both stopping conditions for the CPD algorithm. The algorithm stops when it satisfies either of the stopping conditions.

Data Types: `single` | `double`

`Tolerance` — Tolerance between consecutive CPD iterations
`1e-5` (default) | scalar

Tolerance between consecutive CPD iterations, specified as a scalar. The algorithm stops when the absolute percentage change in the values of the log likelihood function measured between consecutive iterations reaches or falls below the specified tolerance value. Decreasing this value increases the likelihood of a better alignment.

`MaxIterations` and `Tolerance` are both stopping conditions for the CPD algorithm. The algorithm stops when it satisfies either of the stopping conditions.

Data Types: `single` | `double`

`InteractionSigma` — Interaction between points
`2.0` (default) | positive scalar

Interaction between points, specified as positive scalar that represents standard deviation of a Gaussian filter. Typical values are in the range [1.5, 3]. Increasing this value increases interaction between the points in point cloud. As a result, you can observe coherent motion in the point cloud and every point undergoes the same displacement. Alternatively, decreasing this value reduces interaction between the points in point cloud. As a result, you can observe localized displacement of points and the output displacement field exhibits localized deformation.

Note To use this name-value argument, `Transform` must be `"Nonrigid"`.

Data Types: `single` | `double`

`SmoothingWeight` — Motion smoothing weight
`3.0` (default) | positive scalar

Motion smoothing weight, specified as a positive scalar. Typical values are in the range [0.1,10]. Increase this value to produce a more coherent motion in the output displacement field.

Note To use this name-value argument, `Transform` must be `"Nonrigid"`.

Data Types: `single` | `double`

`Verbose` — Display progress information

`false` (default) | `true`

Display progress information, specified as a numeric or logical `0` (`false`) or `1` (`true`). To display the progress information, set `Verbose` to `true`.

Output Arguments

`tform` — Transformation

`rigidtform3d` object | `affinetform3d` object | displacement field

Transformation, returned as a `rigidtform3d` object, an `affinetform3d` object, or a displacement field. `tform` is a 3-D transformation that registers the moving point cloud, `moving` to the fixed point cloud, `fixed`. The output type depends on the value of the `Transform` argument.

`Transform` Value	`tform`
`"Rigid"`	`rigidtform3d` object
`"Affine"`	`affinetform3d` object
`"Nonrigid"`	Displacement field, a numeric matrix of same size and datatype as the `Location` property of the moving point cloud object, `moving`.

`movingReg` — Transformed point cloud

`pointCloud` object

Transformed point cloud, returned as a `pointCloud` object. The transformed point cloud is aligned with the fixed point cloud, `fixed`.

`rmse` — Root mean square error

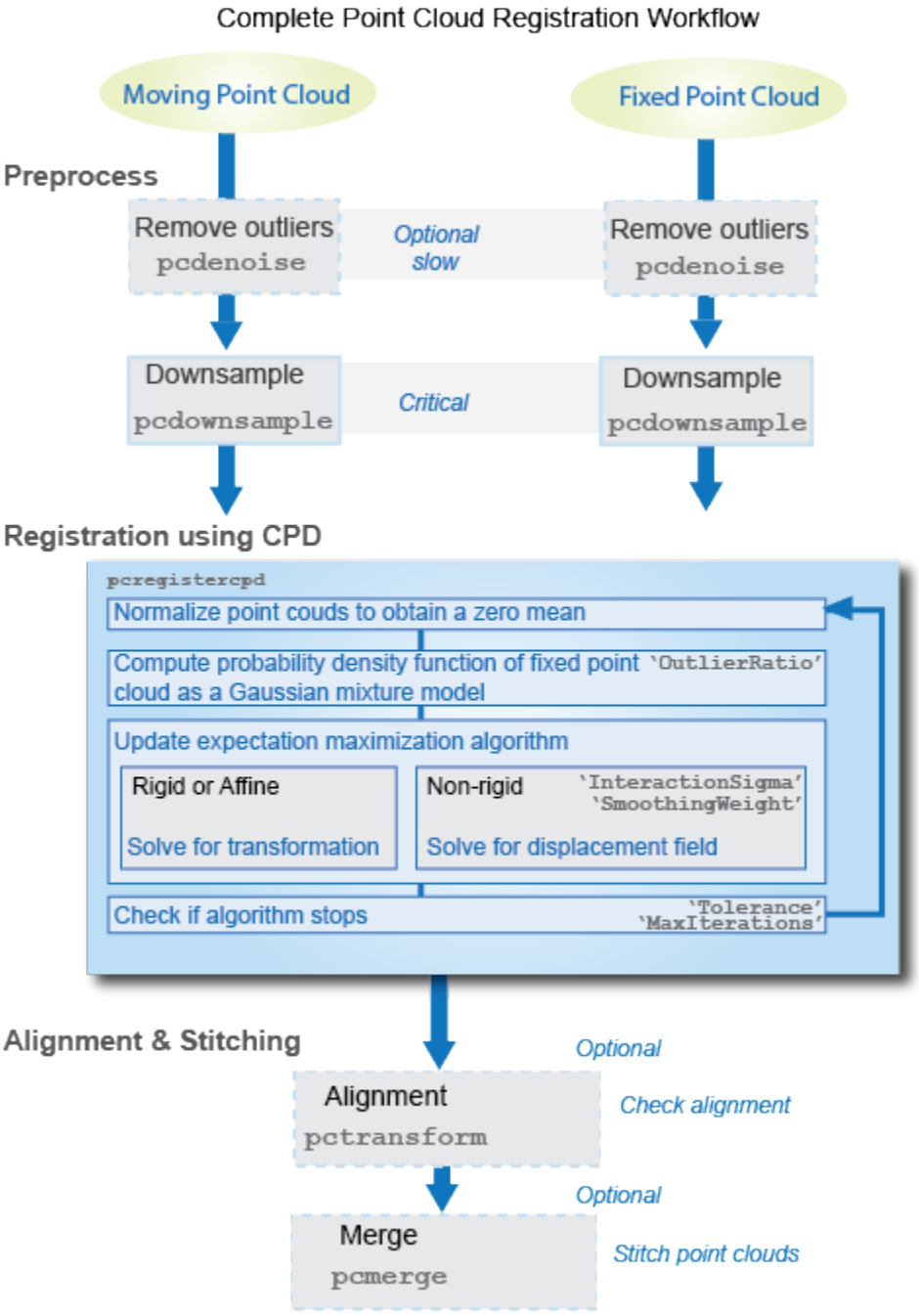
positive real number

Root mean square error, returned as a positive numeric scalar that represents the Euclidean distance between the aligned points. The aligned points is a result of the moving point cloud transformed by the `tform` output and the fixed point cloud. For each point in the fixed point cloud, the algorithm finds the closest point in the transformed point cloud, `movingReg`, then computes the Euclidean distance between the points, and then computes the `rmse`. A low `rmse` value indicates a more accurate registration. The `rmse` calculation is:

```
rmse = sqrt(sum(distance.^2,'all')/numel(distance));
```

Data Types: `double`

Algorithms



Both `MaxIterations` and `Tolerance` are used as stopping criteria. The algorithm stops when it satisfies either of the stopping conditions, i.e., when the number of iteration reaches `MaxIterations` or the absolute percentage change in log likelihood function is less than or equal to `Tolerance`.

Version History

Introduced in R2018b

R2022b: Returns new geometric transformation objects

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, for rigid and affine transformation types, the `pcregistercpd` function now returns `tform` as a `rigidtform3d` or `affinetform3d` object, which use the premultiply convention. Before, the function returned `tform` as a `rigid3d` or `affine3d` object, which use the postmultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Myronenko, A., and X. Song. "Point Set Registration: Coherent Point Drift. "*Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*." Vol 32, Number 12, December 2010, pp. 2262–2275.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The name-value argument `Transform` must be compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`pcregistercorr` | `pcregistericp` | `pcregisterndt` | `pctransform` | `pcshow` | `pcshowpair` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcmerge`

Objects

`pointCloud` | `rigidtform3d` | `affinetform3d`

Topics

“3-D Point Cloud Registration and Stitching”
“Improve CPD Performance Using FPFH Descriptors”
“Implement Point Cloud SLAM in MATLAB”

initializeObject

Set object to track

Syntax

```
initializeObject(hbtracker,I,R)
initializeObject(hbtracker,I,R,N)
```

Description

`initializeObject(hbtracker,I,R)` sets the object to track by extracting it from the [x y width height] region `R` located in the 2-D input image, `I`. The input image, `I`, can be any 2-D feature map that distinguishes the object from the background. For example, the image can be a hue channel of the HSV color space. Typically, `I` will be the first frame in which the object appears. The region, `R`, is also used for the initial search window, in the next call to the object. For best results, the object must occupy the majority of the region, `R`.

`initializeObject(hbtracker,I,R,N)` additionally, lets you specify `N`, the number of histogram bins.

Examples

Track a Face

Track and display a face in each frame of an input video.

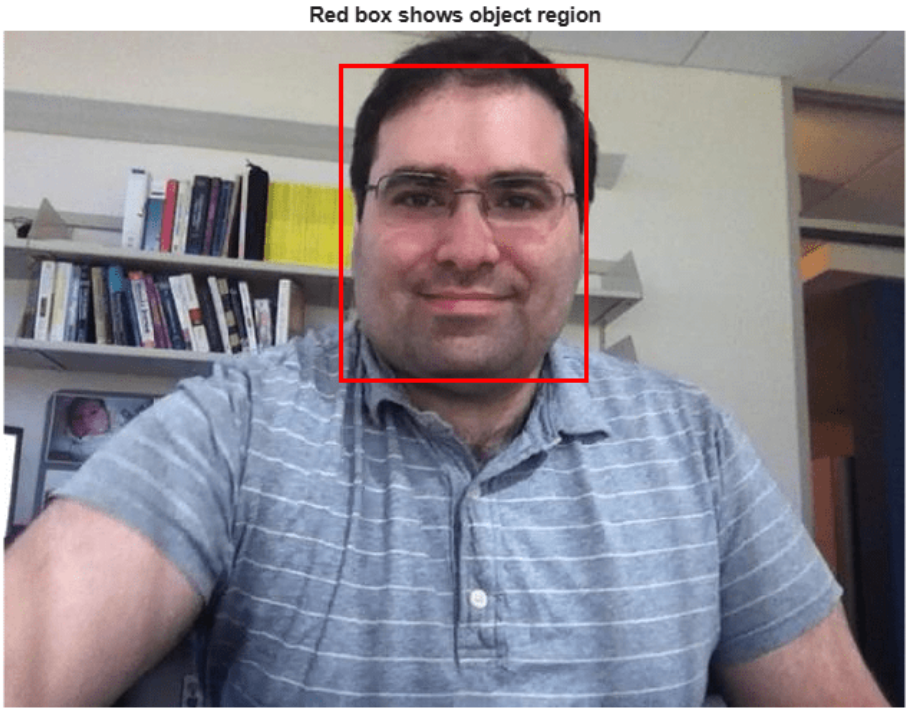
Load in the input video and create a System object displaying video.

```
videoReader = VideoReader("tilted_face.avi");
videoPlayer = vision.VideoPlayer();
```

Read the first video frame, which contains the object. Convert the image to HSV color space. Then define and display the object region.

```
objectFrame = im2single(readFrame(videoReader));
objectHSV = rgb2hsv(objectFrame);
objectRegion = [240, 25, 175, 225];
objectImage = insertShape(objectFrame,"rectangle",objectRegion,LineWidth=3,Color="red");

figure;
imshow(objectImage);
title("Red box shows object region")
```



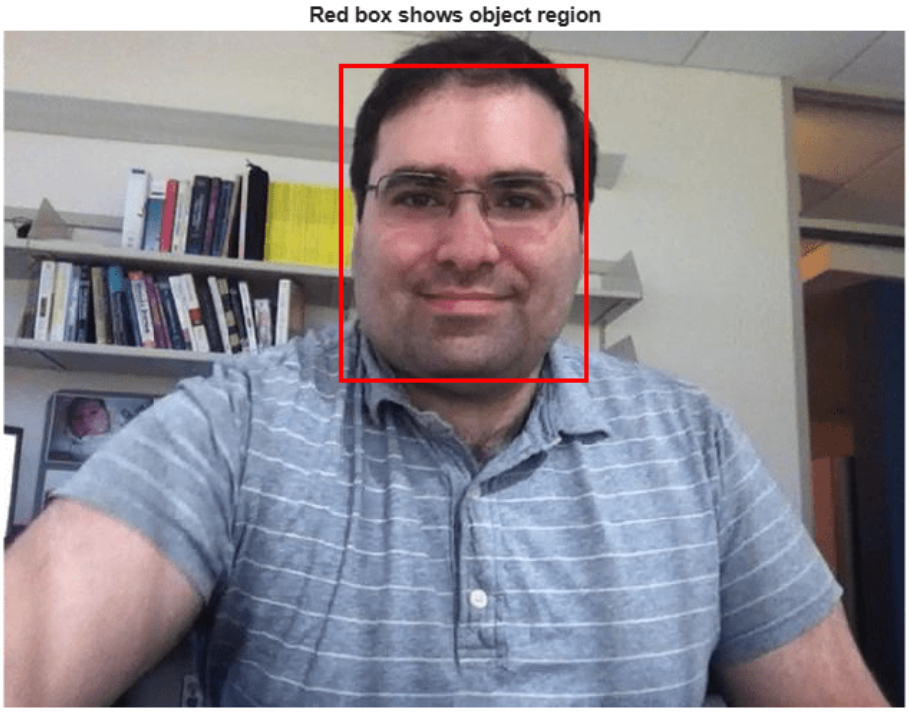
Set the object, based on the hue channel of the first video frame.

```
tracker = vision.HistogramBasedTracker;
initializeObject(tracker, objectHSV(:,:,1),objectRegion);
```

Track and display the object in each video frame. The while loop reads each image frame, converts the image to HSV color space, then tracks the object in the hue channel where it is distinct from the background. Finally, the example draws a box around the object and displays the results.

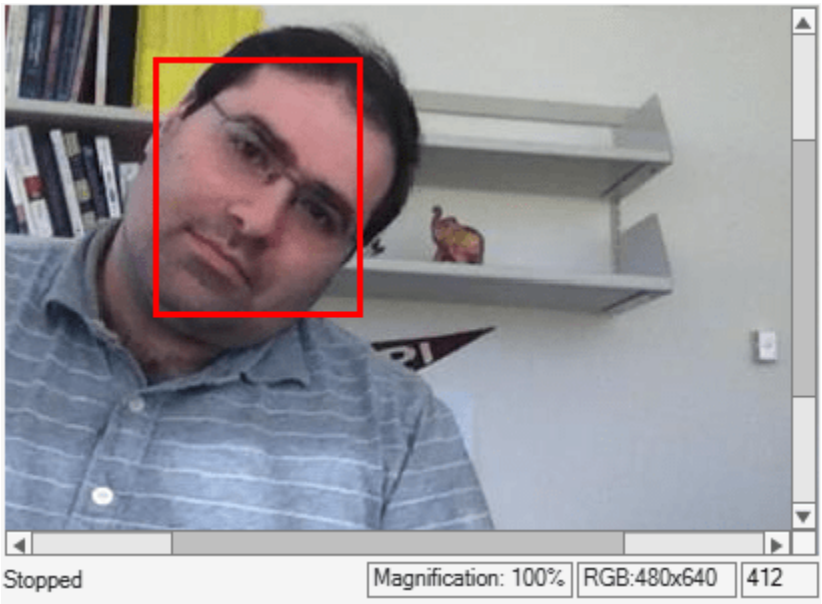
```
while hasFrame(videoReader)
  frame = im2single(readFrame(videoReader));
  hsvFrame = rgb2hsv(frame);
  bbox = tracker(hsvFrame(:,:,1));
  out = insertShape(frame,"rectangle",bbox,LineWidth=3,Color="red");
  videoPlayer(out);
end
```





Release the video player.

```
release(videoPlayer);
```



Copyright 2012-2025 The MathWorks, Inc.

Input Arguments

`hbtracker` — Histogram based tracker

`vision.HistogramBasedTracker` object

Histogram based tracker, specified as a `vision.HistogramBasedTracker` object.

`I` — Video frame

grayscale | truecolor (RGB)

Video frame, specified as grayscale or truecolor (RGB).

`R` — Initial search window

[*x y width height*]

Initial search window, specified in the format [*x y width height*].

`N` — Number of histogram bins

16 (default) | integer

Number of histogram bins, specified as an integer. Increasing the number of bins enhances the ability of the tracker to discriminate the object. However, this approach also narrows the range of changes to the object's visual characteristics that the tracker can accommodate. Consequently, this narrow range increases the likelihood of losing track.

Version History

Introduced in R2012a

See Also

Objects
`vision.HistogramBasedTracker`

hide

Namespace:

Hide player figure

Syntax

```
hide(player)
```

Description

`hide(player)` hides the figure. To redisplay the player, use show on page 3-335(`player`).

Examples

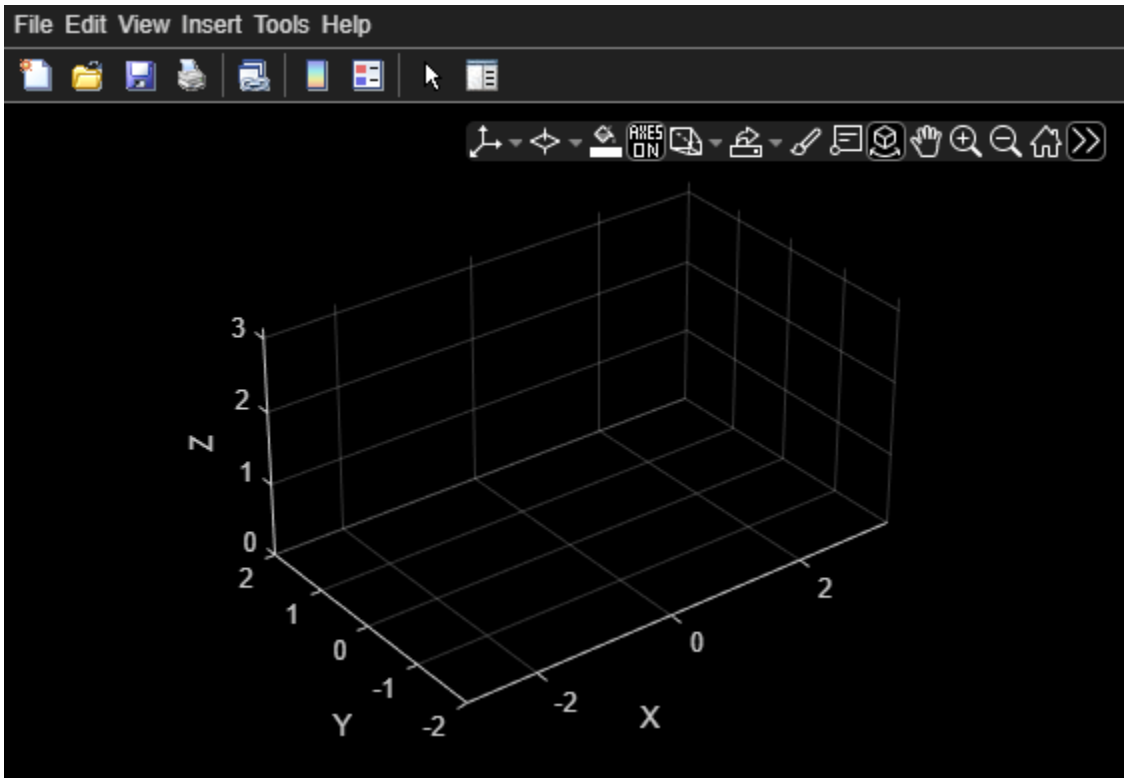
Hide and Show 3-D Point Cloud Figure

Load point cloud.

```
ptCloud = pcread('teapot.ply');
```

Create the player and customize player axis labels.

```
player = pcplayer(ptCloud.XLimits,ptCloud.YLimits,ptCloud.ZLimits);
```

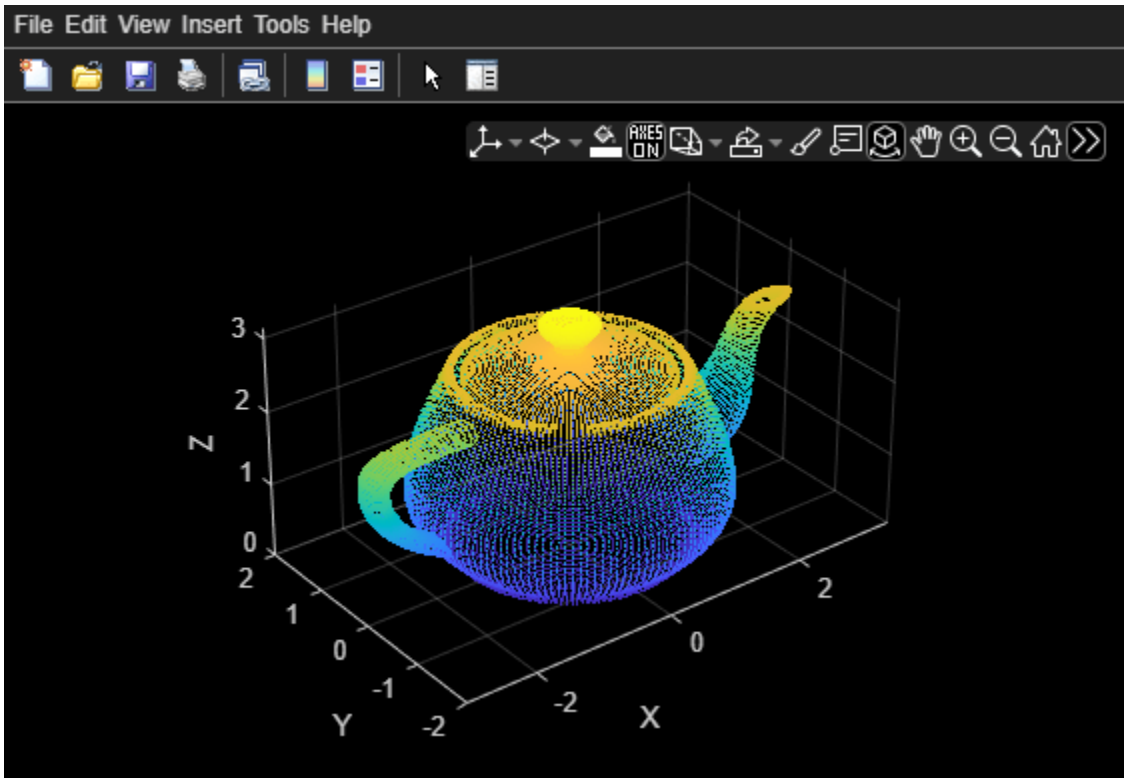


Hide figure.

```
hide(player)
```

Show figure.

```
show(player)
view(player,ptCloud);
```



Input Arguments

`player` — Player
player object

Player for data, specified as a `pcplayer` or `vision.VideoPlayer` object.

Version History

Introduced in R2015b

show

Namespace:

Show player

Syntax

```
show(player)
```

Description

`show(player)` makes the player figure visible again after closing or hiding it.

Examples

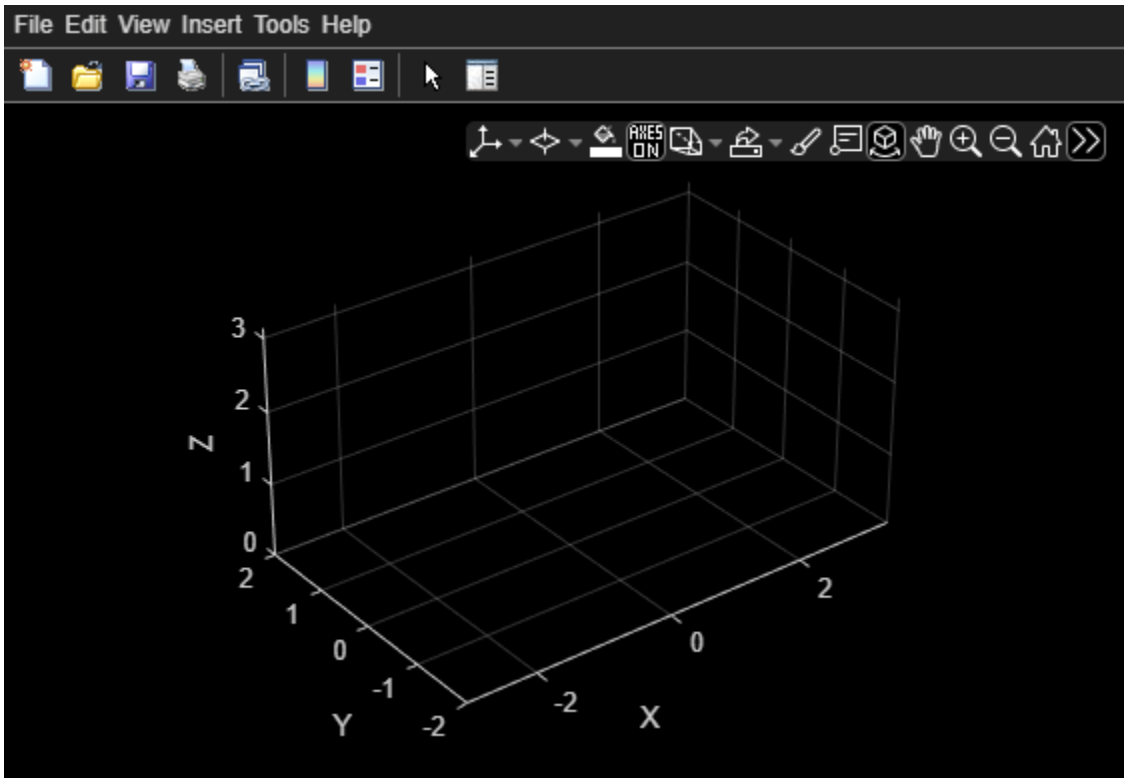
Hide and Show 3-D Point Cloud Figure

Load point cloud.

```
ptCloud = pcread('teapot.ply');
```

Create the player and customize player axis labels.

```
player = pcplayer(ptCloud.XLimits,ptCloud.YLimits,ptCloud.ZLimits);
```

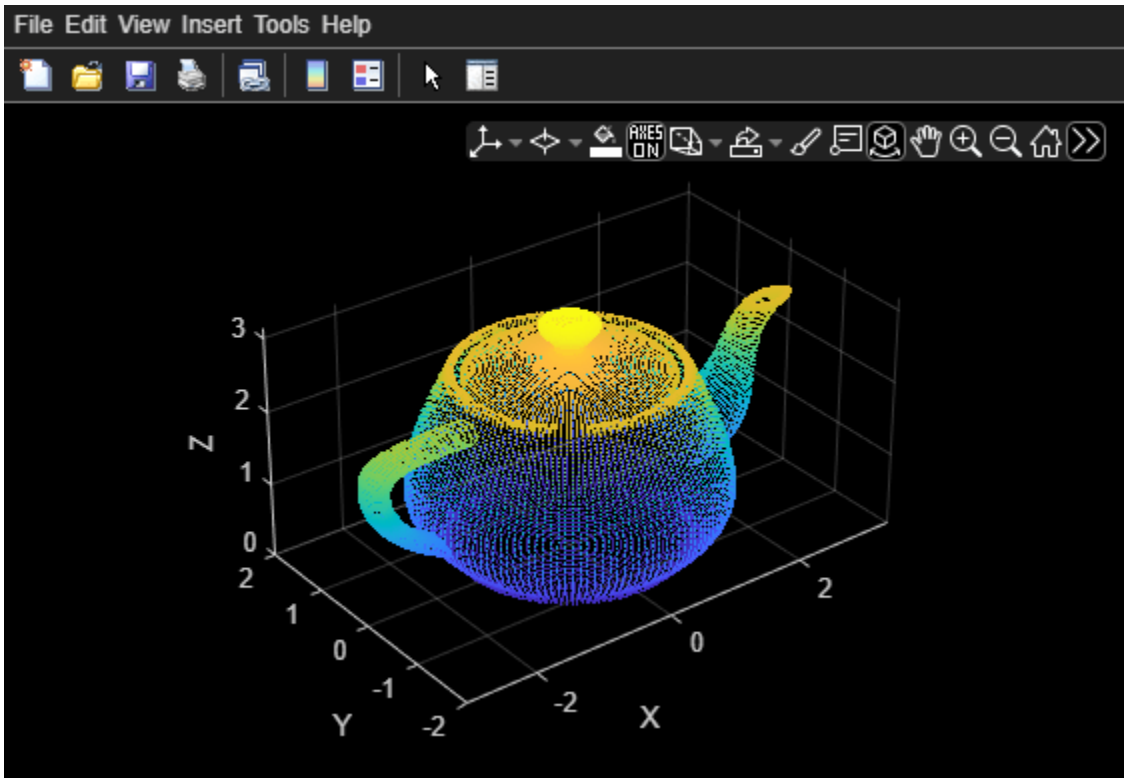


Hide figure.

```
hide(player)
```

Show figure.

```
show(player)
view(player,ptCloud);
```



Input Arguments

`player` — Player

object

Player for visualizing data streams, specified as a `pcplayer` or a `vision.VideoPlayer` object. Use this method to view the figure after you have removed it from display. For example, after you x-out of a figure and you want to view it again. This is particularly useful to use after a while loop that contains display code ends.

Version History

Introduced in R2015b

initializeSearchWindow

Set initial search window

Syntax

```
initializeSearchWindow(hbtracker,R)
```

Description

`initializeSearchWindow(hbtracker,R)` sets the initial search window region,R. The tracker uses this region as the initial window to search for the object. You can also use this function when the tracker loses track of the object. Use the function to reinitialize an object's initial location and size.

Examples

Track a Face

Track and display a face in each frame of an input video.

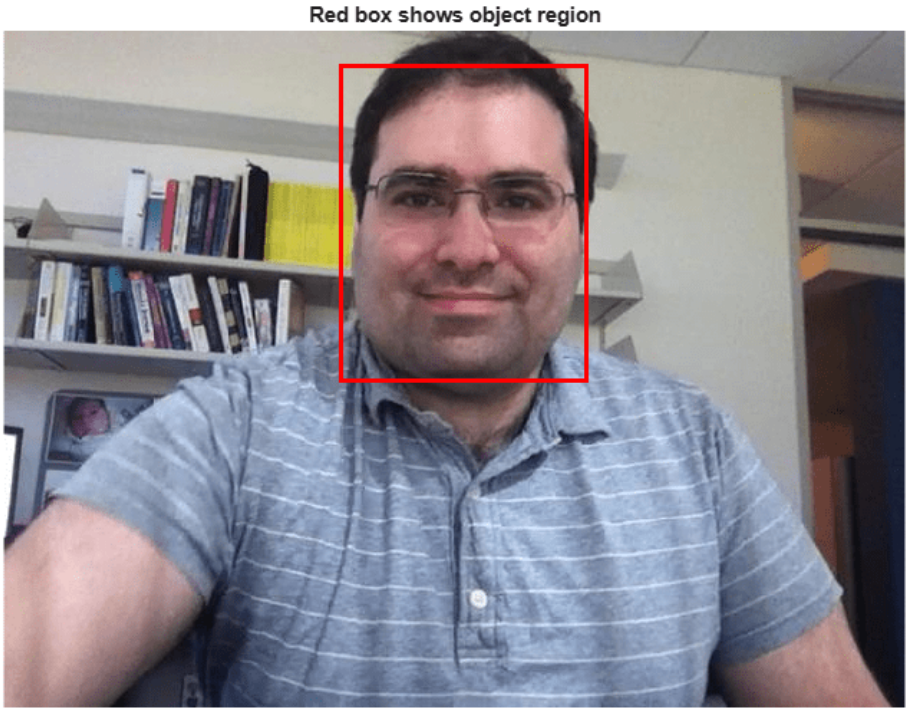
Load in the input video and create a System object displaying video.

```
videoReader = VideoReader("tilted_face.avi");
videoPlayer = vision.VideoPlayer();
```

Read the first video frame, which contains the object. Convert the image to HSV color space. Then define and display the object region.

```
objectFrame = im2single(readFrame(videoReader));
objectHSV = rgb2hsv(objectFrame);
objectRegion = [240, 25, 175, 225];
objectImage = insertShape(objectFrame,"rectangle",objectRegion,LineWidth=3,Color="red");

figure;
imshow(objectImage);
title("Red box shows object region")
```

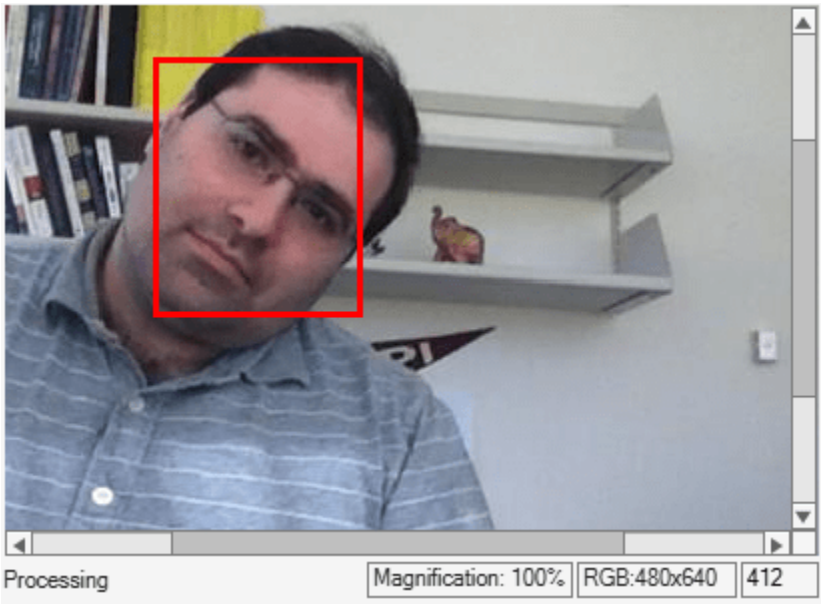


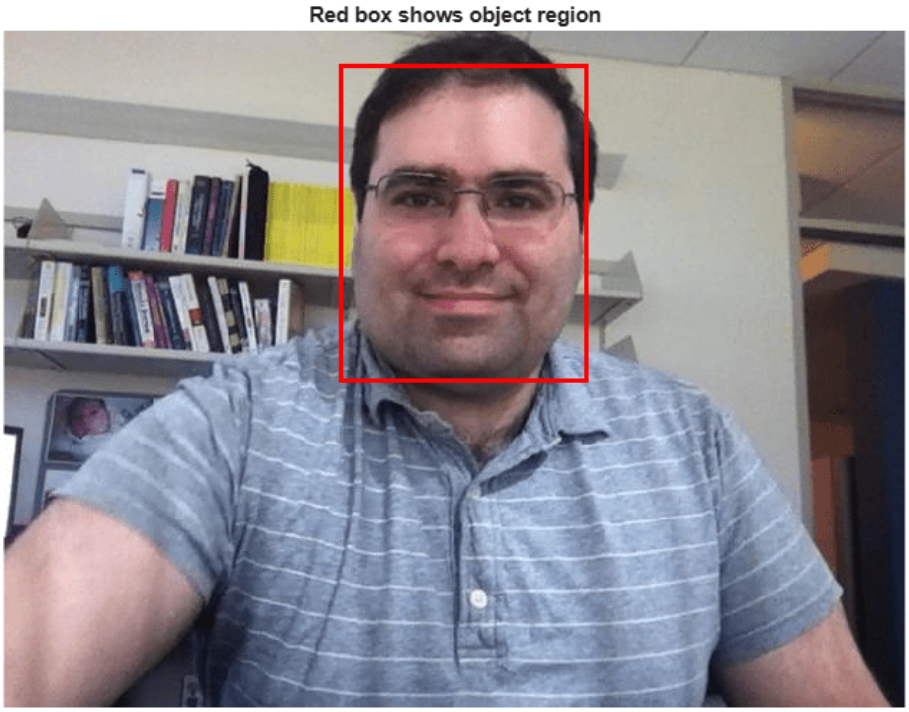
Set the object, based on the hue channel of the first video frame.

```
tracker = vision.HistogramBasedTracker;
initializeObject(tracker, objectHSV(:,:,1),objectRegion);
```

Track and display the object in each video frame. The while loop reads each image frame, converts the image to HSV color space, then tracks the object in the hue channel where it is distinct from the background. Finally, the example draws a box around the object and displays the results.

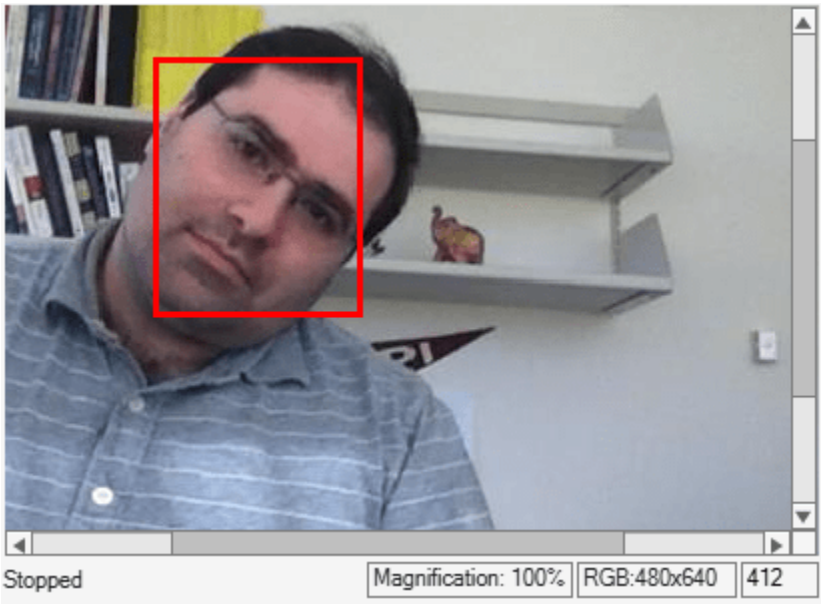
```
while hasFrame(videoReader)
  frame = im2single(readFrame(videoReader));
  hsvFrame = rgb2hsv(frame);
  bbox = tracker(hsvFrame(:,:,1));
  out = insertShape(frame,"rectangle",bbox,LineWidth=3,Color="red");
  videoPlayer(out);
end
```





Release the video player.

```
release(videoPlayer);
```



Copyright 2012-2025 The MathWorks, Inc.

Input Arguments

`hbtracker` — Histogram based tracker
`vision.HistogramBasedTracker` object

Histogram based tracker, specified as a `vision.HistogramBasedTracker` object.

R — Initial search window
[*x y width height*]

Initial search window, specified in the format [*x y width height*].

Version History

Introduced in R2012a

initialize

Initialize video frame and points to track

Syntax

```
initialize(pointTracker,points,I)
```

Description

`initialize(pointTracker,points,I)` initializes points to track and sets the initial video frame. The function sets the *M*-by-2 `points` array of [x y] coordinates with the points to track, and sets the initial video frame, `I`.

If you want to use the point tracker as a persistent variable, you must call `initialize` only during creation. If you call `initialize` in a loop, the previous state is lost and therefore, the tracker cannot maintain tracking.

Examples

Track a Face in Scene

Create System objects for reading and displaying video and for drawing a bounding box of the object.

```
videoReader = VideoReader('visionface.avi');
videoPlayer = vision.VideoPlayer('Position',[100,100,680,520]);
```

Read the first video frame, which contains the object, define the region.

```
objectFrame = readFrame(videoReader);
objectRegion = [264,122,93,93];
```

As an alternative, you can use the following commands to select the object region using a mouse. The object must occupy the majority of the region:

figure; imshow(objectFrame);

objectRegion=round(getPosition(imrect))

Show initial frame with a red bounding box.

```
objectImage = insertShape(objectFrame,'rectangle',objectRegion,'ShapeColor','red');
figure;
imshow(objectImage);
title('Red box shows object region');
```

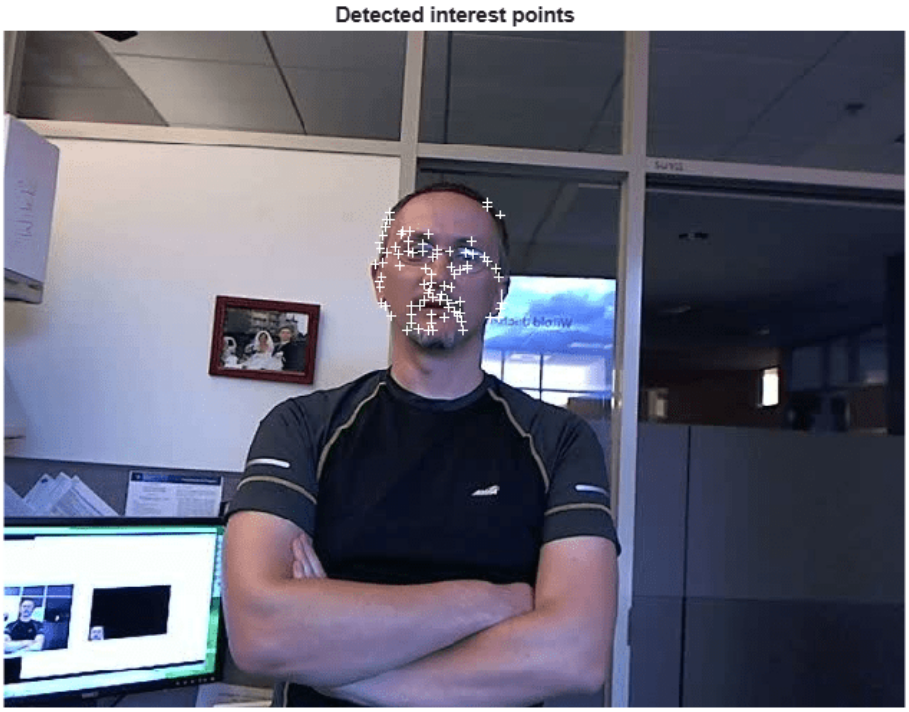


Detect interest points in the object region.

```
points = detectMinEigenFeatures(im2gray(objectFrame),'ROI',objectRegion);
```

Display the detected points.

```
pointImage = insertMarker(objectFrame,points.Location,'+','MarkerColor','white');
figure;
imshow(pointImage);
title('Detected interest points');
```



Create a tracker object.

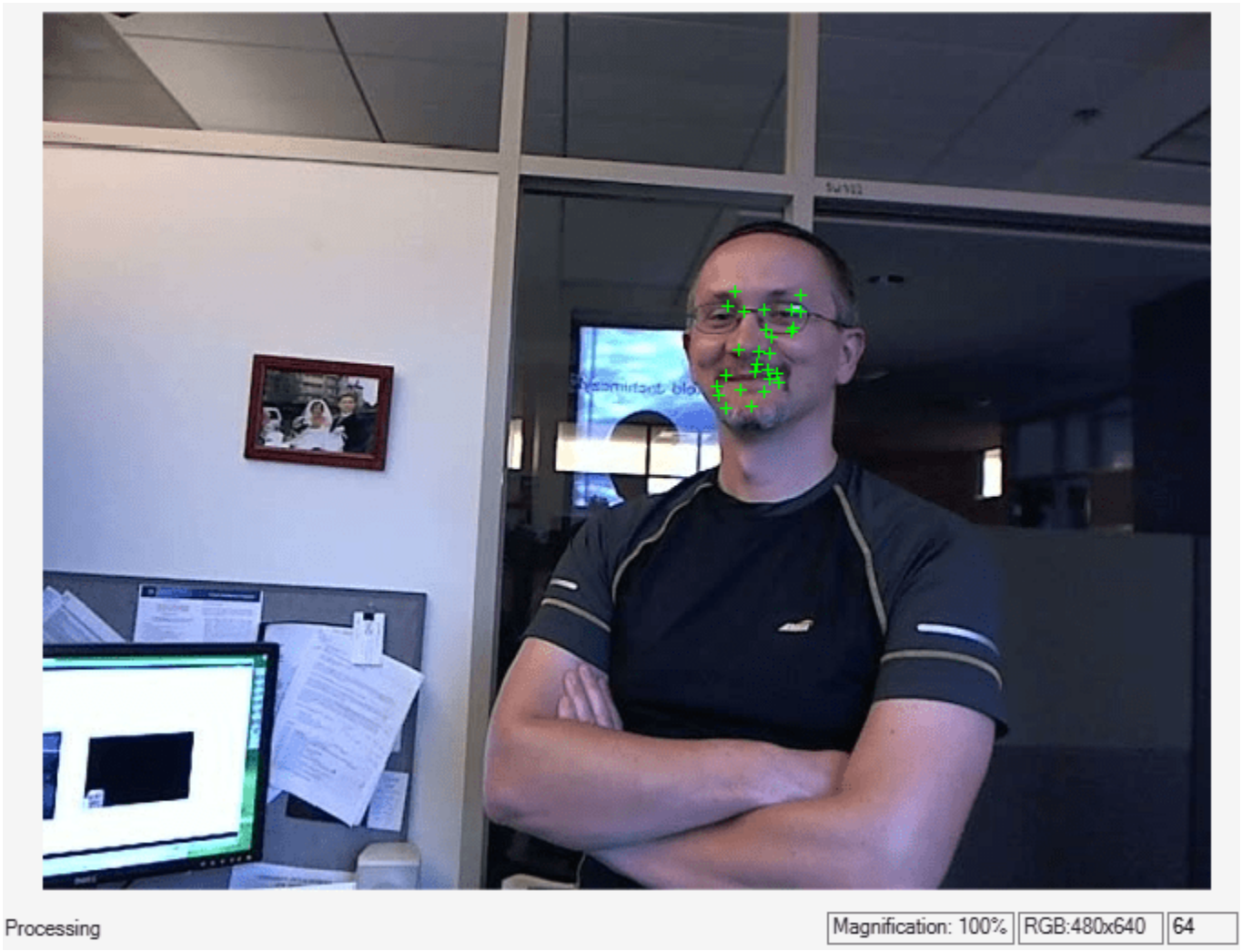
```
tracker = vision.PointTracker('MaxBidirectionalError',1);
```

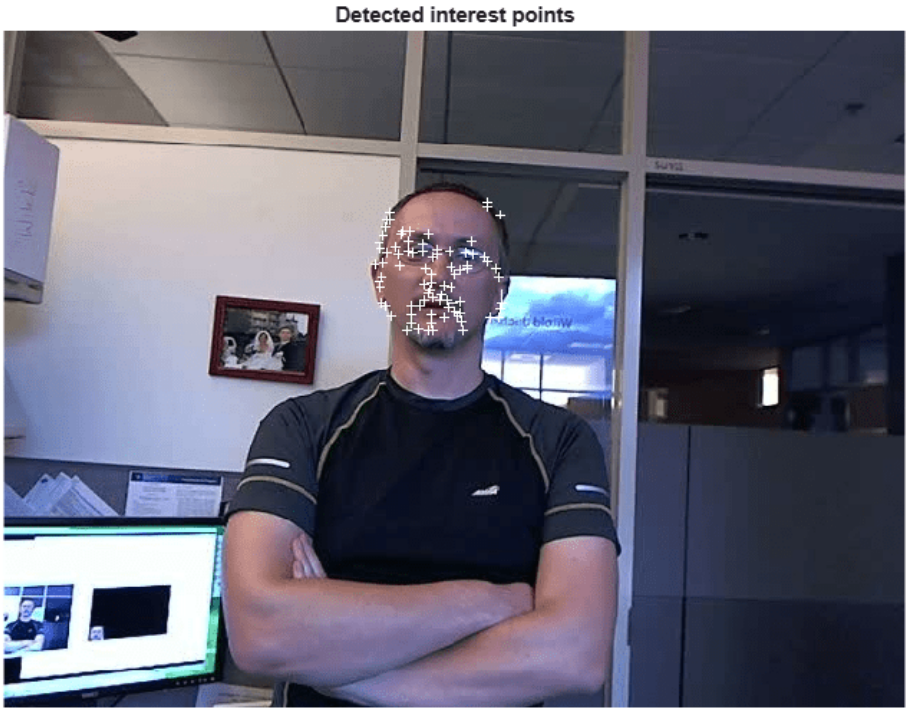
Initialize the tracker.

```
initialize(tracker,points.Location,objectFrame);
```

Read, track, display points, and results in each video frame.

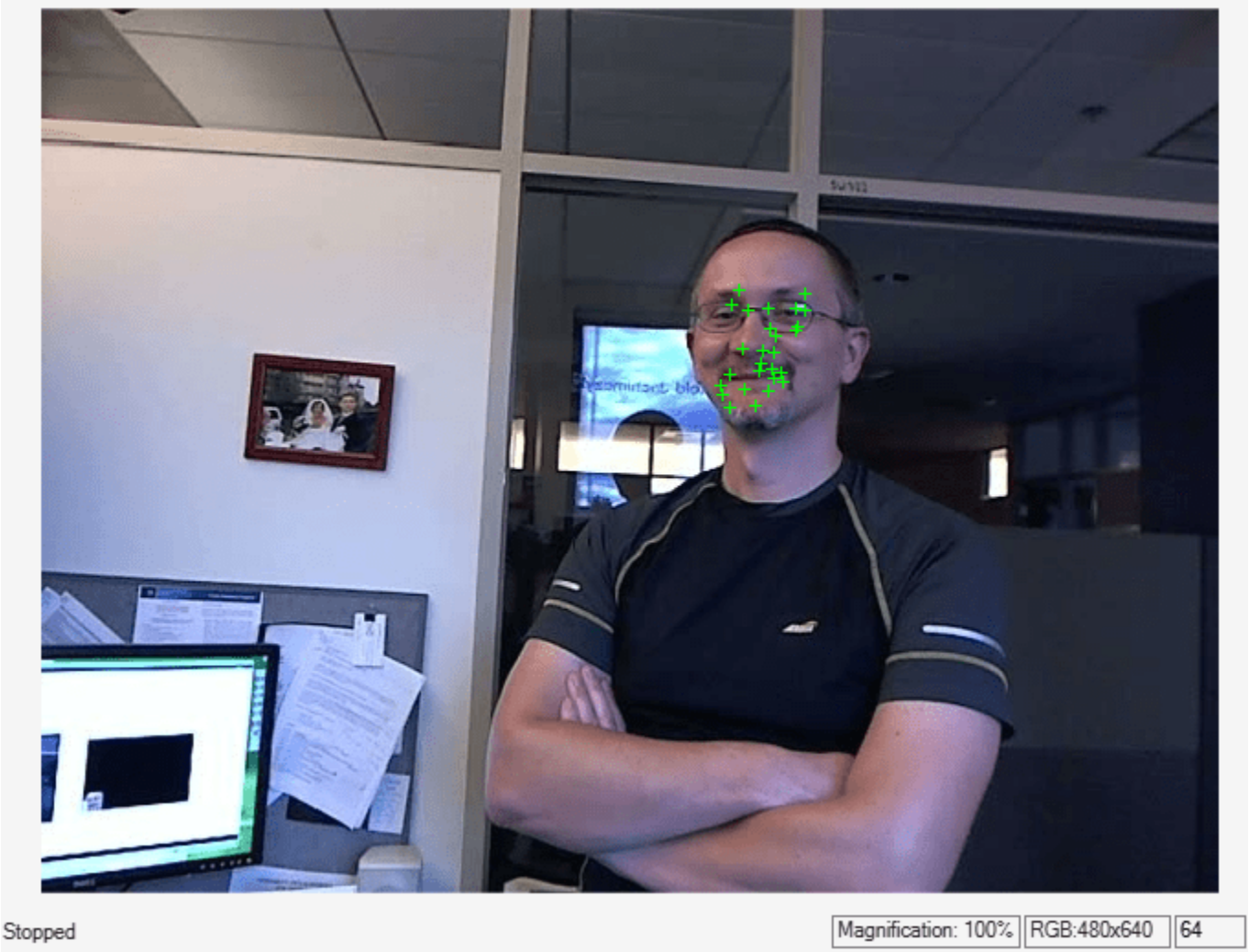
```
while hasFrame(videoReader)
      frame = readFrame(videoReader);
      [points,validity] = tracker(frame);
      out = insertMarker(frame,points(validity, :),'+');
      videoPlayer(out);
end
```


Processing
Magnification: 100%
RGB:480x640
64



Release the video player.

```
release(videoPlayer);
```



Input Arguments

`pointTracker` — Point tracker
`PointTracker` object

Point tracker, specified as a `vision.PointTracker` object.

`points` — Points
M-by-2 array

Points, specified as an *M*-by-2 array of [x y] coordinates that correspond to the locations of the points in the input frame, `I`.

`I` — Video frame
grayscale | truecolor (RGB)

Video frame, specified as grayscale or truecolor (RGB) and must be the same size as the images read into the tracker.

Version History

Introduced in R2012b

unet

Create U-Net convolutional neural network for semantic segmentation

Syntax

```
unetNetwork = unet(imageSize,numClasses)
[unetNetwork,outputSize] = unet(imageSize,numClasses)
___ = unet(imageSize,numClasses,Name=Value)
```

Description

`unetNetwork = unet(imageSize,numClasses)` returns a U-Net network.

Use `unet` to create the U-Net network architecture. You must train the network using the Deep Learning Toolbox function `trainnet`.

`[unetNetwork,outputSize] = unet(imageSize,numClasses)` also returns the size of the output size from the U-Net network.

`___ = unet(imageSize,numClasses,Name=Value)` specifies options using one or more name-value arguments. For example, `unet(imageSize,numClasses,NumFirstEncoderFilters=64)` specifies the number of output channels as `64` for the first encoder stage.

Examples

Create U-Net Network with Custom Encoder-Decoder Depth

Create a U-Net network with an encoder-decoder depth of 3.

```
imageSize = [480 640 3];
numClasses = 5;
encoderDepth = 3;
unetNetwork = unet(imageSize,numClasses,EncoderDepth=encoderDepth)
```

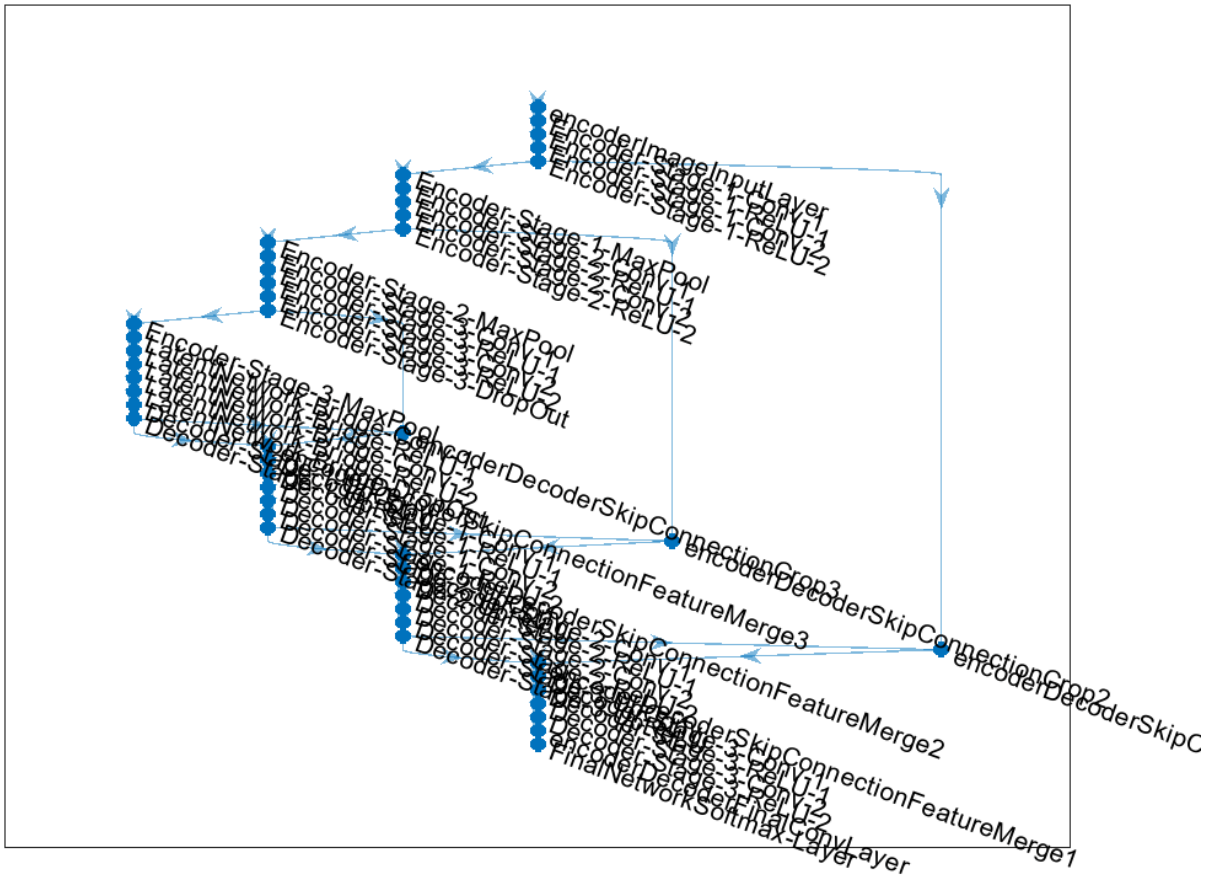
```
unetNetwork =
  dlnetwork with properties:

         Layers: [48×1 nnet.cnn.layer.Layer]
    Connections: [53×2 table]
     Learnables: [36×3 table]
          State: [0×3 table]
     InputNames: {'encoderImageInputLayer'}
    OutputNames: {'FinalNetworkSoftmax-Layer'}
    Initialized: 1

  View summary with summary.
```

Display the network.

```
plot(unetNetwork)
```



Train U-Net Network for Semantic Segmentation

Load training images and pixel labels into the workspace.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
imageDir = fullfile(dataSetDir,"trainingImages");
labelDir = fullfile(dataSetDir,"trainingLabels");
```

Create an `imageDatastore` object to store the training images.

```
imds = imageDatastore(imageDir);
```

Define the class names and their associated label IDs.

```
classNames = ["triangle","background"];
labelIDs   = [255 0];
```

Create a `pixelLabelDatastore` object to store the ground truth pixel labels for the training images.

```
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Create the U-Net network.

```
imageSize = [32 32];
numClasses = 2;
unetNetwork = unet(imageSize, numClasses)
```

```
unetNetwork =
  dlnetwork with properties:

         Layers: [61×1 nnet.cnn.layer.Layer]
    Connections: [68×2 table]
     Learnables: [46×3 table]
          State: [0×3 table]
     InputNames: {'encoderImageInputLayer'}
    OutputNames: {'FinalNetworkSoftmax-Layer'}
    Initialized: 1

  View summary with summary.
```

Create a datastore for training the network.

```
ds = combine(imds,pxds);
```

Set training options.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=1e-3, ...
    MaxEpochs=20, ...
    VerboseFrequency=10);
```

Train the network.

```
net = trainnet(ds,unetNetwork,"crossentropy",options)
```

```
    Iteration    Epoch    TimeElapsed    LearnRate    TrainingLoss
    _________    _____    ___________    _________    ____________

            1        1       00:00:05        0.001          3.2975
           10       10       00:00:48        0.001          0.6778
           20       20       00:01:36        0.001         0.27066
Training stopped: Max epochs completed

net =
  dlnetwork with properties:

         Layers: [61×1 nnet.cnn.layer.Layer]
    Connections: [68×2 table]
     Learnables: [46×3 table]
          State: [0×3 table]
     InputNames: {'encoderImageInputLayer'}
    OutputNames: {'FinalNetworkSoftmax-Layer'}
    Initialized: 1

  View summary with summary.
```

Input Arguments

`imageSize` — Network input image size

2-element vector | 3-element vector

Network input image size, specified as a:

- 2-element vector in the form [*height*, *width*].
- 3-element vector in the form [*height*, *width*, *depth*]. *depth* is the number of image channels. Set *depth* to 3 for RGB images, to 1 for grayscale images, or to the number of channels for multispectral and hyperspectral images.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numClasses` — Number of classes

integer greater than 1

Number of classes in the semantic segmentation, specified as an integer greater than 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `EncoderDepth=3` specifies the encoder depth as 3.

`EncoderNetwork` — Encoder network

`dlnetwork` object

Encoder network that `unet` uses as the encoder, specified as a `dlnetwork` object. You can specify a pretrained or custom encoder network. To use a pretrained encoder network, create the network using the `pretrainedEncoderNetwork` function.

Note If you specify `EncoderNetwork`, the `unet` function does not use these name-value arguments to configure the network:

- `NumFirstEncoderFilters`
- `FilterSize`
- `ConvolutionPadding`

`EncoderDepth` — Encoder depth

4 (default) | positive integer

Encoder depth, specified as a positive integer. U-Net is composed of an encoder subnetwork and a corresponding decoder subnetwork. The depth of these networks determines the number of times the input image is downsampled or upsampled during processing. The encoder network downsamples the input image by a factor of 2^D, where D is the value of `EncoderDepth`. The decoder network upsamples the encoder network output by a factor of 2^D.

Note If you also specify `EncoderNetwork`, specify the value of `EncoderDepth` using the depth of the `EncoderNetwork` input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumFirstEncoderFilters` — Number of output channels for first encoder
64 (default) | positive integer

Number of output channels for the first encoder stage, specified as a positive integer or vector of positive integers. In each subsequent encoder stage, the number of output channels doubles. The `unet` function sets the number of output channels in each decoder stage to match the number in the corresponding encoder stage.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Convolutional layer filter size
3 (default) | positive odd integer | 2-element row vector of positive odd integers

Convolutional layer filter size, specified as a positive odd integer or a 2-element row vector of positive odd integers. Typical values are in the range [3, 7].

`FilterSize`	Description
scalar	The filter is square.
2-element row vector	The filter has the size [*height width*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConvolutionPadding` — Type of padding
`"same"` (default) | `"valid"`

Type of padding, specified as `"same"` or `"valid"`. The type of padding specifies the padding style for the `convolution2dLayer` in the encoder and the decoder subnetworks. The spatial size of the output feature map depends on the type of padding. If you specify type of padding as:

- `"same"` — Zero padding is applied to the inputs to convolution layers such that the output and input feature maps are the same size.
- `"valid"` — Zero padding is not applied to the inputs to convolution layers. The convolution layer returns only values of the convolution that are computed without zero padding. The output feature map is smaller than the input feature map.

Note To ensure that the *height* and *width* of the inputs to max-pooling layers are even, choose the network input image size to confirm to any one of these criteria:

- If you specify `ConvolutionPadding` as `"same"`, then the *height* and *width* of the input image must be a multiple of 2^D.
- If you specify `ConvolutionPadding` as `"valid"`, then the *height* and *width* of the input image must be chosen such that $\frac{1}{2^D}\left(height - \sum_{i=1}^{D} 2^i(f_h - 1)\right)$ and $\frac{1}{2^D}\left(width - \sum_{i=1}^{D} 2^i(f_w - 1)\right)$ are multiples of 2^D.

where f_h and f_w are the height and width of the two-dimensional convolution kernel, respectively. D is the encoder depth.

Data Types: `char` | `string`

Output Arguments

`unetNetwork` — Layers

`dlnetwork` object

Layers that represent the U-Net network architecture, returned as a `dlnetwork` object.

`outputSize` — Network output image size

three-element vector

Network output image size, returned as a three-element vector of the form [*height*, *width*, *channels*]. *channels* is the number of output channels and it is equal to the number of classes specified at the input. The *height* and *width* of the output image from the network depend on the type of padding convolution.

- If you specify `ConvolutionPadding` as `"same"`, then the *height* and *width* of the network output image are the same as that of the network input image.
- If you specify `ConvolutionPadding` as `"valid"`, then the *height* and *width* of the network output image are less than that of the network input image.

Data Types: `double`

More About

U-Net Architecture

- The U-Net architecture consists of an encoder subnetwork and decoder subnetwork that are connected by a bridge section.
- The encoder and decoder subnetworks in the U-Net architecture consists of multiple stages. `EncoderDepth`, which specifies the depth of the encoder and decoder subnetworks, sets the number of stages.
- The stages within the U-Net encoder subnetwork consist of two sets of convolutional and ReLU layers, followed by a 2-by-2 max pooling layer. The decoder subnetwork consists of a transposed convolution layer for upsampling, followed by two sets of convolutional and ReLU layers.
- The bridge section consists of two sets of convolution and ReLU layers.
- The bias term of all convolutional layers is initialized to zero.
- Convolution layer weights in the encoder and decoder subnetworks are initialized using the `'He'` weight initialization method [2].

Tips

- Use `'same'` padding in convolution layers to maintain the same data size from input to output and enable the use of a broad set of input image sizes.

- Use patch-based approaches for seamless segmentation of large images. You can extract image patches by using the `randomPatchExtractionDatastore` function.
- Use `'valid'` padding to prevent border artifacts while you use patch-based approaches for segmentation.
- You can use the network created using `unet` function for GPU code generation after training with `trainnet`. For details and examples, see “Generate Code and Deploy Deep Neural Networks” (Deep Learning Toolbox).

Version History

Introduced in R2024a

References

[1] Ronneberger, O., P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351, 2015, pp. 234–241.

[2] He, K., X. Zhang, S. Ren, and J. Sun. "Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1026–1034.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

You can use the U-Net network for code generation. First, create the network using the `unet` function. Then, use the `trainnet` function to train the network for segmentation. After training and evaluating the network, you can generate code for the `dlnetwork` object by using GPU Coder.

See Also

Objects

`dlnetwork`

Functions

`trainnet` | `deeplabv3plus` | `unet3d` | `pretrainedEncoderNetwork` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Semantic Segmentation of Multispectral Images Using Deep Learning”
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

unet3d

Create 3-D U-Net convolutional neural network for semantic segmentation of volumetric images

Syntax

```
unet3dNetwork = unet3d(inputSize,numClasses)
[unet3dNetwork,outputSize] = unet3d(inputSize,numClasses)
[___] = unet3d(inputSize,numClasses,Name=Value)
```

Description

`unet3dNetwork = unet3d(inputSize,numClasses)` returns a 3-D U-Net network. `unet3d` includes a pixel classification layer in the network to predict the categorical label for each pixel in an input volumetric image.

Use `unet3d` to create the network architecture for 3-D U-Net. Train the network using the Deep Learning Toolbox function `trainnet`.

`[unet3dNetwork,outputSize] = unet3d(inputSize,numClasses)` also returns the size of an output volumetric image from the 3-D U-Net network.

`[___] = unet3d(inputSize,numClasses,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntax. For example, specify `unet3d(inputSize,numClasses,EncoderDepth=4)` to set the encoder depth to 4.

Examples

Create 3-D U-Net Network with Custom Encoder-Decoder Depth

Create a 3-D U-Net network with an encoder-decoder depth of 2. Specify the number of output channels for the first convolution layer as 16.

```
imageSize = [128 128 128 3];
numClasses = 5;
encoderDepth = 2;
unet3dNetwork = unet3d(imageSize,numClasses,EncoderDepth=encoderDepth,NumFirstEncoderFilters=16)
```

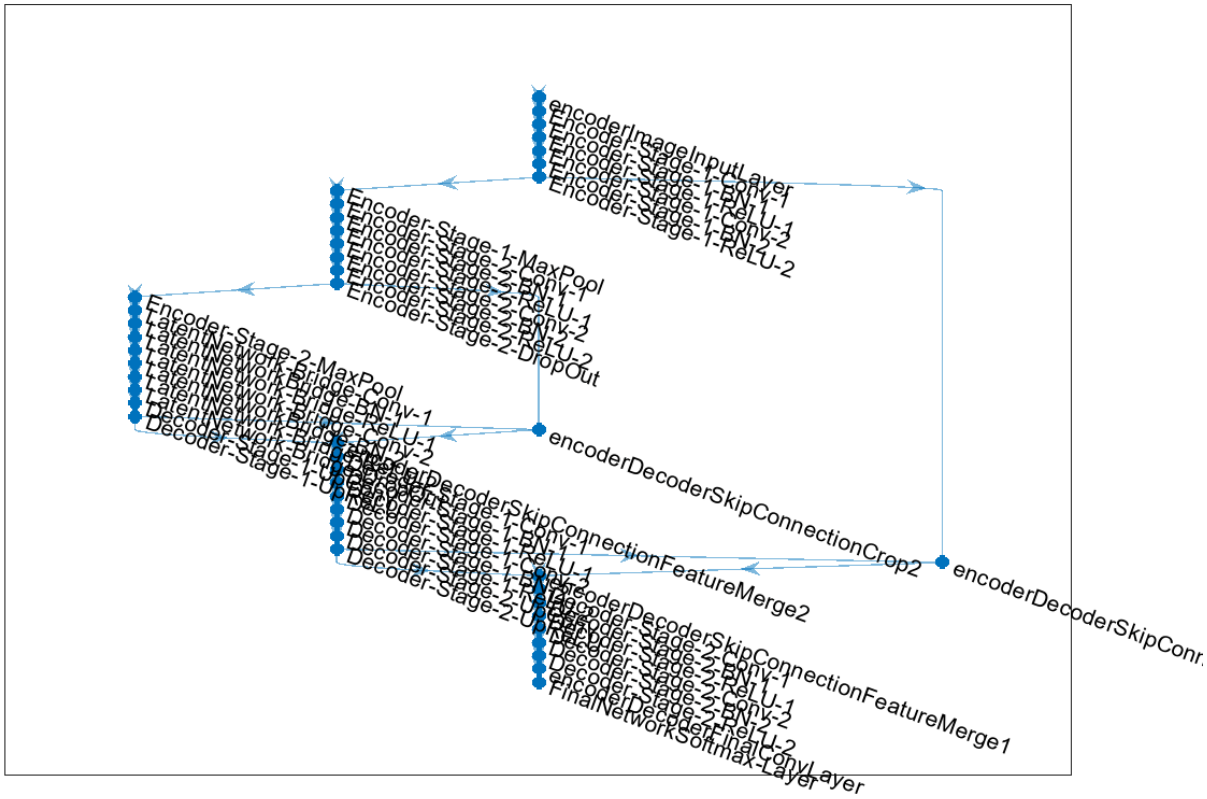
```
unet3dNetwork =
  dlnetwork with properties:

         Layers: [45×1 nnet.cnn.layer.Layer]
    Connections: [48×2 table]
     Learnables: [46×3 table]
          State: [20×3 table]
     InputNames: {'encoderImageInputLayer'}
    OutputNames: {'FinalNetworkSoftmax-Layer'}
    Initialized: 1

  View summary with summary.
```

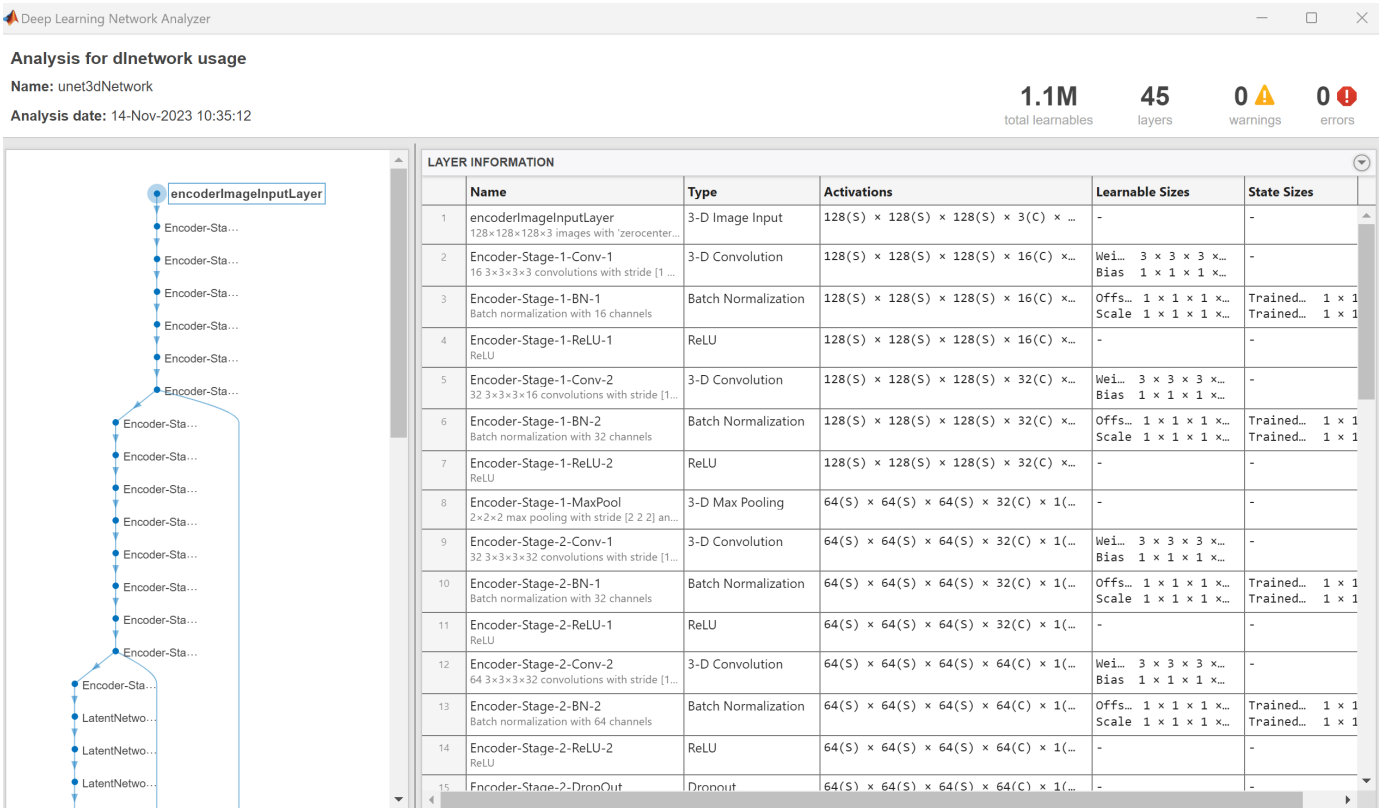
Display the network.

```
figure(Units="normalized",Position=[0 0 0.5 0.55]);
plot(unet3dNetwork)
```



Use the deep learning network analyzer to visualize the 3-D U-Net network.

```
analyzeNetwork(unet3dNetwork);
```



The visualization shows the number of output channels for each encoder stage. The first convolution layers in encoder stages 1 and 2 have 16 and 32 output channels, respectively. The second convolution layers in encoder stages 1 and 2 have 32 and 64 output channels, respectively.

Input Arguments

`inputSize` — Network input image size

three-element vector | four-element vector

Network input image size representing a volumetric image, specified as one of these values:

- Three-element vector of the form [*height width depth*]
- Four-element vector of the form [*height width depth channel*]. *channel* denotes the number of image channels.

Note Network input image size must be chosen such that the dimension of the inputs to the max-pooling layers must be even numbers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numClasses` — Number of classes

scalar greater than 1

Number of classes to segment, specified as a scalar greater than 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `unet3d(inputSize,numClasses,EncoderDepth=4)` sets the encoder depth to 4.

`EncoderNetwork` — Encoder network

`dlnetwork` object

Encoder network that `unet3d` uses as the encoder, specified as a `dlnetwork` object. You can specify a pretrained or custom encoder network. To use a pretrained encoder network, create the network using the `pretrainedEncoderNetwork` function.

Note If you specify `EncoderNetwork`, the `unet3d` function does not use these name-value arguments to configure the network:

- `NumFirstEncoderFilters`
- `FilterSize`
- `ConvolutionPadding`

`EncoderDepth` — Encoder depth

3 (default) | positive integer

Encoder depth, specified as a positive integer. The 3-D U-Net network is composed of an encoder subnetwork and a corresponding decoder subnetwork. The depth of the network determines the number of times the input volumetric image is downsampled or upsampled during processing. The encoder network downsamples the input volumetric image by a factor of 2^D, where D is the value of `EncoderDepth`. The decoder network upsamples the encoder network output by a factor of 2^D. The depth of the decoder subnetwork is same as that of the encoder subnetwork.

Note If you also specify `EncoderNetwork`, specify the value of `EncoderDepth` using the depth of the `EncoderNetwork` input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumFirstEncoderFilters` — Number of output channels for first convolution layer

32 (default) | positive integer

Number of output channels for the first convolution layer in the first encoder stage, specified as a positive integer. The number of output channels for the second convolution layer and the convolution layers in the subsequent encoder stages is set based on this value.

Given stage = {1, 2, ..., *EncoderDepth*}, the number of output channels for the first convolution layer in each encoder stage is equal to

$2^{stage\text{-}1}$ `NumFirstEncoderFilters`

The number of output channels for the second convolution layer in each encoder stage is equal to 2^{stage} `NumFirstEncoderFilters`

The `unet3d` function sets the number of output channels for convolution layers in the decoder stages to match the number in the corresponding encoder stage.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Size of 3-D convolution filter

3 (default) | positive scalar integer | three-element row vector of positive integers

Size of the 3-D convolution filter, specified as a positive scalar integer or a three-element row vector of positive integers of the form [f_h f_w f_d]. Typical values for filter dimensions are in the range [3, 7].

If you specify `FilterSize` as a positive scalar integer of value a, then the convolution kernel is of uniform size [`a a a`].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConvolutionPadding` — Type of padding

`"same"` (default) | `"valid"`

Type of padding, specified as `"same"` or `"valid"`. The type of padding specifies the padding style for the `convolution3dLayer` in the encoder and the decoder subnetworks. The spatial size of the output feature map depends on the type of padding. Specify one of these options:

- `"same"` — Zero padding is applied to the inputs to convolution layers such that the output and input feature maps are the same size.
- `"valid"` — Zero padding is not applied to the inputs to convolution layers. The convolution layer returns only values of the convolution that are computed without zero padding. The output feature map is smaller than the input feature map.

.

Note To ensure that the *height*, *width*, and *depth* values of the inputs to max-pooling layers are even, choose the network input image size to confirm to any one of these criteria:

- If you specify `ConvolutionPadding` as `"same"`, then the *height*, *width*, and *depth* of the input volumetric image must be a multiple of 2^D.
- If you specify `ConvolutionPadding` as `"valid"`, then the *height*, *width*, and *depth* of the input volumetric image must be chosen such that $\frac{1}{2^D}\left(height - \sum_{i=1}^{D} 2^i(f_h - 1)\right)$, $\frac{1}{2^D}\left(width - \sum_{i=1}^{D} 2^i(f_w - 1)\right)$, and $\frac{1}{2^D}\left(depth - \sum_{i=1}^{D} 2^i(f_d - 1)\right)$ are multiples of 2^D.

  where f_h, f_w and f_d are the height, width, and depth of the three-dimensional convolution kernel, respectively. D is the encoder depth.

Data Types: `char` | `string`

Output Arguments

`unet3dNetwork` — Layers
`dlnetwork` object

Layers that represent the 3-D U-Net network architecture, returned as a `dlnetwork` object.

`outputSize` — Network output image size
four-element vector

Network output image size, returned as a four-element vector of the form [*height*, *width*, *depth*, *channels*]. *channels* is the number of output channels and is equal to the number of classes specified at the input. The *height*, *width*, and *depth* of the output image from the network depend on the type of padding convolution.

- If you specify `ConvolutionPadding` as `"same"`, then the *height*, *width*, and *depth* of the network output image are the same as that of the network input image.
- If you specify `ConvolutionPadding` as `"valid"`, then the *height*, *width*, and *depth* of the network output image are less than that of the network input image.

Data Types: `double`

More About

3-D U-Net Architecture

- The 3-D U-Net architecture consists of an encoder subnetwork and decoder subnetwork that are connected by a bridge section.
- The encoder and decoder subnetworks in the 3-D U-Net architecture consist of multiple stages. `EncoderDepth`, which specifies the depth of the encoder and decoder subnetworks, sets the number of stages.
- Each encoder stage in the 3-D U-Net network consists of two sets of convolutional, batch normalization, and ReLU layers. The ReLU layer is followed by a 2-by-2-by-2 max pooling layer. Likewise, each decoder stage consists of a transposed convolution layer for upsampling, followed by two sets of convolutional, batch normalization, and ReLU layers.
- The bridge section consists of two sets of convolution, batch normalization, and ReLU layers.
- The bias term of all convolution layers is initialized to zero.
- Convolution layer weights in the encoder and decoder subnetworks are initialized using the `'He'` weight initialization method.

Tips

- Use `'same'` padding in convolution layers to maintain the same data size from input to output and enable the use of a broad set of input image sizes.
- Use patch-based approaches for seamless segmentation of large images. You can extract image patches by using the `randomPatchExtractionDatastore` function.
- Use `'valid'` padding in convolution layers to prevent border artifacts while you use patch-based approaches for segmentation.

Version History

Introduced in R2024a

References

[1] Çiçek, Ö., A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016. MICCAI 2016. Lecture Notes in Computer Science*. Vol. 9901, pp. 424–432. Springer, Cham.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

You can use the 3-D U-Net network for code generation. First, create the network using the `unet3d` function. Then, use the `trainnet` function to train the network for segmentation. After training and evaluating the network, you can generate code for the `dlnetwork` object by using GPU Coder.

See Also

Objects

`dlnetwork`

Functions

`trainnet` | `unet` | `deeplabv3plus` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

deeplabv3plus

Create DeepLab v3+ convolutional neural network for semantic image segmentation

Syntax

```
deepLabNetwork = deeplabv3plus(imageSize,numClasses,network)
deepLabNetwork = deeplabv3plus( ___ ,DownsamplingFactor=value)
```

Description

`deepLabNetwork = deeplabv3plus(imageSize,numClasses,network)` returns a DeepLab v3+ layer with the specified base network, number of classes, and image size.

`deepLabNetwork = deeplabv3plus( ___ ,DownsamplingFactor=value)` additionally sets the downsampling factor (output stride) [1] to either 8 or 16. The downsampling factor sets the amount the encoder section of DeepLab v3+ downsamples the input image.

Examples

Create DeepLab v3+ Network Based on ResNet-18

Create a DeepLab v3+ network based on ResNet-18.

```
imageSize = [480 640 3];
numClasses = 5;
network = "resnet18";
net = deeplabv3plus(imageSize,numClasses,network, ...
             DownsamplingFactor=16);
```

Display the network.

```
analyzeNetwork(net)
```

Train DeepLab v3+ Network

Load the triangle data set images using an image datastore. The datastore contains 200 grayscale images of random triangles. Each image is 32-by-32.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
imageDir = fullfile(dataSetDir,"trainingImages");
imds = imageDatastore(imageDir);
```

Load the triangle data set pixel labels using a pixel label datastore.

```
labelDir = fullfile(dataSetDir, "trainingLabels");
classNames = ["triangle","background"];
labelIDs   = [255 0];
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Create a DeepLab v3+ network.

```
imageSize = [256 256];
numClasses = numel(classNames);
net = deeplabv3plus(imageSize,numClasses,"resnet18");
```

Combine image and pixel label data for training and apply a preprocessing transform to resize the training images.

```
cds = combine(imds,pxds);
tds = transform(cds, @(data)preprocessTrainingData(data,imageSize));
```

Specify training options. Lower the mini-batch size to reduce memory usage.

```
opts = trainingOptions("sgdm",...
    MiniBatchSize=8,...
    MaxEpochs=3);
```

Train the network.

```
net = trainnet(tds,net,"crossentropy",opts);
```

```
    Iteration    Epoch    TimeElapsed    LearnRate    TrainingLoss
    _________    _____    ___________    _________    ____________
            1        1       00:00:04         0.01         0.93844
           50        2       00:04:09         0.01        0.033749
           75        3       00:05:35         0.01        0.026353
Training stopped: Max epochs completed
```

Read a test image.

```
I = imread("triangleTest.jpg");
```

Resize the test image by a factor equal to the input image size divided by 32 so that the triangles in the test image are roughly equal to the size of the triangles during training.

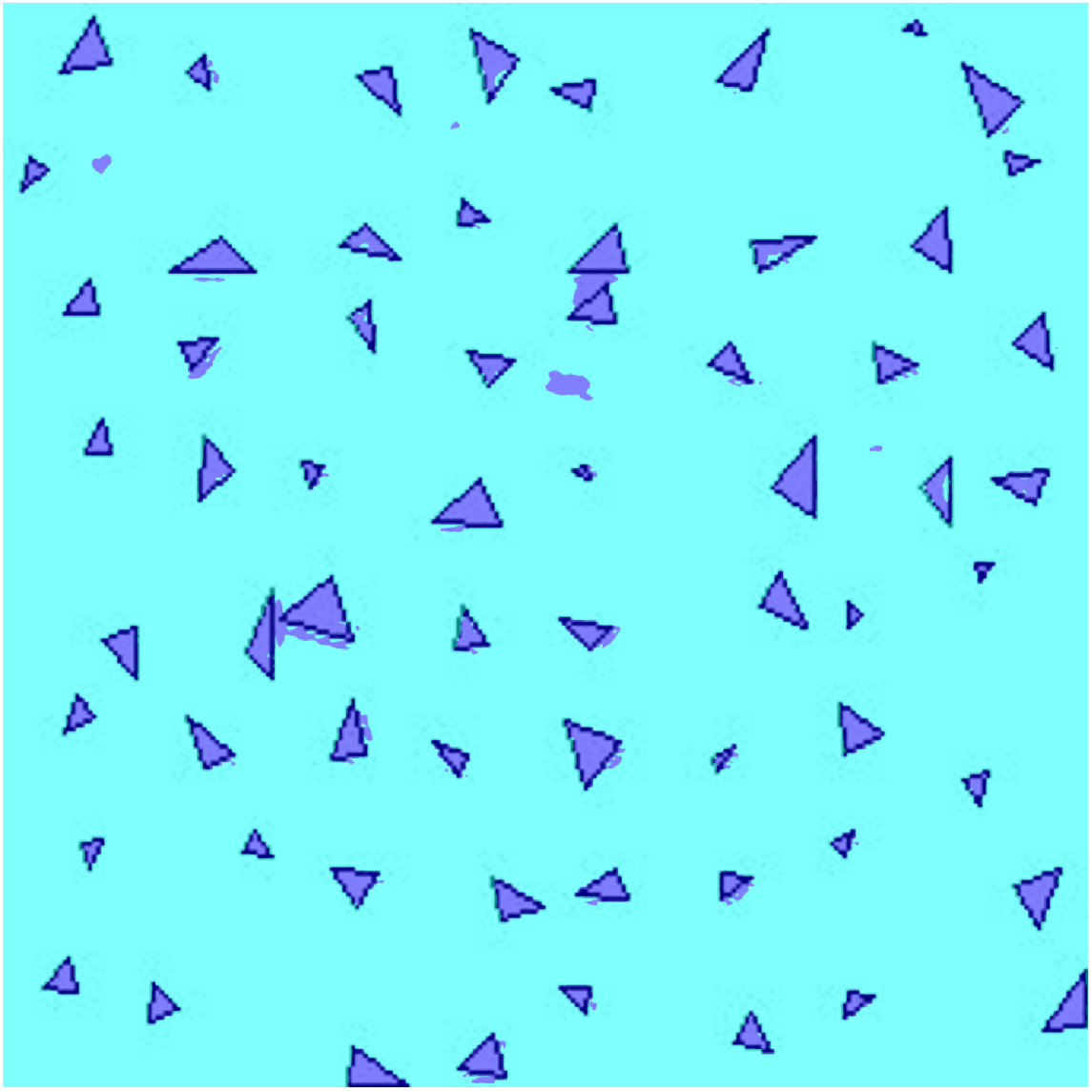
```
I = imresize(I,Scale=imageSize./32);
```

Segment the image.

```
C = semanticseg(I,net);
```

Display the results.

```
B = labeloverlay(I,C);
figure
imshow(B)
```



Supporting Functions

```
function data = preprocessTrainingData(data, imageSize)
% Resize the training image and associated pixel label image.
data{1} = imresize(data{1},imageSize);
data{2} = imresize(data{2},imageSize);

% Convert grayscale input image into RGB for use with ResNet-18, which
% requires RGB image input.
```

```
data{1} = repmat(data{1},1,1,3);
end
```

Input Arguments

`imageSize` — Network input image size

RGB image

Network input image size, specified as an RGB image. The image can have one of these formats.

- 2-element vector in the format [*height width*].
- 3-element vector in the format [*height width*, `3`].

If you specify `imageSize` as a 2-element vector [*height width*], the network will automatically use [*height width* 3] as the input size for RGB images. You must first convert grayscale images to RGB before passing them to the network.

`numClasses` — Number of classes

integer greater than 1

Number of classes for network to classify, specified as an integer greater than 1.

`network` — Base network

`"resnet18"` | `"resnet50"` | `"mobilenetv2"` | `"xception"` | `"inceptionresnetv2"`

Base network, specified as one of these options:

- `"resnet18"` — The base network is a ResNet-18 network.
- `"resnet50"` — The base network is a ResNet-50 network.
- `"mobilenetv2"` — The base network is a MobileNet-v2 network.
- `"xception"` — The base network is an Xception network.
- `"inceptionresnetv2"` — The base network is an Inception-ResNet-v2 network.

To specify each type of base network, you must install the corresponding network add-on. To learn more about these networks, see “Pretrained Deep Neural Networks” (Deep Learning Toolbox).

Output Arguments

`deepLabNetwork` — DeepLab v3+ network

`dlnetwork` object

DeepLab v3+ network, returned as a `dlnetwork` object for semantic image segmentation. The network uses encoder-decoder architecture, dilated convolutions, and skip connections to segment images. You must use the `trainnet` function (requires Deep Learning Toolbox) to train the network before you can use the network for semantic segmentation.

Algorithms

- When you use either the `xception` or `mobilenetv2` base networks to create a DeepLab v3+ network, depth separable convolutions are used in the atrous spatial pyramid pooling (ASPP) and decoder subnetworks. For all other base networks, convolution layers are used.

- This implementation of DeepLab v3+ does not include a global average pooling layer in the ASPP.

Version History

Introduced in R2024a

References

[1] Chen, L., Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation." *Computer Vision — ECCV 2018*, 833-851. Munic, Germany: ECCV, 2018.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation, you must first create a DeepLab v3+ network by using the `deeplabv3plus` function. Then, use the `trainnet` function on the resulting `dlnetwork` object to train the network for segmentation. Once the network is trained and evaluated, you can generate code for the deep learning network object using GPU Coder.

See Also

Objects

`dlnetwork`

Functions

`trainnet` | `unet` | `unet3d` | `semanticseg`

Topics

“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

bisenetv2

Create BiSeNet v2 convolutional neural network for semantic segmentation

Syntax

```
[net,classes] = bisenetv2
net = bisenetv2(imageSize,numClasses)
[net] = bisenetv2(imageSize,numClasses,Name=Value)
```

Description

Use the `bisenetv2` function to semantically segment images using the BiSeNet v2 convolutional neural network. Using the pretrained network, trained on 171 image classes of the COCO-Stuff data set [2], you can perform inference on test images which contain these classes.

To perform semantic segmentation on a custom data set, you must train the network on your data set using the `trainnet` function.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for BiSeNet v2 Semantic Segmentation Network. You can install the Computer Vision Toolbox Model for BiSeNet v2 Semantic Segmentation Network from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[net,classes] = bisenetv2` returns a pretrained BiSeNet v2 network and the names of the 171 classes that it was trained on.

`net = bisenetv2(imageSize,numClasses)` returns a BiSeNet v2 network pretrained on 171 image classes of the COCO-Stuff data set and configured for transfer learning using input images of the specified size `imageSize` and the specified number of classes `numClasses`. Train this network on a custom data set using the using the `trainnet` function.

`[net] = bisenetv2(imageSize,numClasses,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments from the previous syntax. When you specify one or more network options using a name-value argument, you create a custom BiSeNet v2 network with uninitialized weights.

For example, `bisenetv2(ChannelRatio=16)` specifies the channel ratio between the semantic branch and detail branch as `16`.

Examples

Segment Image using Pretrained BiSeNet v2 Network

Create a BiSeNet v2 convolutional neural network for semantic segmentation.

```
[net,classes] = bisenetv2;
```

Read a test image into the workspace.

```
I = imread("kobi.png");
```

Resize the image to the input size of the network.

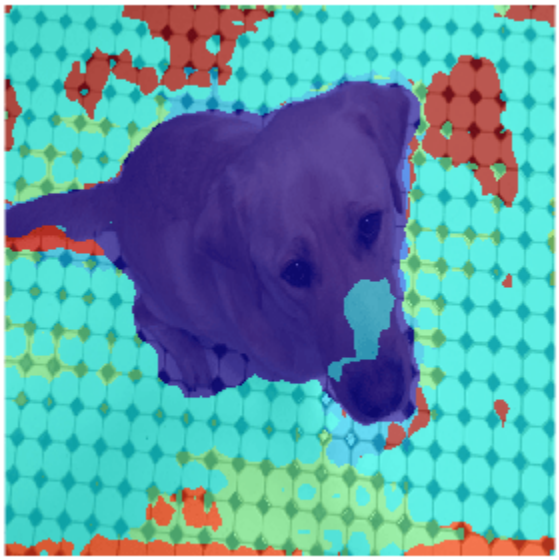
```
inputSize = net.Layers(1).InputSize(1:2);
img = imresize(I,inputSize);
```

Create a segmentation map, a categorical array which relates the labels to each pixel in the input image, of the test image using the `semanticseg` function.

```
segMap = semanticseg(img,net,Classes=classes);
```

Display the segmentation map overlaid on the image, ordered by the smallest object masks on top, using the `labeloverlay` function.

```
segmentedImage = labeloverlay(img,segMap,Transparency=0.4);
imshow(segmentedImage)
```



Input Arguments

`imageSize` — Network input image size

two-element vector | three-element vector

Network input image size, specified as one of these options:

- Two-element vector of the form [*height width*].
- Three-element vector of the form [*height width depth*]. *depth* is the number of image channels. Specify *depth* as 3 for RGB images, as 1 for grayscale images, or as the number of channels for multispectral and hyperspectral images.

`numClasses` — Number of classes
integer greater than 1

Number of classes to segment, specified as an integer greater than 1.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `bisenetv2(ChannelRatio=16)` specifies the channel ratio between the semantic and detail branch as 16.

`ChannelRatio` — Channel ratio
4 (default) | positive integer in the range `[2 64]`

Channel ratio between the semantic branch and detail branch, specified as a positive integer in the range `[2, 64]`. The channel ratio must be a power of 2 no greater than 64.

Tip Decrease this value to enable the detail branch to learn finer features. Increase the channel ratio to enable the semantic branch to capture global context and improve the overall performance of the network, at the expense of less segmented finer features and a slower computing speed. Use empirical analysis to determine the optimal channel ratio for your application.

`ChannelMultiplier` — Channel multiplier
1 (default) | positive scalar

Channel multiplier, specified as a positive scalar. This value scales the number of channels in all convolutional layers other than the inputs to the semantic, detail, and output branches.

Tip Increase this value to improve segmentation accuracy at the cost of larger model size and longer training time.

Depending on the `ChannelRatio` value you specify, the `ChannelMultiplier` argument supports these values:

`ChannelRatio` Value	Supported `ChannelMultiplier` Values
2, 4, 8	1, 1.25, 1.5, 1.75, 2
16	1, 1.5, 2
32	1, 2
64	2

`DepthMultiplier` — Depth multiplier
1 (default) | positive integer

Depth multiplier specified as a positive scalar. This value scales the number of layers of the semantic and detail branches. Increase the depth multiplier to improve learning capacity and accuracy at the

cost of longer processing time. For most applications, use specify a depth multiplier value between 1 and 4.

Output Arguments

`net` — BiSeNet v2 network layers

`dlnetwork` object

BiSeNet v2 network layers, returned as a `dlnetwork` object.

`classes` — Names of classes

categorical array

Names of the classes that the BiSeNet v2 network has been pretrained to segment, returned as a categorical array.

More About

BiSeNet v2 Architecture

- The BiSeNet v2 architecture consists of a detail branch, which captures high-resolution spatial details and a semantic branch, which focuses on contextual information and abstract feature extraction, followed by a feature aggregation layer that combines and balances these pathways.

Version History

Introduced in R2025a

References

[1] Yu, Changqian, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. “BiSeNet V2: Bilateral Network with Guided Aggregation for Real-Time Semantic Segmentation.” *International Journal of Computer Vision* 129, no. 11 (November 2021): 3051–68. https://doi.org/10.1007/s11263-021-01515-2.

[2] Caesar, Holger, Jasper Uijlings, and Vittorio Ferrari. “COCO-Stuff: Thing and Stuff Classes in Context.” In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1209–18. Salt Lake City, UT, USA: IEEE, 2018. https://doi.org/10.1109/CVPR.2018.00132.

See Also

Objects

`dlnetwork`

Functions

`trainnet` | `deeplabv3plus` | `unet` | `unet3d` | `pretrainedEncoderNetwork` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Semantic Segmentation of Multispectral Images Using Deep Learning”
“Get Started with Semantic Segmentation Using Deep Learning”

“Deep Learning in MATLAB” (Deep Learning Toolbox)

fcnLayers

(Removed) Create fully convolutional network layer graph for semantic segmentation

Note `fcnLayers` has been removed. Create an FCN network using a `dlnetwork` object instead. For more information, see “Version History”.

Syntax

```
lgraph = fcnLayers(imageSize,numClasses)
lgraph = fcnLayers(imageSize,numClasses,'Type',type)
```

Description

`lgraph = fcnLayers(imageSize,numClasses)` returns a fully convolutional network (FCN), configured as FCN 8s, for semantic segmentation. The FCN is preinitialized using layers and weights from the VGG-16 network.

`fcnLayers` includes a `pixelClassificationLayer` to predict the categorical label for every pixel in an input image. The pixel classification layer only supports RGB images.

This function requires the Deep Learning Toolbox *Model for VGG-16 Network* support package. If this support package is not installed, then the `vgg16` function provides a download link.

`lgraph = fcnLayers(imageSize,numClasses,'Type',type)` returns an FCN configured as a type specified by `type`.

Examples

Create Fully Convolutional Network 8s

Define the image size and number of classes, then create the network.

```
imageSize = [480 640];
numClasses = 5;
lgraph = fcnLayers(imageSize,numClasses)
```

Display the network.

```
plot(lgraph)
```

Create Fully Convolutional Network 16s

Create a FCN 16s.

```
imageSize = [480 640];
numClasses = 5;
lgraph = fcnLayers(imageSize,numClasses,'Type','16s')
```

Display the network.

```
plot(lgraph)
```

Input Arguments

`imageSize` — Network input image size

2-element vector

Network input image size, specified as a 2-element vector in the format [*height*, *width*]. The minimum image size is [224 224] because an FCN is based on the VGG-16 network.

`numClasses` — Number of classes

integer greater than 1

Number of classes in the semantic segmentation, specified as an integer greater than 1.

`type` — Type of FCN model

`'8s'` (default) | `'16s'` | `'32s'`

Type of FCN model, specified as one of the following:

FCN Model	Description
`'32s'`	Upsamples the final feature map by a factor of 32. This option provides coarse segmentation with a lower computational cost.
`'16s'`	Upsamples the final feature map by a factor of 16 after fusing the feature map from the fourth pooling layer. This additional information from earlier layers provides medium-grain segmentation at the cost of additional computation.
`'8s'`	Upsamples the final feature map by a factor of 8 after fusing feature maps from the third and fourth max pooling layers. This additional information from earlier layers provides finer-grain segmentation at the cost of additional computation.

Output Arguments

`lgraph` — Layers

`LayerGraph` object

Layers that represent the FCN network architecture, returned as a `layerGraph` object.

All transposed convolution layers are initialized using bilinear interpolation weights. All transposed convolution layer bias terms are fixed to zero.

Tips

- Networks produced by `fcnLayers` support GPU code generation for deep learning once they are trained with `trainNetwork`. See “Generate Code and Deploy Deep Neural Networks” (Deep Learning Toolbox) for details and examples.

Version History

Introduced in R2017b

R2026a: Removed
Errors starting in R2026a

The `fcnLayers` function has been removed. Use the `dlnetwork` object instead.

R2024b: Warns
Warns starting in R2024b

The `fcnLayers` function issues a warning that it will be removed in a future release. To update your code, create a `dlnetwork` instead. You can use functions such as `addLayers` and `connectLayers` to build the network.

Do not include output layers in the network. Instead, define a loss function. Here are some sample loss functions appropriate for pixel classification:

```
function loss = modelLoss(Y,T)
  z = generalizedDice(Y,T);
  loss = 1 - mean(z,"all");
end

function loss = modelLoss(Y,T)
  mask = ~isnan(T);
  targets(isnan(T)) = 0;
  loss = crossentropy(Y,T,Mask=mask);
end
```

Specify the loss function when you train the network using the `trainnet` function. For example, this code trains a `dlnetwork` network called `net` using the training data `images` and the loss function `modelLoss`.

```
netTrained = trainnet(images,net,@modelLoss,options);
```

R2024a: To be removed
Not recommended starting in R2024a

The `fcnLayers` function will be removed in a future release.

References

[1] Long, J., E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

See Also

`dlnetwork` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

segnetLayers

(Removed) Create SegNet layer graph for semantic segmentation

Note `segnetLayers` has been removed. For supported semantic segmentation networks, see “Get Started with Semantic Segmentation Using Deep Learning”.

You can create a SegNet network using a `dlnetwork` object. For more information, see “Version History”.

Syntax

```
lgraph = segnetLayers(imageSize,numClasses,model)
lgraph = segnetLayers(imageSize,numClasses,encoderDepth)
lgraph = segnetLayers(imageSize,numClasses,encoderDepth,Name,Value)
```

Description

`lgraph = segnetLayers(imageSize,numClasses,model)` returns SegNet layers, `lgraph`, that is preinitialized with layers and weights from a pretrained `model`.

SegNet is a convolutional neural network for semantic image segmentation. The network uses a `pixelClassificationLayer` to predict the categorical label for every pixel in an input image.

Use `segnetLayers` to create the network architecture for SegNet. You must train the network using the Deep Learning Toolbox function `trainNetwork`.

`lgraph = segnetLayers(imageSize,numClasses,encoderDepth)` returns uninitialized SegNet layers configured using the specified encoder depth.

`lgraph = segnetLayers(imageSize,numClasses,encoderDepth,Name,Value)` returns a SegNet layer with additional options specified by one or more `Name,Value` pair arguments.

Examples

Create SegNet With Custom Encoder-Decoder Depth

Create SegNet layers with an encoder/decoder depth of 4.

```
imageSize = [480 640 3];
numClasses = 5;
encoderDepth = 4;
lgraph = segnetLayers(imageSize,numClasses,encoderDepth)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'inputImage'}
    OutputNames: {'pixelLabels'}
         Layers: [59x1 nnet.cnn.layer.Layer]
```

```
    Connections: [66x2 table]
```

Display the network.

```
figure
plot(lgraph)
```

Train SegNet

Load training images and pixel labels.

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
labelDir = fullfile(dataSetDir,'trainingLabels');
```

Create an image datastore holding the training images.

```
imds = imageDatastore(imageDir);
```

Define the class names and their associated label IDs.

```
classNames = ["triangle", "background"];
labelIDs   = [255 0];
```

Create a pixel label datastore holding the ground truth pixel labels for the training images.

```
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Combine image and pixel label data for training a semantic segmentation network.

```
ds = combine(imds,pxds);
```

Create SegNet layers.

```
imageSize = [32 32];
numClasses = 2;
lgraph = segnetLayers(imageSize,numClasses,2)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'inputImage'}
    OutputNames: {'pixelLabels'}
         Layers: [31x1 nnet.cnn.layer.Layer]
    Connections: [34x2 table]
```

Set up training options.

```
options = trainingOptions('sgdm','InitialLearnRate',1e-3, ...
      'MaxEpochs',20,'VerboseFrequency',10);
```

Train the network.

```
net = trainNetwork(ds,lgraph,options)
```

```
Training on single CPU.
Initializing input data normalization.
```

```
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:03 |       39.75% |       0.7658 |          0.0010 |
|      10 |          10 |       00:00:25 |       49.98% |       0.7388 |          0.0010 |
|      20 |          20 |       00:00:49 |       66.39% |       0.6910 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.


net =
  DAGNetwork with properties:

         Layers: [31x1 nnet.cnn.layer.Layer]
    Connections: [34x2 table]
     InputNames: {'inputImage'}
    OutputNames: {'pixelLabels'}
```

Display the network.

```
plot(lgraph)
```

Input Arguments

`imageSize` — Network input image size

2-element vector | 3-element vector

Network input image size, specified as a:

- 2-element vector in the format [*height*, *width*].
- 3-element vector in the format [*height*, *width*, *depth*]. *depth* is the number of image channels. Set *depth* to `3` for RGB images, `1` for grayscale images, or to the number of channels for multispectral and hyperspectral images.

`numClasses` — Number of classes

integer greater than 1

Number of classes in the semantic segmentation, specified as an integer greater than 1.

`model` — Pretrained network model

`'vgg16'` | `'vgg19'`

Pretrained network model, specified as `'vgg16'` or `'vgg19'`. These models have an encoder depth of 5. When you use a `'vgg16'` model, you must specify RGB inputs. You can convert grayscale images to RGB using the `im2gray` function.

`encoderDepth` — Encoder depth

positive integer

Encoder depth, specified as a positive integer.

SegNet is composed of an encoder and corresponding decoder subnetwork. The depth of these networks determines the number of times the input image is downsampled or upsampled as it is processed. The encoder network downsamples the input image by a factor of 2^D, where D is the value of `encoderDepth`. The decoder network upsamples the encoder network output by a factor of 2^D.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumConvolutionLayers',1`

`NumConvolutionLayers` — Number of convolutional layer sections
2 (default) | positive integer | vector of positive integers

Number of convolutional layers in each encoder and decoder section, specified as a positive integer or vector of positive integers.

`NumConvolutionLayers`	Description
scalar	The same number of layers is used for all encoder and decoder sections.
vector	The *k*th element of `NumConvolutionLayers` is the number of convolution layers in the *k*th encoder section and corresponding decoder section. Typical values are in the range [1, 3].

`NumOutputChannels` — Number of output channels
64 (default) | positive integer | vector of positive integers

Number of output channels for each section in the SegNet encoder network, specified as a positive integer or vector of positive integers. `segnetLayers` sets the number of output channels in the decoder to match the corresponding encoder section.

`NumOutputChannels`	Description
scalar	The same number of output channels is used for all encoder and decoder sections.
vector	The *k*th element of `NumOutputChannels` is the number of output channels of the *k*th encoder section and corresponding decoder section.

`FilterSize` — Convolutional layer filter size
3 (default) | positive odd integer | 2-element row vector of positive odd integers

Convolutional layer filter size, specified as a positive odd integer or a 2-element row vector of positive odd integers. Typical values are in the range [3, 7].

`FilterSize`	Description
scalar	The filter is square.
2-element row vector	The filter has the size [*height width*].

Output Arguments

`lgraph` — Layers

`LayerGraph` object

Layers that represent the SegNet network architecture, returned as a `layerGraph` object.

Tips

- The sections within the SegNet encoder and decoder subnetworks are made up of convolutional, batch normalization, and ReLU layers.
- All convolutional layers are configured such that the bias term is fixed to zero.
- Convolution layer weights in the encoder and decoder subnetworks are initialized using the `'MSRA'` weight initialization method. For `'vgg16'` or `'vgg19'` models, only the decoder subnetwork is initialized using MSRA.[1]
- Networks produced by `segnetLayers` support GPU code generation for deep learning once they are trained with `trainNetwork`. See “Generate Code and Deploy Deep Neural Networks” (Deep Learning Toolbox) for details and examples.

Version History

Introduced in R2017b

R2026a: Removed

Errors starting in R2026a

The `segnetLayers` function has been removed. To update your code, create a `dlnetwork` instead. You can use functions such as `addLayers` and `connectLayers` to build the network.

R2024b: Warns

Warns starting in R2024b

The `segnetLayers` function issues a warning that it will be removed in a future release. To update your code, create a `dlnetwork` instead. You can use functions such as `addLayers` and `connectLayers` to build the network.

Do not include output layers in the network. Instead, define a loss function. Here are some sample loss functions appropriate for pixel classification:

```
function loss = modelLoss(Y,T)
  z = generalizedDice(Y,T);
  loss = 1 - mean(z,"all");
end

function loss = modelLoss(Y,T)
  mask = ~isnan(T);
  targets(isnan(T)) = 0;
  loss = crossentropy(Y,T,Mask=mask);
end
```

Specify the loss function when you train the network using the `trainnet` function. For example, this code trains a `dlnetwork` network called `net` using the training data `images` and the loss function `modelLoss`.

```
netTrained = trainnet(images,net,@modelLoss,options);
```

R2024a: To be removed
Not recommended starting in R2024a

The `segnetLayers` function will be removed in a future release. For supported semantic segmentation networks, see “Get Started with Semantic Segmentation Using Deep Learning”.

References

[1] He, K., X. Zhang, S. Ren, and J. Sun. "Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1026–1034.

[2] Badrinarayanan, V., A. Kendall, and R. Cipolla. "Segnet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." *arXiv*. Preprint arXiv: 1511.0051, 2015.

Extended Capabilities

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

For code generation, you must first create a SegNet network by using the `segnetLayers` function. Then, use the `trainNetwork` function on the resulting `lgraph` object to train the network for segmentation. Once the network is trained and evaluated, you can generate code for the `DAGNetwork` object using GPU Coder.

See Also
`dlnetwork` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

unetLayers

(Removed) Create U-Net layers for semantic segmentation

Note `unetLayers` has been removed. Use the `unet` function instead. For more information, see "Version History".

Syntax

```
lgraph = unetLayers(imageSize,numClasses)
[lgraph,outputSize] = unetLayers(imageSize,numClasses)
___ = unetLayers(imageSize,numClasses,Name,Value)
```

Description

`lgraph = unetLayers(imageSize,numClasses)` returns a U-Net network. `unetLayers` includes a pixel classification layer in the network to predict the categorical label for every pixel in an input image.

Use `unetLayers` to create the U-Net network architecture. You must train the network using the Deep Learning Toolbox function `trainNetwork`.

`[lgraph,outputSize] = unetLayers(imageSize,numClasses)` also returns the size of the output size from the U-Net network.

`___ = unetLayers(imageSize,numClasses,Name,Value)` specifies options using one or more name-value pair arguments. For example, `unetLayers(imageSize,numClasses,'NumFirstEncoderFilters',64)` additionally sets the number of output channels to `64` for the first encoder stage.

Examples

Create U-Net Network with Custom Encoder-Decoder Depth

Create a U-Net network with an encoder-decoder depth of 3.

```
imageSize = [480 640 3];
numClasses = 5;
encoderDepth = 3;
lgraph = unetLayers(imageSize,numClasses,'EncoderDepth',encoderDepth)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
         Layers: [46x1 nnet.cnn.layer.Layer]
```

```
    Connections: [48x2 table]
```

Train U-Net

Load training images and pixel labels into the workspace.

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
labelDir = fullfile(dataSetDir,'trainingLabels');
```

Define the class names and their associated label IDs.

```
classNames = ["triangle","background"];
labelIDs   = [255 0];
```

Create an `imageDatastore` object to store the training images. Create a `pixelLabelDatastore` object to store the ground truth pixel labels for the training images. Then, combine the datastores for training the network.

```
imds = imageDatastore(imageDir);
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
ds = combine(imds,pxds);
```

Create the U-Net network.

```
imageSize = [32 32];
numClasses = 2;
lgraph = unetLayers(imageSize,numClasses)
```

```
lgraph =
  LayerGraph with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Set training options.

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',20, ...
    'VerboseFrequency',10);
```

Train the network.

```
net = trainNetwork(ds,lgraph,options)
```

```
Training on single CPU.
Initializing input data normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:04 |       75.57% |       2.4341 |          0.0010 |
|      10 |          10 |       00:00:36 |       96.02% |       0.4517 |          0.0010 |
|      20 |          20 |       00:01:13 |       97.62% |       0.2324 |          0.0010 |
|========================================================================================|
```

```
net =
  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Input Arguments

`imageSize` — Network input image size

2-element vector | 3-element vector

Network input image size, specified as a:

- 2-element vector in the form [*height*, *width*].
- 3-element vector in the form [*height*, *width*, *depth*]. *depth* is the number of image channels. Set *depth* to 3 for RGB images, to 1 for grayscale images, or to the number of channels for multispectral and hyperspectral images.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numClasses` — Number of classes

integer greater than 1

Number of classes in the semantic segmentation, specified as an integer greater than 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"EncoderDepth",3`

`EncoderDepth` — Encoder depth

4 (default) | positive integer

Encoder depth, specified as a positive integer. U-Net is composed of an encoder subnetwork and a corresponding decoder subnetwork. The depth of these networks determines the number of times the input image is downsampled or upsampled during processing. The encoder network downsamples the input image by a factor of 2^D, where D is the value of `EncoderDepth`. The decoder network upsamples the encoder network output by a factor of 2^D.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumFirstEncoderFilters` — Number of output channels for first encoder

32 (default) | positive integer

Number of output channels for the first encoder stage, specified as a positive integer or vector of positive integers. In each subsequent encoder stage, the number of output channels doubles. The `unetLayers` function sets the number of output channels in each decoder stage to match the number in the corresponding encoder stage.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Convolutional layer filter size
3 (default) | positive odd integer | 2-element row vector of positive odd integers

Convolutional layer filter size, specified as a positive odd integer or a 2-element row vector of positive odd integers. Typical values are in the range [3, 7].

`FilterSize`	Description
scalar	The filter is square.
2-element row vector	The filter has the size [*height width*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConvolutionPadding` — Type of padding
`'same'` (default) | `'valid'`

Type of padding, specified as `'same'` or `'valid'`. The type of padding specifies the padding style for the `convolution2dLayer` in the encoder and the decoder subnetworks. The spatial size of the output feature map depends on the type of padding. If you specify type of padding as:

- `'same'` — Zero padding is applied to the inputs to convolution layers such that the output and input feature maps are the same size.
- `'valid'` — Zero padding is not applied to the inputs to convolution layers. The convolution layer returns only values of the convolution that are computed without zero padding. The output feature map is smaller than the input feature map.

Note To ensure that the *height* and *width* of the inputs to max-pooling layers are even, choose the network input image size to confirm to any one of these criteria:

- If you specify `'ConvolutionPadding'` as `'same'`, then the *height* and *width* of the input image must be a multiple of 2^D.
- If you specify `'ConvolutionPadding'` as `'valid'`, then the *height* and *width* of the input image must be chosen such that $\frac{1}{2^D}\left(height - \sum_{i=1}^{D} 2^i(f_h - 1)\right)$ and $\frac{1}{2^D}\left(width - \sum_{i=1}^{D} 2^i(f_w - 1)\right)$ are multiples of 2^D.

  where f_h and f_w are the height and width of the two-dimensional convolution kernel, respectively. D is the encoder depth.

Data Types: `char` | `string`

Output Arguments

`lgraph` — Layers
`layerGraph` object

Layers that represent the U-Net network architecture, returned as a `layerGraph` object.

`outputSize` — Network output image size
three-element vector

Network output image size, returned as a three-element vector of the form [*height*, *width*, *channels*]. *channels* is the number of output channels and it is equal to the number of classes specified at the input. The *height* and *width* of the output image from the network depend on the type of padding convolution.

- If you specify `'ConvolutionPadding'` as `'same'`, then the *height* and *width* of the network output image are the same as that of the network input image.
- If you specify `'ConvolutionPadding'` as `'valid'`, then the *height* and *width* of the network output image are less than that of the network input image.

Data Types: `double`

More About

U-Net Architecture

- The U-Net architecture consists of an encoder subnetwork and decoder subnetwork that are connected by a bridge section.
- The encoder and decoder subnetworks in the U-Net architecture consists of multiple stages. `EncoderDepth`, which specifies the depth of the encoder and decoder subnetworks, sets the number of stages.
- The stages within the U-Net encoder subnetwork consist of two sets of convolutional and ReLU layers, followed by a 2-by-2 max pooling layer. The decoder subnetwork consists of a transposed convolution layer for upsampling, followed by two sets of convolutional and ReLU layers.
- The bridge section consists of two sets of convolution and ReLU layers.
- The bias term of all convolutional layers is initialized to zero.
- Convolution layer weights in the encoder and decoder subnetworks are initialized using the `'He'` weight initialization method [2].

Tips

- Use `'same'` padding in convolution layers to maintain the same data size from input to output and enable the use of a broad set of input image sizes.
- Use patch-based approaches for seamless segmentation of large images. You can extract image patches by using the `randomPatchExtractionDatastore` function.
- Use `'valid'` padding to prevent border artifacts while you use patch-based approaches for segmentation.
- You can use the network created using `unetLayers` function for GPU code generation after training with `trainNetwork`. For details and examples, see “Generate Code and Deploy Deep Neural Networks” (Deep Learning Toolbox).

Version History

Introduced in R2018b

R2026a: Removed
Errors starting in R2026a

The `unetLayers` function has been removed. Use the `unet` function instead.

R2024b: Warns
Warns starting in R2024b

The `unetLayers` function issues a warning that it will be removed in a future release. Use the `unet` function instead. The `unet` function returns a `dlnetwork` object, which has these advantages over `layerGraph` objects:

- `dlnetwork` objects support a wider range of network architectures which you can then easily train using the `trainnet` function or import from external platforms.
- `dlnetwork` objects provide more flexibility. They have wider support with current and upcoming Deep Learning Toolbox functionality.
- `dlnetwork` objects provide a unified data type that supports network building, prediction, built-in training, compression, and custom training loops.
- `dlnetwork` training and prediction is typically faster than `DAGNetwork` and `SeriesNetwork` training and prediction.

To update your code, replace instances of the `unetLayers` function with the `unet` function. If you want to use a custom or pretrained encoder network, specify the `EncoderNetwork` name-value argument.

Discouraged Usage	Recommended Replacement

This example uses the `unetLayers` function to create a U-Net network, returned as a `LayerGraph` object.
 	
`imageSize = [480 640 3];`
`numClasses = 5;`
`encoderDepth = 3;`
`lgraph = unetLayers(imageSize,numClasses,EncoderDepth=encoderDepth)`
	Here is equivalent code that instead uses the `unet` function to create a U-Net network, which is returned as a `dlnetwork` object.
	
	`imageSize = [480 640 3];`
	`numClasses = 5;`
	`encoderDepth = 3;`
	`unetNetwork = unet(imageSize,numClasses,EncoderDepth=`

R2024a: To be removed
Not recommended starting in R2024a

The `unetLayers` function will be removed in a future release.

R2019b: `NumOutputChannels` argument in `unetLayers` is renamed to `NumFirstEncoderFilters`
Not recommended starting in R2019b

`unetLayers` argument `NumOutputChannels` is renamed to `NumFirstEncoderFilters`. `NumOutputChannels` will not be supported in a future release. Use `NumFirstEncoderFilters` instead. To update your code, replace all instances of `NumOutputChannels` with `NumFirstEncoderFilters`.

References

[1] Ronneberger, O., P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351, 2015, pp. 234–241.

[2] He, K., X. Zhang, S. Ren, and J. Sun. "Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1026–1034.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

You can use the U-Net network for code generation. First, create the network using the `unetLayers` function. Then, use the `trainNetwork` function to train the network for segmentation. After training and evaluating the network, you can generate code for the `DAGNetwork` object by using GPU Coder.

See Also

`unet` | `unet3d` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics

“Semantic Segmentation of Multispectral Images Using Deep Learning”
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

unet3dLayers

(Removed) Create 3-D U-Net layers for semantic segmentation of volumetric images

Note `unet3dLayers` has been removed

Use the `unet3d` function instead. For more information, see “Version History”.

Syntax

```
lgraph = unet3dLayers(inputSize,numClasses)
[lgraph,outputSize] = unet3dLayers(inputSize,numClasses)
[___] = unet3dLayers(inputSize,numClasses,Name,Value)
```

Description

`lgraph = unet3dLayers(inputSize,numClasses)` returns a 3-D U-Net network. `unet3dLayers` includes a pixel classification layer in the network to predict the categorical label for each pixel in an input volumetric image.

Use `unet3dLayers` to create the network architecture for 3-D U-Net. Train the network using the Deep Learning Toolbox function `trainNetwork`.

`[lgraph,outputSize] = unet3dLayers(inputSize,numClasses)` also returns the size of an output volumetric image from the 3-D U-Net network.

`[___] = unet3dLayers(inputSize,numClasses,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in previous syntax.

Examples

Create 3-D U-Net Network with Custom Encoder-Decoder Depth

Create a 3-D U-Net network with an encoder-decoder depth of 2. Specify the number of output channels for the first convolution layer as 16.

```
imageSize = [128 128 128 3];
numClasses = 5;
encoderDepth = 2;
lgraph = unet3dLayers(imageSize,numClasses, ...
    'EncoderDepth',encoderDepth,'NumFirstEncoderFilters',16)
```

```
lgraph =
  LayerGraph with properties:

         Layers: [40×1 nnet.cnn.layer.Layer]
    Connections: [41×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Use the deep learning network analyzer to visualize the 3-D U-Net network.

```
analyzeNetwork(lgraph);
```

The first convolution layers in encoder stages 1 and 2 have 16 and 32 output channels, respectively. The second convolution layers in encoder stages 1 and 2 have 32 and 64 output channels, respectively.

Input Arguments

`inputSize` — Network input image size

three-element vector | four-element vector

Network input image size representing a volumetric image, specified as one of these values:

- Three-element vector of the form [*height width depth*]
- Four-element vector of the form [*height width depth channel*]. *channel* denotes the number of image channels.

Note Network input image size must be chosen such that the dimension of the inputs to the max-pooling layers must be even numbers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numClasses` — Number of classes

scalar greater than 1

Number of classes to segment, specified as a scalar greater than 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `unet3dLayers(inputSize,numClasses,'EncoderDepth',4)`

`EncoderDepth` — Encoder depth

3 (default) | positive integer

Encoder depth, specified as a positive integer. The 3-D U-Net network is composed of an encoder subnetwork and a corresponding decoder subnetwork. The depth of the network determines the number of times the input volumetric image is downsampled or upsampled during processing. The encoder network downsamples the input volumetric image by a factor of 2^D, where D is the value of `EncoderDepth`. The decoder network upsamples the encoder network output by a factor of 2^D. The depth of the decoder subnetwork is same as that of the encoder subnetwork.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumFirstEncoderFilters` — Number of output channels for first convolution layer
32 (default) | positive integer

Number of output channels for the first convolution layer in the first encoder stage, specified as a positive integer. The number of output channels for the second convolution layer and the convolution layers in the subsequent encoder stages is set based on this value.

Given stage = {1, 2, ..., *EncoderDepth*}, the number of output channels for the first convolution layer in each encoder stage is equal to

$2^{stage-1}$ `NumFirstEncoderFilters`

The number of output channels for the second convolution layer in each encoder stage is equal to

2^{stage} `NumFirstEncoderFilters`

The `unet3dLayers` function sets the number of output channels for convolution layers in the decoder stages to match the number in the corresponding encoder stage.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Size of 3-D convolution filter
3 (default) | positive scalar integer | three-element row vector of positive integers

Size of the 3-D convolution filter, specified as a positive scalar integer or a three-element row vector of positive integers of the form $[f_h \ f_w \ f_d]$. Typical values for filter dimensions are in the range [3, 7].

If you specify `FilterSize` as a positive scalar integer of value *a*, then the convolution kernel is of uniform size `[a a a]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConvolutionPadding` — Type of padding
`'same'` (default) | `'valid'`

Type of padding, specified as `'same'` or `'valid'`. The type of padding specifies the padding style for the `convolution3dLayer` in the encoder and the decoder subnetworks. The spatial size of the output feature map depends on the type of padding. Specify one of these options:

- `'same'` — Zero padding is applied to the inputs to convolution layers such that the output and input feature maps are the same size.
- `'valid'` — Zero padding is not applied to the inputs to convolution layers. The convolution layer returns only values of the convolution that are computed without zero padding. The output feature map is smaller than the input feature map.

.

Note To ensure that the *height*, *width*, and *depth* values of the inputs to max-pooling layers are even, choose the network input image size to confirm to any one of these criteria:

- If you specify `'ConvolutionPadding'` as `'same'`, then the *height*, *width*, and *depth* of the input volumetric image must be a multiple of 2^D.

- If you specify `'ConvolutionPadding'` as `'valid'`, then the *height*, *width*, and *depth* of the input volumetric image must be chosen such that $\frac{1}{2^D}\left(height - \sum_{i=1}^{D} 2^i(f_h - 1)\right)$, $\frac{1}{2^D}\left(width - \sum_{i=1}^{D} 2^i(f_w - 1)\right)$, and $\frac{1}{2^D}\left(depth - \sum_{i=1}^{D} 2^i(f_d - 1)\right)$ are multiples of 2^D.

  where f_h, f_w and f_d are the height, width, and depth of the three-dimensional convolution kernel, respectively. D is the encoder depth.

Data Types: `char` | `string`

Output Arguments

`lgraph` — Layers

`layerGraph` object

Layers that represent the 3-D U-Net network architecture, returned as a `layerGraph` object.

`outputSize` — Network output image size

four-element vector

Network output image size, returned as a four-element vector of the form [*height*, *width*, *depth*, *channels*]. *channels* is the number of output channels and is equal to the number of classes specified at the input. The *height*, *width*, and *depth* of the output image from the network depend on the type of padding convolution.

- If you specify `ConvolutionPadding` as `'same'`, then the *height*, *width*, and *depth* of the network output image are the same as that of the network input image.
- If you specify `ConvolutionPadding` as `'valid'`, then the *height*, *width*, and *depth* of the network output image are less than that of the network input image.

Data Types: `double`

More About

3-D U-Net Architecture

- The 3-D U-Net architecture consists of an encoder subnetwork and decoder subnetwork that are connected by a bridge section.
- The encoder and decoder subnetworks in the 3-D U-Net architecture consist of multiple stages. `EncoderDepth`, which specifies the depth of the encoder and decoder subnetworks, sets the number of stages.
- Each encoder stage in the 3-D U-Net network consists of two sets of convolutional, batch normalization, and ReLU layers. The ReLU layer is followed by a 2-by-2-by-2 max pooling layer. Likewise, each decoder stage consists of a transposed convolution layer for upsampling, followed by two sets of convolutional, batch normalization, and ReLU layers.
- The bridge section consists of two sets of convolution, batch normalization, and ReLU layers.
- The bias term of all convolution layers is initialized to zero.

- Convolution layer weights in the encoder and decoder subnetworks are initialized using the `'He'` weight initialization method.

Tips

- Use `'same'` padding in convolution layers to maintain the same data size from input to output and enable the use of a broad set of input image sizes.
- Use patch-based approaches for seamless segmentation of large images. You can extract image patches by using the `randomPatchExtractionDatastore` function.
- Use `'valid'` padding in convolution layers to prevent border artifacts while you use patch-based approaches for segmentation.

Version History

Introduced in R2019b

R2026a: Removed

Errors starting in R2026a

The `unet3dLayers` function has been removed. Use the `unet3d` function instead.

R2024b: Warns

Warns starting in R2024b

The `unet3dLayers` function issues a warning that it will be removed in a future release. Use the `unet3d` function instead. The `unet3d` function returns a `dlnetwork` object, which has these advantages over `layerGraph` objects:

- `dlnetwork` objects support a wider range of network architectures which you can then easily train using the `trainnet` function or import from external platforms.
- `dlnetwork` objects provide more flexibility. They have wider support with current and upcoming Deep Learning Toolbox functionality.
- `dlnetwork` objects provide a unified data type that supports network building, prediction, built-in training, compression, and custom training loops.
- `dlnetwork` training and prediction is typically faster than `DAGNetwork` and `SeriesNetwork` training and prediction.

To update your code, replace instances of the `unet3dLayers` function with the `unet3d` function. If you want to use a custom or pretrained encoder network, specify the `EncoderNetwork` name-value argument.

Discouraged Usage	Recommended Replacement

This example uses the `unet3dLayers` function to create a 3-D U-Net network, returned as a `LayerGraph` object.
 	
`imageSize = [128 128 128 3];`
`numClasses = 5;`
`encoderDepth = 2;`
`lgraph = unet3dLayers(imageSize,numClasses,EncoderDepth=encoderDepth,`
`    NumFirstEncoderFilters=16)`
	Here is equivalent code that instead uses the `u3dnet` function to create a U-Net network, is returned as a `dlnetwork` object.
	
	`imageSize = [128 128 128 3];`
	`numClasses = 5;`
	`encoderDepth = 2;`
	`unet3dNetwork = u3dnet(imageSize,numClasses,EncoderDe`
	`    NumFirstEncoderFilters=16);`

R2024a: To be removed
Not recommended starting in R2024a

The `unet3dLayers` function will be removed in a future release.

References

[1] Çiçek, Ö., A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016. MICCAI 2016. Lecture Notes in Computer Science*. Vol. 9901, pp. 424–432. Springer, Cham.

See Also

`unet` | `unet3d` | `trainnet` | `semanticseg` | `evaluateSemanticSegmentation`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)

countEachLabel

Count occurrence of pixel or box labels

Syntax

```
counts = countEachLabel(ds)
```

Description

`counts = countEachLabel(ds)` returns a table containing information about the pixel or box labels and count for the input datastore, `ds`.

Examples

Count Unique Labels in Datastore

Load a table that contains bounding boxes with labels for vehicles.

```
load('vehicleTrainingData.mat');
```

Load a table that contains bounding boxes with labels for stop signs and cars.

```
load('stopSignsAndCars.mat');
```

Combine ground truth boxes and labels, excluding the image filenames in the first column.

```
vehiclesTbl  = vehicleTrainingData(:,2:end);
stopSignsTbl = stopSignsAndCars(:,2:end);
```

Create a boxLabelDatastore using 2 tables, one with vehicle label data and the other with stop signs label data.

```
blds = boxLabelDatastore(vehiclesTbl,stopSignsTbl);
tbl = countEachLabel(blds)
```

```
tbl=4×3 table
     Label      Count    ImageCount
    ________    _____    __________

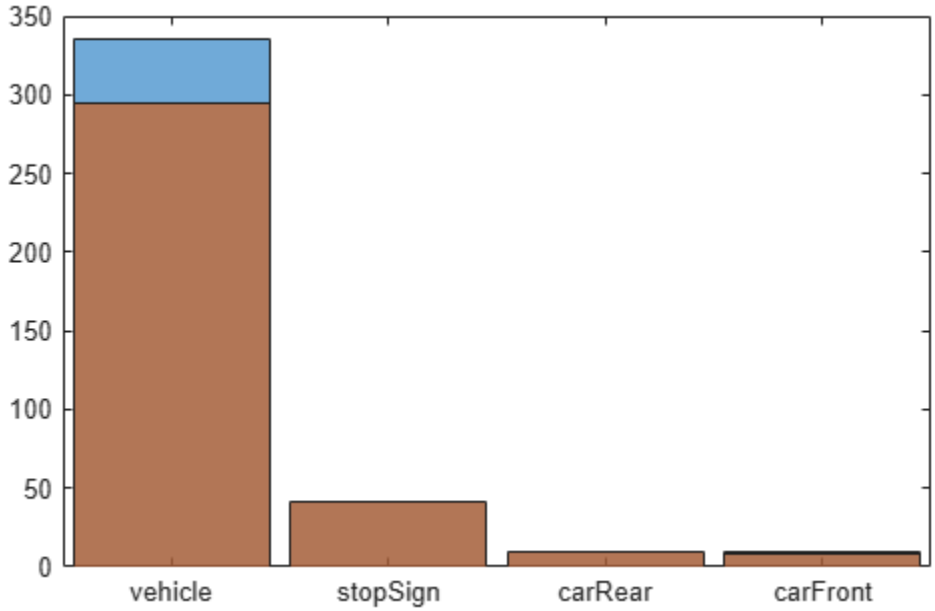
    vehicle      336        295
    stopSign      42         41
    carRear       10          9
    carFront       9          8
```

Create a histogram plot using the labels and the respective label counts.

```
histogram('Categories',tbl.Label,'BinCounts',tbl.Count);
```

Create another histogram overlaying the respective image counts.

```
hold on;
histogram('Categories',tbl.Label,'BinCounts',tbl.ImageCount);
```



Input Arguments

`ds` — Datastore with labeled data

`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

Output Arguments

`counts` — Label information

table

Label information, returned as a table. The labeled data table contain three variables.

For `pixelLabelDatastore` inputs, the `counts` output contains:

Pixel Count Variables	Description
`Name`	Pixel label class name
`PixelCount`	Number of pixels in class
`ImagePixelCount`	Total number of pixels in images that had an instance of the class

For `boxLabelDatastore` inputs, the `counts` output table contains:

Box Count Variables	Description
`Label`	Box label class name
`Count`	Total number of labels of the class across all images
`ImageCount`	Total number of images that contain one or more instances of the class

Tips

The output of `countEachLabel` can be used to calculate class weights for class balancing. For example, for labeled pixel data information in `tbl`:

- Uniform class balancing weights each class such that each contains a uniform prior probability:

  ```
  numClasses = height(tbl)
  prior = 1/numClasses;
  classWeights = prior./tbl.PixelCount
  ```

- Inverse frequency balancing weights each class such that underrepresented classes are given higher weight:

  ```
  totalNumberOfPixels = sum(tbl.PixelCount)
  frequency = tbl.PixelCount / totalNumberOfPixels;
  classWeights = 1./frequency
  ```

- Median frequency balancing weights each class using the median frequency. The weight for each class is defined as median(*imageFreq*)/*imageFreq*(c), where *imageFreq*(c) represents the number of pixels of the class divided by the total number of pixels in images that had an instance of the class (c):

  ```
  imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount
  classWeights = median(imageFreq) ./ imageFreq
  ```

When training the network using `trainnet`, you can pass the calculated class weights to the loss function. For an example, see “Semantic Segmentation Using Deep Learning”.

Version History

Introduced in R2017b

See Also

Functions
`trainnet`

Objects
`pixelLabelDatastore` | `boxLabelDatastore`

reset

Reset label datastore to initial state

Syntax

```
reset(ds)
```

Description

`reset(ds)` resets the label datastore specified by `ds` to the state where no data has been read from it. Resetting the datastore enables you to read from it again.

Input Arguments

`ds` — Datastore with labeled data

`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

Version History

Introduced in R2017b

See Also

`trainnet` | `pixelLabelDatastore` | `boxLabelDatastore`

readimage

Read specified pixel label data file

Syntax

```
C = readimage(pxds,k)
[C,info] = readimage(pxds,k)
```

Description

`C = readimage(pxds,k)` returns the k`th` file in the pixel label datastore specified by `pxds`.

`[C,info] = readimage(pxds,k)` also returns information about the extracted data in `info`, including metadata.

Input Arguments

pxds — Input pixel label datastore

`PixelLabelDatastore` object

Input pixel label datastore, specified as a `PixelLabelDatastore` object.

k — File number

positive integer

File number to read from the pixel label datastore, specified as a positive integer.

Output Arguments

C — Output data

categorical matrix

Output data, returned as a categorical matrix.

info — Information about read data

structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`Filename`	Fully resolved path containing the path string, name of the file, and file extension.
`FileSize`	Total file size, in bytes.

Version History

Introduced in R2017b

See Also

`datastore`

readall

Read all data in label datastore

Syntax

```
data = readall(ds)
```

Description

`data = readall(ds)` returns all the data contained in the label datastore specified by `ds`.

If all the data in the datastore does not fit in memory, then `readall` returns an error.

Input Arguments

ds — Datastore with labeled data

`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

Output Arguments

data — Output data

cell array of categorical matrices | *M*-by-2 cell array | table

Output data, returned as an *M*-by-2 cell array, cell array of categorical matrices, or a table.

Datastore	Output Description
`PixelLabelDatastore`	Table with `MiniBatchSize` number of rows. For the last batch of data in the datastore, `numObservations` must be divisible by `MiniBatchSize` or `read` returns a partial batch containing all the remaining observations in the datastore.
`boxLabelDatastore`	*N*-by-2 cell matrix. *N* must be less than or equal to `ReadSize(ds)`. The first column must be a cell vector that contains bounding boxes. Each element in the cell vector contains *M*-by-4 matrices in the format [*x*,*y*,*width*,*height*]. The second column must be a cell vector that contains the label names corresponding to each bounding box. Label names are represented as an *M*-by-1 categorical vector.

Version History

Introduced in R2017b

See Also

`trainnet` | `boxLabelDatastore` | `pixelLabelDatastore`

partition

Partition label datastore

Syntax

```
subds = partition(ds,N,index)

subds = partition(ds,'Files',fileIndex)
subds = partition(ds,'Files',filename)
```

Description

`subds = partition(ds,N,index)` partitions a datastore `ds` into *N* parts and returns the partition corresponding to `index`.

`subds = partition(ds,'Files',fileIndex)` partitions the `PixelLabelDatastore` by files. The partitioned datastore corresponds to the `index` in the `Files` property of the datastore.

`subds = partition(ds,'Files',filename)` partitions the `PixelLabelDatastore` by files. The partitioned datastore corresponds to the file specified by `filename`.

Examples

Partition a Datastore

Load training data that contains bounding boxes with labels for vehicles.

```
data = load('vehicleTrainingData.mat');
trainingData = data.vehicleTrainingData;
blds = boxLabelDatastore(trainingData(:,2:end));
```

Use the `partition` function to partition the box label datastore. The output contains the first 5 rows of the training data.

```
subds = partition(blds,59,5);
```

Read the data. Use the `hasdata` function to check for data.

```
while hasdata(subds)
      % Read one row of box labels at a time
      bxLabels = read(subds);
end
```

Input Arguments

ds — Datastore with labeled data

`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

`N` — Number of partitions

positive integer

Number of partitions to divide datastore, specified as a positive integer. To obtain an estimate for a reasonable number of partitions, use the `numpartitions` function.

```
numWorkers = 3;
p = parpool('local',numWorkers);
n = numpartitions(ds,p);

parfor ii=1:n
    subds = partition(ds,n,ii);
    while hasdata(subds)
        data = read(subds);
    end
end
```

Example: 3

Data Types: `double`

`index` — Index

positive integer

Index, specified as an integer. The index points to a partitioned datastore of the `N` number of partitioned datastores.

Example: 1

Data Types: `double`

`fileIndex` — File index

positive integer

File index of a file stored within the `Files` property of a `pixelLabelDatastore`, specified as a positive integer.

Example: 1

Data Types: `double`

`filename` — Filename

character vector

Filename, specified as a character vector.

Example: `'file1.csv'`

Example: `'../dir/data/file1.csv'`

Example: `'hdfs://myserver:7867/data/file1.txt'`

Data Types: `char`

Output Arguments

subds — Output datastore
`datastore` object

Output datastore, returned as a datastore object. The output datastore is of the same type as the input datastore, `ds`.

Version History

Introduced in R2017b

See Also

`trainnet` | `boxLabelDatastore` | `pixelLabelDatastore`

numpartitions

Number of partitions for label datastore

Syntax

```
N = numpartitions(ds)
N = numpartitions(ds,pool)
```

Description

`N = numpartitions(ds)` returns the default number of partitions for a label datastore `ds`.

`N = numpartitions(ds,pool)` returns the number of partitions needed to parallelize datastore access over the parallel pool specified by `pool`. To parallelize datastore access, you must have Parallel Computing Toolbox installed.

Input Arguments

`ds` — Datastore with labeled data
`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

`pool` — Parallel pool
parallel pool object

Parallel pool object.

Example: `gcp`

Version History

Introduced in R2017b

See Also

Objects
`boxLabelDatastore` | `pixelLabelDatastore`

hasdata

Determine if data is available to read from label datastore

Syntax

```
tf = hasdata(ds)
```

Description

`tf = hasdata(ds)` returns logical `1` (`true`) if there is data available to read from the datastore specified by `ds`. Otherwise, it returns logical `0` (`false`).

Examples

Determine Pixel Data Available to Read

Check if pixel label data can be read from a datastore.

Set the location of the image and pixel label data.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata');
imDir = fullfile(dataDir,'building');
pxDir = fullfile(dataDir,'buildingPixelLabels');
```

Create an image and pixel label datastore.

```
imds = imageDatastore(imDir);
classNames = ["sky" "grass" "building" "sidewalk"];
pixelLabelID = [1 2 3 4];
pxds = pixelLabelDatastore(pxDir,classNames,pixelLabelID);
```

While data is available in the datastore, read the data.

```
while hasdata(pxds)
    T = read(pxds);
end
```

Input Arguments

ds — Datastore with labeled data

`PixelLabelDatastore` object | `boxLabelDatastore` object

Datastore with labeled data for training a semantic segmentation network or an object detection network, specified as a `pixelLabelDatastore` or `boxLabelDatastore` object.

Output Arguments

tf — Datastore data status

`1` | `0`

Datastore data status, returned as a logical `1` (`true`) if there is data available to read from the datastore specified by `ds`. Otherwise, it returns logical `0` (`false`).

Version History

Introduced in R2017b

See Also

`trainnet` | `pixelLabelDatastore` | `boxLabelDatastore`

semanticseg

Semantic image segmentation using deep learning

Syntax

```
C = semanticseg(I,network)
[C,score,allScores] = semanticseg(I,network)
[___] = semanticseg(I,network,roi)

pxds = semanticseg(ds,network)

[___] = semanticseg(___,Name=Value)
```

Description

`C = semanticseg(I,network)` returns a semantic segmentation of the input image using deep learning.

`[C,score,allScores] = semanticseg(I,network)` also returns the classification scores for each categorical label in `C`. The function returns the scores in an array that corresponds to each pixel or voxel in the input image.

`[___] = semanticseg(I,network,roi)` returns a semantic segmentation for a rectangular subregion of the input image.

`pxds = semanticseg(ds,network)` returns the semantic segmentation for a collection of images in `ds`, a datastore object.

The function supports parallel computing using multiple MATLAB workers. You can enable parallel computing using the “Computer Vision Toolbox Preferences” dialog.

`[___] = semanticseg(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ExecutionEnvironment="gpu"` sets the hardware resource for processing images to `gpu`.

Examples

Semantic Image Segmentation

Perform semantic segmentation of a test image and display the results.

Load a pretrained network.

```
load("triangleSegmentationNetwork")
```

List the network layers.

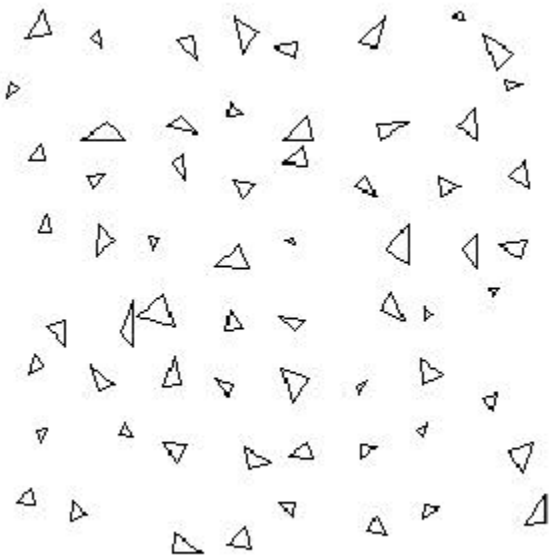
```
net.Layers
```

```
ans =
  9×1 Layer array with layers:
```

```
     1   'imageinput'        Image Input                  32×32×1 images with 'zerocenter' norma
     2   'conv_1'            2-D Convolution              64 3×3×1 convolutions with stride [1  
     3   'relu_1'            ReLU                         ReLU
     4   'maxpool'           2-D Max Pooling              2×2 max pooling with stride [2  2] and
     5   'conv_2'            2-D Convolution              64 3×3×64 convolutions with stride [1
     6   'relu_2'            ReLU                         ReLU
     7   'transposed-conv'   2-D Transposed Convolution   64 4×4×64 transposed convolutions with
     8   'conv_3'            2-D Convolution              2 1×1×64 convolutions with stride [1  
     9   'softmax'           Softmax                      Softmax
```

Read and display the test image.

```
I = imread("triangleTest.jpg");
imshow(I)
```

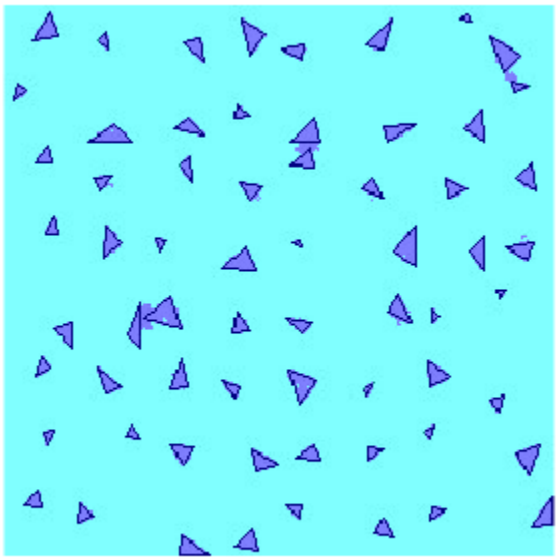


Define the two classes on which the network was trained, then perform semantic image segmentation.

```
classNames = ["triangle" "background"];
[C,scores] = semanticseg(I,net,Classes=classNames,MiniBatchSize=32);
```

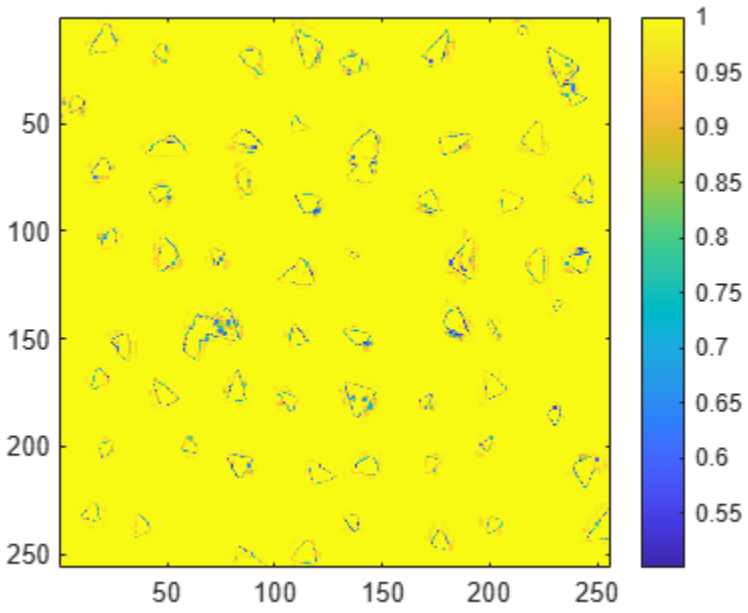
Overlay segmentation results on the image and display the results.

```
B = labeloverlay(I,C);
imshow(B)
```



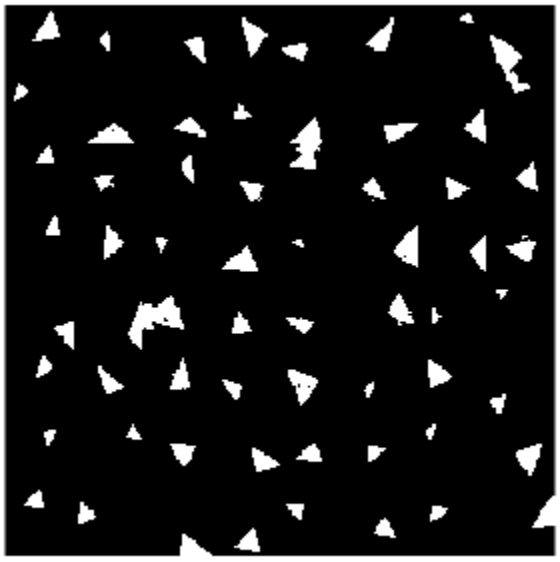
Display the classification confidence scores.

```
imagesc(scores)
axis square
colorbar
```



Create a binary mask with only the triangles.

```
BW = C=="triangle";
imshow(BW)
```



Evaluate Semantic Segmentation Test Set

Run semantic segmentation on a test set of images and compare the results against ground truth data.

Load a pretrained network.

```
data = load("triangleSegmentationNetwork");
net = data.net;
```

Load test images using `imageDatastore`.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
testImageDir = fullfile(dataDir,"testImages");
imds = imageDatastore(testImageDir)
```

```
imds =
  ImageDatastore with properties:

                      Files: {
                             ' ...\toolbox\vision\visiondata\triangleImages\testImages\image_001
                             ' ...\toolbox\vision\visiondata\triangleImages\testImages\image_002
                             ' ...\toolbox\vision\visiondata\triangleImages\testImages\image_003
                              ... and 97 more
                             }
                    Folders: {
                             'B:\runnable\matlab\toolbox\vision\visiondata\triangleImages\testIm
                             }
    AlternateFileSystemRoots: {}
                   ReadSize: 1
```

```
                  Labels: {}
      SupportedOutputFormats: ["png"    "jpg"    "jpeg"    "tif"    "tiff"]
         DefaultOutputFormat: "png"
                    ReadFcn: @readDatastoreImage
```

Load ground truth test labels.

```
testLabelDir = fullfile(dataDir,"testLabels");
classNames = ["triangle" "background"];
pixelLabelID = [255 0];
pxdsTruth = pixelLabelDatastore(testLabelDir,classNames,pixelLabelID);
```

Run semantic segmentation on all of the test images with a batch size of 4. You can increase the batch size to increase throughput based on your systems memory resources.

```
pxdsResults = semanticseg(imds,net,Classes=classNames, ...
    MiniBatchSize=4,WriteLocation=tempdir);
```

```
Running semantic segmentation network
-------------------------------------
* Processed 100 images.
```

Compare the results against the ground truth.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth)
```

```
Evaluating semantic segmentation results
----------------------------------------
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 100 images.
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

       0.99074          0.99183       0.91118      0.98299        0.80563

metrics =
  semanticSegmentationMetrics with properties:

              ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
               DataSetMetrics: [1×5 table]
                 ClassMetrics: [2×3 table]
                 ImageMetrics: [100×5 table]
```

Semantic Segmentation Using Dilated Convolutions

This example shows how to train a semantic segmentation network using dilated convolutions.

A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for

autonomous driving and cancer cell segmentation for medical diagnosis. To learn more, see “Get Started with Semantic Segmentation Using Deep Learning”.

Semantic segmentation networks like Deeplab v3+ [1 on page 3-417] make extensive use of dilated convolutions (also known as atrous convolutions) because they can increase the receptive field of the layer (the area of the input which the layers can see) without increasing the number of parameters or computations.

Load Training Data

The example uses a simple dataset of 32-by-32 triangle images for illustration purposes. The dataset includes accompanying pixel label ground truth data. Load the training data using an `imageDatastore` and a `pixelLabelDatastore`.

```
dataFolder = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
imageFolderTrain = fullfile(dataFolder,"trainingImages");
labelFolderTrain = fullfile(dataFolder,"trainingLabels");
```

Create an `imageDatastore` for the images.

```
imdsTrain = imageDatastore(imageFolderTrain);
```

Create a `pixelLabelDatastore` for the ground truth pixel labels.

```
classNames = ["triangle" "background"];
labels = [255 0];
pxdsTrain = pixelLabelDatastore(labelFolderTrain,classNames,labels)
```

```
pxdsTrain =
  PixelLabelDatastore with properties:

                      Files: {200×1 cell}
                 ClassNames: {2×1 cell}
                   ReadSize: 1
                    ReadFcn: @readDatastoreImage
    AlternateFileSystemRoots: {}
```

Create Semantic Segmentation Network

This example uses a simple semantic segmentation network based on dilated convolutions.

Create a data source for training data and get the pixel counts for each label.

```
ds = combine(imdsTrain,pxdsTrain);
tbl = countEachLabel(pxdsTrain)
```

```
tbl=2×3 table
      'triangle'     10326    204800
    'background'    194474    204800
```

The majority of pixel labels are for background. This class imbalance biases the learning process in favor of the dominant class. To fix this, use class weighting to balance the classes. You can use several methods to compute class weights. One common method is inverse frequency weighting where the class weights are the inverse of the class frequencies. This method increases the weight given to under represented classes. Calculate the class weights using inverse frequency weighting.

```
numberPixels = sum(tbl.PixelCount);
frequency = tbl.PixelCount / numberPixels;
classWeights = dlarray(1 ./ frequency,"C");
```

Create a network for pixel classification by using an image input layer with an input size corresponding to the size of the input images. Next, specify three blocks of convolution, batch normalization, and ReLU layers. For each convolutional layer, specify 32 3-by-3 filters with increasing dilation factors and pad the inputs so they are the same size as the outputs by setting the `Padding` name-value argument as `"same"`. To classify the pixels, include a convolutional layer with K 1-by-1 convolutions, where K is the number of classes, followed by a softmax layer. The classification of pixels is done with a custom model loss within the built-in trainer, `trainnet`.

```
inputSize = [32 32 1];
filterSize = 3;
numFilters = 32;
numClasses = numel(classNames);

layers = [
    imageInputLayer(inputSize)

    convolution2dLayer(filterSize,numFilters,DilationFactor=1,Padding="same")
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filterSize,numFilters,DilationFactor=2,Padding="same")
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filterSize,numFilters,DilationFactor=4,Padding="same")
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(1,numClasses)
    softmaxLayer];
```

Model Loss Function

The semantic segmentation network can be trained using different loss functions. The built-in trainer `trainnet` (Deep Learning Toolbox) supports custom loss functions as well as some standard loss functions such as "crossentropy" and "mse". A custom loss function manually computes the loss for each batch of training data by comparing the network's predictions to the actual ground truth or target values. Custom loss functions use a function handle with the function syntax *loss = f(Y1,...,Yn,T1,...,Tm)*, where Y1,...,Yn are `dlarray` objects that correspond to the n network predictions and *T1*,...,*Tm* are `dlarray` objects that correspond to the *m* targets.

This example enables you to select from two different loss functions that account for the class imbalance seen in the data. These loss functions are:

1. Weighted cross-entropy loss, which uses the `crossentropy` (Deep Learning Toolbox) function. Weighted cross-entropy loss gives stronger favor to the underrepresented class by scaling the error of that class during training.
2. A custom loss function called `tverskyLoss` that calculates the Tversky loss [2 on page 3-417]. Tversky loss is more specialized loss for class imbalance.

The Tversky loss is based on the Tversky index for measuring overlap between two segmented images. The Tversky index TI_c between one image Y and the corresponding ground truth T is given by

$$\mathrm{TI}_c = \frac{\sum_{m=1}^{M} Y_{\mathrm{cm}} T_{\mathrm{cm}}}{\sum_{m=1}^{M} Y_{\mathrm{cm}} T_{\mathrm{cm}} + \alpha \sum_{m=1}^{M} Y_{\mathrm{cm}} T_{\bar{c}m} + \beta \sum_{m=1}^{M} Y_{\bar{c}m} T_{\mathrm{cm}}}$$

- c corresponds to the class and $\bar{c}$ corresponds to not being in class c.
- M is the number of elements along the first two dimensions of Y.
- α and β are weighting factors that control the contribution that false positives and false negatives for each class make to the loss.

The loss L over the number of classes C is given by

$$L = \sum_{c=1}^{C} 1 - \mathrm{TI}_c$$

Select the loss function to use during training.

```
lossFunction = Tversky Loss
```

```
lossFunction =
"tverskyLoss"
```

```
if strcmp(lossFunction,"tverskyLoss")
    % Declare Tversky loss weighting coefficients for false positives and
    % false negatives. These coefficients are set and passed to the
    % training loss function using trainnet.
    alpha = 0.7;
    beta = 0.3;
    lossFcn = @(Y,T) tverskyLoss(Y,T,alpha,beta);
else
    % Use weighted cross-entropy loss during training.
    lossFcn = @(Y,T) crossentropy(Y,T,classWeights,NormalizationFactor="all-elements");
end
```

Train Network

Specify the training options.

```
options = trainingOptions("sgdm",...
    MaxEpochs=100,...
    MiniBatchSize= 64,...
    InitialLearnRate=1e-2,...
    Verbose=false);
```

Train the network using `trainnet` (Deep Learning Toolbox). Specify the loss as the loss function `lossFcn`.

```
net = trainnet(ds,layers,lossFcn,options);
```

Test Network

Load the test data. Create an `imageDatastore` for the images. Create a `pixelLabelDatastore` for the ground truth pixel labels.

```
imageFolderTest = fullfile(dataFolder,"testImages");
imdsTest = imageDatastore(imageFolderTest);
labelFolderTest = fullfile(dataFolder,"testLabels");
pxdsTest = pixelLabelDatastore(labelFolderTest,classNames,labels);
```

Make predictions using the test data and trained network.

```
pxdsPred = semanticseg(imdsTest,net,...
    Classes=classNames,...
    MiniBatchSize=32,...
    WriteLocation=tempdir);
```

```
Running semantic segmentation network
-------------------------------------
* Processed 100 images.
```

Evaluate the prediction accuracy using `evaluateSemanticSegmentation`.

```
metrics = evaluateSemanticSegmentation(pxdsPred,pxdsTest);
```

```
Evaluating semantic segmentation results
----------------------------------------
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 100 images.
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

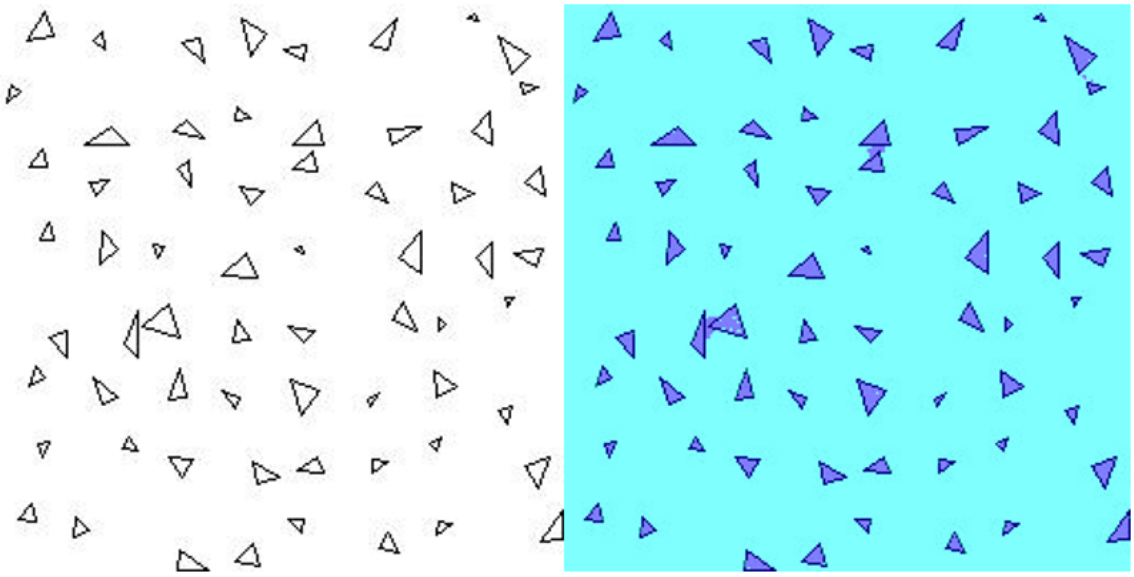
       0.99674           0.98562       0.96447      0.99362        0.92831
```

Segment New Image

Read the test image `triangleTest.jpg` and segment the test image using `semanticseg`. Display the results using `labeloverlay`.

```
imgTest = imread("triangleTest.jpg");
[C,scores] = semanticseg(imgTest,net,classes=classNames);

B = labeloverlay(imgTest,C);
montage({imgTest,B})
```



Supporting Functions

```
function loss = tverskyLoss(Y,T,alpha,beta)
    % loss = tverskyLoss(Y,T,alpha,beta) returns the Tversky loss
    % between the predictions Y and the training targets T.

    Pcnot = 1-Y;
    Gcnot = 1-T;
    TP = sum(sum(Y.*T,1),2);
    FP = sum(sum(Y.*Gcnot,1),2);
    FN = sum(sum(Pcnot.*T,1),2);

    epsilon = 1e-8;
    numer = TP + epsilon;
    denom = TP + alpha*FP + beta*FN + epsilon;

    % Compute tversky index.
    lossTIc = 1 - numer./denom;
    lossTI = sum(lossTIc,3);

    % Return average Tversky index loss.
    N = size(Y,4);
    loss = sum(lossTI)/N;
end
```

References

[1] Chen, Liang-Chieh et al. “Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation.” ECCV (2018).

[2] Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus, and Ali Gholipour. "Tversky loss function for image segmentation using 3D fully convolutional deep networks." *International Workshop on Machine Learning in Medical Imaging*. Springer, Cham, 2017.

Input Arguments

`I` — Input image

numeric array

Input image, specified as one of the following.

Image Type	Data Format
Single 2-D grayscale image	2-D matrix of size *H*-by-*W*
Single 2-D color image or 2-D multispectral image	3-D array of size *H*-by-*W*-by-*C*. The number of color channels *C* is 3 for color images.
Series of *P* 2-D images	4-D array of size *H*-by-*W*-by-*C*-by-*P*. The number of color channels *C* is 1 for grayscale images and 3 for color images.
Single 3-D grayscale image with depth *D*	3-D array of size *H*-by-*W*-by-*D*
Single 3-D color image or 3-D multispectral image	4-D array of size *H*-by-*W*-by-*D*-by-*C*. The number of color channels *C* is 3 for color images.
Series of *P* 3-D images	5-D array of size *H*-by-*W*-by-*D*-by-*C*-by-*P*

The input image can also be a `gpuArray` containing one of the preceding image types (requires Parallel Computing Toolbox).

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`network` — Network

`dlnetwork` object | `taylorPrunableNetwork` object

Network, specified as a `dlnetwork` or `taylorPrunableNetwork` object.

`roi` — Region of interest

4-element numeric vector | 6-element vector

Region of interest, specified as one of the following.

Image Type	ROI Format
2-D image	4-element vector of the form [*x*,*y*,*width*,*height*]
3-D image	6-element vector of the form [*x*,*y*,*z*,*width*,*height*,*depth*]

The vector defines a rectangular or cuboidal region of interest fully contained in the input image. Image pixels outside the region of interest are assigned the `<undefined>` categorical label. If the input image consists of a series of images, then `semanticseg` applies the same `roi` to all images in the series.

Note When using the `roi` argument, the semantic segmentation network must produce an output size that matches the size of the specified ROI. If your network cannot produce an output of this size,

manually crop the input image to the desired ROI using the `imcrop` function, and then process the cropped image with `semanticseg`.

`ds` — Collection of images
datastore object

Collection of images, specified as a datastore. The `read` function of the datastore must return a numeric array, cell array, or table. For cell arrays or tables with multiple columns, the function processes only the first column.

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ExecutionEnvironment="gpu"` sets the hardware resource for processing images to `"gpu"`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`OutputType` — Returned segmentation type
`"categorical"` (default) | `"double"` | `"uint8"`

Returned segmentation type, specified as `"categorical"`, `"double"`, or `"uint8"`. When you specify `"double"` or `"uint8"`, the function returns the segmentation results as a label array containing label IDs. The IDs are integer values that correspond to the class names defined in the classification layer used in the input network.

You cannot use the `OutputType` property with an `ImageDatastore` object input.

`MiniBatchSize` — Group of images
128 (default) | integer

Group of images, specified as an integer. Images are grouped and processed together as a batch. Batches are used for processing a large collection of images and they improve computational efficiency. Increasing the '`MiniBatchSize`' value increases the efficiency, but it also takes up more memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource for processing images with a network, specified as `"auto"`, `"gpu"`, or `"cpu"`.

`ExecutionEnvironment`	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

ExecutionEnvironment	Description
"gpu"	Use the GPU. If a suitable GPU is not available, the function returns an error message.
"cpu"	Use the CPU.

Acceleration — Performance optimization
"auto" (default) | "mex" | "none"

Performance optimization, specified as "auto", "mex", or "none".

Acceleration	Description
"auto"	Automatically apply a number of optimizations suitable for the input network and hardware resource.
"mex"	Compile and execute a MEX function. This option is available when using a GPU only. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).
"none"	Disable all acceleration.

The default option is "auto". If you use the "auto" option, then MATLAB does not ever generate a MEX function.

Using the Acceleration name-value argument options "auto" and "mex" can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The "mex" option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The "mex" option is only available when you are using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox). If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error.

"mex" acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Classes — Classes into which pixels or voxels are classified
"auto" (default) | cell array of character vectors | string vector | categorical vector

Classes into which pixels or voxels are classified, specified as "auto", a cell array of character vectors, a string vector, or a categorical vector. If the value is a categorical vector *Y*, then the elements of the vector are sorted and ordered according to categories(Y).

If the network is a dlnetwork object, then the number of classes specified by 'Classes' must match the number of channels in the output of the network predictions. By default, when 'Classes' has the value "auto", the classes are numbered from 1 through *C*, where *C* is the number of channels in the output layer of the network.

If the network is a SeriesNetwork or DAGNetwork object, then the number of classes specified by the Classes name-value argument must match the number of classes in the classification output

layer. By default, when `Classes` has the value `"auto"`, the classes are automatically set using the classification output layer.

`WriteLocation` — Folder location
`pwd` (current working folder) (default) | string scalar | character vector

Folder location, specified as `pwd` (your current working folder), a string scalar, or a character vector. The specified folder must exist and have write permissions.

This property applies only when using a datastore that can process images.

`NamePrefix` — Prefix applied to output filenames
`"pixelLabel"` (default) | string scalar | character vector

Prefix applied to output filenames, specified as a string scalar or character vector. The image files are named as follows:

- *<prefix>*`_`*<N>*`.png`, where *N* corresponds to the index of the input image file, `imds.Files`(*N*).

This property applies only when using a datastore that can process images.

`OutputFolderName` — Output folder name
`"semanticsegOutput"` (default) | string scalar | character vector

Output folder name for segmentation results, specified as a string scalar or a character vector. This folder is in the location specified by the value of the `WriteLocation` name-value argument.

If the output folder already exists, the function creates a new folder with the string `"_1"` appended to the end of the name. Set `OutputFoldername` to `""` to write all the results to the folder specified by `WriteLocation`.

`NameSuffix` — Suffix to add to the output image filename
string scalar | character vector

Suffix to add to the output image filename, specified as a string scalar or a character vector. The function appends the specified suffix to the output filename as:

- *<prefix>*`_`*<N>*`<`*suffix*`>.png`, where *N* corresponds to the index of the input image file, `imds.Files`(*N*).

If you do not specify the suffix, the function uses the input filenames as the output file suffixes. The function extracts the input filenames from the `info` output of the `read` object function of the datastore. When the datastore does not provide the filename, the function does not add a suffix.

`Verbose` — Display progress information
`false` or `0` (default) | `true` or `1`

Display progress information, specified as a logical `0` (`false`) or `1` (`true`). Specify `Verbose` as `true` to display progress information. This property applies only when using a datastore that can process images.

Output Arguments

`C` — Categorical labels
categorical array

Categorical labels, returned as a categorical array. The categorical array relates a label to each pixel or voxel in the input image. The images returned by `readall(datastore)` have a one-to-one correspondence with the categorical matrices returned by `readall(pixelLabelDatastore)`. The elements of the label array correspond to the pixel or voxel elements of the input image. If you select an ROI, then the labels are limited to the area within the ROI. Image pixels and voxels outside the region of interest are assigned the `<undefined>` categorical label.

Image Type	Categorical Label Format
Single 2-D image	2-D matrix of size *H*-by-*W*. Element `C`(*i*,*j*) is the categorical label assigned to the pixel `I`(*i*,*j*).
Series of *P* 2-D images	3-D array of size *H*-by-*W*-by-*P*. Element `C`(*i*,*j*,*p*) is the categorical label assigned to the pixel `I`(*i*,*j*,*p*).
Single 3-D image	3-D array of size *H*-by-*W*-by-*D*. Element `C`(*i*,*j*,*k*) is the categorical label assigned to the voxel `I`(*i*,*j*,*k*).
Series of *P* 3-D images	4-D array of size *H*-by-*W*-by-*D*-by-*P*. Element `C`(*i*,*j*,*k*,*p*) is the categorical label assigned to the voxel `I`(*i*,*j*,*k*,*p*).

`score` — Confidence scores

numeric array

Confidence scores for each categorical label in `C`, returned as an array of values between `0` and `1`. The scores represents the confidence in the predicted labels `C`. Higher score values indicate a higher confidence in the predicted label.

Image Type	Score Format
Single 2-D image	2-D matrix of size *H*-by-*W*. Element `score`(*i*,*j*) is the classification score of the pixel `I`(*i*,*j*).
Series of *P* 2-D images	3-D array of size *H*-by-*W*-by-*P*. Element `score`(*i*,*j*,*p*) is the classification score of the pixel `I`(*i*,*j*,*p*).
Single 3-D image	3-D array of size *H*-by-*W*-by-*D*. Element `score`(*i*,*j*,*k*) is the classification score of the voxel `I`(*i*,*j*,*k*).
Series of *P* 3-D images	4-D array of size *H*-by-*W*-by-*D*-by-*P*. Element `score`(*i*,*j*,*k*,*p*) is the classification score of the voxel `I`(*i*,*j*,*k*,*p*).

`allScores` — Scores for all label categories

numeric array

Scores for all label categories that the input network can classify, returned as a numeric array. The format of the array is described in the following table. *L* represents the total number of label categories.

Image Type	All Scores Format
Single 2-D image	3-D array of size *H*-by-*W*-by-*L*. Element `allScores`(*i*,*j*,*q*) is the score of the *q*th label at the pixel `I`(*i*,*j*).
Series of *P* 2-D images	4-D array of size *H*-by-*W*-by-*L*-by-*P*. Element `allScores`(*i*,*j*,*q*,*p*) is the score of the *q*th label at the pixel `I`(*i*,*j*,*p*).
Single 3-D image	4-D array of size *H*-by-*W*-by-*D*-by-*L*. Element `allScores`(*i*,*j*,*k*,*q*) is the score of the *q*th label at the voxel `I`(*i*,*j*,*k*).

Image Type	All Scores Format
Series of *P* 3-D images	5-D array of size *H*-by-*W*-by-*D*-by-*L*-by-*P*. Element `allScores`(*i*,*j*,*k*,*q*,*p*) is the score of the *q*th label at the voxel `I`(*i*,*j*,*k*,*p*).

`pxds` — Semantic segmentation results
`PixelLabelDatastore` object

Semantic segmentation results, returned as a `pixelLabelDatastore` object. The object contains the semantic segmentation results for all the images contained in the `ds` input object. The result for each image is saved as separate `uint8` label matrices of PNG images. You can use `read`(`pxds`) to return the categorical labels assigned to the images in `ds`.

The images in the output of `readall`(`ds`) have a one-to-one correspondence with the categorical matrices in the output of `readall`(`pxds`).

More About

Network Output Size and ROI Input

The output size of a semantic segmentation network is determined by the architecture of its layers, particularly the use of padding in convolutional and pooling layers. Due to these factors, the output size may not always match the input size. In some cases, certain input sizes will produce matching output sizes, while others will not. For example, a network might produce an output of size 100-by-100 pixels for an input of 100-by-100 pixels, but produce an output of 94-by-94 pixels for an input of 95-by-95 pixels.

When specifying a region of interest (ROI) using the `roi` argument, ensure that the network can produce an output that exactly matches the ROI size. If this is not possible, crop the input image to the ROI, and then apply `semanticseg` to the cropped image.

Version History

Introduced in R2017b

R2024a: `DAGNetwork` and `SeriesNetwork` objects are not recommended

Not recommended starting in R2024a

Starting in R2024a, `DAGNetwork` and `SeriesNetwork` objects are not recommended. Instead, specify the semantic segmentation network as a `dlnetwork` object.

There are no plans to remove support for `DAGNetwork` and `SeriesNetwork` objects. However, `dlnetwork` objects have these advantages:

- `dlnetwork` objects support a wider range of network architectures which you can then easily train using the `trainnet` function or import from external platforms.
- `dlnetwork` objects provide more flexibility. They have wider support with current and upcoming Deep Learning Toolbox functionality.
- `dlnetwork` objects provide a unified data type that supports network building, prediction, built-in training, compression, and custom training loops.
- `dlnetwork` training and prediction is typically faster than `DAGNetwork` and `SeriesNetwork` training and prediction.

R2023a: Output file location
Behavior changed in R2023a

The `semanticseg` function now writes output files to the folder specified by the `WriteLocation` and `OutputFolderName` name-value arguments as `<WriteLocation>/<OutputFolderName>`. Prior to R2023a, the function wrote output files directly into the location specified by `WriteLocation`. To get the same results as in previous releases, set the `OutputFolderName` name-value argument to `""`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `semanticSeg` object for code generation, use `coder.loadDeepLearningNetwork`.
- For code generation, the `semanticSeg` object does not support `imageDatastore` and 3-D images as input.
- For code generation, `ExcecutionEnvironment` and `Acceleration` name value pairs are ignored with compile time warnings.
- For code generation, only `OutputType`, `MiniBatchSize`, and `Classes` name value arguments are supported. These name value arguments must be compile time constants.
- For code generation, `Classes` name value argument does not support string array as input.
- For code generation, `pxds` output argument is not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `semanticSeg` object for code generation, use `coder.loadDeepLearningNetwork`.
- For code generation, the `semanticSeg` object does not support `imageDatastore` and 3-D images as input.
- For code generation, `ExcecutionEnvironment` and `Acceleration` name value pairs are ignored with compile time warnings.
- For code generation, only `OutputType`, `MiniBatchSize`, and `Classes` name value arguments are supported. These name value arguments must be compile time constants.
- For code generation, `Classes` name value argument does not support string array as input.
- For code generation, `pxds` output argument is not supported.

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `semanticseg` function uses a GPU if one is available. You can specify the hardware that the `semanticseg` function uses by setting the `ExecutionEnvironment` argument.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainnet` | `labeloverlay` | `evaluateSemanticSegmentation`

Objects
`ImageDatastore` | `pixelLabelDatastore` | `dlnetwork`

Topics
“Get Started with Semantic Segmentation Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Datastores for Deep Learning” (Deep Learning Toolbox)

External Websites
What is Semantic Segmentation?

detectKAZEFeatures

Detect KAZE features

Syntax

```
points = detectKAZEFeatures(I)
points = detectKAZEFeatures(I,Name=Value)
```

Description

`points = detectKAZEFeatures(I)` returns a `KAZEPoints` object containing information about KAZE keypoints detected in a 2-D grayscale or binary image. The function uses nonlinear diffusion to construct a scale space for the given image. It then detects multiscale corner features from the scale space.

`points = detectKAZEFeatures(I,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `detectKAZEFeatures(I,Threshold=0.003)` sets the threshold to exclude less significant local extrema to `0.003`.

Examples

Detect KAZE Point Features in Image

Read an image.

```
I = imread("cameraman.tif");
```

Detect KAZE points in the image.

```
points = detectKAZEFeatures(I);
```

Plot the 20 strongest points.

```
imshow(I)
hold on
plot(selectStrongest(points,20))
hold off
```



Detect KAZE Features and Display Specific Points

Read an image into the workspace.

```
I = imread("cameraman.tif");
```

Detect KAZE features in the image.

```
points = detectKAZEFeatures(I);
```

Select and display the last 5 points detected.

```
imshow(I);
hold on;
plot(points(end-4:end));
hold off;
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detectKAZEFeatures(I,Threshold=0.003)` sets the threshold to exclude less significant local extrema to `0.003`.

`Diffusion` — Method to compute conductivity

`"region"` (default) | `"sharpedge"` | `"edge"`

Method to compute conductivity, specified as `"region"`, `"sharpedge"`, or `"edge"`. The computation is based on first-order derivatives of a layer in scale space.

Method	Selected Features
`"region"`	Large regions. Uses the Perona and Malik conductivity coefficient, `1/(1 + dL^2/k^2)`.

Method	Selected Features
`"sharpedge"`	High-contrast edges. Uses the Perona and Malik conductivity coefficient, `exp(-|dL|^2/k^2)`.
`"edge"`	Smoothing on both sides of an edge rather than across it. Uses the Weickert conductivity coefficient.

`Threshold` — Local extrema
`0.0001` (default) | scalar

Local extrema, specified as a scalar greater than or equal to `0`. Increase this value to exclude less significant local extrema.

`NumOctaves` — Multiscale detection factor
3 (default) | positive integer

Multiscale detection factor, specified as a positive integer. Increase this value to detect larger features. To disable multiscale detection, set `NumOctaves` to `1`. When you set the value to `1`, the function performs the detection at the scale of the input image. Recommended values are between 1 and 4.

`NumScaleLevels` — Scale levels
4 (default) | integer

Scale levels, specified as an integer in the range [3,10]. Increase this value to achieve smoother scale changes. Increasing this value also provides additional intermediate scales between octaves.

`ROI` — Rectangular region size
`[1 1 size(I,2) size(I,1)]` (default) | 4-element vector

Rectangular region size for corner detection, specified as a 4-element vector in the format [*y x width height*]. The [*y x*] values are measured from the upper left corner of the rectangle.

Output Arguments

`points` — KAZE points object
`KAZEpoints` object

KAZE points, returned as a `KAZEPoints` object. The object contains information about the point features detected in the 2-D grayscale or binary input image.

Version History

Introduced in R2017b

References

[1] Alcantarilla, P.F., A. Bartoli, and A.J. Davison. "KAZE Features." *ECCV 2012, Part VI, LNCS 7577*. 2012, p. 214

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`detectSURFFeatures` | `detectMinEigenFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `detectORBFeatures` | `extractFeatures` | `extractHOGFeatures` | `detectHarrisFeatures` | `detectFASTFeatures` | `matchFeatures`

Objects

`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

selectUniform

Select uniformly distributed subset of point features

Syntax

```
pointsOut = selectUniform(pointsIn,N,imageSize)
[pointsOut,indices] = selectUniform(pointsIn,N,imageSize)
```

Description

`pointsOut = selectUniform(pointsIn,N,imageSize)` returns `N` uniformly distributed points with the strongest metrics.

`[pointsOut,indices] = selectUniform(pointsIn,N,imageSize)` returns the indices of the selected points from the input `pointsIn`.

Examples

Select Uniformly Distributed Subset of KAZE Features

Read an image.

```
im = imread("yellowstone_left.png");
```

Detect and display KAZE features.

```
points1 = detectKAZEFeatures(im2gray(im))
```

```
points1 =
  6713×1 KAZEPoints array with properties:

       Location: [6713×2 single]
         Metric: [6713×1 single]
          Count: 6713
          Scale: [6713×1 single]
    Orientation: [6713×1 single]
```

```
subplot(1,2,1)
imshow(im)
hold on
plot(points1)
hold off
title("Original Points")
```

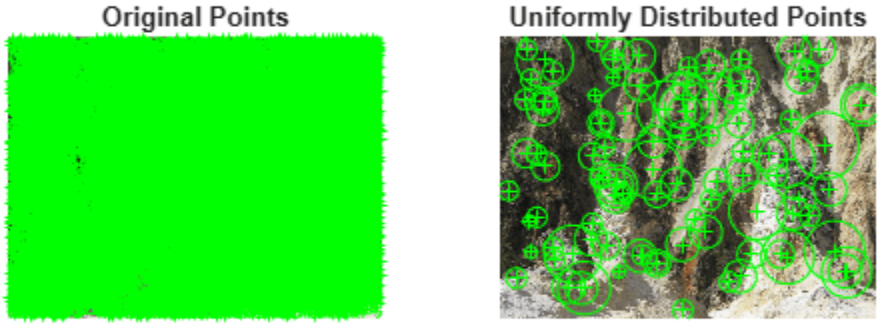
Select a uniformly distributed subset of points.

```
numPoints = 100;
points2 = selectUniform(points1,numPoints,size(im))
```

```
points2 =
  100×1 KAZEPoints array with properties:
```

```
       Location: [100×2 single]
         Metric: [100×1 single]
          Count: 100
          Scale: [100×1 single]
    Orientation: [100×1 single]
```

```
subplot(1,2,2)
imshow(im)
hold on
plot(points2)
hold off
title("Uniformly Distributed Points")
```



Select Uniformly Distributed Subset of Harris Features From Image

Load an image.

```
im = imread("yellowstone_left.png");
```

Detect many corners by reducing the quality threshold.

```
points1 = detectHarrisFeatures(im2gray(im),MinQuality=0.05);
```

Plot image with detected corners.

```
subplot(1,2,1);
imshow(im);
hold on
plot(points1);
```

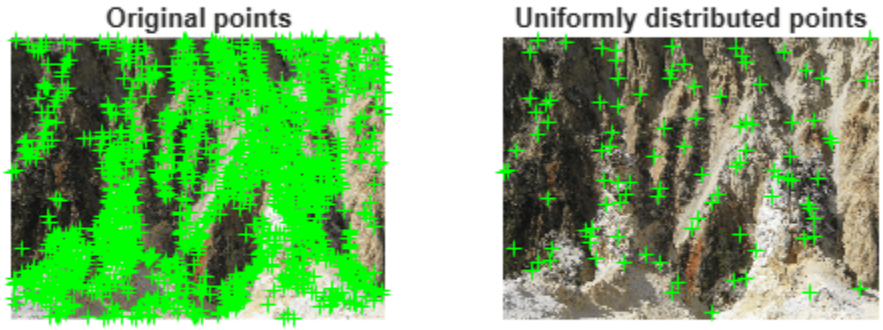
```
hold off
title("Original points");
```

Select a uniformly distributed subset of points.

```
numPoints = 100;
points2 = selectUniform(points1,numPoints,size(im));
```

Plot images showing original and subset of points.

```
subplot(1, 2, 2);
imshow(im);
hold on
plot(points2);
hold off
title("Uniformly distributed points");
```



Select A Uniformly Distributed Subset of BRISK Features From Image

Load an image into the workspace.

```
im = imread("yellowstone_left.png");
```

Detect many corners by reducing the quality threshold.

```
points1 = detectBRISKFeatures(im2gray(im),MinQuality=0.05);
```

Plot image with detected corners.

```
subplot(1,2,1);
imshow(im);
```

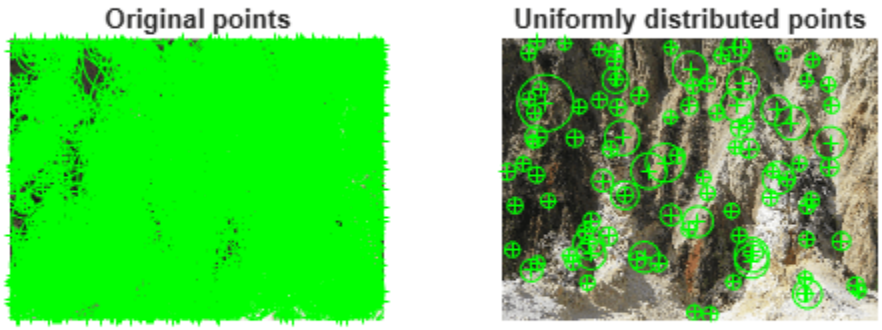
```
hold on
plot(points1);
hold off
title("Original points");
```

Select a uniformly distributed subset of points.

```
numPoints = 100;
points2 = selectUniform(points1,numPoints,size(im));
```

Plot images showing original and subset of points.

```
subplot(1, 2, 2);
imshow(im);
hold on
plot(points2);
hold off
title("Uniformly distributed points");
```



Select Uniformly Distributed Subset of SURF Features from an Image

Load an image.

```
im = imread("yellowstone_left.png");
```

Detect and display SURF features.

```
points1 = detectSURFFeatures(im2gray(im));
subplot(1,2,1);
imshow(im);
hold on
```

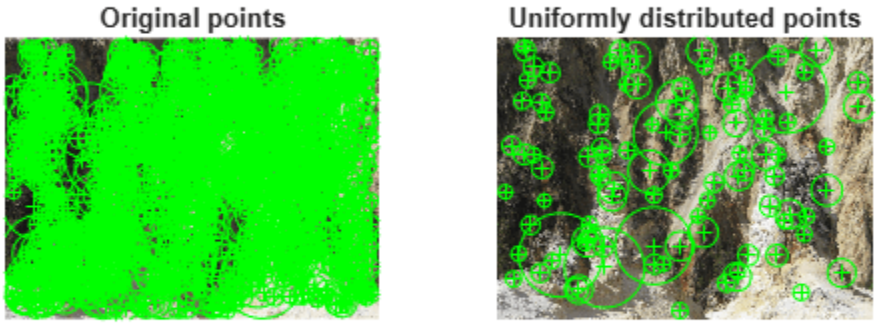
```
plot(points1);
hold off
title("Original points");
```

Select a uniformly distributed subset of points.

```
numPoints = 100;
points2 = selectUniform(points1,numPoints,size(im));
```

Plot images showing original and subset of points.

```
subplot(1, 2, 2);
imshow(im);
hold on
plot(points2);
hold off
title("Uniformly distributed points");
```



Input Arguments

`pointsIn` — Points object

point feature object

Points object,specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The object contains information about the point features detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

`N` — Number of points

integer

Number of uniformly distributed points to select, specified as an integer.

`imageSize` — Size of image
2-element vector | 3-element vector

Size of image, specified as a 2-element vector for grayscale images or a 3-element vector for truecolor images.

Output Arguments

`pointsOut` — Uniformly distributed points
points feature object

Uniformly distributed points, returned as a points feature object.

`indices` — Indices of selected points
N-element vector

Indices of selected points from the input, returned as an *N*-element vector of linear indices into `pointsIn`.

Version History

Introduced in R2012a

R2026a: Function returns indices of selected points

The function can return the indices of selected points from the input.

See Also

Functions
`extractFeatures` | `matchFeatures`

Topics
“Point Feature Types”

size

Return size of points object

Syntax

```
sz = size(points)
sz = size(points,1)
sz = size(points,dimension)
[M,N] = size(points)
```

Description

`sz = size(points)` returns the size of the points object. `sz` is defined as the vector `[length(points),1]`.

`sz = size(points,1)` returns the length of `points`.

`sz = size(points,dimension)` returns the length of the points in the specified dimension.

`[M,N] = size(points)` returns `length(points)` for *M* and 1 for *N*

Examples

Find Size of Points Object

Read an image.

```
I = imread('cameraman.tif');
```

Detect corner features.

```
featurePoints = detectHarrisFeatures(I);
```

Find the size of the feature points object.

```
sz = size(featurePoints)
```

```
sz = 1×2

   184     1
```

Plot feature image with detected features.

```
imshow(I); hold on;
plot(featurePoints);
```



Input Arguments

`points` — Points object

point feature object

Points object,specified as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

`dimension` — Dimension

integer

Dimension, returned as an integer. For `dim >= 2`, the object returns `1`.

Version History

Introduced in R2012a

See Also

Functions

`detectKAZEFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectORBFeatures` | `detectSIFTFeatures` | `extractFeatures` | `matchFeatures` | `detectSURFFeatures`

Objects

`MSERRegions` | `SURFPoints` | `BRISKPoints` | `KAZEPoints` | `cornerPoints` | `ORBPoints` | `SIFTPoints`

Topics
“Point Feature Types”

selectStrongest

Select points with strongest metrics

Syntax

```
strongestPoints = selectStrongest(pointsIn,N)
[strongestPoints,indices] = selectStrongest(pointsIn,N)
```

Description

`strongestPoints = selectStrongest(pointsIn,N)` returns `N` number of points that have the strongest metrics.

`[strongestPoints,indices] = selectStrongest(pointsIn,N)` returns `N` number of points that have the strongest metrics.

Examples

Select Strongest KAZE Features

Create a `KAZEPoints` object holding 50 points.

```
points = KAZEPoints(ones(50,2),Metric=1:50);
```

Keep the two strongest features.

```
points = selectStrongest(points,2)
```

```
points =
  2×1 KAZEPoints array with properties:

       Location: [2×2 single]
         Metric: [2×1 single]
          Count: 2
          Scale: [2×1 single]
    Orientation: [2×1 single]
```

Input Arguments

`pointsIn` — Points object

M-by-2 matrix | point feature object

Points object,specified as an *M*-by-2 matrix of *M* number of [*x y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

`N` — Number of points
integer

Number of strongest points to select, specified as an integer.

Output Arguments

`strongestPoints` — Points object with strongest metrics
M-by-2 matrix | points object

Points object with strongest metrics, returned as an *M*-by-2 matrix or a points object.

`indices` — Indices of selected points
N-element vector

Indices of selected points from the input, returned as an *N*-element vector of linear indices into `pointsIn`.

Version History

Introduced in R2017b

R2026a: Function returns indices of selected points

The function can return the indices of selected points from the input.

See Also

Functions
`detectKAZEFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectORBFeatures` | `detectSIFTFeatures` | `extractFeatures` | `matchFeatures` | `detectSURFFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `KAZEPoints` | `cornerPoints` | `ORBPoints` | `SIFTPoints`

Topics
“Point Feature Types”

plot

Plot detected feature points

Syntax

```
plot(points)
plot(points,ax)
plot(points,ax,Name=Value)
```

Description

`plot(points)` plots detected feature points in the current axis. The input `points` is a feature point object that stores the locations and related information of the detected features.

`plot(points,ax)` displays feature points in the specified axis.

`plot(points,ax,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `plot(ShowOrientation=true)` renders the orientation in the display plot.

Examples

Plot KAZE Points

Read an image.

```
I = imread("cameraman.tif");
```

Detect KAZE points from the image.

```
points = detectKAZEFeatures(I);
```

Extract KAZE features from the detected points.

```
[features,valid_points] = extractFeatures(I,points);
```

Plot the 10 strongest points and show their orientations.

```
imshow(I)
hold on
strongestPoints = selectStrongest(valid_points,10);
plot(strongestPoints,showOrientation=true)
hold off
```



Plot SURF Features

```
I = imread("cameraman.tif");
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I,points);
```

Visualize 10 strongest SURF features, including their scales and orientation which were determined during the descriptor extraction process.

```
imshow(I);
hold on;
strongestPoints = valid_points.selectStrongest(10);
strongestPoints.plot("showOrientation",true);
```



Input Arguments

`points` — Points object

M-by-2 matrix | point feature object

Points object,specified as an *M*-by-2 matrix of *M* number of [*x y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

`ax` — Axes handle

handle

Handle to use for display. You can set the handle using `gca`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `plot(ShowOrientation=true)` renders the orientation in the display plot.

`ShowScale` — Display scaled circle

`true` (default) | `false`

Display scaled circle, specified as `true` or `false`. When you set this value to `true`, the object draws a circle proportional to the scale of the detected feature, with the feature point located at its center. When you set this value to `false`, the object turns the display of the circle off.

The algorithm represents the scale of the feature with a circle of `6*Scale` radius. The algorithm uses this equivalent size of circular area to compute the orientation of the feature.

`ShowOrientation` — Display orientation line
`true` (default) | `false`

Display feature point orientation, specified as `true` or `false`. When you set this value to `true`, the object draws a line corresponding to the point's orientation. The object draws the line from the feature point location to the edge of the circle, indicating the scale.

Version History

Introduced in R2011b

See Also

Functions
`extractFeatures` | `matchFeatures` | `plot`

plot

Plot points

Syntax

```
plot(points)
plot(points,ax)
```

Description

`plot(points)` plots points in the current axis.

`plot(points,ax)` plots points in the specified axis.

Examples

Plot Corner Features

Read an image.

```
I = imread("cameraman.tif");
```

Detect corner features.

```
featurePoints = detectHarrisFeatures(I);
```

Plot feature image with detected features.

```
imshow(I);
hold on;
plot(featurePoints);
```



Input Arguments

`points` — Points object

M-by-2 matrix | point feature object

Points object,specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

ax — Axes handle

handle

Handle to use for display. You can set the handle using `gca`.

Version History

Introduced in R2011b

See Also

Functions

`extractFeatures` | `matchFeatures`

plot

Plot parametric model

Syntax

```
plot(model)
plot(model,Name=Value)
H = plot( ___ )
```

Description

`plot(model)` plots a model within the axis limits of the current figure.

`plot(model,Name=Value)` specifies options using one or more name-value arguments in addition to argument from previous syntax. For example, `plot(model,Parent=gca)` sets the color of the plot to red.

`H = plot( ___ )` plots a model and returns a handle `H` to `surf`, a 3-D shaded surface plot or `patch`, a filled polygon region.

Examples

Detect Cylinder in Point Cloud

Load a MAT file containing a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud.

```
figure
pcshow(ptCloud)
title("Detect a Cylinder in a Point Cloud")
```

Set the maximum point-to-cylinder distance for cylinder fitting to 5mm.

```
maxDistance = 0.005;
```

Specify a region of interest (ROI) to constrain the fitting function.

```
roi = [0.4 0.6; -inf 0.2; 0.1 inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set the orientation constraint for the fitting function

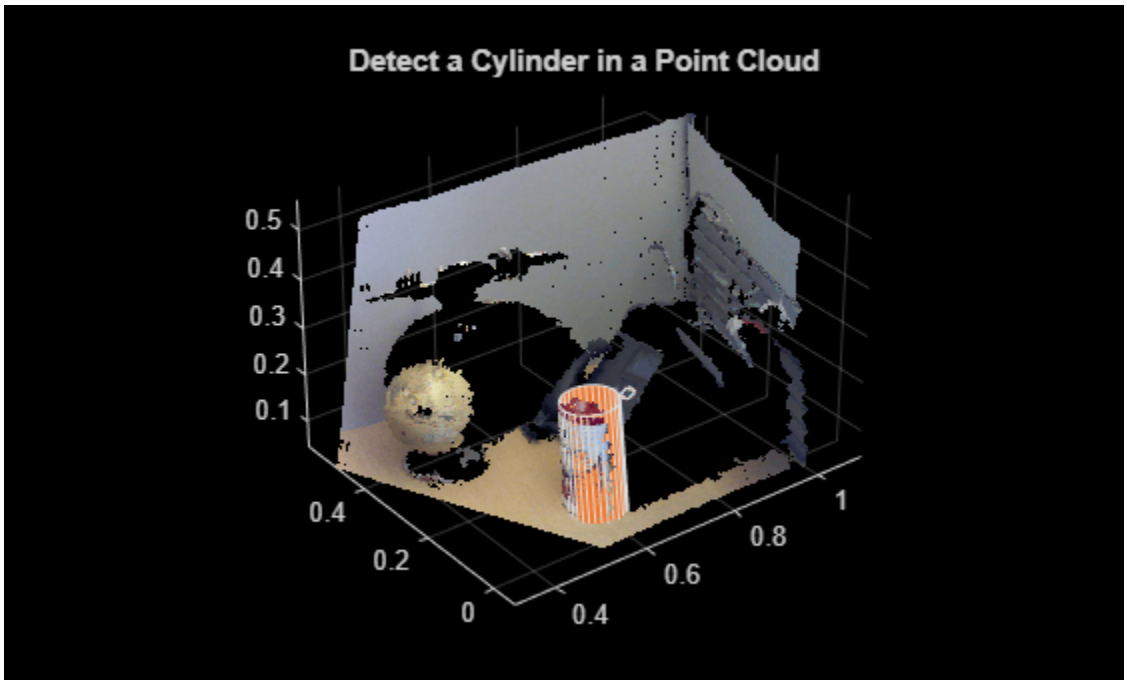
```
referenceVector = [0 0 1];
```

Detect the cylinder in the specified ROI of the point cloud and extract it.

```
model = pcfitcylinder(ptCloud,maxDistance,referenceVector, ...
        SampleIndices=sampleIndices);
```

Plot the model of the detected cylinder.

```
hold on
plot(model)
```



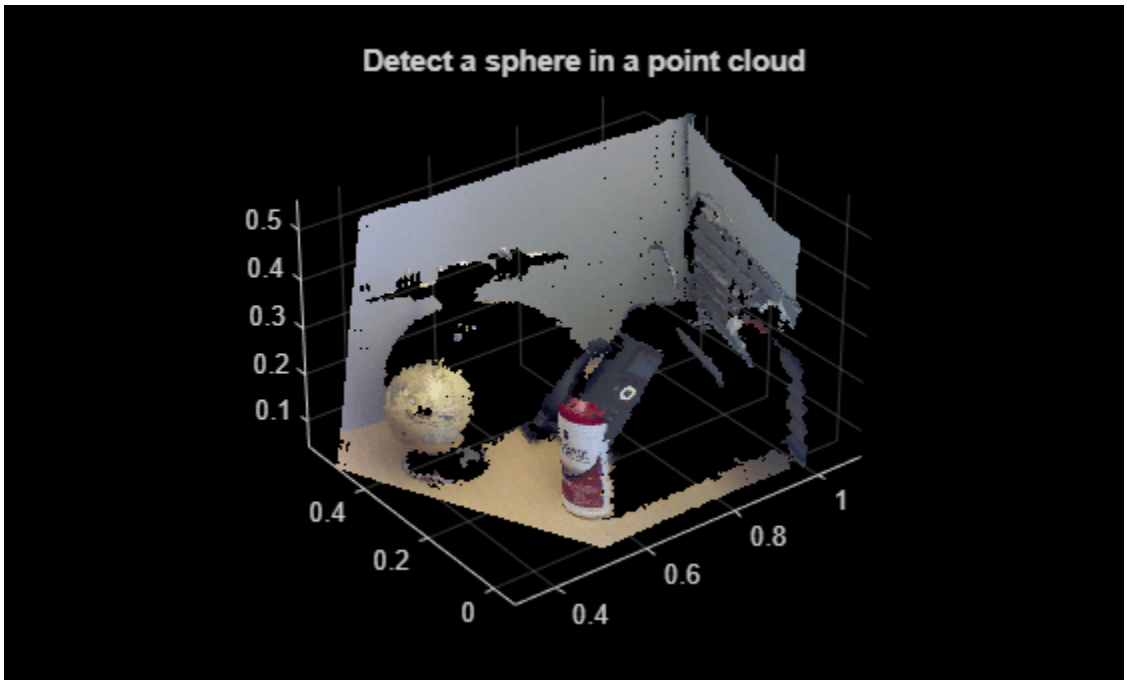
Detect Sphere in Point Cloud

Load a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud and label the figure.

```
figure
pcshow(ptCloud)
title("Detect a sphere in a point cloud")
```



Set the maximum point-to-sphere distance for sphere fitting to 1cm.

```
maxDistance = 0.01;
```

Set the region of interest to constrain the search.

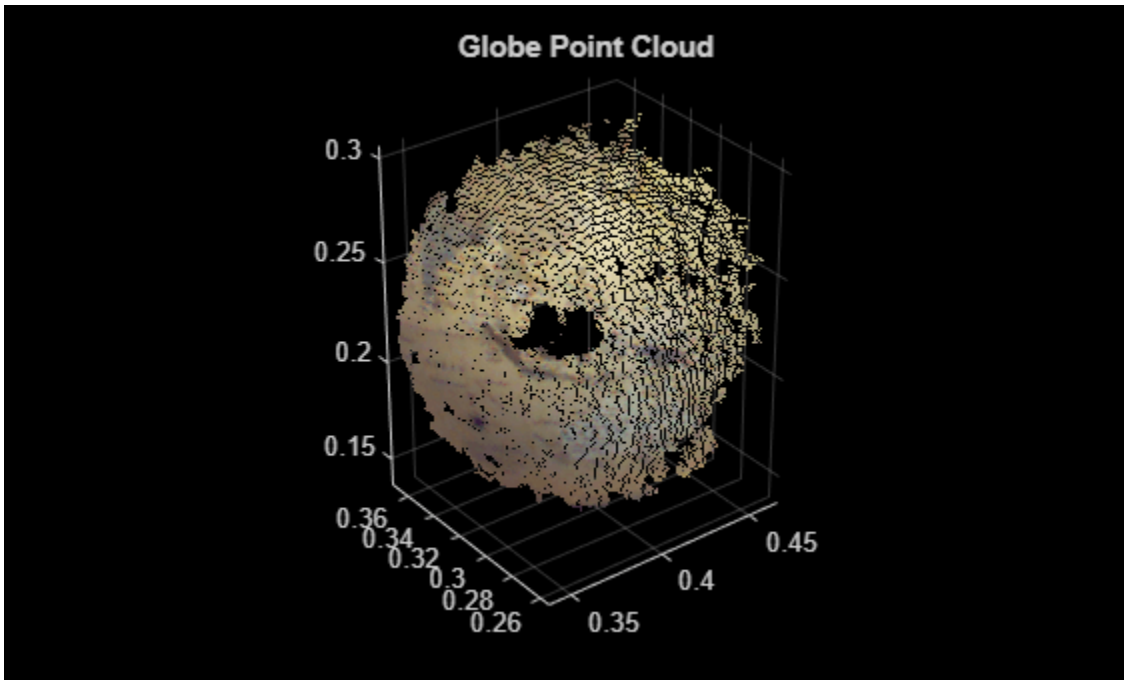
```
roi = [-inf,0.5;0.2,0.4;0.1,inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Detect the globe in the point cloud and extract it.

```
[model,inlierIndices] = pcfitsphere(ptCloud,maxDistance,SampleIndices=sampleIndices);
globe = select(ptCloud,inlierIndices);
```

Plot the extracted globe.

```
figure
pcshow(globe)
title("Globe Point Cloud")
```



Detect Plane in Point Cloud

Load a MAT file containing a point cloud.

```
load("object3d.mat");
```

Display the point cloud into the workspace.

```
figure
pcshow(ptCloud)
title("Detect a Plane in a Point Cloud")
```

Set the maximum point-to-plane distance for plane fitting to 2cm.

```
maxDistance = 0.02;
```

Specify the normal vector plane for the fitted plane.

```
referenceVector = [0 0 1];
```

Set the maximum angular distance for the fitted plane to 5 degrees.

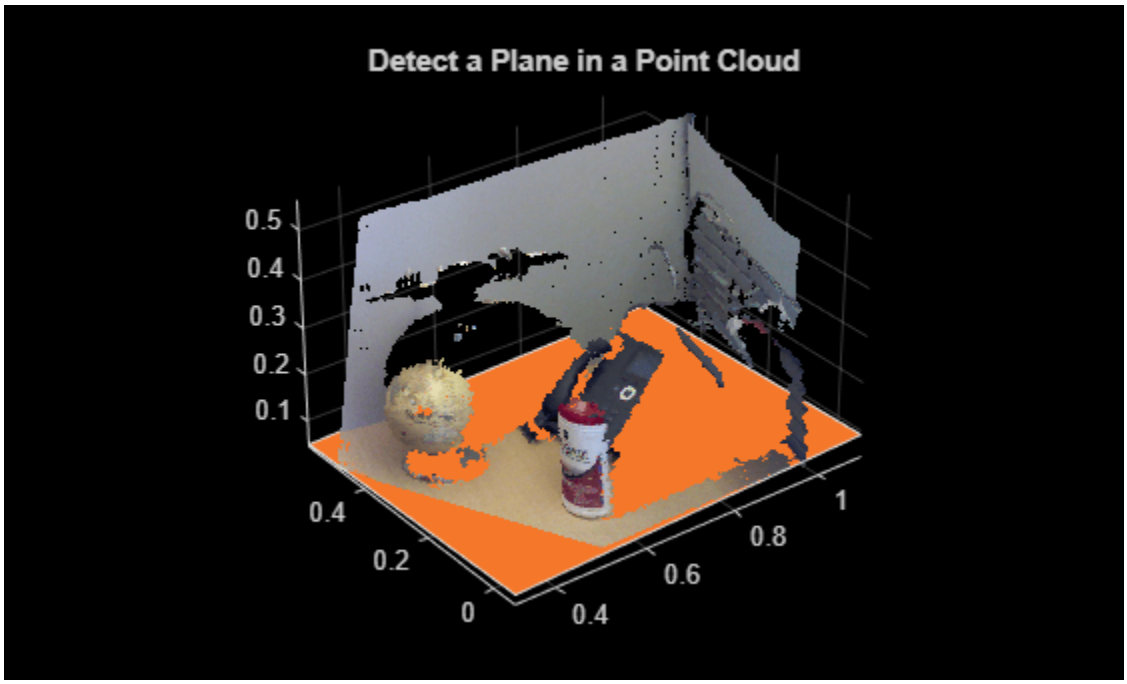
```
maxAngularDistance = 5;
```

Detect the plane, the table, in the point cloud and extract it.

```
model = pcfitplane(ptCloud,maxDistance,referenceVector,maxAngularDistance);
```

Plot the plane.

```
hold on
plot(model)
```



Input Arguments

`model` — Parametric model
cylinder model object | plane model object | sphere model object

Parametric model returned by the `cylinderModel`, `planeModel`, or the `sphereModel` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(model,Parent=gca)` sets the output axes to the current axes in the current figure.

`Parent` — Output axes
`gca` (default) | `axes`

Output axes, specified as `gca` (the current axes) or `axes` (a custom set axes) for displaying the visualization.

`Color` — Color of the plane
`[245 119 41]` (default) | 1-by-3 RGB vector | short name of color | long name of color

Color of the plane, specified as a 1-by-3 RGB vector or a short or long color name. The default color [245 119 41] is well-suited for visualizing the model when fitting it to point clouds.

Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	
"green"	"g"	[0 1 0]	
"blue"	"b"	[0 0 1]	
"cyan"	"c"	[0 1 1]	
"magenta"	"m"	[1 0 1]	
"yellow"	"y"	[1 1 0]	
"black"	"k"	[0 0 0]	
"white"	"w"	[1 1 1]	

Output Arguments

`H` — Handle to plot

handle

Handle to a plot, returned as a `surf` 3-D shaded surface plot or as a `patch` filled polygon region.

Version History

Introduced in R2015b

See Also

Objects

`cylinderModel` | `sphereModel` | `planeModel`

Functions

`pcfitcylinder` | `pcfitplane` | `pcfitsphere`

length

Number of stored points

Syntax

```
length(points)
```

Description

`length(points)` returns the number of stored points in the points object.

Examples

Check Number of Stored Points

Read an image.

```
I = imread("cameraman.tif");
```

Detect KAZE points from the image.

```
points = detectKAZEFeatures(I);
```

Find number of stored points.

```
numPoints = length(points)
```

```
numPoints =
631
```

Extract KAZE features from the detected points.

```
[features,validPoints] = extractFeatures(I,points);
```

Plot the ten strongest valid extracted points and show their orientations.

```
imshow(I)
hold on
strongestPoints = selectStrongest(validPoints,10);
plot(strongestPoints,showOrientation=true)
hold off
```



Input Arguments

`points` — Points object

point feature object

Points object,specified as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

Version History

Introduced in R2017b

See Also

Functions

`detectKAZEFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectORBFeatures` | `detectSIFTFeatures` | `extractFeatures` | `matchFeatures` | `detectSURFFeatures`

Objects

`MSERRegions` | `SURFPoints` | `BRISKPoints` | `KAZEPoints` | `cornerPoints` | `ORBPoints` | `SIFTPoints`

Topics

“Point Feature Types”

isempty

Determine if points object is empty

Syntax

```
isempty(points)
```

Description

`isempty(points)` returns a `true` value, if the points object is empty.

Examples

Check If Points Object Is Empty

Read an image.

```
I = imread('cameraman.tif');
```

Detect KAZE points from the image.

```
points = detectKAZEFeatures(I);
```

Check for points.

```
if isempty(points)
    return;
end
```

Extract KAZE features from the detected points.

```
[features,validPoints] = extractFeatures(I,points);
```

Plot the ten strongest valid extracted points and show their orientations.

```
imshow(I)
hold on
strongestPoints = selectStrongest(validPoints,10);
plot(strongestPoints,'showOrientation',true)
hold off
```



Input Arguments

`points` — Points object

point feature object

Points object,specified as one of the point feature objects described in “Point Feature Types”. The object contains information about the feature points detected in the input image. To obtain points, use the appropriate detect function that pairs with the point feature type.

Version History

Introduced in R2017b

See Also

Functions

`detectKAZEFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectORBFeatures` | `detectSIFTFeatures` | `extractFeatures` | `matchFeatures` | `detectSURFFeatures`

Objects

`MSERRegions` | `SURFPoints` | `BRISKPoints` | `KAZEPoints` | `cornerPoints` | `ORBPoints` | `SIFTPoints`

Topics

“Point Feature Types”

gather

Retrieve cornerPoints from the GPU

Syntax

```
pointsCPU = gather(pointsGPU)
```

Description

`pointsCPU = gather(pointsGPU)` returns a `cornerPoints` object with data gathered from the GPU for the `Location` and `Metric` properties.

Examples

Find and Plot Corner Points in Image

Read an image.

```
I = imread('cameraman.tif')
```

Create a GPU array object.

```
I = gpuArray(I);
```

Find and display Harris features in the image.

```
pointsGPU = detectHarrisFeatures(I);
imshow(I);
hold on;
plot(pointsGPU.selectStrongest(50));
```



Copy the corner points to the CPU for further processing.

```
pointsCPU = gather(pointsGPU);
```

Input Arguments

pointsGPU — GPU points
`cornerPoints` object

GPU points, specified as a `cornerPoints` object.

Version History

Introduced in R2012a

See Also

`cornerPoints`

plot

Plot MSER regions

Syntax

```
plot(points)
plot(points,ax)
plot(points,ax,Name=Value)
```

Description

`plot(points)` plots points in the current axis.

`plot(points,ax)` plots points in the specified axis.

`plot(points,ax,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `plot(points,ax,ShowOrientation=true)` sets the logical value to true to show the orientation.

Examples

Plot MSER Regions

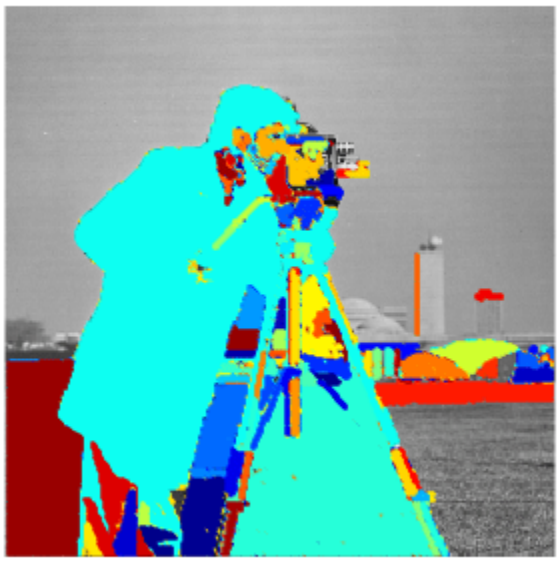
Read image and extract MSER features.

```
I = imread("cameraman.tif");
regions = detectMSERFeatures(I);
imshow(I); hold on;
plot(regions);
```



Plot MSER Regions.

```
figure; imshow(I);
hold on;
plot(regions,showPixelList=true,showEllipses=false);
hold off;
```



Input Arguments

`points` — Points
`MSERRegions` object

Points, specified as a points object. The object contains information about the feature points detected in the 2-D grayscale input image.

`ax` — Axes handle
handle

Handle to use for display. You can set the handle using `gca`.

Name-Value Pair Arguments

Example: `plot(points,ax,ShowOrientation=true)` sets the logical value to true to show the orientation.

`showEllipses` — Display ellipsis
`true` (default) | `false`

Display ellipsis around feature, specified as `true` or `false`. When you set this value to `true`, the object draws an ellipse with the same 2nd order moments as the region. When you set this value to `false`, only the ellipses centers are plotted.

`ShowOrientation` — Display orientation line
`true` (default) | `false`

Display feature point orientation, specified as `true` or `false`. When you set this value to `true`, the object draws a line corresponding to the point's orientation. The object draws the line from the feature point location to the edge of the circle, indicating the scale.

`showPixelList` — Display regions using JET colormap
`true` (default) | `false`

Display regions using JET colormap, specified as `true` or `false`.

Version History

Introduced in R2012a

See Also

`detectMSERFeatures`

assignDetectionsToTracks

Assign detections to tracks for multiobject tracking

Syntax

```
[assignments,unassignedTracks,unassignedDetections] =
assignDetectionsToTracks(costMatrix,costOfNonAssignment)
[assignments,unassignedTracks,unassignedDetections] =
assignDetectionsToTracks(costMatrix, unassignedTrackCost,
unassignedDetectionCost)
```

Description

`[assignments,unassignedTracks,unassignedDetections] = assignDetectionsToTracks(costMatrix,costOfNonAssignment)` assigns detections to tracks in the context of multiple object tracking using the James Munkres's variant of the Hungarian assignment algorithm. It also determines which tracks are missing and which detections should begin new tracks. It returns the indices of assigned and unassigned tracks, and unassigned detections. The `costMatrix` must be an *M*-by-*N* matrix. In this matrix, *M* represents the number of tracks, and *N* is the number of detections. Each value represents the cost of assigning the N^{th} detection to the M^{th} track. The lower the cost, the more likely that a detection gets assigned to a track. The `costOfNonAssignment` scalar input represents the cost of a track or a detection remaining unassigned.

`[assignments,unassignedTracks,unassignedDetections] = assignDetectionsToTracks(costMatrix, unassignedTrackCost, unassignedDetectionCost)` specifies the cost of unassigned tracks and detections separately. The `unassignedTrackCost` must be a scalar value, or an *M*-element vector, where *M* represents the number of tracks. For the *M*-element vector, each element represents the cost of not assigning any detection to that track. The `unassignedDetectionCost` must be a scalar value or an *N*-element vector, where *N* represents the number of detections.

Examples

Assign Detections to Tracks in a Single Video Frame

This example shows you how to assign a detection to a track for a single video frame.

Set the predicted locations of objects in the current frame. Obtain predictions using the Kalman filter System object.

```
predictions = [1,1;2,2];
```

Set the locations of the objects detected in the current frame. For this example, there are 2 tracks and 3 new detections. Thus, at least one of the detections is unmatched, which can indicate a new track.

```
detections = [1.1,1.1;2.1,2.1;1.5,3];
```

Preallocate a cost matrix.

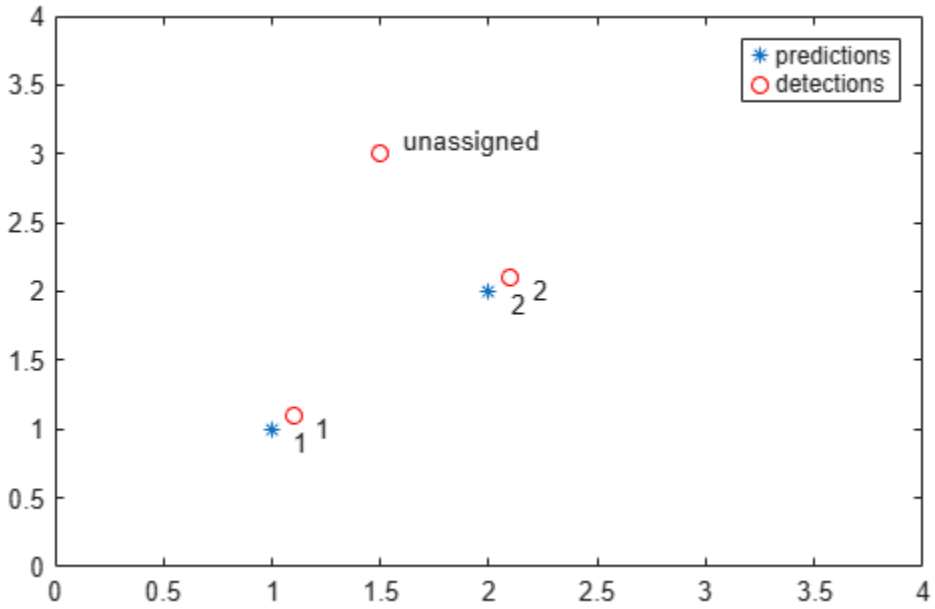
```
cost = zeros(size(predictions,1),size(detections,1));
```

Compute the cost of each prediction matching a detection. The cost here, is defined as the Euclidean distance between the prediction and the detection.

```
for i = 1:size(predictions, 1)
      diff = detections - repmat(predictions(i,:),[size(detections,1),1]);
      cost(i, :) = sqrt(sum(diff .^ 2,2));
end
```

Associate detections with predictions. Detection 1 should match to track 1, and detection 2 should match to track 2. Detection 3 should be unmatched.

```
[assignment,unassignedTracks,unassignedDetections] = ...
            assignDetectionsToTracks(cost,0.2);
  figure;
  plot(predictions(:,1),predictions(:,2),'*',detections(:,1),...
            detections(:,2),'ro');
  hold on;
  legend('predictions','detections');
  for i = 1:size(assignment,1)
    text(predictions(assignment(i, 1),1)+0.1,...
            predictions(assignment(i,1),2)-0.1,num2str(i));
    text(detections(assignment(i, 2),1)+0.1,...
            detections(assignment(i,2),2)-0.1,num2str(i));
  end
  for i = 1:length(unassignedDetections)
    text(detections(unassignedDetections(i),1)+0.1,...
            detections(unassignedDetections(i),2)+0.1,'unassigned');
  end
  xlim([0,4]);
  ylim([0,4]);
```

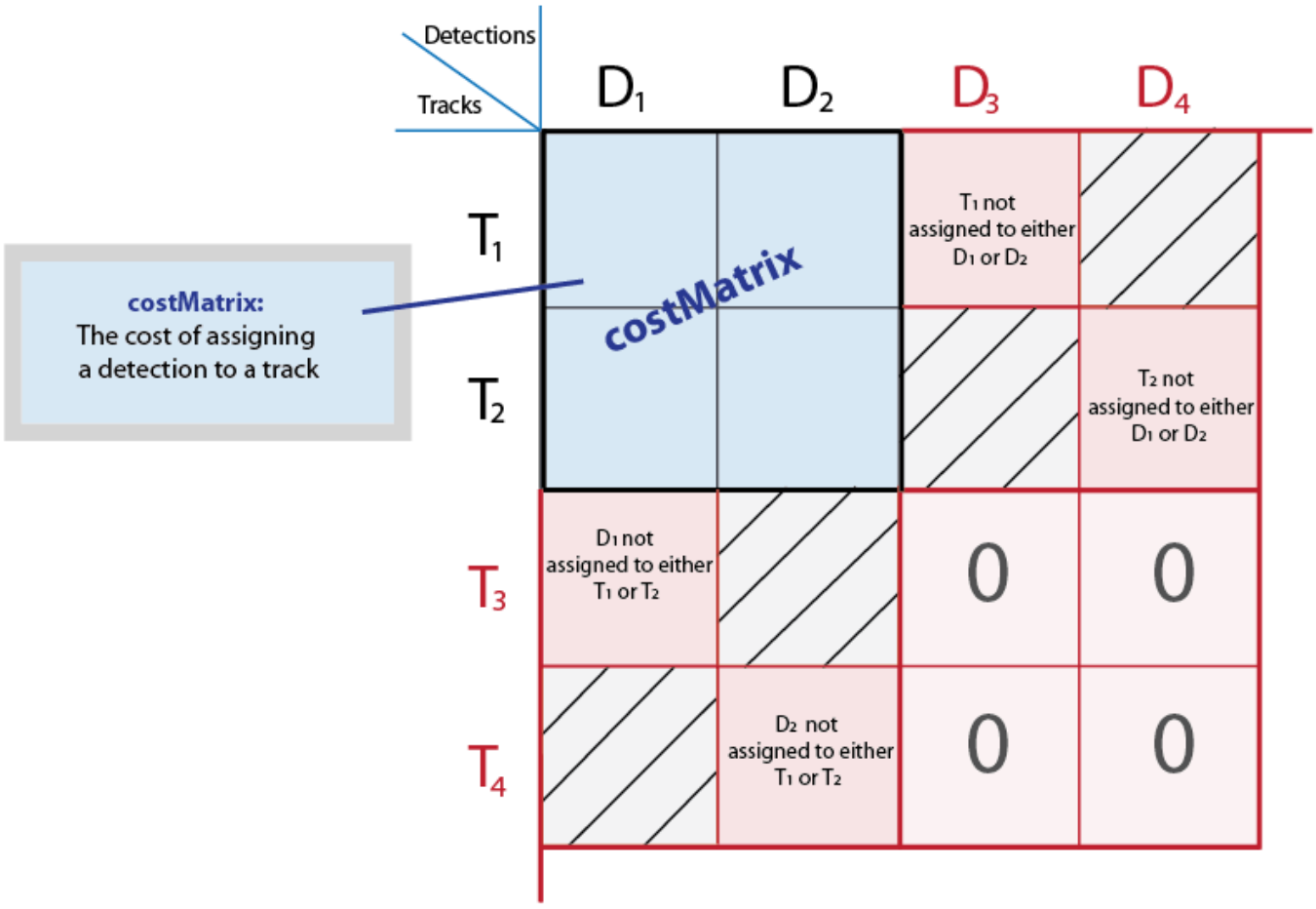


Input Arguments

`costMatrix` — Cost of assigning detection to track

M-by-*N* matrix

Cost of assigning a detection to a track, specified as an *M*-by-*N* matrix, where *M* represents the number of tracks, and *N* is the number of detections. The cost matrix value must be real, nonsparse, and numeric. The lower the cost, the more likely that a detection gets assigned to a track. Each value represents the cost of assigning the N^{th} detection to the M^{th} track. If there is no likelihood of an assignment between a detection and a track, the `costMatrix` input is set to `Inf`. Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. The function applies the Hungarian assignment algorithm to the padded matrix.



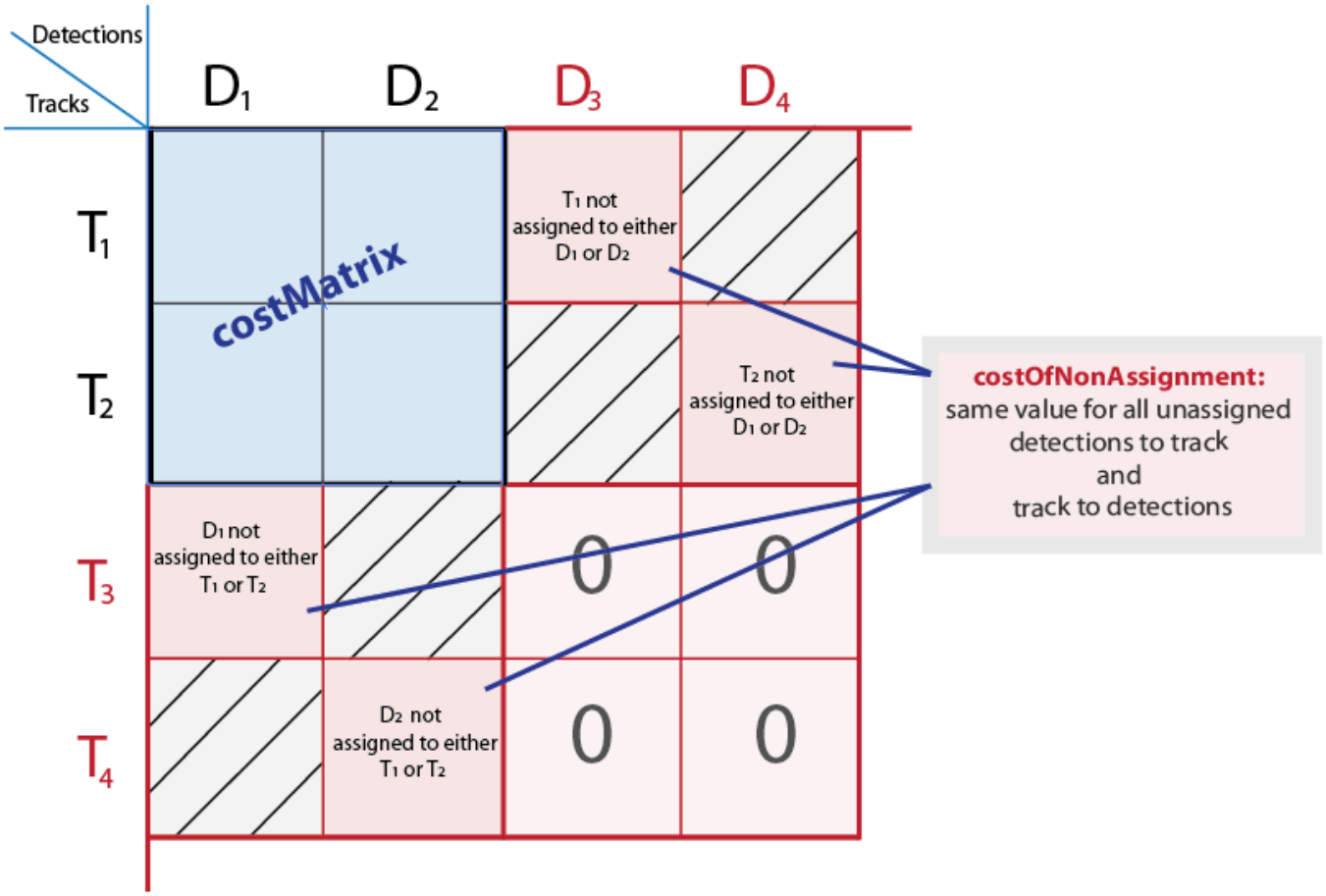
Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double`

`costOfNonAssignment` — Cost of not assigning detection to any track or track to any detection

scalar | finite

Cost of not assigning detection to any track or track to detection. You can specify this value as a scalar value representing the cost of a track or a detection remaining unassigned. An unassigned detection may become the start of a new track. If a track is unassigned, the object does not appear. The higher the `costOfNonAssignment` value, the higher the likelihood that every track will be assigned a detection.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input. To vary the values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs.



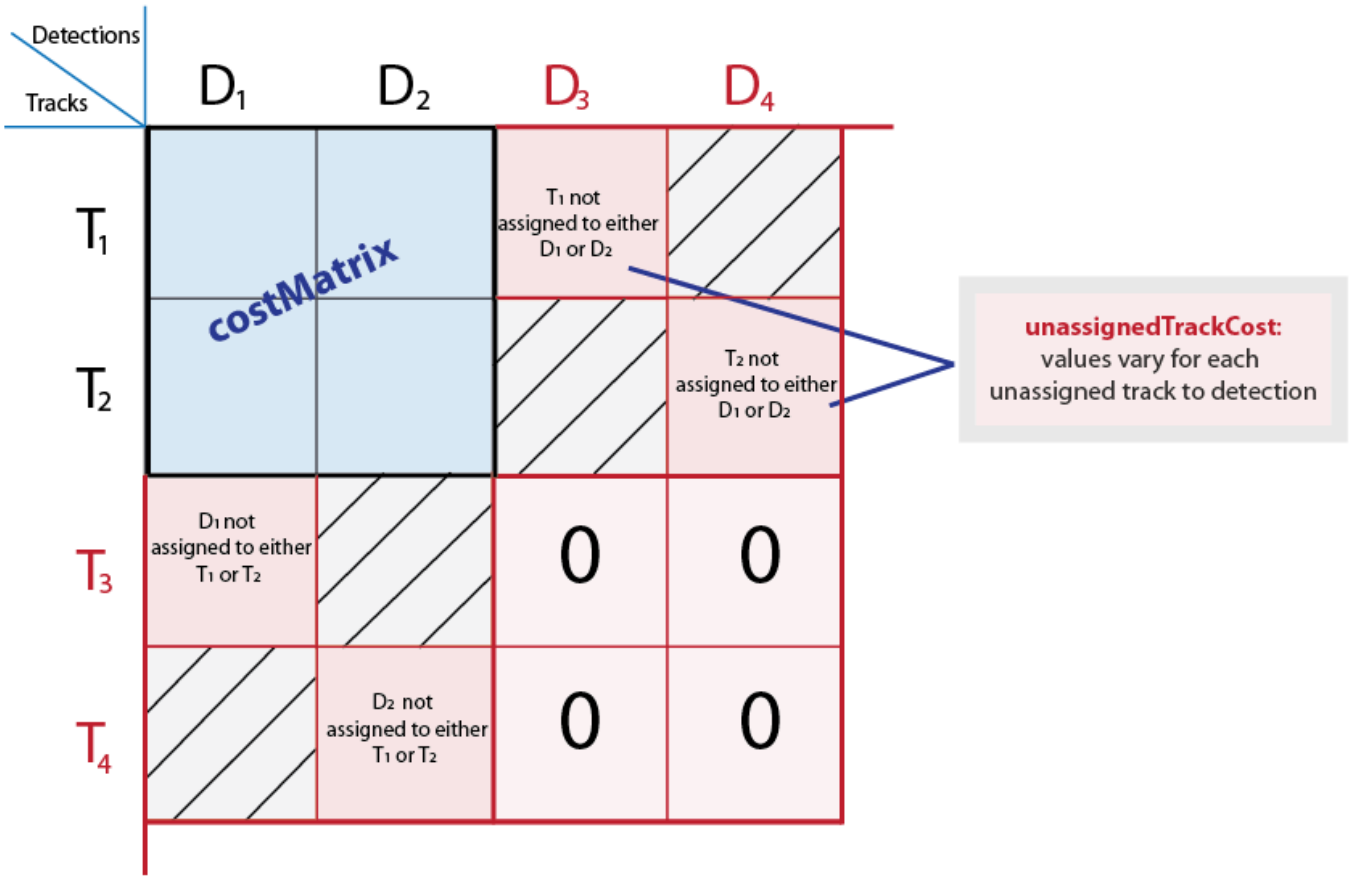
Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double`

`unassignedTrackCost` — Cost or likelihood of an unassigned track
M-element vector | scalar | finite

Cost or likelihood of an unassigned track. You can specify this value as a scalar value, or an *M*-element vector, where *M* represents the number of tracks. For the *M*-element vector, each element represents the cost of not assigning any detection to that track. A scalar input represents the same cost of being unassigned for all tracks. The cost may vary depending on what you know about each

track and the scene. For example, if an object is about to leave the field of view, the cost of the corresponding track being unassigned should be low.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To vary the values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input.



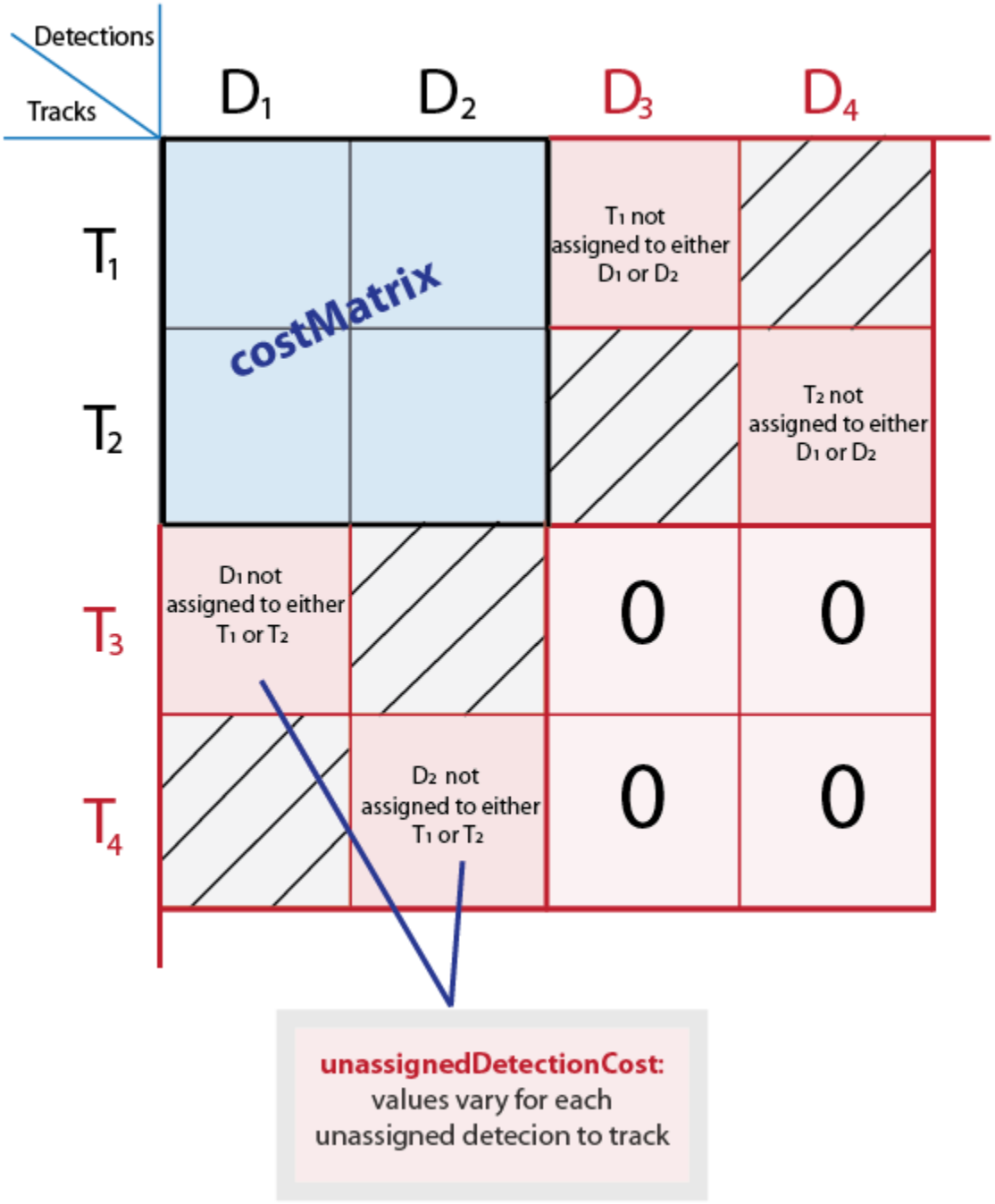
Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double`

`unassignedDetectionCost` — Cost of unassigned detection
N-element vector | scalar | finite

Cost of unassigned detection, specified as a scalar value or an *N*-element vector, where *N* represents the number of detections. For the *N*- element vector, each element represents the cost of starting a new track for that detection. A scalar input represents the same cost of being unassigned for all tracks. The cost may vary depending on what you know about each detection and the scene. For example, if a detection appears close to the edge of the image, it is more likely to be a new object.

Internally, this function pads the cost matrix with dummy rows and columns to account for the possibility of unassigned tracks and detections. The padded rows represent detections not assigned to any tracks. The padded columns represent tracks not associated with any detections. To vary the

values for different detections or tracks, use the syntax with the `unassignedTrackCost` and `unassignedDetectionCost` inputs. To apply the same value to all elements in both the rows and columns, use the syntax with the `costOfNonAssignment` input.



Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double`

Output Arguments

`assignments` — Index pairs of tracks and corresponding detections

L-by-2 matrix

Index pairs of tracks and corresponding detections. This value is returned as an *L*-by-2 matrix of index pairs, with *L* number of pairs. The first column represents the track index and the second column represents the detection index.

Data Types: `uint32`

`unassignedTracks` — Unassigned tracks

P-element vector

Unassigned tracks, returned as a *P*-element vector. *P* represents the number of unassigned tracks. Each element represents a track to which no detections are assigned.

Data Types: `uint32`

`unassignedDetections` — Unassigned detections

Q-element vector

Unassigned detections, returned as a *Q*-element vector, where *Q* represents the number of unassigned detections. Each element represents a detection that was not assigned to any tracks. These detections can begin new tracks.

Data Types: `uint32`

Version History

Introduced in R2012b

References

[1] Miller, Matt L., Harold S. Stone, and Ingemar J. Cox, “Optimizing Murty's Ranked Assignment Method,” *IEEE Transactions on Aerospace and Electronic Systems*, 33(3), 1997.

[2] Munkres, James, “Algorithms for Assignment and Transportation Problems,” *Journal of the Society for Industrial and Applied Mathematics*, Volume 5, Number 1, March, 1957.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`vision.KalmanFilter` | `configureKalmanFilter`

Topics

“Multiple Object Tracking”

External Websites
Munkres' Assignment Algorithm Modified for Rectangular Matrices

bbox2points

Convert rectangle to corner points list

Syntax

```
points = bbox2points(rectangle)
```

Description

`points = bbox2points(rectangle)` converts the input rectangle, into a list of four [*x* *y*] corner points. The `rectangle` input must be either a single bounding box or a set of bounding boxes.

Examples

Convert Bounding Box to List of Points and Apply Rotation

Define a bounding box.

```
bbox = [10,20,50,60];
```

Convert the bounding box to a list of four points.

```
points = bbox2points(bbox);
```

Define a rotation transformation.

```
theta = 10;
tform = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1]);
```

Apply the rotation.

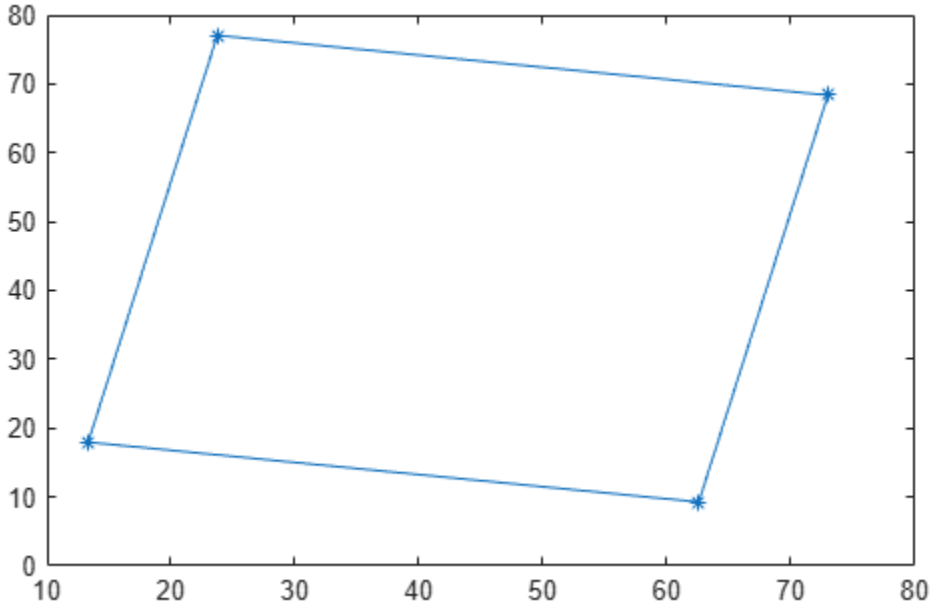
```
points2 = transformPointsForward(tform,points);
```

Close the polygon for display.

```
points2(end+1,:) = points2(1,:);
```

Plot the rotated box.

```
plot(points2(:,1),points2(:,2), '*-');
```



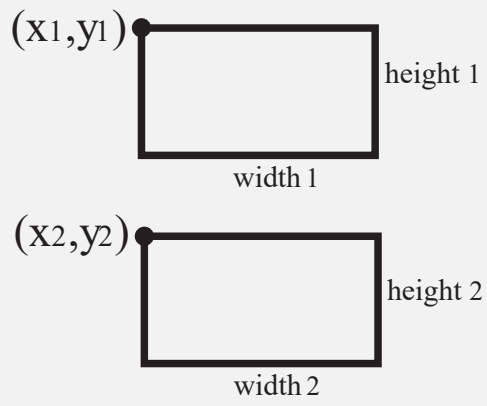
Input Arguments

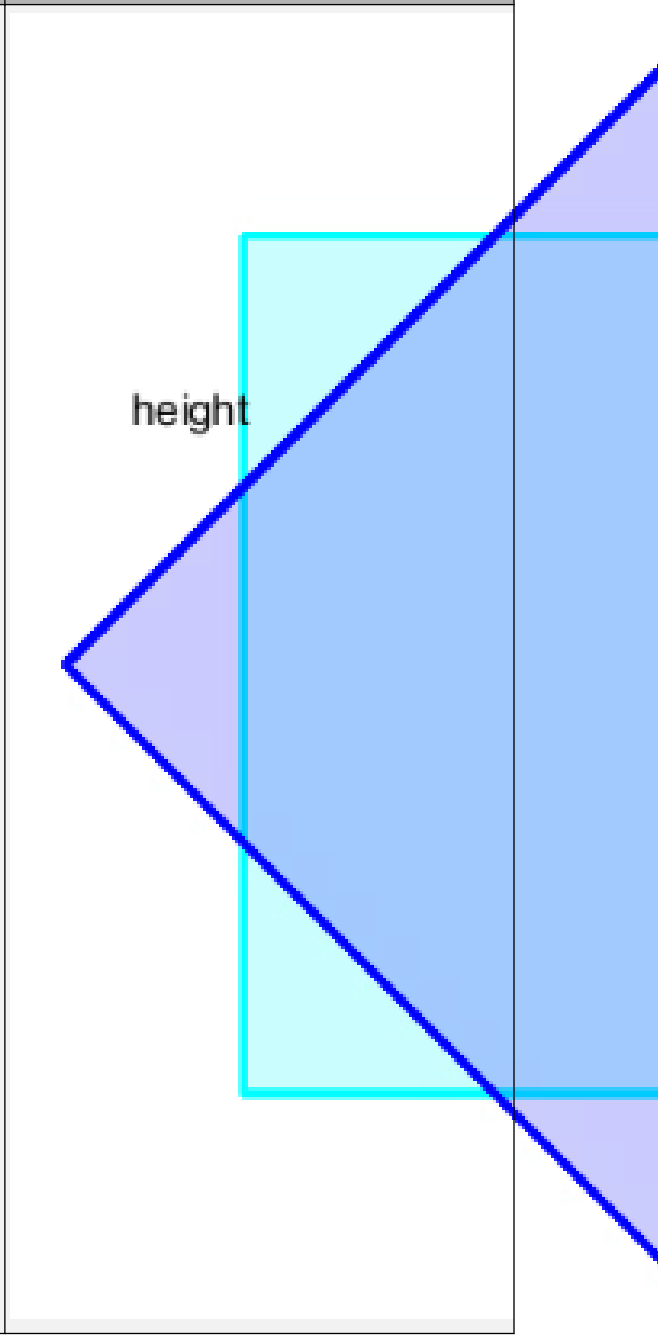
`rectangle` — Bounding box

4-element vector | 5-element vector | *M*-by-4 matrix | *M*-by-5 matrix

Bounding box, specified as a 4-element vector, 5-element vector, *M*-by-4 matrix, or *M*-by-5 matrix.

Bounding Box	Example

For one or more rectangles, specify one of:
• An *M*-by-4 matrix where each row specifies a rectangle as [*x y width height*].
$\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$
• An *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where:
  • *M* is the number of rotated rectangles.
  • *xctr* and *yctr* specify the center of the rectangle.
  • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation.
  • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation.
  • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box.
	

Bounding Box	Example
	

Data Types: `single` | `double` | `int16` | `int32` | `uint16` | `uint32`

Output Arguments

`points` — Rectangle corner coordinates
4-by-2 matrix | 4-by-2-by-*M* array

Rectangle corner coordinates, returned as a 4-by-2 matrix of [*x*,*y*] coordinates, or a 4-by-2-by-*M* array of [*x*,*y*] coordinates. The output points for the rectangle are listed counterclockwise starting from the upper-left corner.

- For a single input bounding box, the function returns the 4-by-2 matrix.
- For multiple input bounding boxes, the function returns the 4-by-2-*M* array for *M* bounding boxes.

Data Types: `single` | `double` | `int16` | `int32` | `uint16` | `uint32`

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`affine2d` | `projective2d`

bboxcrop

Crop bounding boxes

Syntax

```
bboxB = bboxcrop(bboxA,window)
[bboxB,indices] = bboxcrop(bboxA,window)
[___] = bboxcrop(___,Name=Value)
```

Description

`bboxB = bboxcrop(bboxA,window)` crops bounding boxes from a set of input bounding boxes, `bboxA`, located in the cropping area, `window`. `bboxB` contains the cropped bounding boxes. This function supports 2-D and 3-D bounding boxes.

`[bboxB,indices] = bboxcrop(bboxA,window)` additionally returns a vector of indices that indicate which bounding boxes in `bboxA` are within the cropping window, `window`.

`[___] = bboxcrop(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `OverlapThreshold=1`, sets the positive overlap threshold to `1`.

Examples

Center Crop Image and Corresponding Bounding Boxes

Read an image.

```
I = imread('peppers.png');
```

Define bounding boxes and labels.

```
bboxA = [
    410 230 100 90
    186 78  80  60
    ]
```

```
bboxA = 2×4

   410   230   100    90
   186    78    80    60
```

```
labelsA = [
    "garlic"
    "onion"
    ];
```

Create a center cropping window.

```
targetSize = [256 256];
win = centerCropWindow2d(size(I),targetSize);
```

Center crop the image.

```
[r,c] = deal(win.YLimits(1):win.YLimits(2),win.XLimits(1):win.XLimits(2));
J = I(r,c,:);
```

Center crop boxes and labels. Boxes outside the cropping window are removed.

```
[bboxB,indices] = bboxcrop(bboxA,win);
labelsB = labelsA(indices);
```

Display the results.

```
figure
I = insertObjectAnnotation(I,'Rectangle',bboxA,labelsA);
J = insertObjectAnnotation(J,'Rectangle',bboxB,labelsB);
imshowpair(I,J,'montage')
```



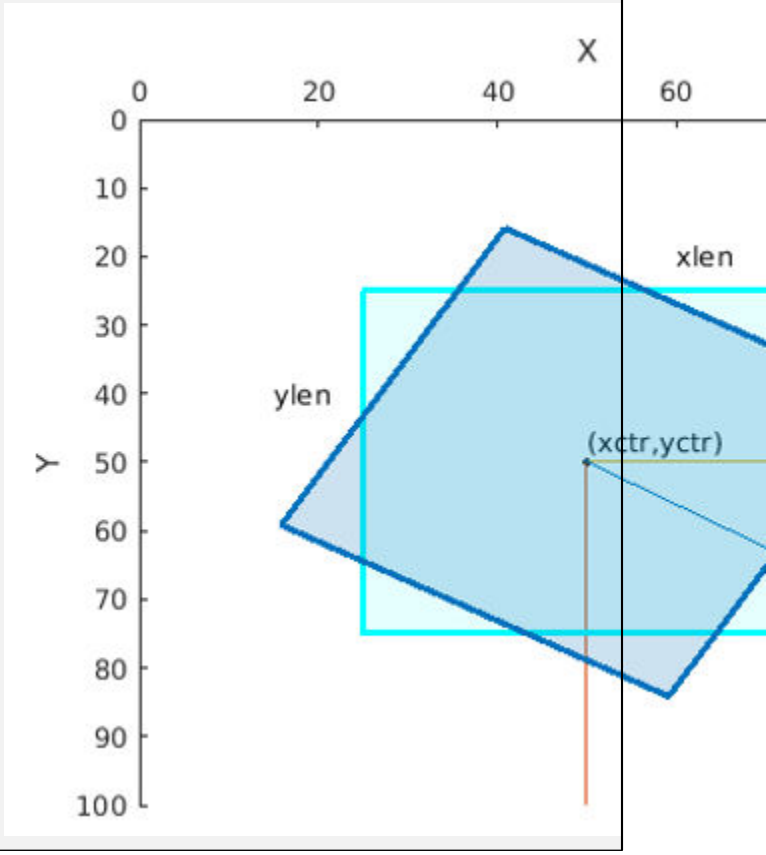
Input Arguments

bboxA — Bounding boxes

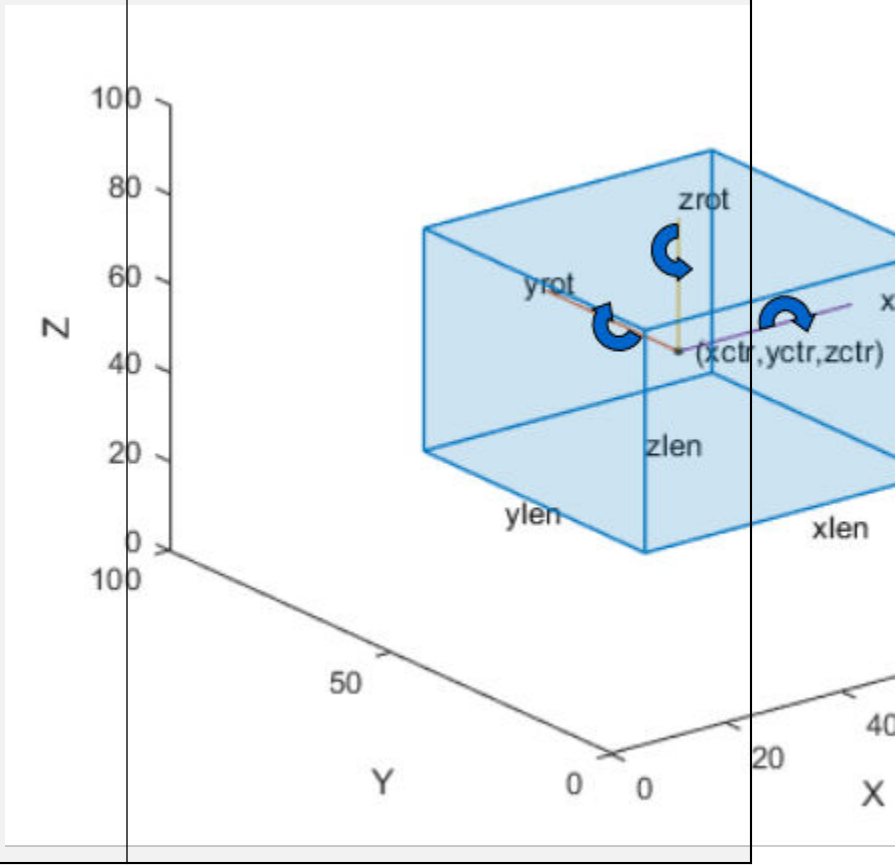
M-by-4 matrix | *M*-by-5 matrix | *M*-by-9 matrix

Bounding boxes, specified as an *M*-by-4, *M*-by-5, or *M*-by-9 nonsparse numeric matrix. *M* is the number of bounding boxes. Each row of the matrix defines a bounding box as either an axis-aligned rectangle, a rotated rectangle, or a cuboid. This table describes the format for each bounding box.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

Bounding Box	Description
Cuboid	Defined in spatial coordinates as an *M*-by-9 numeric matrix with rows of the form [*xctr yctr zctr xlen ylen zlen xrot yrot zrot*], where: • *M* is the number of cuboids. • *xctr*, *yctr*, and *zctr* specify the center of the cuboid. • *xlen*, *ylen*, and *zlen* specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation. • *xrot*, *yrot*, and *zrot* specify the rotation angles of the cuboid around the *x*-axis, *y*-axis, and *z*-axis, respectively. The *xrot*, *yrot*, and *zrot* rotation angles are in degrees about the cuboid center. Each rotation is clockwise-positive with respect to the positive direction of the associated spatial axis. The function computes rotation matrices assuming ZYX order Euler angles [*xrot yrot zrot*]. The figure shows how these values determine the position of a cuboid.



`window` — Cropping window
four-element vector (default) | `images.spatialref.Rectangle` | six-element vector | `images.spatialref.Cuboid` | numeric | image

Cropping window, specified as a four-element vector, a six-element vector, or a `Cuboid` object. The output bounding box positions are relative to the location of the cropping window.

When you specify `bboxA` as a rectangular input, the cropping window must be a four-element vector in the format [*x*,*y*,*width*,*height*], or a `Rectangle` object.

When you specify `bboxA` as a cuboid, the cropping window must be a six-element vector in the format [*x*,*y*,*zwidth*,*height*,*depth*], or a `Cuboid` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `OverlapThreshold=1`, sets the positive overlap threshold to `1`.

`OverlapThreshold` — Overlap threshold
`1` (default) | positive scalar less than or equal to `1`

Overlap threshold, specified as a positive scalar less than or equal to `1`. The amount of overlap between the input boxes, `bboxA`, and the cropping area, `window`, is defined as:

`area`(`intersect` (`bboxA`,`window`))/`area`(`union`(`bboxB`,`window`)).

If the computed overlap value is greater than the value of the `threshold` property, then the boxes are clipped to the bounding rectangle border. Otherwise, the boxes are discarded. Lowering the threshold can result in parts of the object getting discarded.

Note For rotated rectangle bounding boxes, the function only supports the overlap threshold value of 1.

Output Arguments

`bboxB` — Cropped bounding boxes
M2-by-*N* matrix | nonsparse numeric

Cropped bounding boxes, returned as an *M2*-by-*N* matrix of *M2* bounding boxes. The number of bounding boxes returned is less than the number of bounding boxes in the input. Each row, *M2*, of the matrix defines one bounding box of the same type as the input `bboxA`. The output bounding box positions are relative to the location of the cropping window.

`indices` — Indices
vector of integers

Indices, returned as a vector of integers. The indices indicate which bounding boxes in the input, `bboxA`, are within the cropping window.

Version History

Introduced in R2019b

R2022a: Bounding Box Coordinates: Data augmentation for object detection using spatial coordinates

Behavior changed in R2022a

The `bboxresize`, `bboxcrop`, `bboxwarp`, and `showShape` functions assume the input bounding box coordinates for axis-aligned rectangles are specified in spatial coordinates and return the transformed bounding boxes in spatial coordinates.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”

See Also

`imcrop` | `bboxresize` | `bboxwarp` | `bboxerase`

bboxresize

Resize bounding boxes

Syntax

```
bboxB = bboxresize(bboxA,scale)
```

Description

`bboxB = bboxresize(bboxA,scale)` resizes bounding boxes in `bboxA` by the amount specified by `scale`. This function supports 2-D and 3-D bounding boxes.

Examples

Resize Image and Corresponding Bounding Boxes

Read an image.

```
I = imread('peppers.png');
```

Define bounding boxes and labels.

```
bboxA = [
    410 230 100 90
    186 78  80  60
    ]
```

```
bboxA = 2×4

   410   230   100    90
   186    78    80    60
```

```
labelsA = [
    "garlic"
    "onion"
    ];
```

Resize the image and the bounding boxes.

```
scale = 1.5;
J = imresize(I,scale);
bboxB = bboxresize(bboxA,scale);
```

Display the results.

```
figure
I = insertObjectAnnotation(I,'Rectangle',bboxA,labelsA);
J = insertObjectAnnotation(J,'Rectangle',bboxB,labelsA);
imshowpair(I,J,'montage')
```



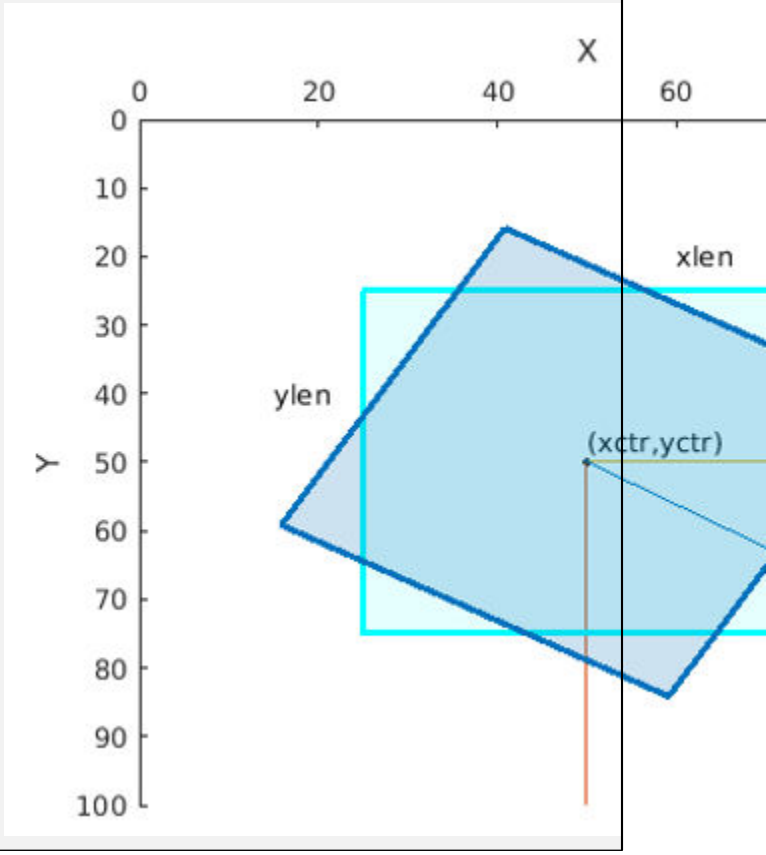
Input Arguments

bboxA — Bounding boxes

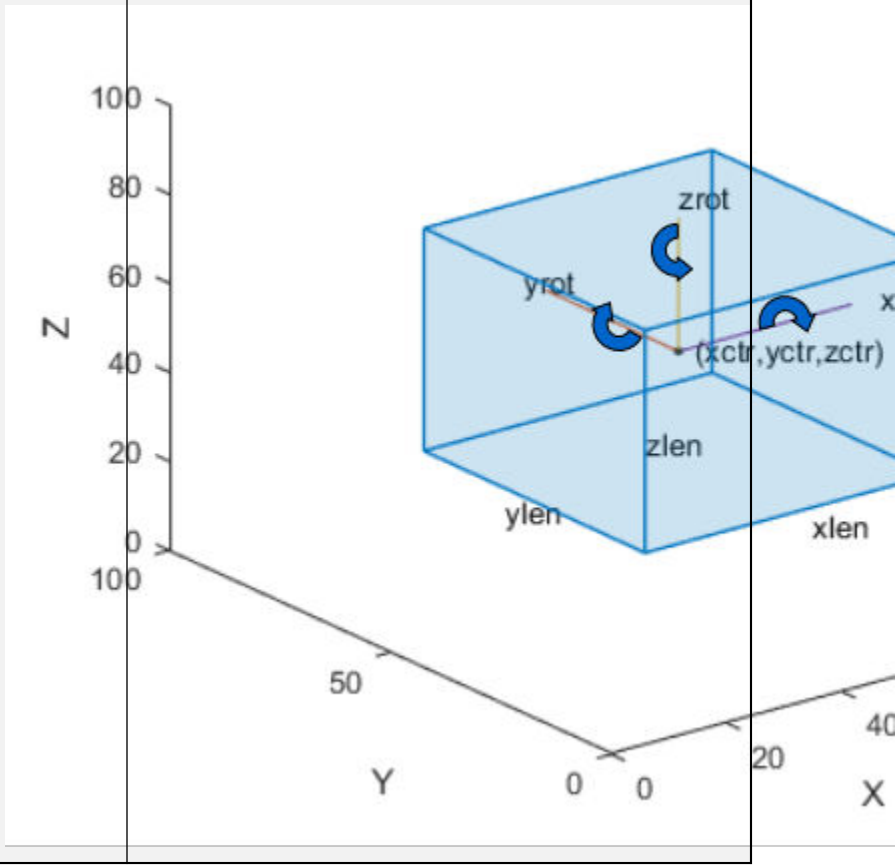
M-by-4 matrix | *M*-by-5 matrix | *M*-by-9 matrix | nonsparse numeric

Bounding boxes, specified as an *M*-by-4, *M*-by-5, or *M*-by-9 nonsparse numeric matrix of *M* bounding boxes. Each row, *M*, of the matrix defines a bounding box as either an axis-aligned rectangle, a rotate rectangle, or a cuboid. The table below describes the format of the bounding boxes.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

Bounding Box	Description
Cuboid	Defined in spatial coordinates as an *M*-by-9 numeric matrix with rows of the form [*xctr yctr zctr xlen ylen zlen xrot yrot zrot*], where: • *M* is the number of cuboids. • *xctr*, *yctr*, and *zctr* specify the center of the cuboid. • *xlen*, *ylen*, and *zlen* specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation. • *xrot*, *yrot*, and *zrot* specify the rotation angles of the cuboid around the *x*-axis, *y*-axis, and *z*-axis, respectively. The *xrot*, *yrot*, and *zrot* rotation angles are in degrees about the cuboid center. Each rotation is clockwise-positive with respect to the positive direction of the associated spatial axis. The function computes rotation matrices assuming ZYX order Euler angles [*xrot yrot zrot*]. The figure shows how these values determine the position of a cuboid.



`scale` — Scale

scalar (default) | row vector

Scale, specified as a scalar or a row vector. When you specify a scalar, the function applies the same scale factor to the height and width of the bounding boxes in `bboxA`. When you specify a row vector, the function applies the factor in the first element of the vector to resize the height and the second element to resize the width of the bounding boxes.

Output Arguments

`bboxB` — Warped bounding boxes

M2-by-*N* matrix | nonsparse numeric

Warped bounding boxes, returned as an *M2*-by-*N* matrix of *M2* bounding boxes. The number of bounding boxes returned is less than the number of bounding boxes in the input. Each row, *M2*, of the matrix defines one bounding box of the same type as the input `bboxA`. When `bboxB` contains floating point data, the function returns it with the same type as `bboxA`. Otherwise, the function returns `bboxB` as type `single`.

Version History

Introduced in R2019b

R2022a: Bounding Box Coordinates: Data augmentation for object detection using spatial coordinates

Behavior changed in R2022a

The `bboxresize`, `bboxcrop`, `bboxwarp`, and `showShape` functions assume the input bounding box coordinates for axis-aligned rectangles are specified in spatial coordinates and return the transformed bounding boxes in spatial coordinates.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”

See Also

`bboxcrop` | `bboxwarp` | `bboxerase` | `imresize`

bboxwarp

Apply geometric transformation to bounding boxes

Syntax

```
bboxB = bboxwarp(bboxA,tform,ref)
bboxB = bboxwarp(bboxA,tform,ref,"OverlapThreshold",overlapThreshold)
[bboxB,indices] = bboxwarp( ___ )
```

Description

`bboxB = bboxwarp(bboxA,tform,ref)` transforms bounding boxes in `bboxA` according to the geometric transformation defined by `tform`. Bounding boxes can be axis-aligned rectangles, rotated rectangles, or cuboids. The spatial reference object, `ref`, defines the output view into which the boxes are transformed. This function supports 2-D and 3-D bounding boxes.

`bboxB = bboxwarp(bboxA,tform,ref,"OverlapThreshold",overlapThreshold)` also specifies the positive overlap threshold.

`[bboxB,indices] = bboxwarp( ___ )` additionally returns a vector of indices that indicate which bounding boxes in `bboxA` correspond to the warped versions in the output, `bboxB`. You can specify either of the preceding sets of input arguments.

Examples

Transform Images and Corresponding Bounding Boxes

Read an image.

```
I = imread("peppers.png");
```

Define bounding boxes and labels.

```
bboxA = [
    410 230 100 90
    186 78  80  60
    ]
```

```
bboxA = 2×4

   410   230   100    90
   186    78    80    60
```

```
labelsA = [
    "garlic"
    "onion"
    ];
```

Define an affine transform to horizontally flip and translate the image.

```
tform = affinetform2d([-1 0 50; 0 1 50; 0 0 1]);
```

Create an output view for `imwarp`.

```
rout = affineOutputView(size(I),tform);
```

Warp the image.

```
J = imwarp(I,tform,"OutputView",rout);
```

Warp the boxes.

```
[bboxB,indices] = bboxwarp(bboxA,tform,rout);
labelsB = labelsA(indices);
```

Display the results.

```
annotatedI = insertObjectAnnotation(I,"Rectangle",bboxA,labelsA);
annotatedJ = insertObjectAnnotation(J,"Rectangle",bboxB,labelsB);
figure
montage({annotatedI,annotatedJ})
```



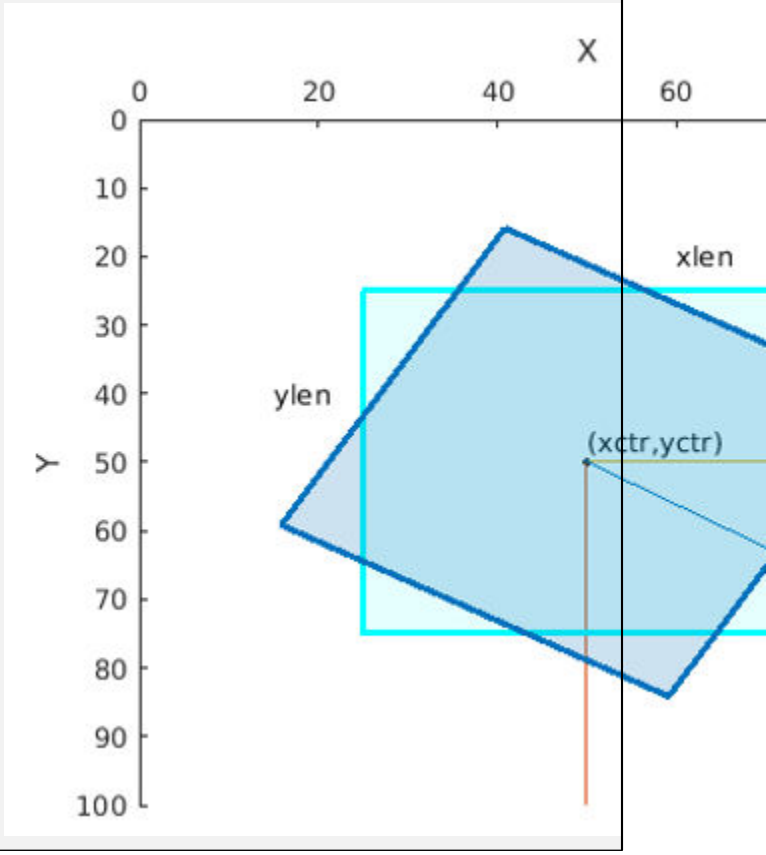
Input Arguments

bboxA — Bounding boxes

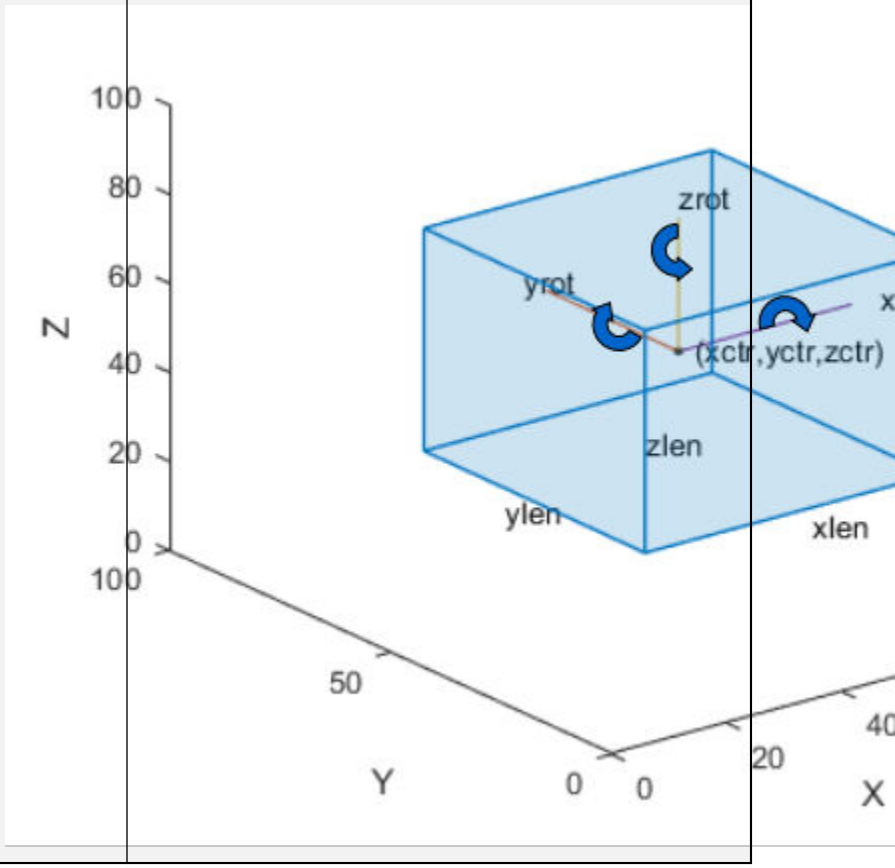
M-by-4 matrix | *M*-by-5 matrix | *M*-by-9 matrix

Bounding boxes, specified as an *M*-by-4, *M*-by-5, or *M*-by-9 nonsparse numeric matrix of *M* bounding boxes. Each row, *M*, of the matrix defines a bounding box as either an axis-aligned rectangle, a rotate rectangle, or a cuboid. The table below describes the format of the bounding boxes.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an M-by-4 numeric matrix with rows of the form [x y w h], where: • M is the number of axis-aligned rectangles. • x and y specify the upper-left corner of the rectangle. • w specifies the width of the rectangle, which is its length along the x-axis. • h specifies the height of the rectangle, which is its length along the y-axis.

Bounding Box	Description
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

Bounding Box	Description
Cuboid	Defined in spatial coordinates as an *M*-by-9 numeric matrix with rows of the form [*xctr yctr zctr xlen ylen zlen xrot yrot zrot*], where: • *M* is the number of cuboids. • *xctr*, *yctr*, and *zctr* specify the center of the cuboid. • *xlen*, *ylen*, and *zlen* specify the length of the cuboid along the *x*-axis, *y*-axis, and *z*-axis, respectively, before rotation. • *xrot*, *yrot*, and *zrot* specify the rotation angles of the cuboid around the *x*-axis, *y*-axis, and *z*-axis, respectively. The *xrot*, *yrot*, and *zrot* rotation angles are in degrees about the cuboid center. Each rotation is clockwise-positive with respect to the positive direction of the associated spatial axis. The function computes rotation matrices assuming ZYX order Euler angles [*xrot yrot zrot*]. The figure shows how these values determine the position of a cuboid.



`tform` — Geometric transformation

`affinetform2d` object | `affinetform3d` object

Geometric transformation, specified as an `affinetform2d` object for rectangular inputs or an `affinetform3d` object for cuboid inputs. The bboxwarp function supports affine transformations consisting only of scale, rotation, and translation.

Note Rotated rectangles do not support reflection or shear transformations.

`ref` — Spatial reference

`imref2d` object | `imref3d` object

Spatial reference, specified as an `imref2d` object for rectangular inputs or `imref3d` object for cuboid inputs. To obtain one of these objects, you can use the `imwarp` or the `affineOutputView` function. The object defines the output view to transform boxes. Boxes that are transformed completely outside of the output view defined by `ref` are discarded.

```
[J,rout] = imwarp(I,tform);
[bboxB,indices] = bboxwarp(bboxA,tform,rout);

rout = affineOutputView(size(I),tform)
J = imwarp(I,tform,OutputView=rout);
[bboxB,indices] = bboxwarp(bboxA,tform,rout);
```

`overlapThreshold` — Overlap threshold

1 (default) | positive scalar less than or equal to 1

Overlap threshold, specified as a positive scalar less than or equal to `1`. The amount of overlap between transformed boxes and the region *W*, defined by the output view, is defined as:

`area`(`intersect`(`bboxA`,*W*))/`area`(`bboxB`,*W)*.

If the computed overlap value is greater than the value of the `threshold` property, then the transformed boxes are clipped to the bounding rectangle border. Otherwise, the boxes are discarded. Lowering the threshold can result in parts of the object getting discarded.

Output Arguments

`bboxB` — Warped bounding boxes

M2-by-*N* matrix

Warped bounding boxes, returned as an *M2*-by-*N* matrix of *M2* bounding boxes. The number of bounding boxes returned is less than the number of bounding boxes in the input. Each row, *M2*, of the matrix defines one bounding box of the same type as the input `bboxA`. When `bboxB` contains floating point data, the function returns it with the same type as `bboxA`. Otherwise, the function returns `bboxB` as type `single`.

`indices` — Indices

vector of integers

Indices, returned as a vector of integers. The indices indicate which bounding boxes in the input, `bboxA`, that correspond to the warped versions in the output, `bboxB`.

Version History

Introduced in R2019b

R2022b: Supports new geometric transformation objects

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify `tform` as an `affinetform2d` or `affinetform3d` geometric transformation object, which use the premultiply convention.

Although you can still specify `tform` as an `affine2d` or `affine3d` geometric transformation object, these objects are not recommended because they use the postmultiply convention. You can streamline your geometric transformation workflows by switching to the new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2022a: Bounding Box Coordinates: Data augmentation for object detection using spatial coordinates

Behavior changed in R2022a

The `bboxresize`, `bboxcrop`, `bboxwarp`, and `showShape` functions assume the input bounding box coordinates for axis-aligned rectangles are specified in spatial coordinates and return the transformed bounding boxes in spatial coordinates.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”

See Also

`imcrop` | `bboxcrop` | `bboxresize` | `imwarp` | `bboxerase`

Topics

“Get Started with Object Detection Using Deep Learning”
“Augment Bounding Boxes for Object Detection”

bboxOverlapRatio

Compute bounding box overlap ratio

Syntax

```
overlapRatio = bboxOverlapRatio(bboxA,bboxB)
overlapRatio = bboxOverlapRatio(bboxA,bboxB,ratioType)
```

Description

`overlapRatio = bboxOverlapRatio(bboxA,bboxB)` returns the overlap ratio between each pair of bounding boxes `bboxA` and `bboxB`. The function returns the `overlapRatio` value between `0` and `1`, where `1` implies a perfect overlap.

`overlapRatio = bboxOverlapRatio(bboxA,bboxB,ratioType)` additionally lets you specify the method to use for computing the ratio. You must set the `ratioType` to either `'Union'` or `'Min'`.

Examples

Compute the Overlap Ratio Between Two Bounding Boxes

Define two bounding boxes in the format [x y width height].

```
bboxA = [150,80,100,100];
bboxB = bboxA + 50;
```

Display the bounding boxes on an image.

```
I = imread('peppers.png');
RGB = insertShape(I,'filled-rectangle',bboxA,'ShapeColor','green');
RGB = insertShape(RGB,'filled-rectangle',bboxB,'ShapeColor','yellow');
imshow(RGB)
```



Compute the overlap ratio between the two bounding boxes.

```
overlapRatio = bboxOverlapRatio(bboxA,bboxB)
```

```
overlapRatio =
0.0833
```

Compute Overlap Ratio Between Each Pair of Bounding Boxes

Randomly generate two sets of bounding boxes.

```
bboxA = 10*rand(5,4);
bboxB = 10*rand(10,4);
```

Ensure that the width and height of the boxes are positive.

```
bboxA(:,3:4) = bboxA(:,3:4) + 10;
bboxB(:,3:4) = bboxB(:,3:4) + 10;
```

Compute the overlap ratio between each pair.

```
overlapRatio = bboxOverlapRatio(bboxA,bboxB)
```

```
overlapRatio = 5×10

    0.2431    0.2329    0.3418    0.5117    0.7972    0.1567    0.1789    0.4339    0.0906    0.5
    0.3420    0.1655    0.7375    0.5188    0.2786    0.3050    0.2969    0.4350    0.2477    0.2
    0.4844    0.3290    0.3448    0.1500    0.1854    0.4976    0.5629    0.4430    0.5027    0.2
    0.3681    0.0825    0.3499    0.0840    0.0658    0.5921    0.6498    0.1930    0.7433    0.0
    0.3752    0.1114    0.3114    0.0696    0.0654    0.5408    0.6234    0.2046    0.7557    0.0
```

Input Arguments

bboxA — Bounding box

M-by-4 matrix | *M*-by-5 matrix

Bounding boxes, specified as an *M*-by-4 or *M*-by-5 nonsparse numeric matrix. *M* is the number of bounding boxes. Each row of the matrix defines a bounding box as either an axis-aligned rectangle or a rotated rectangle. This table describes the format for each bounding box.

Bounding Box	Row	Description
Axis-aligned rectangle	[x_{min}, y_{min}, *width*, *height*]	This type of bounding box is defined in pixel coordinates as an *M*-by-4 matrix representing *M* bounding boxes
Rotated rectangle	[x_{center}, y_{center}, *width*, *height*, *yaw*]	This type of bounding box is defined in spatial coordinates as an *M*-by-5 matrix representing *M* bounding boxes. The x_{center} and y_{center} coordinates represent the center of the bounding box. The *width* and *height* elements represent the length of the box along the *x* and *y* axes, respectively. The *yaw* represents the rotation angle in degrees. The amount of rotation about the center of the bounding box is measured in the clockwise direction.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

bboxB — Bounding box

M-by-4 matrix | *M*-by-5 matrix

Bounding boxes, specified as an *M*-by-4 or *M*-by-5 nonsparse numeric matrix. *M* is the number of bounding boxes. Each row, *M*, of the matrix defines a bounding box as either an axis-aligned rectangle or a rotated rectangle. This table describes the format for each bounding box.

Bounding Box	Row	Description
Axis-aligned rectangle	[x_{min}, y_{min}, *width*, *height*]	This type of bounding box is defined in pixel coordinates as an *M*-by-4 matrix representing *M* bounding boxes

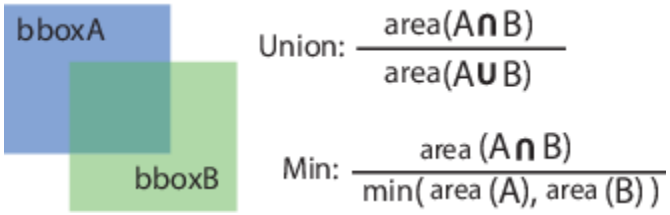
Bounding Box	Row	Description
Rotated rectangle	[x_{center}, y_{center}, *width*, *height*, *yaw*]	This type of bounding box is defined in spatial coordinates as an *M*-by-5 matrix representing *M* bounding boxes. The x_{center} and y_{center} coordinates represent the center of the bounding box. The *width* and *height* elements represent the length of the box along the *x* and *y* axes, respectively. The *yaw* represents the rotation angle in degrees. The amount of rotation about the center of the bounding box is measured in the clockwise direction.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`ratioType` — Ratio type
`'Union'` (default) | `'Min'`

Ratio type, specified as the character vector `'Union'` or `'Min'`.

- Set the ratio type to `'Union'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the area of the union of the two.
- Set the ratio type to `'Min'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the minimum area of the two bounding boxes.



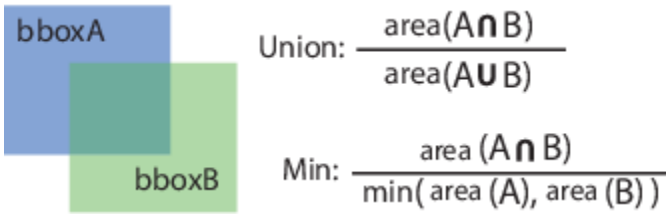
Data Types: `char`

Output Arguments

`overlapRatio` — Overlap ratio between two bounding boxes
M-by-*N* matrix

Overlap ratio between two bounding boxes, returned as an *M*-by-*N* matrix. Each (*I*, *J*) element in the output matrix corresponds to the overlap ratio between row *I* in `bboxA` and row *J* in `bboxB`. The function returns `overlapRatio` in the between 0 and 1, where `1` implies a perfect overlap. If either `bboxA` or `bboxB` is double, then the function returns `overlapRatio` as `double`. Otherwise, the function returns it as `single`.

The function computes the overlap ratio based on the ratio type. You can set `ratioType` to `'Union'` or `'Min'`:



Data Types: `single` | `double`

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`selectStrongestBbox` | `selectStrongestBboxMulticlass` | `bboxPrecisionRecall`

Topics

“Multiple Object Tracking”

bboxPrecisionRecall

Compute bounding box precision and recall against ground truth

Syntax

```
[precision,recall] = bboxPrecisionRecall(bboxes,groundTruthBboxes)
[precision,recall] = bboxPrecisionRecall(bboxes,groundTruthBboxes,threshold)
```

Description

`[precision,recall] = bboxPrecisionRecall(bboxes,groundTruthBboxes)` measures the accuracy of bounding box overlap between `bboxes` and `groundTruthBboxes`. Precision is a ratio of true positive instances to all positive instances of objects in the detector, based on the ground truth. Recall is a ratio of true positive instances to the sum of true positives and false negatives in the detector, based on the ground truth.

If the bounding box is associated with a class label, `precision` and `recall` contain metrics for each class. If the bounding box is also associated with a confidence score for ranking, use the `evaluateObjectDetection` function.

`[precision,recall] = bboxPrecisionRecall(bboxes,groundTruthBboxes,threshold)` specifies the overlap threshold for assigning a given box to a ground truth box.

Examples

Evaluate Bounding Box Overlap Accuracy

Create two ground truth boxes.

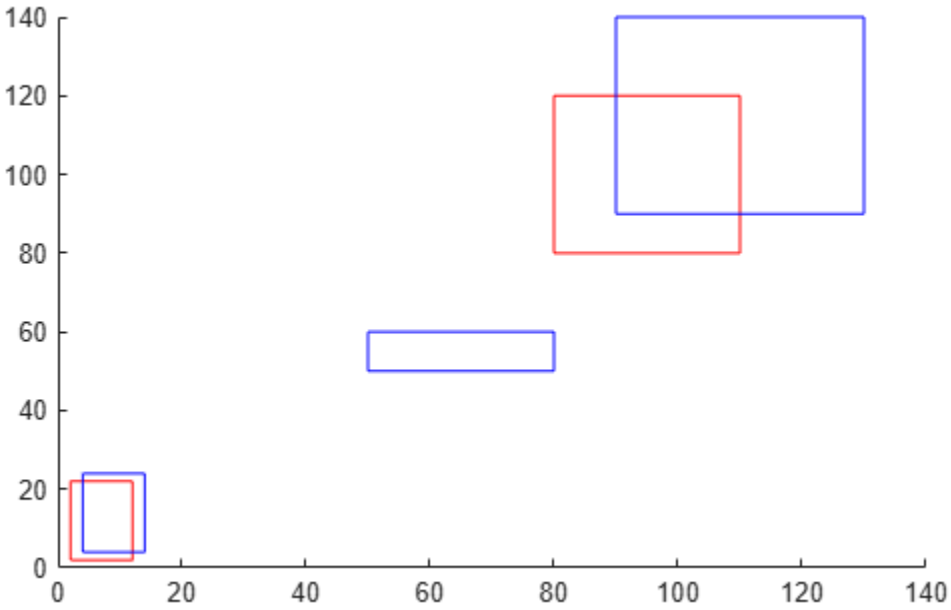
```
groundTruthBoxes = [2 2 10 20; 80 80 30 40];
```

Create three boxes for evaluation.

```
boundingBoxes = [4 4 10 20; 50 50 30 10; 90 90 40 50];
```

Plot the boxes.

```
figure
hold on
for i=1:2
    rectangle('Position',groundTruthBoxes(i,:),'EdgeColor','r');
end
for i=1:3
    rectangle('Position',boundingBoxes(i,:),'EdgeColor','b');
end
```



Evaluate the overlap accuracy against the ground truth data.

```
[precision,recall] = bboxPrecisionRecall(boundingBoxes,groundTruthBoxes)
```

```
precision =
0.3333

recall =
0.5000
```

Evaluate Bounding Box Overlap for Three Classes

Define class names.

```
classNames = ["A","B","C"];
```

Create bounding boxes for evaluation.

```
predictedLabels = {...
    categorical("A",classNames); ...
    categorical(["C";"B"],classNames)};
bboxes = {...
    [10 10 20 30]; ...
    [60 18 20 10; 120 120 5 10]};
boundingBoxes = table(bboxes,predictedLabels,'VariableNames',...
    {'PredictedBoxes','PredictedLabels'});
```

Create ground truth boxes.

```
A = {[10 10 20 28]; []};
B = {[]; [118 120 5 10]};
```

```
C = {[]; [59 19 20 10]};
groundTruthData = table(A,B,C);
```

Evaluate overlap accuracy against ground truth data.

```
[precision,recall] = bboxPrecisionRecall(boundingBoxes,groundTruthData)
```

```
precision = 3×1

     1
     0
     1


recall = 3×1

     1
     0
     1
```

Input Arguments

bboxes — Bounding boxes

M-by-4 matrix | table with *M* rows

Bounding boxes, specified as one of the following. *M* is the number of bounding boxes.

- For single-class bounding boxes, `bboxes` can be an *M*-by-4 matrix, or a table with *M* rows and one column. Each row of the matrix or element in the table represents a bounding box, specified in the format [*x y width height*], where *x* and *y* correspond to the upper left corner of the bounding box.
- For multi-class bounding boxes, `bboxes` is a table with *M* rows and two columns. Each element in the first column represents a bounding box, specified in the format [*x y width height*]. The second column contains the predicted label for each box. The label must be a categorical type defined by the variable (column) names of the `groundTruthBboxes` table.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

groundTruthBboxes — Ground truth bounding boxes

M-by-4 matrix | table with *M* rows

Ground truth bounding boxes, specified as one of the following. *M* is the number of ground truth bounding boxes.

- For single-class bounding boxes, `groundTruthBboxes` can be an *M*-by-4 matrix, or a table with *M* rows and one column. Each row of the matrix or element in the table represents a bounding box, specified in the format [*x y width height*], where *x* and *y* correspond to the upper left corner of the bounding box.
- For multi-class bounding boxes, `groundTruthBboxes` is a table with *M* rows and multiple columns. Each column represents a different class, and the column name specifies the class label. Each element in the table has the format [*x y width height*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`threshold` — Overlap threshold
`0.5` | numeric scalar

Overlap threshold for assigned a detection to a ground truth box, specified as a numeric scalar. The overlap ratio is computed as the IoU (intersection over union).

Output Arguments

`precision` — Precision
numeric scalar | numeric vector

Precision, returned as a numeric scalar for single-class bounding boxes, or a numeric vector consisting of metrics for each class for multi-class bounding boxes. The class order follows the same column order as the `groundTruthBboxes` table.

`recall` — Recall
numeric scalar | numeric vector

Recall, returned as a numeric scalar for single-class bounding boxes, or a numeric vector consisting of metrics for each class for multi-class bounding boxes. The class order follows the same column order as the `groundTruthBboxes` table.

Version History

Introduced in R2018a

See Also

`bboxOverlapRatio` | `evaluateObjectDetection`

bundleAdjustment

Adjust collection of 3-D points and camera poses

Syntax

```
[xyzRefinedPoints,refinedPoses] = bundleAdjustment(xyzPoints,pointTracks,
cameraPoses,intrinsics)
[wpSetRefined,vSetRefined,pointIndex] = bundleAdjustment(wpSet,vSet,viewIDs,
intrinsics)
[___,reprojectionErrors] = bundleAdjustment(___)
[___] = bundleAdjustment(___,Name=Value)
```

Description

`[xyzRefinedPoints,refinedPoses] = bundleAdjustment(xyzPoints,pointTracks, cameraPoses,intrinsics)` refines 3-D points and camera poses to minimize reprojection errors. The refinement procedure is a variant of the Levenberg-Marquardt algorithm. The function uses the same global reference coordinate system to return both the 3-D points and camera poses.

`[wpSetRefined,vSetRefined,pointIndex] = bundleAdjustment(wpSet,vSet,viewIDs, intrinsics)` refines 3-D points from the world point set, `wpSet`, and refines camera poses from the image view set, `vSet`. `viewIDs` specify the camera poses in `vSet` to refine.

`[___,reprojectionErrors] = bundleAdjustment(___)` returns the mean reprojection error for each 3-D world point, in addition to the arguments from the previous syntax.

`[___] = bundleAdjustment(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MaxIterations=50` sets the number of iterations to `50`. Unspecified arguments have default values.

Examples

Refine 3-D Points and Camera Poses from Single Camera

Load data for initialization.

```
data = load("globeBA.mat");
```

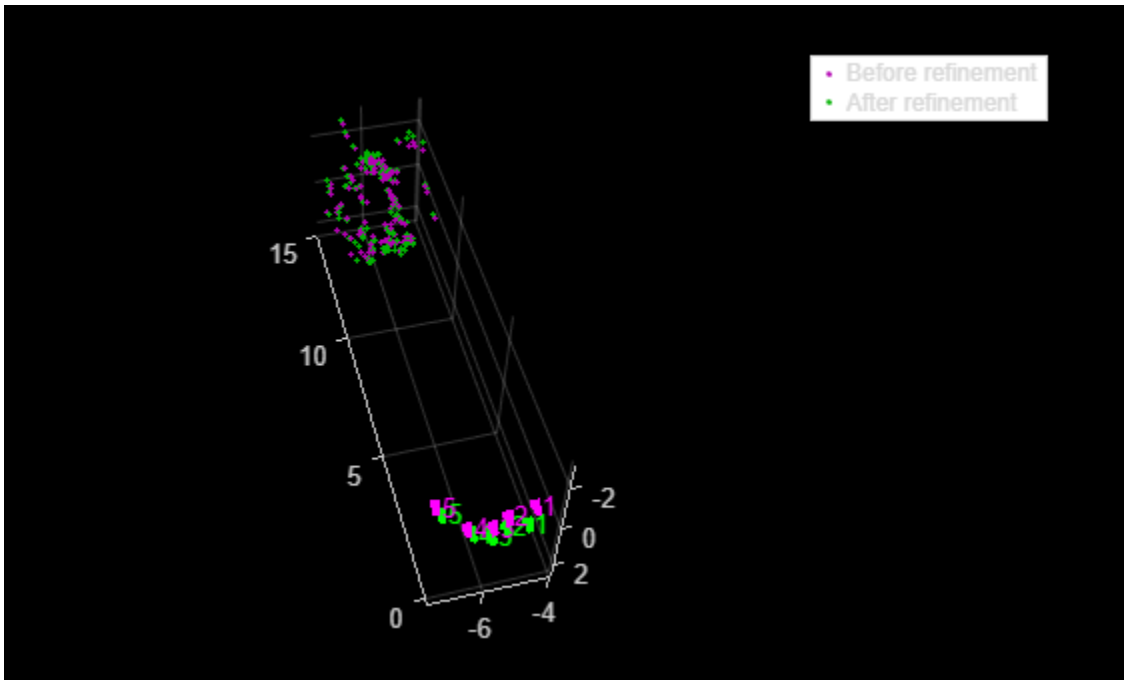
Refine the camera poses and points.

```
[xyzRefinedPoints,refinedPoses] = ...
    bundleAdjustment(data.xyzPoints,data.pointTracks,data.cameraPoses,data.intrinsics);
```

Display the 3-D points and camera poses before and after the refinement.

```
pcshowpair(pointCloud(data.xyzPoints), pointCloud(xyzRefinedPoints), ...
    AxesVisibility="on", VerticalAxis="y", VerticalAxisDir="down", MarkerSize=40);
hold on
plotCamera(data.cameraPoses, Size=0.1, Color="m");
```

```
plotCamera(refinedPoses, Size=0.1, Color="g");
legend("Before refinement", "After refinement", color="w");
```



Input Arguments

`xyzPoints` — Unrefined 3-D points

M-by-3 matrix

Unrefined 3-D points, specified as an *M*-by-3 matrix of [*x* *y* *z*] locations.

`pointTracks` — Matching points across multiple images

N-element array of `pointTrack` objects

Matching points across multiple images, specified as an *N*-element array of `pointTrack` objects. Each element contains two or more matching points across multiple images.

`cameraPoses` — Camera pose information

two-column table

Camera pose information, specified as a two-column table with columns `ViewId` and `AbsolutePose`. The view IDs relate to the IDs of the objects in the `pointTracks` argument. You can use the `poses` object function to obtain the `cameraPoses` table.

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | *N*-element array of `cameraIntrinsics` objects

Camera intrinsics, specified as a `cameraIntrinsics` object or an *N*-element array of `cameraIntrinsics` objects. *N* is the number of camera poses or the number of IDs in `viewIDs`. Use a single `cameraIntrinsics` object when images are captured using the same camera. Use a vector `cameraIntrinsics` objects when images are captured by different cameras.

wpSet — 3-D world points
`worldpointset` object

3-D world points, specified as a `worldpointset` object.

vSet — Camera poses
`imageviewset` object

Camera poses, specified as an `imageviewset` object.

viewIDs — View identifiers
N-element array

View identifiers, specified as an *N*-element array. The `viewIDs` represent which camera poses to refine specifying their related views in `imageviewset`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxIterations=50` sets the number of iterations to `50`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

MaxIterations — Maximum number of iterations
`50` (default) | positive integer

Maximum number of iterations before the Levenberg-Marquardt algorithm stops, specified as a positive integer.

AbsoluteTolerance — Absolute termination tolerance
`1.0` (default) | positive scalar

Absolute termination tolerance of the mean squared reprojection error in pixels, specified as positive scalar.

RelativeTolerance — Relative termination tolerance
`1e-5` (default) | positive scalar

Relative termination tolerance of the reduction in reprojection error between iterations, specified as positive scalar.

PointsUndistorted — Flag to indicate lens distortion
`false` (default) | `true`

Flag to indicate lens distortion, specified as `false` or `true`. When you set `PointsUndistorted` to `false`, the 2-D points in `pointTracks` or in `vSet` must be from images with lens distortion. To use undistorted points, first use the `undistortImage` function to remove distortions from the images, then set `PointsUndistorted`.

FixedViewIDs — View IDs for fixed camera pose
`[]` (default) | vector of nonnegative integers

View IDs for fixed camera pose, specified as a vector of nonnegative integers. Each ID corresponds to the `ViewId` of a fixed camera pose in `cameraPoses`. An empty value for `FixedViewIDs` means that all camera poses are optimized.

`Solver` — Solver
`"sparse-linear-algebra"` (default) | `"preconditioned-conjugate-gradient"`

Solver, specified as `"sparse-linear-algebra"` or `"preconditioned-conjugate-gradient"`. Use the `"sparse-linear-algebra"` solver for low sparsity images. Low sparsity indicates that many camera views observe some of the same world points. Use the `"preconditioned-conjugate-gradient"` (PCG) solver, from the general graphic optimization (g2o) library, for high sparsity images. High sparsity indicates that each camera view observes, only a small portion of the world points, specified by `xyzPoints`.

`Verbose` — Display progress information
`false` (default) | `true`

Display progress information, specified as `false` or `true`.

Output Arguments

`xyzRefinedPoints` — 3-D locations of refined world points
M-by-3 matrix

3-D locations of refined world points, returned as an *M*-by-3 matrix of [*x y z*] locations.

Data Types: `single` | `double`

`refinedPoses` — Refined camera poses
two-column table

Refined camera poses, returned as a two-column table. The table contains columns for `ViewId` and `AbsolutePose`.

`wpSetRefined` — Refined 3-D world points
`worldpointset` object

Refined 3-D world points, returned as a `worldpointset` object.

`vSetRefined` — Refined camera poses
`imageviewset` object

Refined camera poses, specified as an `imageviewset` object.

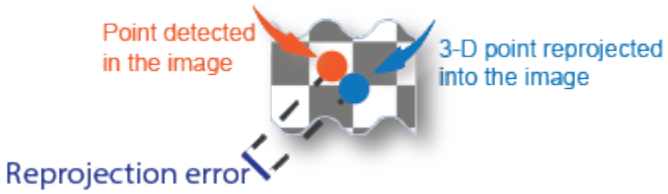
`pointIndex` — Indices of refined 3-D world points
M-element array

Indices to `wpSetRefined` of refined 3-D world points, returned as an *M*-element array.

`reprojectionErrors` — Reprojection errors
M-element vector

Reprojection errors, returned as an *M*-element vector. The function projects each world point back into each camera. Then, in each image, the function calculates the reprojection error as the distance

between the detected and the reprojected point. The `reprojectionErrors` vector contains the average reprojection error for each world point.



Version History

Introduced in R2016a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. The `bundleAdjustment` function now supports both the premultiply and the postmultiply convention.

Regardless of the convention, the `cameraPoses` argument must be a two-column table with variables `ViewId` and `AbsolutePose`. You can no longer specify the `cameraPoses` argument as a three-column table with variables `ViewID`, `Orientation`, and `Location`. The `bundleAdjustment` function chooses the convention according to the value of the `AbsolutePose` variable:

- When you specify `AbsolutePose` as a `rigidtform3d` object, the `bundleAdjustment` function uses the premultiply convention and returns `refinedPoses` with the `AbsolutePose` variable as a `rigidtform3d` object.
- When you specify `AbsolutePose` as a `rigid3d` object, the `bundleAdjustment` function uses the postmultiply convention and returns `refinedPoses` with the `AbsolutePose` variable as a `rigid3d` object.

Before, the function returned `refinedPoses` as a three-column table with variables `ViewID`, `Orientation`, and `Location`. If you need to obtain the values of `Orientation` and `Location`, which were in the postmultiply convention, then you can query properties of the geometric transformation object stored in the `AbsolutePose` variable:

- When `AbsolutePose` returns a `rigidtform3d` object, you can query the `R` and `Translation` properties of the object. The value of `R` is the transpose of `Orientation`.
- When `AbsolutePose` returns a `rigid3d` object, you can query the `Rotation` and `Translation` properties of the object. The value of `Rotation` is identical to `Orientation`.

You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Lourakis, Manolis I. A., and Antonis A. Argyros. "SBA: A Software Package for Generic Sparse Bundle Adjustment." *ACM Transactions on Mathematical Software 36*, no. 1 (March 2009): 2:1–2:30.

[2] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[3] Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle Adjustment — A Modern Synthesis." In *Proceedings of the International Workshop on Vision Algorithms*, 298–372. Springer-Verlag, 1999.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports code generation in only host target platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`bundleAdjustmentMotion` | `bundleAdjustmentStructure` | `estrelpose` | `undistortImage` | `undistortPoints` | `cameraProjection` | `triangulateMultiview`

Objects

`cameraIntrinsics` | `pointTrack` | `cameraParameters` | `worldpointset` | `imageviewset`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Structure from Motion?”

bundleAdjustmentMotion

Adjust collection of 3-D points and camera poses using motion-only bundle adjustment

Syntax

```
refinedPose = bundleAdjustmentMotion(xyzPoints,imagePoints,absolutePose,
intrinsics)
[refinedPose,reprojectionErrors] = bundleAdjustmentMotion( ___ )
[ ___ ] = bundleAdjustmentMotion( ___ ,Name,Value)
```

Description

`refinedPose = bundleAdjustmentMotion(xyzPoints,imagePoints,absolutePose, intrinsics)` returns the refined absolute camera pose that minimizes reprojection errors.

The motion-only refinement procedure is a special case of the Levenberg-Marquardt algorithm for bundle adjustment with 3-D points fixed during optimization. The 3-D points and the camera pose are placed in the same world coordinate system.

`[refinedPose,reprojectionErrors] = bundleAdjustmentMotion( ___ )` additionally returns an *N*-element vector containing the mean reprojection error for each 3-D world point using the arguments from the previous syntax.

`[ ___ ] = bundleAdjustmentMotion( ___ ,Name,Value)` uses additional options specified by one or more name-value arguments. Unspecified arguments have default values.

Examples

Refine Absolute Camera Pose

Load data for initialization into the workspace.

```
data = load("globeMotionOnlyBA.mat");
```

Refine the absolute camera poses.

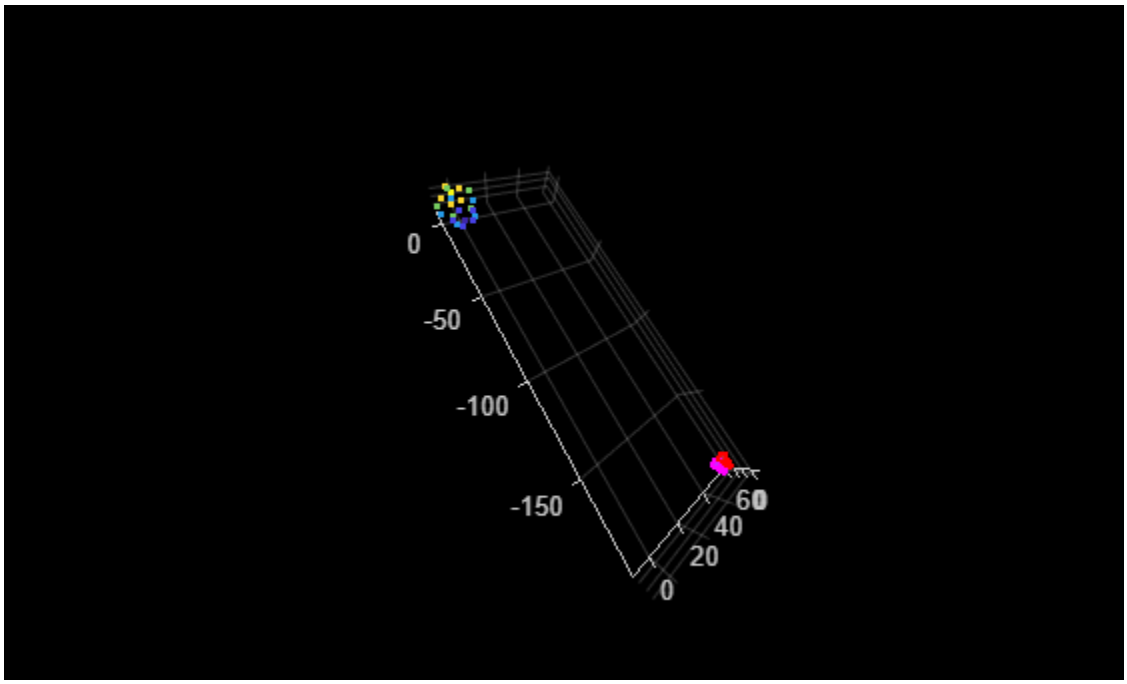
```
refinedPose = bundleAdjustmentMotion(data.xyzPoints,data.imagePoints,data.absPose,data.intrinsics
```

Display the 3-D world points.

```
pcshow(data.xyzPoints,AxesVisibility="on",VerticalAxis="y",VerticalAxisDir="down",MarkerSize=45)
hold on
```

Plot the absolute camera poses before and after refinement.

```
plotCamera(AbsolutePose=data.absPose,Color="r",Size=2);
plotCamera(AbsolutePose=refinedPose,Color="m",Size=2);
```



Input Arguments

`xyzPoints` — Unrefined 3-D points
M-by-3 matrix

Unrefined 3-D points, specified as an *M*-by-3 matrix of [*x*,*y*,*z*] locations.

Data Types: `single` | `double`

`imagePoints` — Image points
M-by-2 matrix | *M*-element feature point array

Image points, specified as an *M*-by-2 matrix or an *M*-element "Point Feature Types" array.

`absolutePose` — Absolute camera pose
`rigidtform3d` object

Absolute camera pose, specified as a `rigidtform3d` object.

`intrinsics` — Camera intrinsics
`cameraIntrinsics` object

Camera intrinsics, specified as a `cameraIntrinsics` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"MaxIterations",50`

`MaxIterations` — Maximum number of iterations
`50` (default) | positive integer

Maximum number of iterations before the Levenberg-Marquardt algorithm stops, specified as a positive integer.

`AbsoluteTolerance` — Absolute termination tolerance
`1.0` (default) | positive scalar

Absolute termination tolerance of the mean squared reprojection error in pixels, specified as a positive scalar.

`RelativeTolerance` — Relative termination tolerance
`1e-5` (default) | positive scalar

Relative termination tolerance of the reduction in reprojection error between iterations, specified as a positive scalar.

`PointsUndistorted` — Flag to indicate lens distortion
`false` (default) | `true`

Flag to indicate lens distortion, specified as `false` or `true`. When you set `PointsUndistorted` to `false`, the 2-D points in `pointTracks` must be from images with lens distortion. To use undistorted points, use the `undistortImage` function first, then set `PointsUndistorted` to `true`.

`Verbose` — Display progress information
`False` (default) | `true`

Display progress information, specified as `false` or `true`.

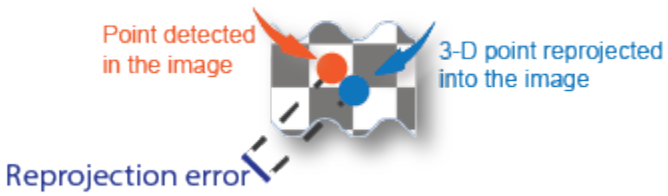
Output Arguments

`refinedPose` — Refined absolute pose
`rigidtform3d` object

Refined absolute pose of the camera, returned as a `rigidtform3d` object.

`reprojectionErrors` — Reprojection errors
M-element vector

Reprojection errors, returned as an *M*-element vector. The function projects each world point back into each camera. Then in each image, the function calculates the reprojection error as the distance between the detected and the reprojected point. The `reprojectionErrors` vector contains the average reprojection error for each world point.



Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `bundleAdjustmentMotion` function now supports specifying `absolutePose` as a `rigidtform3d` object, which uses the premultiply convention. Although you can still specify `absolutePose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. The `bundleAdjustmentMotion` function returns `refinedPose` as a geometric transformation object of the same type as `absolutePose`.

You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Lourakis, Manolis I. A., and Antonis A. Argyros. "SBA: A Software Package for Generic Sparse Bundle Adjustment." *ACM Transactions on Mathematical Software 36*, no. 1 (March 2009): 2:1–2:30.

[2] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[3] Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle Adjustment — A Modern Synthesis." In *Proceedings of the International Workshop on Vision Algorithms*, 298–372. Springer-Verlag, 1999.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports code generation in only host target platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects

`rigidtform3d` | `imageviewset` | `cameraParameters` | `cameraIntrinsics`

Functions
`bundleAdjustment` | `bundleAdjustmentStructure`

Topics
“Point Feature Types”
“What Is Structure from Motion?”

bundleAdjustmentStructure

Refine 3-D points using structure-only bundle adjustment

Syntax

```
xyzRefinedPoints = bundleAdjustmentStructure(xyzPoints,pointTracks,
cameraPoses,intrinsics)
[wpSetRefined,vSetRefined,pointIndex] = bundleAdjustmentStructure(wpSet,vSet,
viewID,intrinsics)
[xyzRefinedPoints,reprojectionErrors] = bundleAdjustmentStructure( ___ )
[ ___ ] = bundleAdjustmentStructure( ___ ,Name,Value)
```

Description

`xyzRefinedPoints = bundleAdjustmentStructure(xyzPoints,pointTracks, cameraPoses,intrinsics)` returns the refined 3-D points that minimize reprojection errors.

The structure-only refinement procedure is a special case of bundle adjustment with camera poses fixed during optimization. The 3-D points and the camera poses are placed in the same world coordinate system.

`[wpSetRefined,vSetRefined,pointIndex] = bundleAdjustmentStructure(wpSet,vSet, viewID,intrinsics)` refines 3-D points from the world point set, `wpSet`, and refines camera poses from the image view set, `vSet`. `viewID` specifies the IDs if the views in `vSet`.

`[xyzRefinedPoints,reprojectionErrors] = bundleAdjustmentStructure( ___ )` additionally returns an *N*-element vector containing the mean reprojection error for each 3-D world point using the arguments from the previous syntax.

`[ ___ ] = bundleAdjustmentStructure( ___ ,Name,Value)` uses additional options specified by one or more name-value arguments. Unspecified arguments have default values.

Examples

Refine 3-D Points from Single Camera

Load data for initialization.

```
data = load('globeBA.mat');
```

Refine the 3-D world points.

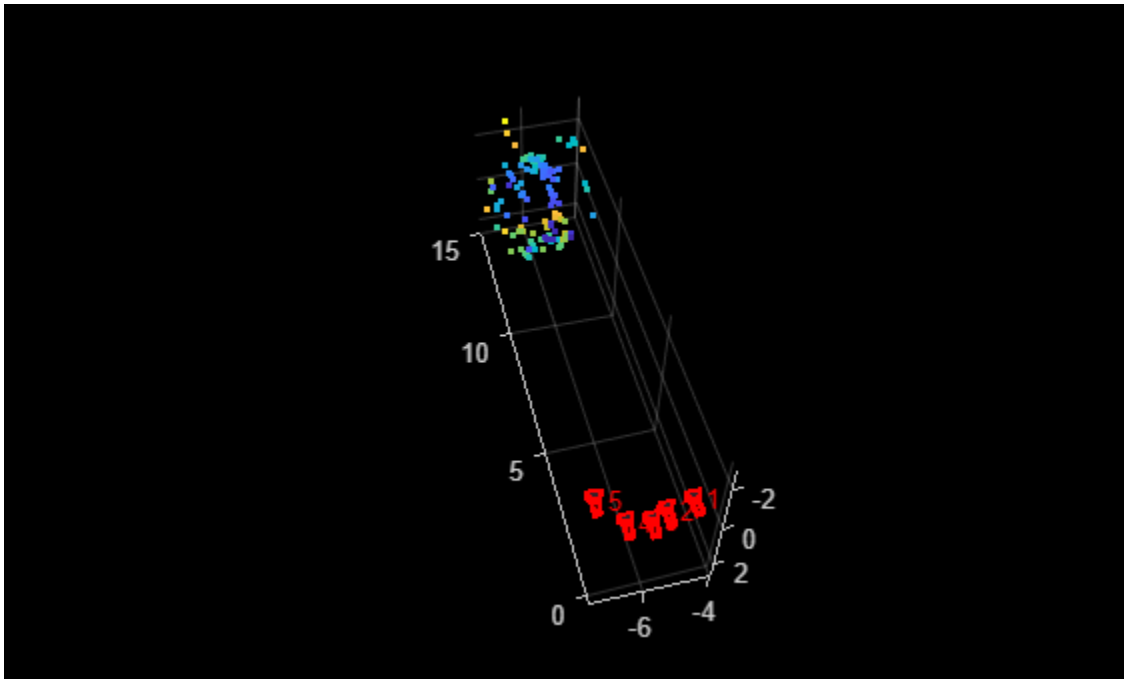
```
xyzRefinedPoints = bundleAdjustmentStructure(data.xyzPoints,data.pointTracks,data.cameraPoses,da
```

Display the refined 3-D world points.

```
pcshow(xyzRefinedPoints,AxesVisibility="on",VerticalAxis="y",VerticalAxisDir="down",MarkerSize=4
```

Display the cameras.

```
hold on
plotCamera(data.cameraPoses,Size=0.2);
hold off
grid on
```



Input Arguments

`xyzPoints` — Unrefined 3-D points
M-by-3 matrix

Unrefined 3-D points, specified as an *M*-by-3 matrix of [*x*,*y*,*z*] locations.

Data Types: `single` | `double`

`pointTracks` — Matched points
N-element array of `pointTracks` objects

Matched points across multiple images, specified as an *N*-element array of `pointTracks` objects.

`cameraPoses` — Camera poses
table

Camera poses, specified as a table containing the two columns `ViewId` and `AbsolutePose`. You can get the `cameraPoses` table using the `poses` object function of the `imageviewset` object.

`wpSet` — 3-D world points
`worldpointset` object

3-D world points, specified as a `worldpointset` object.

`vSet` — Camera poses
`imageviewset` object

Camera poses, specified as an `imageviewset` object.

`viewID` — View identifiers of the views in `vSet`
N-element array

View identifiers of the views in `vSet`, specified as an *N*-element array. The `viewIDs` represent which camera poses to refine specifying their related views in `imageviewset`.

`intrinsics` — Camera intrinsics
`cameraIntrinsics` object | *N*-element array of `cameraIntrinsics` objects

Camera intrinsics, specified as a `cameraIntrinsics` object or an *N*-element array of `cameraIntrinsics` objects. *N* is the number of camera poses or the number of IDs in `viewIDs`. Use a single `cameraIntrinsics` object when images are captured using the same camera. Use a vector `cameraIntrinsics` objects when images are captured by different cameras.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxIterations=50` sets the number of iterations to `50`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`MaxIterations` — Maximum number of iterations
`50` (default) | positive integer

Maximum number of iterations before the Levenberg-Marquardt algorithm stops, specified as a positive integer.

`AbsoluteTolerance` — Absolute termination tolerance
`1.0` (default) | positive scalar

Absolute termination tolerance of the mean squared reprojection error in pixels, specified as a positive scalar.

`RelativeTolerance` — Relative termination tolerance
`1e-5` (default) | positive scalar

Relative termination tolerance of the reduction in reprojection error between iterations, specified as a positive scalar.

`PointsUndistorted` — Flag to indicate lens distortion
`false` (default) | `true`

Flag to indicate lens distortion, specified as `false` or `true`. When you set `PointsUndistorted` to `false`, the 2-D points in `pointTracks` or in `vSet` must be from images with lens distortion. To use undistorted points, first use the `undistortImage` function to remove distortions from the images, then set `PointsUndistorted`.

`Verbose` — Display progress information
`False` (default) | `true`

Display progress information, specified as `false` or `true`.

Output Arguments

`xyzRefinedPoints` — 3-D locations of refined world points
M-by-3 matrix

3-D locations of refined world points, returned as an *M*-by-3 matrix of [*x*,*y*, *z*] locations.

Data Types: `single` | `double`

`wpSetRefined` — Refined 3-D world points
`worldpointset` object

Refined 3-D world points, returned as a `worldpointset` object.

`vSetRefined` — Refined camera poses
`imageviewset` object

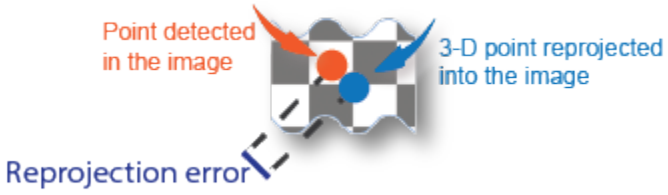
Refined camera poses, returned as a `imageviewset` object.

`pointIndex` — Indices of refined 3-D world points
M-element array

Indices to `wpSetRefined` of refined 3-D world points, returned as an *M*-element array.

`reprojectionErrors` — Reprojection errors
M-element vector

Reprojection errors, returned as an *M*-element vector. The function projects each world point back into each camera. Then in each image, the function calculates the reprojection error as the distance between the detected and the reprojected point. The `reprojectionErrors` vector contains the average reprojection error for each world point.



Version History

Introduced in R2020a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `bundleAdjustmentStructure` function now accepts `cameraPoses` as a table whose `AbsolutePose` column stores a `rigidtform3d` object, which uses the premultiply convention. This format is consistent with poses returned by the `poses` function of the `imageviewset` object.

Although you can still specify `cameraPoses` as a table whose `AbsolutePose` column stores a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Lourakis, Manolis I. A., and Antonis A. Argyros. "SBA: A Software Package for Generic Sparse Bundle Adjustment." *ACM Transactions on Mathematical Software 36*, no. 1 (March 2009): 2:1–2:30.

[2] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[3] Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle Adjustment — A Modern Synthesis." In *Proceedings of the International Workshop on Vision Algorithms*, 298–372. Springer-Verlag, 1999.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports code generation in only host target platforms.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects

`rigidtform3d` | `cameraParameters` | `cameraIntrinsics` | `imageviewset` | `worldpointset`

Functions

`bundleAdjustment` | `bundleAdjustmentMotion`

listTrueTypeFonts

List available TrueType fonts

Syntax

```
fontNames = listTrueTypeFonts
```

Description

`fontNames = listTrueTypeFonts` returns a cell array of sorted TrueType font names installed on the system.

Use `listTrueTypeFonts` function with the `insertText` function.

Note To display fonts that are available with MATLAB graphics, use the `listfonts` function.

Examples

List Available TrueType Fonts

```
listTrueTypeFonts
```

```
ans = 360×1 cell
    {'Agency FB'               }
    {'Agency FB Bold'          }
    {'Algerian'                }
    {'Arial'                   }
    {'Arial Black'             }
    {'Arial Bold'              }
    {'Arial Bold Italic'       }
    {'Arial Italic'            }
    {'Arial Narrow'            }
    {'Arial Narrow Bold'       }
    {'Arial Narrow Bold Italic'}
    {'Arial Narrow Italic'     }
    {'Arial Rounded MT Bold'   }
    {'Bahnschrift'             }
    {'Baskerville Old Face'    }
    {'Bauhaus 93'              }
    {'Bell MT'                 }
    {'Bell MT Bold'            }
    {'Bell MT Italic'          }
    {'Berlin Sans FB'          }
    {'Berlin Sans FB Bold'     }
    {'Berlin Sans FB Demi Bold'}
    {'Bernard MT Condensed'    }
    {'Blackadder ITC'          }
    {'Bodoni MT'               }
    {'Bodoni MT Black'         }
    {'Bodoni MT Black Italic'  }
```

```
    {'Bodoni MT Bold'          }
    {'Bodoni MT Bold Italic'   }
    {'Bodoni MT Condensed'     }
      ⋮
```

List All TrueType 'Lucida' Fonts

```
fontNames = listTrueTypeFonts;
LucidaFonts = fontNames(~cellfun(@isempty,regexp(fontNames,'^Lucida')))
```

```
LucidaFonts = 20×1 cell
    {'Lucida Bright'                    }
    {'Lucida Bright Demibold'           }
    {'Lucida Bright Demibold Italic'    }
    {'Lucida Bright Italic'             }
    {'Lucida Calligraphy Italic'        }
    {'Lucida Console'                   }
    {'Lucida Fax Demibold'              }
    {'Lucida Fax Demibold Italic'       }
    {'Lucida Fax Italic'                }
    {'Lucida Fax Regular'               }
    {'Lucida Handwriting Italic'        }
    {'Lucida Sans Demibold Italic'      }
    {'Lucida Sans Demibold Roman'       }
    {'Lucida Sans Italic'               }
    {'Lucida Sans Regular'              }
    {'Lucida Sans Typewriter Bold'      }
    {'Lucida Sans Typewriter Bold Oblique'}
    {'Lucida Sans Typewriter Oblique'   }
    {'Lucida Sans Typewriter Regular'   }
    {'Lucida Sans Unicode'              }
```

Output Arguments

`fontNames` — Available TrueType fonts on system

cell array

Available TrueType fonts on system, returned as a cell array of sorted TrueType font names.

Version History

Introduced in R2015b

See Also

`insertObjectAnnotation` | `insertText` | `listfonts`

pcfromkinect

Point cloud from Kinect for Windows

Syntax

```
ptCloud = pcfromkinect(depthDevice,depthImage)
ptCloud = pcfromkinect(depthDevice,depthImage,colorImage)
ptCloud = pcfromkinect(depthDevice,depthImage,colorImage,alignment)
```

Description

`ptCloud = pcfromkinect(depthDevice,depthImage)` returns a point cloud from a Kinect depth image. The `depthDevice` input can be either a `videoinput` object or an `imaq.VideoDevice` object configured for Kinect (Versions 1 and 2) for Windows.

This function requires the Image Acquisition Toolbox™ software, which supports Kinect for Windows.

`ptCloud = pcfromkinect(depthDevice,depthImage,colorImage)` adds color to the returned point cloud, specified by the `colorImage` input.

The Kinect for Windows system, designed for gaming, produces `depthImage` and `colorImage` as mirror images of the scene. The returned point cloud is corrected to match the actual scene.

`ptCloud = pcfromkinect(depthDevice,depthImage,colorImage,alignment)` additionally specifies the direction of the alignment of the image coordinate system, which can be color centric or depth centric.

Examples

Plot Color Point Cloud from Kinect for Windows

Plot a color point cloud from Kinect images. This example requires the Image Acquisition Toolbox software and the Kinect camera and a connection to the camera.

Create a System object for the color device.

```
colorDevice = imaq.VideoDevice('kinect',1)
```

Create a System object for the depth device.

```
depthDevice = imaq.VideoDevice('kinect',2)
```

Initialize the camera.

```
step(colorDevice);
step(depthDevice);
```

Load one frame from the device.

```
colorImage = step(colorDevice);
depthImage = step(depthDevice);
```

Extract the point cloud.

```
ptCloud = pcfromkinect(depthDevice,depthImage,colorImage);
```

Initialize a point cloud player to visualize 3-D point cloud data. The axis is set appropriately to visualize the point cloud from Kinect.

```
player = pcplayer(ptCloud.XLimits,ptCloud.YLimits,ptCloud.ZLimits,...
    'VerticalAxis','y','VerticalAxisDir','down');

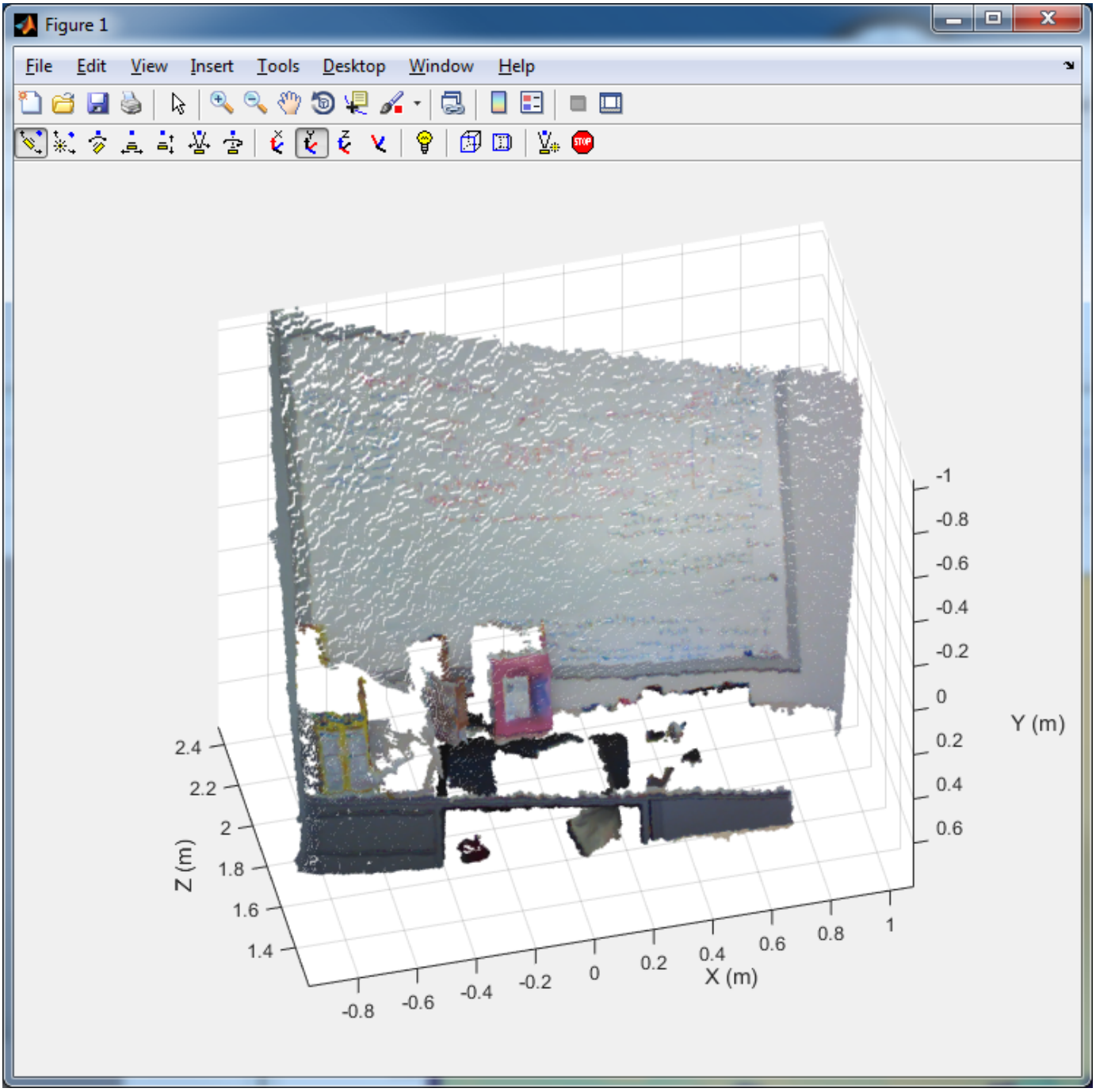
xlabel(player.Axes,'X (m)');
ylabel(player.Axes,'Y (m)');
zlabel(player.Axes,'Z (m)');
```

Acquire and view 500 frames of live Kinect point cloud data.

```
for i = 1:500
   colorImage = step(colorDevice);
   depthImage = step(depthDevice);

   ptCloud = pcfromkinect(depthDevice,depthImage,colorImage);

   view(player,ptCloud);
end
```



Release the objects.

```
release(colorDevice);
release(depthDevice);
```

Input Arguments

depthDevice — Input video object

`videoinput` object | `imaq.VideoDevice` object

Input video object, specified as either a `videoinput` object or an `imaq.VideoDevice` object configured for Kinect for Windows.

depthImage — Depth image

M-by-*N* matrix

Depth image, specified as an *M*-by-*N* pixel matrix. The original images, `depthImage` and `colorImage`, from Kinect are mirror images of the scene.

The Kinect depth camera has limited range. The limited range of the Kinect depth camera can cause pixel values in the depth image to not have corresponding 3-D coordinates. These missing pixel values are set to `NaN` in the `Location` property of the returned point cloud.

Data Types: `uint16`

colorImage — Color image

M-by-*N*-by-3 RGB truecolor image

Color image, specified as an *M*-by-*N*-by-3 RGB truecolor image that the Kinect returns. The original images, `depthImage` and `colorImage`, from Kinect are mirror images of the scene.

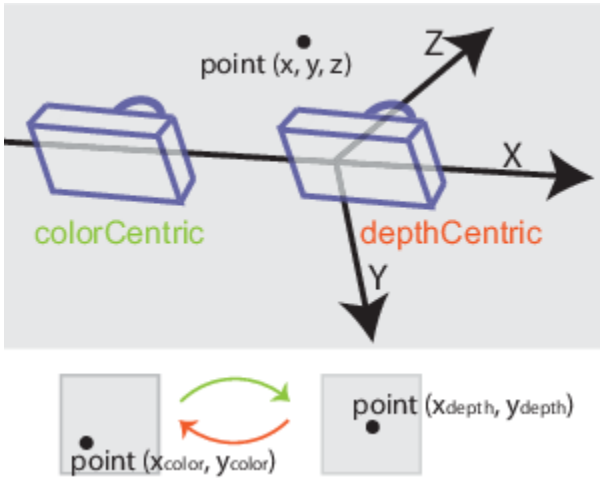
Data Types: `uint8`

alignment — Direction of the image coordinate system

`'colorCentric'` (default) | `'depthCentric'`

Direction of the image coordinate system, specified as the character vector `'colorCentric'` or `'depthCentric'`. Set this value to `'colorCentric'` to align `depthImage` with `colorImage`. Set `alignment` to `'depthCentric'` to align `colorImage` with `depthImage`.

The origin of a right-handed world coordinate system is at the center of the depth camera. The *x*-axis of the coordinate system points to the right, the *y*-axis points downward, and the *z*-axis points away from the camera.



Note For consistency across Computer Vision Toolbox use of coordinates systems, the coordinate system defined by this function is different from the one defined by Kinect Skeletal metadata.

Output Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, returned as a `pointCloud` object. The origin of the coordinate system of the returned point cloud is at the center of the depth camera.

Version History

Introduced in R2015b

See Also

Functions

`pcplayer` | `plot3` | `pcwrite` | `pcread` | `pcshow` | `pcmerge` | `scatter3` | `reconstructScene` | `triangulate` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp` | `videoinput`

Objects

`pointCloud` | `planeModel` | `imaq.VideoDevice`

Topics

“Structure from Motion from Two Views”
“Depth Estimation from Stereo Video”
“Coordinate Systems”

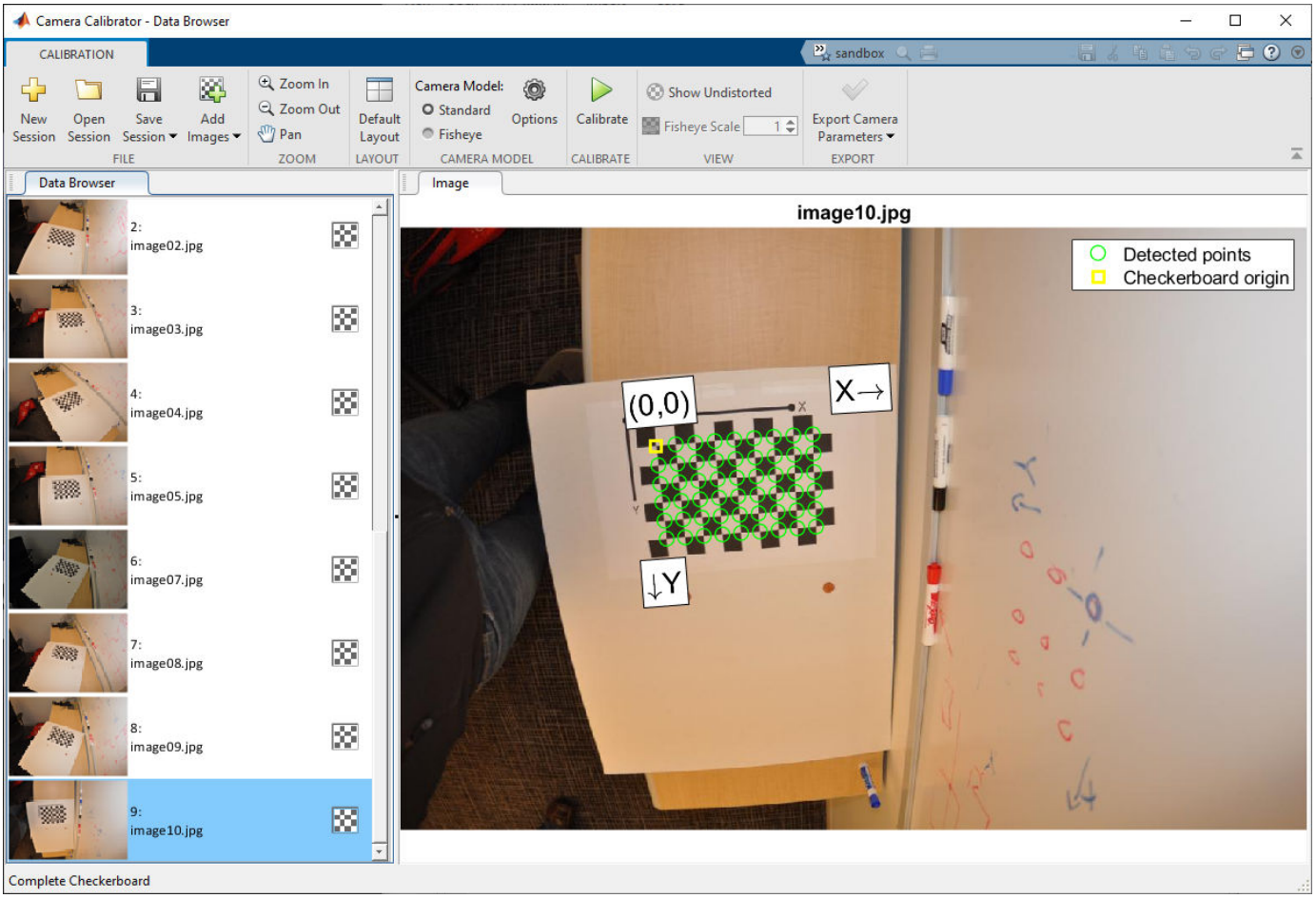
Camera Calibrator

Estimate geometric parameters of a single camera

Description

The Camera Calibrator app allows you to estimate camera intrinsics, extrinsics, and lens distortion parameters. You can use these camera parameters for various computer vision applications. These applications include removing the effects of lens distortion from an image, measuring planar objects, or reconstructing 3-D scenes from multiple cameras.

Note Ensure that the images you are using were captured specifically for single camera calibration following the guidelines outlined in “Prepare Camera and Capture Images for Camera Calibration”. Using images captured for multi-sensor calibration can lead to inaccurate camera intrinsics.



Open the Camera Calibrator App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `cameraCalibrator`.

Examples

Open Camera Calibrator App

This example shows you the two ways to open the Camera Calibrator app.

Type `cameraCalibrator` on the MATLAB command line or select it from the MATLAB desktop **Apps** tab.

- “Evaluating the Accuracy of Single Camera Calibration”
- “Measuring Planar Objects with a Calibrated Camera”
- “Structure from Motion from Two Views”
- “Structure from Motion from Multiple Views”
- “Depth Estimation from Stereo Video”

Programmatic Use

`cameraCalibrator` opens the Camera Calibrator app, which enables you to compute parameters needed to remove the effects of lens distortion from an image.

`cameraCalibrator(imageFolder,squareSize)` opens the app and loads calibration images from a specified folder and specifies the size of the checkerboard squares contained in the images. The `squareSize` input must be a scalar in millimeters.

`cameraCalibrator(imageFolder,squareSize,squareSizeUnits)` opens the app, loads calibration images, specifies the checkerboard square size, and the square size units. The `squareSizeUnits` input must be specified as `'millimeters'` (default), `'centimeters'`, or `'inches'`.

`cameraCalibrator( ___ ,highDistortion)` opens the app as described by previous syntaxes and additionally specifies a high distortion logical `highDistortion`. Set `highDistortion` to `true` when the images contain a high level of distortion, which is typical of a wide field of view camera. Set `highDistortion` to `false` (default) when the images do not contain a high level of distortion.

`cameraCalibrator(sessionFile)` opens the app and loads a saved camera calibration session. Set the `sessionFile` to the name of the saved session file. The name must include the path to the MAT file containing the saved session.

Version History

Introduced in R2013b

See Also

Apps
Stereo Camera Calibrator

Classes
`cameraParameters` | `stereoParameters`

Functions
`estimateCameraParameters` | `showExtrinsics` | `showReprojectionErrors` | `undistortImage` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `generateCharucoBoard` | `generateArucoMarker` | `detectAprilGridPoints` | `rectifyStereoImages` | `estimateExtrinsics` | `triangulate`

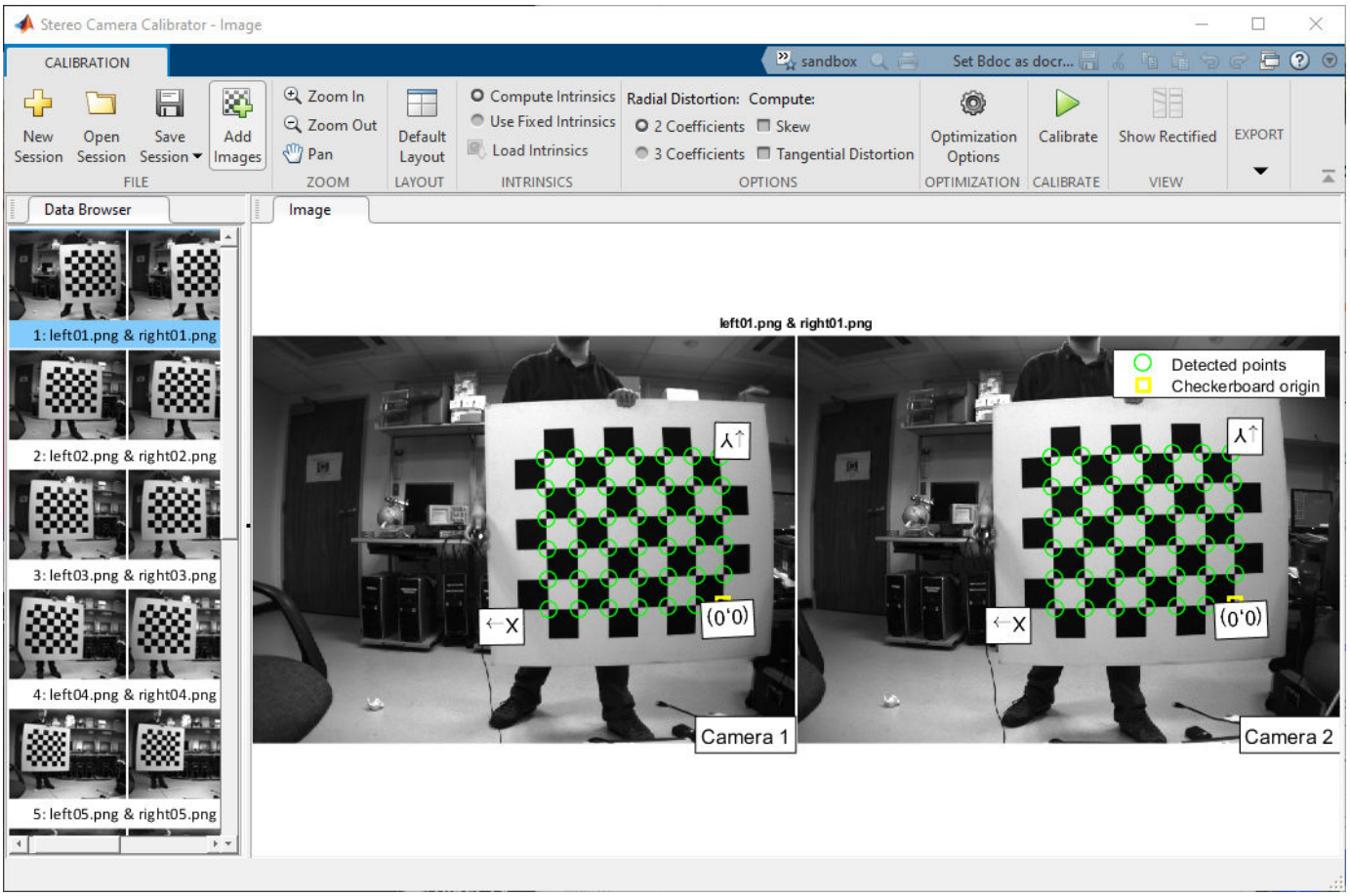
Topics
“Evaluating the Accuracy of Single Camera Calibration”
“Measuring Planar Objects with a Calibrated Camera”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

Stereo Camera Calibrator

Estimate geometric parameters of a stereo camera

Description

The Stereo Camera Calibrator app allows you to estimate the intrinsic and extrinsic parameters of each camera in a stereo pair. You can also use the app to estimate the translation and rotation between the two cameras.



Open the Stereo Camera Calibrator App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `stereoCameraCalibrator`.

Examples

Open Stereo Camera Calibrator App

This example shows you the two ways to open the Stereo Camera Calibrator app.

Type `stereoCameraCalibrator` on the MATLAB command line or select it from the MATLAB desktop **Apps** tab.

- “Evaluating the Accuracy of Single Camera Calibration”
- “Measuring Planar Objects with a Calibrated Camera”
- “Structure from Motion from Two Views”
- “Structure from Motion from Multiple Views”
- “Depth Estimation from Stereo Video”

Programmatic Use

`stereoCameraCalibrator` opens the Stereo Camera Calibrator app. You can use this app to estimate the intrinsic and extrinsic parameters of each camera in a stereo pair. You can also use the app to estimate the translation and rotation between the two cameras.

`stereoCameraCalibrator(folder1,folder2,squareSize)` opens the Stereo Camera Calibrator app and loads the stereo calibration images. The app uses the checkerboard square size specified by the `squareSize` input. It also uses `folder1` images for camera 1 and `folder2` for camera 2.

`stereoCameraCalibrator(folder1,folder2,squareSize,squareSizeUnits)` additionally specifies the units of the square size. If you do not specify units, the app sets `squareSizeUnits` to `'millimeters'`. Units can be `'millimeters'`, `'centimeters'`, or `'inches'`.

`stereoCameraCalibrator(sessionFile)` opens the app and loads a saved stereo calibration session. Set the `sessionFile` to the name of the saved session MAT-file.

Version History

Introduced in R2014b

See Also

Apps
Camera Calibrator

Objects
`cameraParameters` | `stereoParameters`

Functions
`estimateCameraParameters` | `showExtrinsics` | `showReprojectionErrors` | `undistortImage` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `generateCheckerboardPoints` | `generateCharucoBoard` | `generateArucoMarker` | `detectAprilGridPoints`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Measuring Planar Objects with a Calibrated Camera”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

cameraMatrix

(Not recommended) Camera projection matrix

Note `cameraMatrix` is not recommended. Use the `cameraProjection` function instead. For more information, see “Version History”.

Syntax

```
camMatrix = cameraMatrix(cameraParams,tform)
camMatrix = cameraMatrix(cameraParams,rotationMatrix,translationVector)
```

Description

`camMatrix = cameraMatrix(cameraParams,tform)` returns a 4-by-3 camera projection matrix `camMatrix`, which can be used to project a 3-D world point in homogeneous coordinates into an image. `cameraParams` can be a `cameraParameters` object or a `cameraIntrinsics` object.

`camMatrix = cameraMatrix(cameraParams,rotationMatrix,translationVector)` returns a 4-by-3 camera projection matrix. You can use this matrix to project 3-D world points in homogeneous coordinates into an image.

Examples

Compute Camera Matrix

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","slr"));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0). The square size is in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

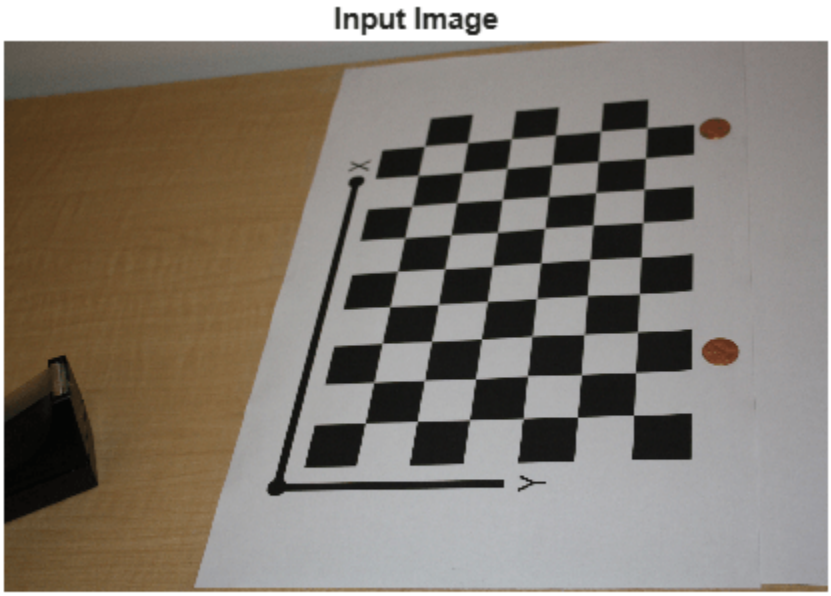
Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1),size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Load image at new location.

```
imOrig = imread(fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration","slr","image9.jpg"));
```

```
figure;
imshow(imOrig);
title("Input Image");
```



Undistort image.

```
im = undistortImage(imOrig,cameraParams);
```

Find reference object in new image.

```
[imagePoints,patternDims] = detectCheckerboardPoints(im);
```

Compute new extrinsics.

```
[rotationMatrix,translationVector] = extrinsics(...
    imagePoints,worldPoints,cameraParams);
```

Calculate camera matrix.

```
P = cameraMatrix(cameraParams,rotationMatrix,translationVector)
```

```
P = 4×3
10^5 ×

    0.0157   -0.0271    0.0000
    0.0404   -0.0046   -0.0000
    0.0199    0.0387    0.0000
    8.9399    9.4399    0.0072
```

Input Arguments

`cameraParams` — Object for storing camera parameters

`cameraParameters` object | `cameraIntrinsics` object

Camera parameters, specified as a `cameraParameters` or `cameraIntrinsics` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`tform` — Transformation

`rigid3d` object

Transformation from world coordinates to camera coordinates, specified as a `rigid3d` object. You can use the `extrinsics` function to obtain the rotation and translation to create the `tform` object.

`rotationMatrix` — Rotation of camera

3-by-3 matrix

Rotation of camera, specified as a 3-by-3 matrix. You can obtain this matrix using the `extrinsics` function. You can also obtain the matrix using the `relativeCameraPose` function by transposing its `orientation` output. The `rotationMatrix` and `translationVector` inputs must be real, nonsparse, and of the same class.

`translationVector` — Translation of camera

1-by-3 vector

Translation of camera, specified as a 1-by-3 vector. The translation vector describes the transformation from the world coordinates to the camera coordinates. You can obtain this vector using the `extrinsics` function. You can also obtain the vector using the `location` and `orientation` outputs of the `relativeCameraPose` function:

- `translationVector` = `-relativeLocation` * `relativeOrientation'`

The `translationVector` inputs must be real, nonsparse, and of the same class.

Output Arguments

`camMatrix` — Camera projection matrix

4-by-3 matrix

Camera projection matrix, returned as a 4-by-3 matrix. The matrix contains the 3-D world points in homogenous coordinates that are projected into the image. When you set `rotationMatrix` and `translationVector` to `double`, the function returns `camMatrix` as `double`. Otherwise it returns `camMatrix` as `single`.

The function computes `camMatrix` as follows:

`camMatrix` = [`rotationMatrix`; `translationVector`] × K.
K: the intrinsic matrix

Then, using the camera matrix and homogeneous coordinates, you can project a world point onto the image.

$w \times [x,y,1] = [X,Y,Z,1] \times$ `camMatrix`.

(X,Y,Z): world coordinates of a point
(x,y): coordinates of the corresponding image point
w: arbitrary scale factor

Data Types: `single` | `double`

Version History

Introduced in R2014b

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `cameraMatrix` function uses the postmultiply convention. Although there are no plans to remove `cameraMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `cameraProjection` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `cameraMatrix` to `cameraProjection`.
- Specify the `cameraParams` argument as a `cameraIntrinsics` object. If you have a `cameraParameters` object, then you can get a `cameraIntrinsics` object by querying the `Intrinsics` property. If the `Intrinsics` property is empty according to the `isempty` function, then set the `ImageSize` property of the `cameraParameters` object to an arbitrary vector before querying the `Intrinsics` property. For example:

  ```
  load worldToImageCorrespondences.mat
  if(isempty(cameraParams.Intrinsics))
      cameraParams.ImageSize = [128 128];
  end
  intrinsics = cameraParams.Intrinsics;
  ```

- Specify the transformation as a `rigidtform3d` object using the `tform` argument. `cameraProjection` does not support the `rotationMatrix` and `translationVector` input arguments. Note that you create the `rigidtform3d` object using the transpose of `rotationMatrix` or the transpose of the transformation matrix in the `T` property of `tform`.

Discouraged Usage	Recommended Replacement

This example specifies the transformation as a rotation matrix and a translation vector, then calculates the camera projection matrix using the `cameraMatrix` function.
 	
`camMatrix = cameraMatrix(cameraParams,rotationMatrix,translationVector)`
	This example specifies the transformation as a `rigidtform3d` object, then calculates the camera projection matrix using the `cameraProjection` function. Note that you create the `rigidtform3d` object using the transpose of the rotation matrix, `rotationMatrix`.
	
	`intrinsics = cameraParams.Intrinsics;`
	`tform = rigidtform3d(rotationMatrix',translationVecto`
	`camMatrix = cameraProjection(intrinsics,tform);`

Discouraged Usage	Recommended Replacement

This example specifies the transformation as a `rigid3d` object.
 	
`camMatrix = cameraMatrix(cameraParams,tformOld);`
	This example starts with an existing `rigid3d` object `tformOld`, creates a `rigidtform3d` object using the transpose of the `T` property of `tformOld`, then calculates the camera projection matrix using the `cameraProjection` function.
	
	`intrinsics = cameraParams.Intrinsics;`
	`tform = rigidtform3d(tformOld.T');`
	`camMatrix = cameraProjection(intrinsics,tform);`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`cameraProjection` | `estimateCameraProjection` | `estimateExtrinsics` | `triangulate` | `estimateCameraParameters` | `estrelpose`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

cameraPose

Compute relative rotation and translation between camera poses

Syntax

```
cameraPose
```

Description

`cameraPose` returns the camera extrinsics.

Note `cameraPose` was renamed to `relativeCameraPose`. Use the new function name instead. You can also use the `estrelpose` function instead.

Version History

Introduced in R2015b

R2020a: `cameraPose` renamed to `relativeCameraPose`
Behavior changed in R2020a

`cameraPose` was renamed to `relativeCameraPose`. Use the new function name instead. You can also use the `estrelpose` function instead.

cameraProjection

Camera projection matrix

Syntax

```
camProjection = cameraProjection(intrinsics,tform)
```

Description

`camProjection = cameraProjection(intrinsics,tform)` returns a 3-by-4 camera projection matrix `camProjection`. You can use `camProjection` to project a 3-D world point in homogeneous coordinates into an image according to the transformation `tform`.

Examples

Calculate Camera Projection Matrix

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration","slr"));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0). The square size is in millimeters.

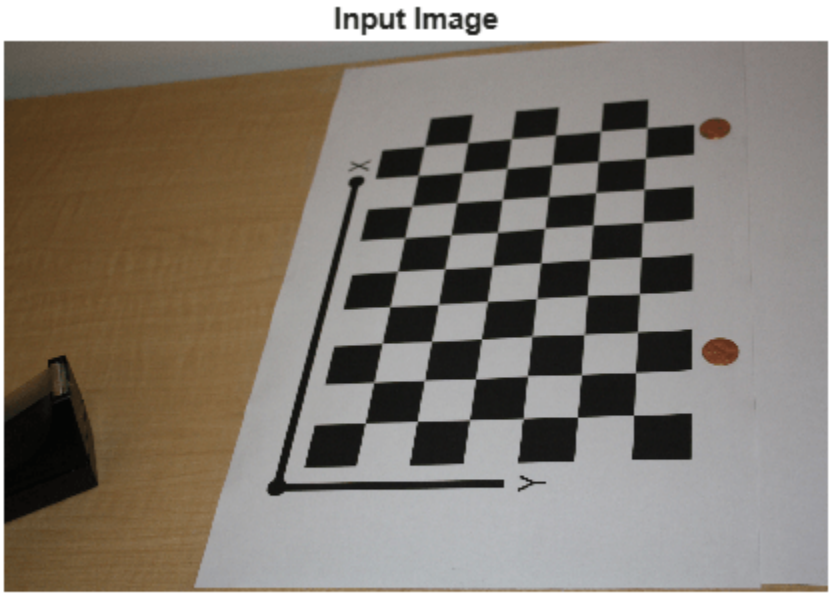
```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1) size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
intrinsics = cameraParams.Intrinsics;
```

Load an image at a new location.

```
imOrig = readimage(images,9);
figure
imshow(imOrig)
title("Input Image")
```



Undistort the image.

```
[im,newIntrinsics] = undistortImage(imOrig,intrinsics,OutputView="full");
```

Find the reference object in the new image.

```
[imagePoints,patternDims] = detectCheckerboardPoints(im);
```

Compensate for the image coordinate system shift.

```
newOrigin = intrinsics.PrincipalPoint-newIntrinsics.PrincipalPoint;
imagePoints = imagePoints+newOrigin;
```

Calculate new extrinsics.

```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,intrinsics);
```

Calculate the camera projection matrix.

```
P = cameraProjection(intrinsics,camExtrinsics)
```

```
P = 3×4
10^5 ×

    0.0157    0.0404    0.0199    8.9399
   -0.0271   -0.0046    0.0387    9.4399
    0.0000   -0.0000    0.0000    0.0072
```

Input Arguments

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object

Camera intrinsics, specified as `cameraIntrinsics` object. The object stores information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

`tform` — Transformation
`rigidtform3d` object

Transformation from world coordinates to camera coordinates, specified as a `rigidtform3d` object. You can use the `estimateExtrinsics` function to create the `tform` object. The `R` and `Translation` properties of the object represent the rotation matrix and translation needed to project 3-D world points in homogeneous coordinates into an image.

Output Arguments

`camProjection` — Camera projection matrix
3-by-4 matrix

Camera projection matrix, returned as a 3-by-4 matrix. The matrix contains the 3-D world points in homogenous coordinates that are projected into the image.

The function calculates `camProjection` using the intrinsic matrix K and the `R` and `Translation` properties of the `tform` object as follows:

`camProjection` = $K \times$ `[tform.R tform.Translation']`

Using the camera projection matrix and homogeneous coordinates, you can project a world point onto the image.

$w \times [x,y,1]'$ = `camProjection`$\times [X,Y,Z,1]'$

(X,Y,Z) are the world coordinates of a point
(x,y) are the coordinates of the corresponding image point
w is an arbitrary scale factor

When the `R` and `Translation` properties of the transformation `tform` are of data type `double`, the function returns `camProjection` as `double`. Otherwise the function returns `camProjection` as `single`.

Data Types: `single` | `double`

Version History

Introduced in R2022b

R2022b: Recommended over `cameraMatrix`

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `cameraMatrix` function uses the postmultiply convention. Although there are no plans to remove `cameraMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `cameraProjection` function, which supports the premultiply convention. For more information, see "Migrate Geometric Transformations to Premultiply Convention".

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`estimateExtrinsics` | `triangulate` | `estimateCameraParameters` | `estworldpose` | `estrelpose` | `pose2extr` | `estimateCameraProjection`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

relativeCameraPose

(Not recommended) Calculate relative rotation and translation between camera poses

Note `relativeCameraPose` is not recommended. Use the `estrelpose` function instead. For more information, see “Version History”.

Syntax

```
[relativeOrientation,relativeLocation] = relativeCameraPose(M,cameraParams,
inlierPoints1,inlierPoints2)
[relativeOrientation,relativeLocation] = relativeCameraPose(M,cameraParams1,
cameraParams2,inlierPoints1,inlierPoints2)
[relativeOrientation,relativeLocation,validPointsFraction] =
relativeCameraPose(M, ___ )
```

Description

`[relativeOrientation,relativeLocation] = relativeCameraPose(M,cameraParams, inlierPoints1,inlierPoints2)` returns the orientation and location of a calibrated camera relative to its previous pose. The two poses are related by `M`, which must be a fundamental, essential, `affine2d`, or `projective2d` matrix. The function computes the camera location up to scale and returns `relativeLocation` as a unit vector.

`[relativeOrientation,relativeLocation] = relativeCameraPose(M,cameraParams1, cameraParams2,inlierPoints1,inlierPoints2)` returns the orientation and location of the second camera relative to the first one.

`[relativeOrientation,relativeLocation,validPointsFraction] = relativeCameraPose(M, ___ )` additionally returns the fraction of the inlier points that project in front of both cameras.

Input Arguments

`M` — Fundamental, essential, or homography matrix

3-by-3 matrix

Fundamental, essential matrix, or a homography matrix, specified as a 3-by-3 matrix, an affine2d object, or a `projective2d` object containing a homography matrix. You can obtain the 3-by-3 matrix using one of the following functions:

- `estimateFundamentalMatrix` for the fundamental matrix.
- `estimateEssentialMatrix` for the essential matrix
- `estimateGeometricTransform` for the projective or affine 2-D object.

Data Types: `single` | `double`

`cameraParams` — Object for storing camera parameters

`cameraParameters` object | `cameraIntrinsics` object

Camera parameters, specified as a `cameraParameters` or `cameraIntrinsics` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`cameraParams1` — Camera parameters for camera 1
`cameraParameters` object | `cameraIntrinsics` object

Camera parameters for camera 1, specified as a `cameraParameters` or `cameraIntrinsics` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`cameraParams2` — Camera parameters for camera 2
`cameraParameters` object | `cameraIntrinsics` object

Camera parameters for camera 2, specified as a `cameraParameters` or `cameraIntrinsics` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

`inlierPoints1` — Coordinates of corresponding points in view 1
`SIFTPoints` | `SURFPoints` | `cornerPoints` | `MSERRegions` | *M*-by-2 matrix of [*x*,*y*] coordinates

Coordinates of corresponding points in view 1, specified as an *M*-by-2 matrix of *M* number of [*x*,*y*] coordinates, or as a `SIFTPoints`, `SURFPoints`, `MSERRegions`, or `cornerPoints` object. You can obtain these points using the `estimateFundamentalMatrix` function or the `estimateEssentialMatrix`.

`inlierPoints2` — Coordinates of corresponding points in view 2
`SIFTPoints` | `SURFPoints` | `cornerPoints` | `MSERRegions` | *M*-by-2 matrix of [*x*,*y*] coordinates

Coordinates of corresponding points in view 2, specified as an *M*-by-2 matrix of *M* number of [*x*,*y*] coordinates, or as a `SIFTPoints`, `SURFPoints`, `MSERRegions`, or `cornerPoints` object. You can obtain these points using the `estimateFundamentalMatrix` function or the `estimateEssentialMatrix`.

Output Arguments

`relativeOrientation` — Orientation of camera
3-by-3 matrix

Orientation of camera, returned as a 3-by-3 matrix. If you use only one camera, the matrix describes the orientation of the second camera pose relative to the first camera pose. If you use two cameras, the matrix describes the orientation of camera 2 relative to camera 1.

Data Types: `single` | `double`

`relativeLocation` — Location of camera
1-by-3 vector

Location of camera, returned as a 1-by-3 unit vector. If you use only one camera, the vector describes the location of the second camera pose relative to the first camera pose. If you use two cameras, the vector describes the location of camera 2 relative to camera 1.

Data Types: `single` | `double`

`validPointsFraction` — Fraction of valid inlier points
scalar

Fraction of valid inlier points that project in front of both cameras, returned as a scalar. If `validPointsFraction` is too small, e.g. less than 0.9, it can indicate that the fundamental matrix is incorrect.

Tips

- You can compute the camera extrinsics, `rotationMatrix` and `translationVector`, corresponding to the camera pose, from `relativeOrientation` and `relativeLocation`:

  ```
  [rotationMatrix,translationVector] = cameraPoseToExtrinsics(relativeOrientation,relativeLocati
  ```

  The orientation of the previous camera pose is the identity matrix, `eye(3)`, and its location is, `[0,0,0]`.
- You can then use `rotationMatrix` and `translationVector` as inputs to the `cameraMatrix` function.
- You can compute four possible combinations of orientation and location from the input fundamental matrix. Three of the combinations are not physically realizable, because they project 3-D points behind one or both cameras. The `relativeCameraPose` function uses `inlierPoints1` and `inlierPoints2` to determine the realizable combination.

Version History

Introduced in R2016b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `relativeCameraPose` function uses the postmultiply convention. Although there are no plans to remove `relativeCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `estrelpose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `relativeCameraPose` to `estrelpose`.
- Specify the `cameraParams` argument as a `cameraIntrinsics` object. If you have a `cameraParameters` object, then you can get a `cameraIntrinsics` object by querying the `Intrinsics` property. If the `Intrinsics` property is empty according to the `isempty` function, then set the `ImageSize` property of the `cameraParameters` object to an arbitrary vector before querying the `Intrinsics` property. For example:

  ```
  load worldToImageCorrespondences.mat
  if(isempty(cameraParams.Intrinsics))
      cameraParams.ImageSize = [128 128];
  end
  intrinsics = cameraParams.Intrinsics;
  ```

- Replace the two output arguments `relativeOrientation` and `relativeLocation` with a single output argument, `relativePose`. If you need to obtain the orientation matrix and location vector, then you can query the `R` and the `Translation` properties of the `rigidtform3d` object returned by the `worldPose` argument. Note that the value of `R` is the transpose of `relativeOrientation`.

The table shows examples of how to update your code.

Discouraged Usage	Recommended Replacement

This example calculates relative camera poses of a single camera using the `relativeCameraPose` function with the `cameraParams` argument specified as a `cameraParameters` object.
 	
`[relativeOrientationOld,relativeLocation] = relativeCameraPose( ...`
`    M,cameraParams,inlierPoints1,inlierPoints2)`
	This example gets the camera intrinsics using the `Intrinsics` property of a `cameraParameters` object, then estimates the relative camera pose using the `estrelpose` function.
	
	`intrinsics = cameraParams.Intrinsics;`
	`relativePose = estrelpose( ...`
	`    M,intrinsics,inlierPoints1,inlierPoints2);`
	
	If you need to obtain the relative camera orientation and location, then you can query properties of `relativePose`.
	
	`relativeOrientation = relativePose.R;`
	`relativeLocation = relativePose.Translation;`
	
	If you want the orientation in the postmultiply convention, take the transpose of `relativePose.R`.
	
	`relativeOrientationOld = relativePose.R';`

This example calculates the relative camera pose of camera 2 with respect to camera 1 using the `relativeCameraPose` function.
 	
`[relativeOrientation,relativeLocation] = relativeCameraPose( ...`
`    M,cameraParams1,cameraParams2,inlierPoints1,inlierPoints2)`
	This example gets the camera intrinsics of two cameras using the `Intrinsics` property of two `cameraParameters` objects, then estimates the relative camera pose using the `estrelpose` function.
	
	`intrinsics1 = cameraParams1.Intrinsics;`
	`intrinsics2 = cameraParams2.Intrinsics;`
	`relativePose = estrelpose( ...`
	`    M,intrinsics1,intrinsics2,inlierPoints1,inlierPo`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator

Functions
`estrelpose` | `estworldpose` | `estimateCameraParameters` | `estimateEssentialMatrix` | `estimateFundamentalMatrix` | `cameraProjection` | `plotCamera` | `triangulate` | `triangulateMultiview` | `pose2extr`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Point Feature Types”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“What Is Structure from Motion?”

estimateCameraMatrix

(Not recommended) Estimate camera projection matrix from world-to-image point correspondences

Note `estimateCameraMatrix` is not recommended. Use the `estimateCameraProjection` function instead. For more information, see “Version History”.

Syntax

```
camMatrix = estimateCameraMatrix(imagePoints,worldPoints)
[camMatrix,reprojectionErrors] = estimateCameraMatrix(imagePoints,
worldPoints)
```

Description

`camMatrix = estimateCameraMatrix(imagePoints,worldPoints)` returns the camera projection matrix determined from known world points and their corresponding image projections by using the direct linear transformation (DLT) approach.

`[camMatrix,reprojectionErrors] = estimateCameraMatrix(imagePoints, worldPoints)` also returns the reprojection error that quantifies the accuracy of the projected image coordinates.

Examples

Estimate Camera Projection Matrix

Load a 3-D point cloud data captured by an RGB-D sensor into the workspace.

```
ld = load('object3d.mat');
ptCloud = ld.ptCloud;
```

Remove points with `Inf` or `NaN` coordinates from the point cloud.

```
[validPtCloud,validIndices] = removeInvalidPoints(ptCloud);
```

Read the valid world point coordinates. Each entry specifies the *x*, *y*, *z* coordinates of a point in the point cloud.

```
worldPoints = validPtCloud.Location;
```

Define the corresponding image point coordinates as a orthographic projection of point cloud data onto the *yz*-plane.

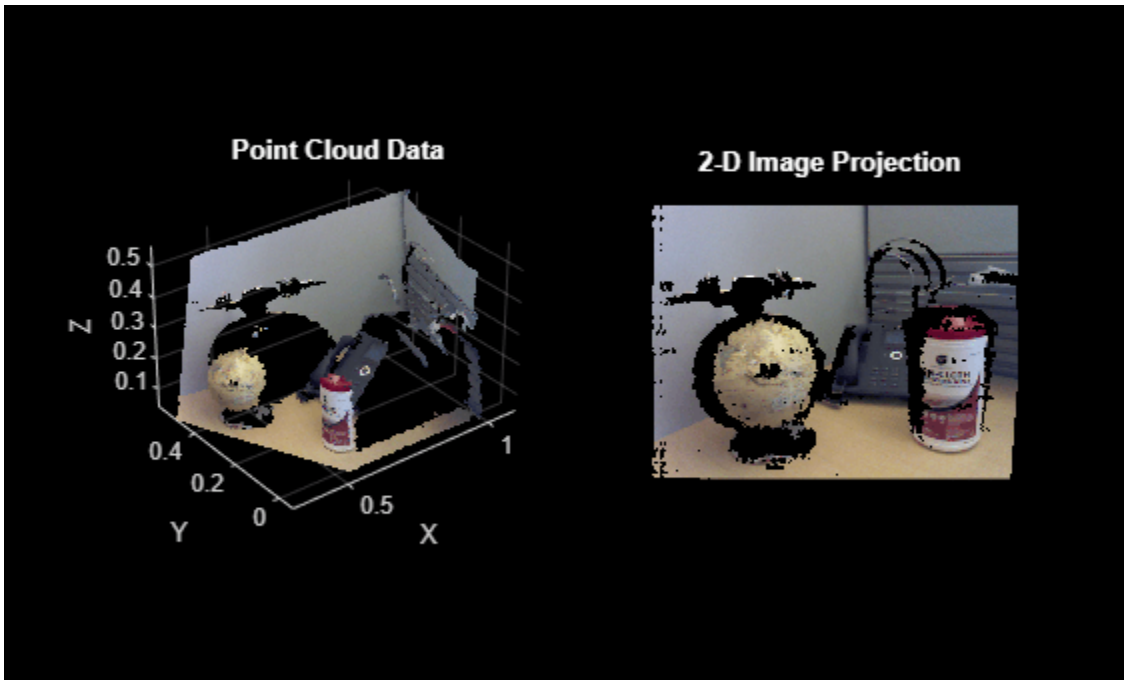
```
indices = 1:ptCloud.Count;
[y,z] = ind2sub([size(ptCloud.Location,1),size(ptCloud.Location,2)],indices);
imagePoints = [y(validIndices)' z(validIndices)'];
```

Generate the 2-D image projection by using the image point coordinates and their color values.

```
projImage = zeros(max(imagePoints(:,1)),max(imagePoints(:,2)),3);
rgb = validPtCloud.Color;
for j = 1:length(rgb)
projImage(imagePoints(j,1),imagePoints(j,2),:) = rgb(j,:);
end
```

Display the point cloud data and the corresponding 2-D image projection.

```
figure
subplot(1,2,1)
pcshow(ptCloud)
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Point Cloud Data','Color',[1 1 1])
subplot(1,2,2)
imshow(uint8(projImage))
title('2-D Image Projection','Color',[1 1 1])
```



Estimate the camera projection matrix and reprojection error by using the known world points and the image points.

```
[camMatrix,reprojectionErrors] = estimateCameraMatrix(imagePoints,worldPoints);
```

Use the estimated camera projection matrix as input to the `findNearestNeighbors` function and find the nearest neighbors of a query point.

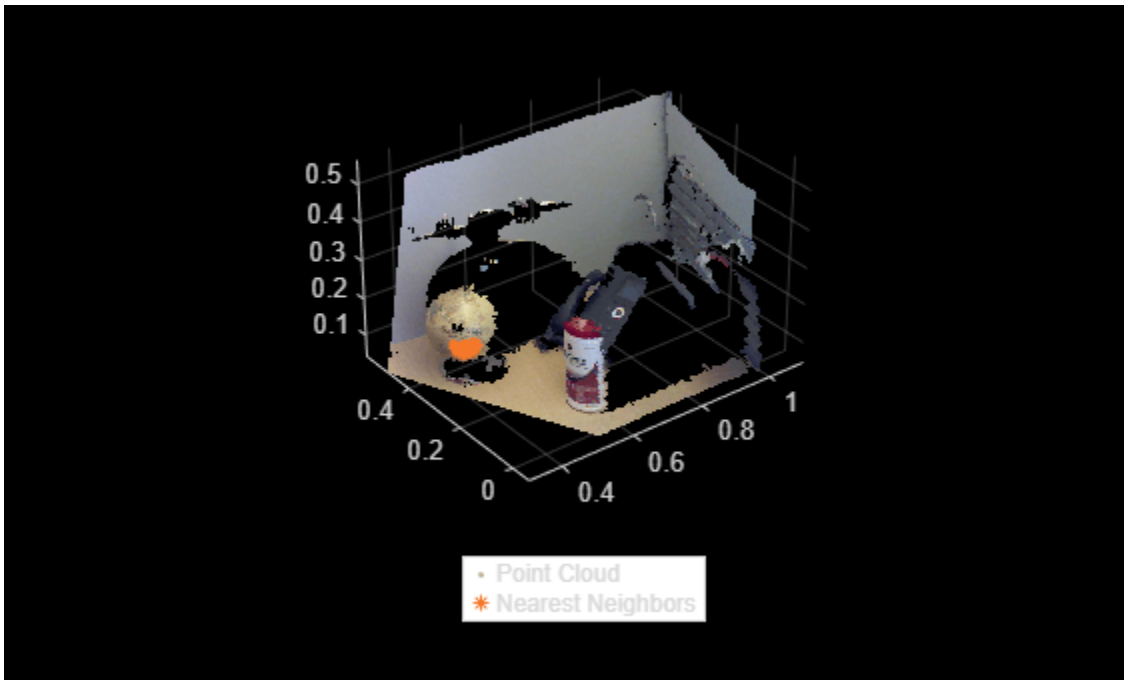
```
point = [0.4 0.3 0.2];  % Specify the query point
K = 1000;  % Specify the number of nearest neighbors to be determined
[indices,dists] = findNearestNeighbors(ptCloud,point,K,camMatrix);  % Get the indices and distanc
```

Use the `select` function to get the point cloud data of nearest neighbors.

```
ptCloudB = select(ptCloud,indices);
```

Display the input point cloud and its nearest neighbors.

```
figure
pcshow(ptCloud)
hold on
plot3(ptCloudB.Location(:,1),ptCloudB.Location(:,2),ptCloudB.Location(:,3),'*')
hold off
legend('Point Cloud','Nearest Neighbors','Location','southoutside','Color',[1 1 1])
```



Input Arguments

`imagePoints` — Coordinates of image projection points

M-by-2 matrix

Coordinates of image projection points, specified as an *M*-by-2 matrix of (*x*, *y*) coordinates. *M* is the number of points and it must be greater than or equal to 6. The input image points must correspond to an undistorted image plane.

Data Types: `single` | `double`

`worldPoints` — 3-D world points

M-by-3 matrix

3-D world points, specified as an *M*-by-3 matrix of (*x*, *y*, *z*) coordinates. *M* is the number of points and it must be greater than or equal to 6. The input world coordinates must be non-coplanar points.

Data Types: `single` | `double`

Output Arguments

`camMatrix` — Camera projection matrix

4-by-3 matrix

Camera projection matrix, returned as a 4-by-3 matrix. The matrix maps the 3-D world points, in homogenous coordinates to the 2-D image coordinates of the projections onto the image plane.

Data Types: `double`

`reprojectionErrors` — Reprojection errors

M-by-1 vector

Reprojection errors, returned as a *M*-by-1 vector. The reprojection error is the error between the reprojected image points and the input image points. For more information on the computation of reprojection errors, see “Algorithms” on page 3-551.

Data Types: `double`

Tips

You can use the `estimateCameraMatrix` function to estimate a camera projection matrix:

- If the world-to-image point correspondences are known, and the camera intrinsics and extrinsics parameters are not known, you can use the `cameraMatrix` function.
- To compute 2-D image points from 3-D world points, refer to the equations in `camMatrix`.
- For use with the `findNearestNeighbors` object function of the `pointCloud` object. The use of a camera projection matrix speeds up the nearest neighbors search in a point cloud generated by an RGB-D sensor, such as Microsoft Kinect.

Algorithms

Given the world points $\boldsymbol{X}$ and the image points $\boldsymbol{x}$, the camera projection matrix C, is obtained by solving the equation

$$\lambda\boldsymbol{x} = C\boldsymbol{X}.$$

The equation is solved using the direct linear transformation (DLT) approach [1]. This approach formulates a homogeneous linear system of equations, and the solution is obtained through generalized eigenvalue decomposition.

Because the image point coordinates are given in pixel values, the approach for computing the camera projection matrix is sensitive to numerical errors. To avoid numerical errors, the input image point coordinates are normalized, so that their centroid is at the origin. Also, the root mean squared distance of the image points from the origin is `sqrt(2)`. These steps summarize the process for estimating the camera projection matrix.

1. Normalize the input image point coordinates with transform T.
2. Estimate camera projection matrix C^N from the normalized input image points.
3. Compute the denormalized camera projection matrix C as $C^N T^{-1}$.
4. Compute the reprojected image point coordinates $\boldsymbol{x}^E$ as $C\boldsymbol{X}$.
5. Compute the reprojection errors as $reprojectionErrors = |\boldsymbol{x} - \boldsymbol{x}^E|$.

Version History

Introduced in R2019a

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateCameraMatrix` function uses the postmultiply convention. Although there are no plans to remove `estimateCameraMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `estimateCameraProjection` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `estimateCameraMatrix` to `estimateCameraProjection`.
- The `estimateCameraProjection` function returns the camera projection matrix in a format consistent with the premultiply convention. The returned camera projection matrix is the transpose of the `camMatrix on page 3-0` argument.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Objects

`stereoParameters` | `cameraCalibrationErrors` | `intrinsicsEstimationErrors` | `extrinsicsEstimationErrors` | `cameraIntrinsics`

Functions

`estimateCameraProjection` | `estimateCameraParameters` | `showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `cameraProjection` | `estworldpose` | `estimateEssentialMatrix` | `estimateFundamentalMatrix` | `findNearestNeighbors`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

estimateCameraProjection

Estimate camera projection matrix from world-to-image point correspondences

Syntax

```
camProjection = estimateCameraProjection(imagePoints,worldPoints)
[camProjection,reprojectionErrors] = estimateCameraProjection(imagePoints,
worldPoints)
```

Description

`camProjection = estimateCameraProjection(imagePoints,worldPoints)` returns the camera projection matrix determined from known world points and their corresponding image projections by using the direct linear transformation (DLT) approach.

`[camProjection,reprojectionErrors] = estimateCameraProjection(imagePoints, worldPoints)` also returns the reprojection error that quantifies the accuracy of the projected image coordinates.

Examples

Estimate Camera Projection Matrix

Load a 3-D point cloud data captured by an RGB-D sensor into the workspace.

```
load("object3d.mat");
```

Remove points with `Inf` or `NaN` coordinates from the point cloud.

```
[validPtCloud,validIndices] = removeInvalidPoints(ptCloud);
```

Read the valid world point coordinates. Each entry specifies the (*x*, *y*, *z*) coordinates of a point in the point cloud.

```
worldPoints = validPtCloud.Location;
```

Define the corresponding image point coordinates as an orthographic projection of point cloud data onto the *yz*-plane.

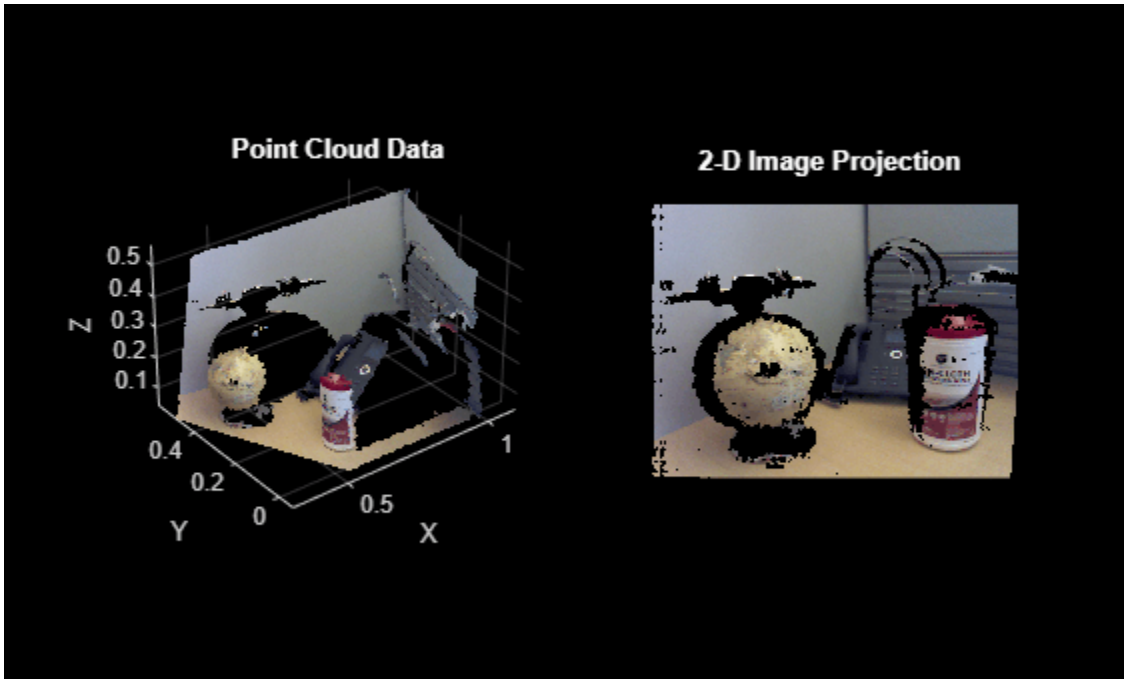
```
indices = 1:ptCloud.Count;
[y,z] = ind2sub([size(ptCloud.Location,1),size(ptCloud.Location,2)],indices);
imagePoints = [y(validIndices)' z(validIndices)'];
```

Generate the 2-D image projection by using the image point coordinates and their color values.

```
projImage = zeros(max(imagePoints(:,1)),max(imagePoints(:,2)),3);
rgb = validPtCloud.Color;
for j = 1:length(rgb)
    projImage(imagePoints(j,1),imagePoints(j,2),:) = rgb(j,:);
end
```

Display the point cloud data and the corresponding 2-D image projection.

```
figure
subplot(1,2,1)
pcshow(ptCloud)
xlabel("X");
ylabel("Y");
zlabel("Z");
title("Point Cloud Data",Color=[1 1 1])
subplot(1,2,2)
imshow(uint8(projImage))
title("2-D Image Projection",Color=[1 1 1])
```



Estimate the camera projection matrix and reprojection error by using the known world points and the image points.

```
[camProjection,reprojectionErrors] = estimateCameraProjection(imagePoints,worldPoints);
```

Use the estimated camera projection matrix as input to the `findNearestNeighbors` function and find the nearest neighbors of a query point.

```
point = [0.4 0.3 0.2];
K = 50;    % Specify the number of nearest neighbors to be determined
[indices,dists] = findNearestNeighbors(ptCloud,point,K,camProjection');
```

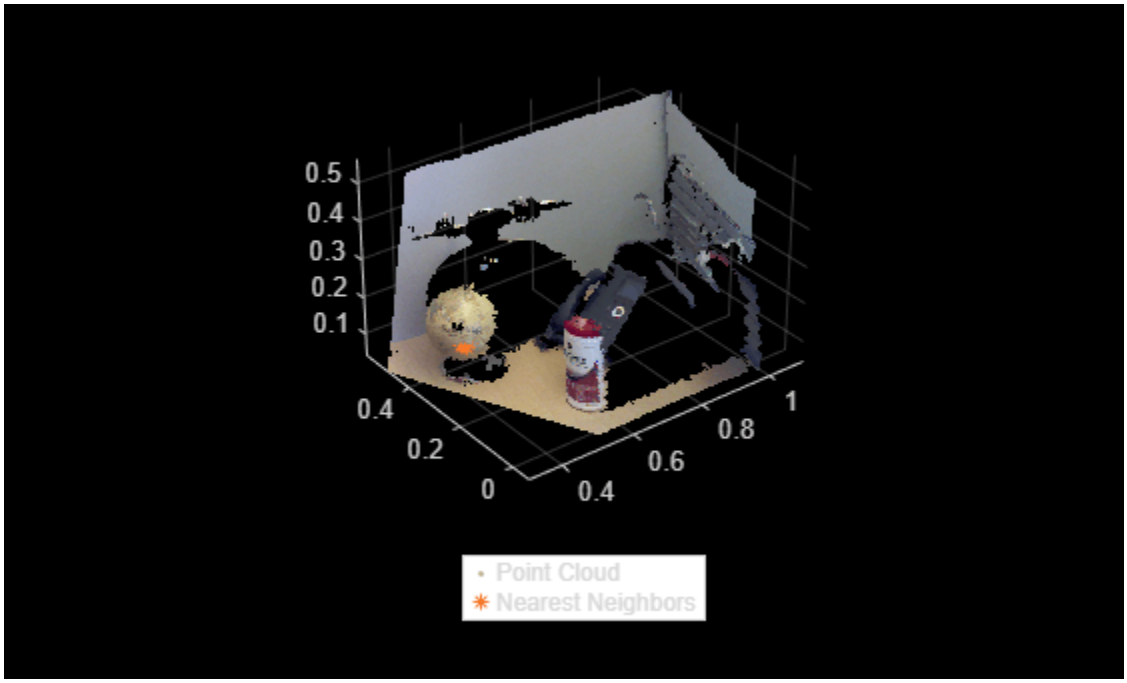
Use the `select` function to get the point cloud data of nearest neighbors.

```
ptCloudB = select(ptCloud,indices);
```

Display the input point cloud and its nearest neighbors.

```
figure
pcshow(ptCloud)
hold on
```

```
plot3(ptCloudB.Location(:,1),ptCloudB.Location(:,2),ptCloudB.Location(:,3),"*")
hold off
legend("Point Cloud","Nearest Neighbors",Location="southoutside",Color=[1 1 1])
```



Input Arguments

`imagePoints` — Coordinates of image projection points

M-by-2 matrix

Coordinates of image projection points, specified as an *M*-by-2 matrix of (*x*, *y*) coordinates. *M* is the number of points and it must be greater than or equal to 6. The input image points must correspond to an undistorted image plane.

Data Types: `single` | `double`

`worldPoints` — 3-D world points

M-by-3 matrix

3-D world points, specified as an *M*-by-3 matrix of non-coplanar (*x*, *y*, *z*) coordinates. *M* is the number of points and must be greater than or equal to 6.

Data Types: `single` | `double`

Output Arguments

`camProjection` — Camera projection matrix

3-by-4 matrix

Camera projection matrix, returned as a 3-by-4 matrix. The matrix maps the 3-D world points, in homogenous coordinates, to the 2-D image coordinates of the projections onto the image plane.

Using the camera projection matrix and homogeneous coordinates, you can project a world point onto the image.

$w \times [x,y,1]' = \text{camProjection} \times [X,Y,Z,1]'$

(X,Y,Z) are the world coordinates of a point
(x,y) are the coordinates of the corresponding image point
w is an arbitrary scale factor

Data Types: `double`

`reprojectionErrors` — Reprojection errors
M-by-1 vector

Reprojection errors, returned as a *M*-by-1 vector. The reprojection error is the error between the reprojected image points and the input image points. For more information on the computation of reprojection errors, see “Algorithms” on page 3-551.

Data Types: `double`

Tips

- This function assumes the camera has negligible lens distortion. If there is lens distortion, calibrate the camera and use the `estworldpose` and `cameraProjection` functions instead.
- If the camera intrinsics and extrinsics parameters are known, you can estimate the camera projection matrix using the `cameraProjection` function.
- You can use the camera projection matrix, `camProjection`, with the `findNearestNeighbors` function to speed up the nearest neighbors search in a point cloud generated by an RGB-D sensor, such as Microsoft Kinect.

Algorithms

Given the world points $\boldsymbol{X}$ and the image points $\boldsymbol{x}$, the camera projection matrix C, is obtained by solving the equation

$\lambda \boldsymbol{x} = C\boldsymbol{X}$.

The equation is solved using the direct linear transformation (DLT) approach [1]. This approach formulates a homogeneous linear system of equations, and the solution is obtained through generalized eigenvalue decomposition.

Because the image point coordinates are given in pixel values, the approach for computing the camera projection matrix is sensitive to numerical errors. To avoid numerical errors, the input image point coordinates are normalized, so that their centroid is at the origin. Also, the root mean squared distance of the image points from the origin is `sqrt(2)`. These steps summarize the process for estimating the camera projection matrix.

1. Normalize the input image point coordinates with transform T.
2. Estimate camera projection matrix C^N from the normalized input image points.
3. Compute the denormalized camera projection matrix C as $C^N T^{-1}$.
4. Compute the reprojected image point coordinates $\boldsymbol{x}^{\boldsymbol{E}}$ as $C\boldsymbol{X}$.
5. Compute the reprojection errors as
   $reprojectionErrors = |\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{E}}|$.

Version History

Introduced in R2022b

R2022b: Recommended over `estimateCameraMatrix`

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateCameraMatrix` function uses the postmultiply convention. Although there are no plans to remove `estimateCameraMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `estimateCameraProjection` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions

`cameraProjection` | `estimateCameraParameters` | `estworldpose` | `estimateEssentialMatrix` | `estimateFundamentalMatrix`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Depth Estimation from Stereo Video”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Camera Calibration?”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

estimateExtrinsics

Calculate location of calibrated camera

Syntax

```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,intrinsics)
```

Description

`camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,intrinsics)` returns the camera extrinsics, which are 3-D rigid transformation that enables you to transform points from the world coordinate to the camera coordinate system.

Examples

Calculate Camera Extrinsics

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration","slr"));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0). The square size is in millimeters.

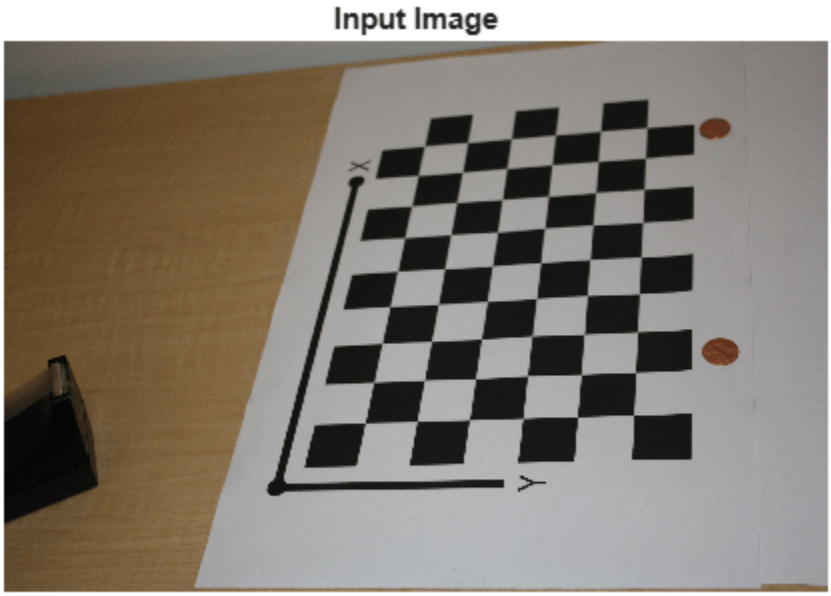
```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1) size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
intrinsics = cameraParams.Intrinsics;
```

Load an image at a new location.

```
imOrig = readimage(images,9);
figure
imshow(imOrig)
title("Input Image")
```



Undistort the image.

```
[im,newIntrinsics] = undistortImage(imOrig,intrinsics,OutputView="full");
```

Find the reference object in the new image.

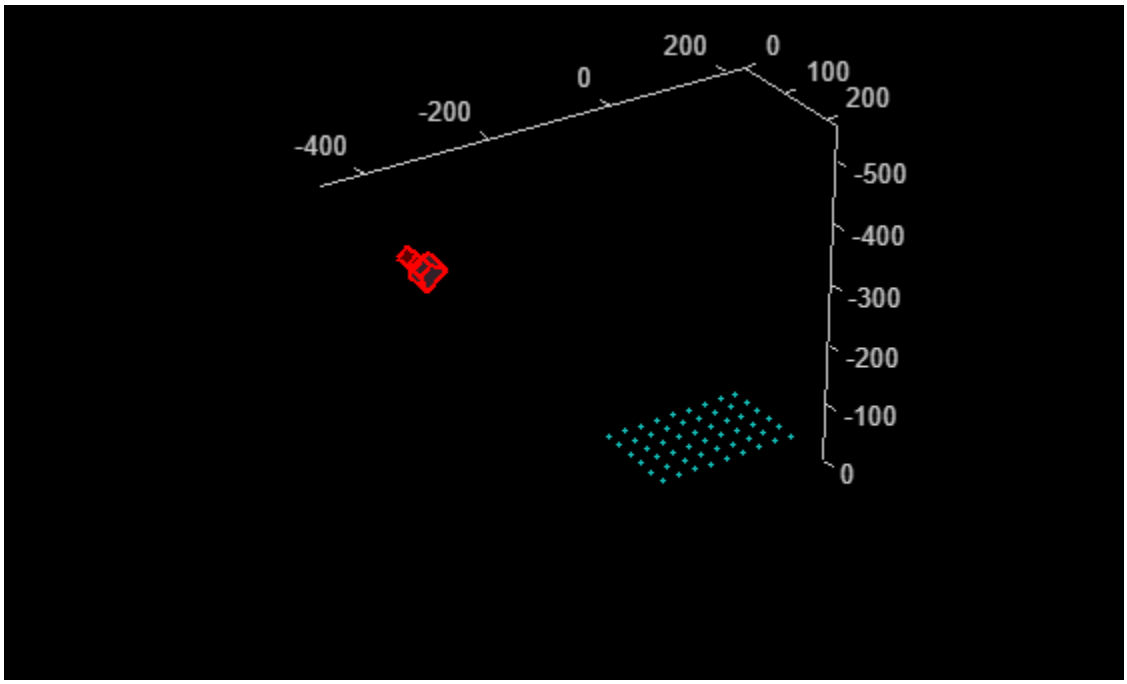
```
[imagePoints,patternDims] = detectCheckerboardPoints(im);
```

Calculate new extrinsics.

```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,newIntrinsics);
```

Calculate the camera pose.

```
camPose = extr2pose(camExtrinsics);
figure
plotCamera(AbsolutePose=camPose,Size=20);
hold on
pcshow([worldPoints,zeros(size(worldPoints,1),1)], ...
  VerticalAxisDir="down",MarkerSize=40);
```



Input Arguments

`imagePoints` — Image coordinates of points

M-by-2 array

Image coordinates of points, specified as an *M*-by-2 array. The array contains *M* number of [*x*, *y*] coordinates. The `imagePoints` and `worldPoints` inputs must both be `double` or both be `single`.

Data Types: `single` | `double`

`worldPoints` — World coordinates corresponding to image coordinates

M-by-2 matrix

World coordinates corresponding to image coordinates, specified as an *M*-by-2 matrix. The `imagePoints` and `worldPoints` inputs must both be `double` or both be `single`. The function assumes that the points are coplanar with $z = 0$ and the number of points, *M*, must be at least 4.

Data Types: `single` | `double`

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | `fisheyeIntrinsics` object | `cameraIntrinsicsKB` object

Camera intrinsics, specified as a `cameraIntrinsics`, `fisheyeIntrinsics`, or a `cameraIntrinsicsKB` object. The object stores information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

Output Arguments

`camExtrinsics` — Camera extrinsics

`rigidtform3d` object

Camera extrinsics, returned as a `rigidtform3d` object. The `"R"` and the `"Translation"` properties of the object represent the orientation and location of the camera.

Tips

- This function does not account for lens distortion when `intrinsics` is a `cameraIntrinsics` object. In this case, you can either undistort the image using the `undistortImage` function before detecting the points, or you can undistort the detected points using the `undistortPoints` function.

Algorithms

The `estimateExtrinsics` function uses two different algorithms to compute the extrinsics depending on whether `worldPoints` are specified as an *M*-by-2 matrix. Use an *M*-by-2 matrix for coplanar points where $z = 0$.

The `estimateExtrinsics` function computes the rotation matrix and translation vector for a single image in closed form. During calibration, the extrinsics are estimated numerically to minimize the reprojection errors for all calibration images. Therefore, using the `estimateExtrinsics` function on one of the calibration images returns rotation matrix and translation vector slightly different from the ones obtained during calibration.

Version History

Introduced in R2022b

R2022b: Recommended over `extrinsics`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `extrinsics` function uses the postmultiply convention. Although there are no plans to remove `extrinsics` at this time, you can streamline your geometric transformation workflows by switching to the `estimateExtrinsics` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2024a: Added support for the `cameraIntrinsicsKB` object

Support added for the `cameraIntrinsicsKB` object as the `instrinsics` input.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.
- Use of a `fisheyeIntrinsics` object as the `intrinsics` input is not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`estimateCameraParameters` | `triangulate` | `undistortPoints` | `cameraProjection` | `estimateFisheyeParameters` | `plotCamera` | `img2world2d` | `world2img` | `extr2pose` | `pose2extr`

Objects
`cameraIntrinsicsKB` | `cameraParameters` | `cameraIntrinsics` | `stereoParameters` | `fisheyeIntrinsics`

Topics
“What Is Camera Calibration?”
“Fisheye Calibration Basics”
“Code Generation for Depth Estimation from Stereo Video”

extractLBPFeatures

Extract local binary pattern (LBP) features

Syntax

```
features = extractLBPFeatures(I)
features = extractLBPFeatures(I,Name,Value)
```

Description

`features = extractLBPFeatures(I)` returns extracted uniform local binary pattern (LBP) from a grayscale image. The LBP features encode local texture information.

`features = extractLBPFeatures(I,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

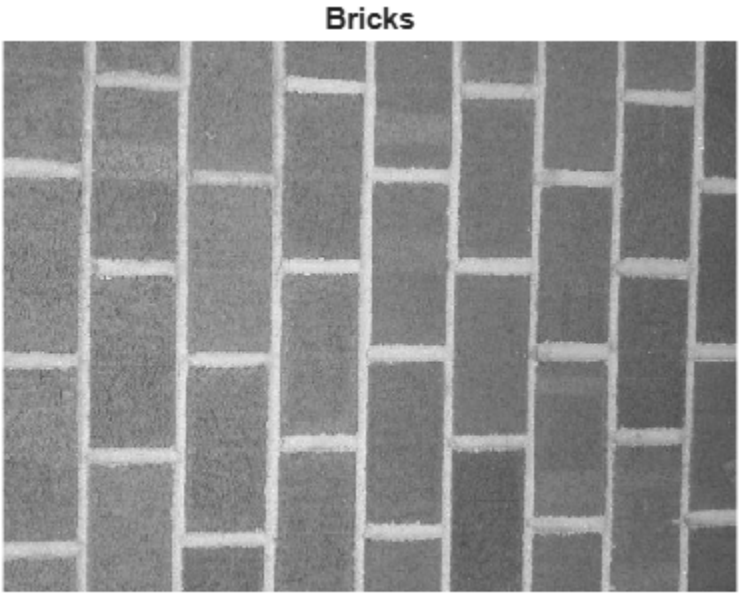
Using LBP Features to Differentiate Images by Texture

Read images that contain different textures.

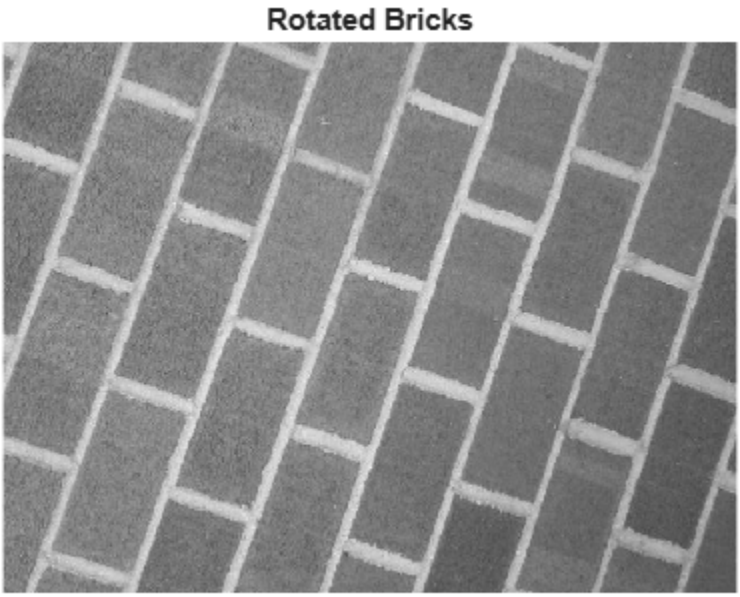
```
brickWall = imread('bricks.jpg');
rotatedBrickWall = imread('bricksRotated.jpg');
carpet = imread('carpet.jpg');
```

Display the images.

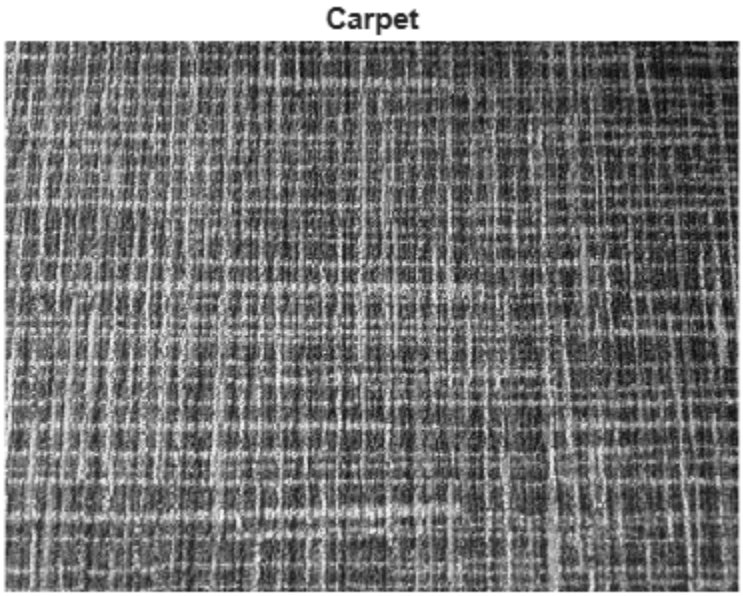
```
figure
imshow(brickWall)
title('Bricks')
```




```
figure
imshow(rotatedBrickWall)
title('Rotated Bricks')
```




```
figure
imshow(carpet)
title('Carpet')
```



Extract LBP features from the images to encode their texture information.

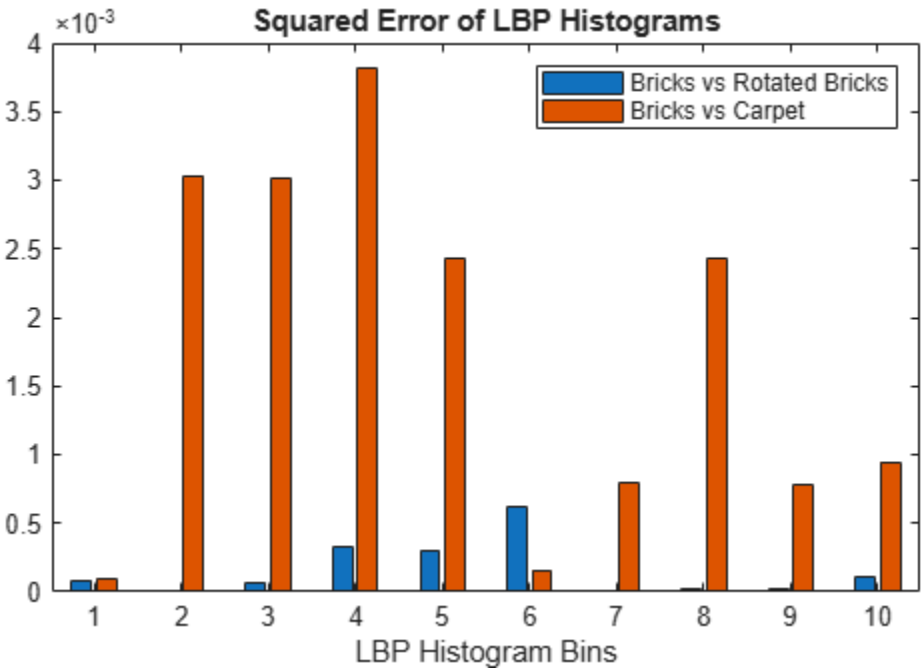
```
lbpBricks1 = extractLBPFeatures(brickWall,'Upright',false);
lbpBricks2 = extractLBPFeatures(rotatedBrickWall,'Upright',false);
lbpCarpet = extractLBPFeatures(carpet,'Upright',false);
```

Gauge the similarity between the LBP features by computing the squared error between them.

```
brickVsBrick = (lbpBricks1 - lbpBricks2).^2;
brickVsCarpet = (lbpBricks1 - lbpCarpet).^2;
```

Visualize the squared error to compare bricks versus bricks and bricks versus carpet. The squared error is smaller when images have similar texture.

```
figure
bar([brickVsBrick; brickVsCarpet]','grouped')
title('Squared Error of LBP Histograms')
xlabel('LBP Histogram Bins')
legend('Bricks vs Rotated Bricks','Bricks vs Carpet')
```



Apply L1 Normalization to LBP Features

Read in a sample image and convert it to grayscale.

```
I = imread('gantrycrane.png');
I = im2gray(I);
```

Extract unnormalized LBP features so that you can apply a custom normalization.

```
lbpFeatures = extractLBPFeatures(I,'CellSize',[32 32],'Normalization','None');
```

Reshape the LBP features into a *number of neighbors* -by- *number of cells* array to access histograms for each individual cell.

```
numNeighbors = 8;
numBins = numNeighbors*(numNeighbors-1)+3;
lbpCellHists = reshape(lbpFeatures,numBins,[]);
```

Normalize each LBP cell histogram using L1 norm.

```
lbpCellHists = bsxfun(@rdivide,lbpCellHists,sum(lbpCellHists));
```

Reshape the LBP features vector back to 1-by- *N* feature vector.

```
lbpFeatures = reshape(lbpCellHists,1,[]);
```

Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale image

Input image, specified as an *M*-by-*N* 2-D grayscale image that is real, and non-sparse.

Data Types: `logical` | `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumNeighbors',8`

Algorithm Parameters
The LBP algorithm parameters control how local binary patterns are computed for each pixel in the input image.

`NumNeighbors` — Number of neighbors
8 (default) | positive integer

Number of neighbors used to compute the LBP for each pixel in the input image, specified as the comma-separated pair consisting of '`NumNeighbors`' and a positive integer. The set of neighbors is selected from a circularly symmetric pattern around each pixel. Increase the number of neighbors to encode greater detail around each pixel. Typical values range from `4` to `24`.

`Radius` — Radius of circular pattern to select neighbors
1 (default) | positive integer

Radius of circular pattern used to select neighbors for each pixel in the input image, specified as the comma-separated pair consisting of '`Radius`' and a positive integer. To capture detail over a larger spatial scale, increase the radius. Typical values range from `1` to `5`.

`Upright` — Rotation invariance flag
`true` | logical scalar

Rotation invariance flag, specified as the comma-separated pair consisting of '`Upright`' and a logical scalar. When you set this property to `true`, the LBP features do not encode rotation information. Set '`Upright`' to `false` when rotationally invariant features are required.

`Interpolation` — Interpolation method
`'Linear'` (default) | `'Nearest'`

Interpolation method used to compute pixel neighbors, specified as the comma-separated pair consisting of '`Interpolation`' and either `'Linear'` or `'Nearest'`. Use `'Nearest'` for faster computation, but with less accuracy.

Histogram Parameters
The histogram parameters determine how the distribution of binary patterns is aggregated over the image to produce the output features.

`CellSize` — Cell size
`size(I)` (default) | 2-element vector

Cell size, specified as the comma-separated pair consisting of 'CellSize' and a 2-element vector. The number of cells is calculated as `floor(size(I)/CellSize)`.

`Normalization` — Type of normalization
`'L2'` (default) | `'None'`

Type of normalization applied to each LBP cell histogram, specified as the comma-separated pair consisting of 'Normalization' and either `'L2'` or `'None'`. To apply a custom normalization method as a post-processing step, set this value to `'None'`.

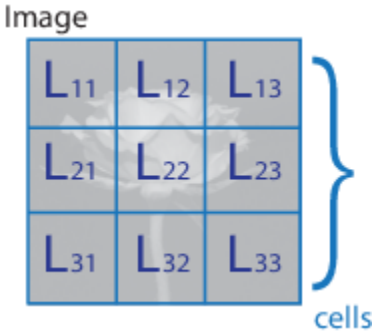
Output Arguments

`features` — LBP feature vector
1-by-*N* vector

LBP feature vector, returned as a 1-by-*N* vector of length *N* representing the number of features. LBP features encode local texture information, which you can use for tasks such as classification, detection, and recognition. The function partitions the input image into non-overlapping cells. To collect information over larger regions, select larger cell sizes. However, when you increase the cell size, you lose local detail. *N*, depends on the number of cells in the image, *numCells*, the number of neighbors, *P*, and the `Upright` parameter.

The number of cells is calculated as:
numCells = prod(`floor(size(I)/CellSize)`)

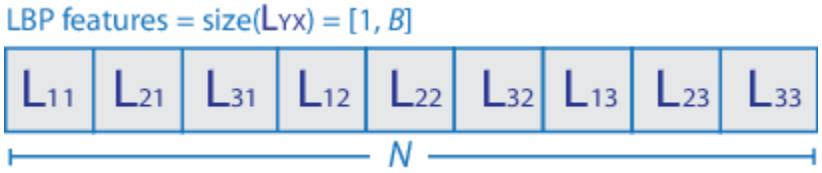
The figure shows an image with nine cell histograms. Each histogram describes an LBP feature.



The size of the histogram in each cell is [1,*B*], where *B* is the number of bins in the histogram. The number of bins depends on the `Upright` property and the number of neighbors, *P*.

Upright	Number of Bins
`true`	(P x *P*–1) + 3)
`false`	(*P* + 2)

The overall LBP feature length, *N*, depends on the number of cells and the number of bins, *B*:
N = *numCells* x *B*



Version History

Introduced in R2015b

References

[1] Ojala, T., M. Pietikainen, and T. Maenpaa. “Multiresolution Gray Scale and Rotation Invariant Texture Classification With Local Binary Patterns.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 24, Issue 7, July 2002, pp. 971-987.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not generate a platform-dependent library.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`extractHOGFeatures` | `extractFeatures` | `detectBRISKFeatures` | `detectMSERFeatures` | `matchFeatures` | `detectSURFFeatures` | `SURFPoints` | `MSERRegions` | `detectHarrisFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures`

Topics

“Local Feature Detection and Extraction”

configureKalmanFilter

Create Kalman filter for object tracking

Syntax

```
kalmanFilter = configureKalmanFilter(MotionModel,InitialLocation,
InitialEstimateError,MotionNoise,MeasurementNoise)
```

Description

`kalmanFilter = configureKalmanFilter(MotionModel,InitialLocation, InitialEstimateError,MotionNoise,MeasurementNoise)` returns a `vision.KalmanFilter` object configured to track a physical object. This object moves with constant velocity or constant acceleration in an *M*-dimensional Cartesian space. The function determines the number of dimensions, *M*, from the length of the `InitialLocation` vector.

This function provides a simple approach for configuring the `vision.KalmanFilter` object for tracking a physical object in a Cartesian coordinate system. The tracked object may move with either constant velocity or constant acceleration. The statistics are the same along all dimensions. If you need to configure a Kalman filter with different assumptions, use the `vision.KalmanFilter` object directly.

Examples

Track an Occluded Object

Detect and track a ball using Kalman filtering, foreground detection, and blob analysis.

Create System objects to read the video frames, detect foreground physical objects, and display results.

```
videoReader = VideoReader('singleball.mp4');
videoPlayer = vision.VideoPlayer('Position',[100,100,500,400]);
foregroundDetector = vision.ForegroundDetector('NumTrainingFrames',10,...
                'InitialVariance',0.05);
blobAnalyzer = vision.BlobAnalysis('AreaOutputPort',false,...
                'MinimumBlobArea',70);
```

Process each video frame to detect and track the ball. After reading the current video frame, the example searches for the ball by using background subtraction and blob analysis. When the ball is first detected, the example creates a Kalman filter. The Kalman filter determines the ball?s location, whether it is detected or not. If the ball is detected, the Kalman filter first predicts its state at the current video frame. The filter then uses the newly detected location to correct the state, producing a filtered location. If the ball is missing, the Kalman filter solely relies on its previous state to predict the ball's current location.

```
  kalmanFilter = []; isTrackInitialized = false;
   while hasFrame(videoReader)
     colorImage  = readFrame(videoReader);
```

```
    foregroundMask = step(foregroundDetector,im2gray(im2single(colorImage)));
    detectedLocation = step(blobAnalyzer,foregroundMask);
    isObjectDetected = size(detectedLocation, 1) > 0;

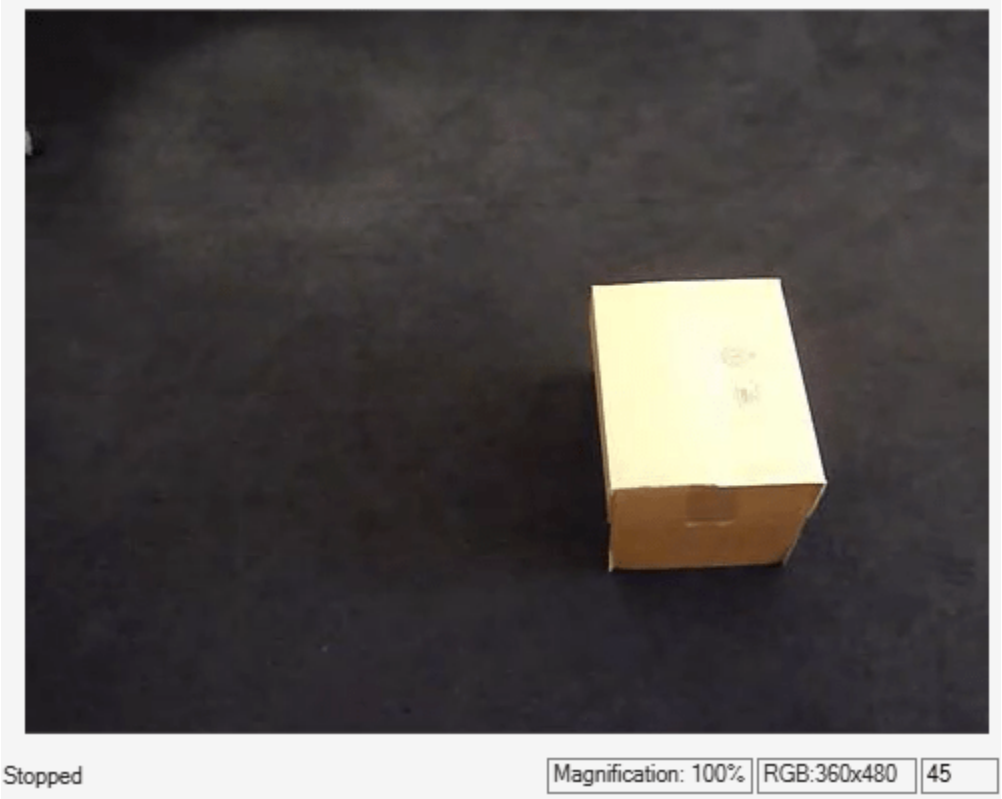
    if ~isTrackInitialized
      if isObjectDetected
        kalmanFilter = configureKalmanFilter('ConstantAcceleration',...
                 detectedLocation(1,:), [1 1 1]*1e5, [25, 10, 10], 25);
        isTrackInitialized = true;
      end
      label = ''; circle = zeros(0,3);
    else
      if isObjectDetected
        predict(kalmanFilter);
        trackedLocation = correct(kalmanFilter, detectedLocation(1,:));
        label = 'Corrected';
      else
        trackedLocation = predict(kalmanFilter);
        label = 'Predicted';
      end
      circle = [trackedLocation, 5];
    end

    colorImage = insertObjectAnnotation(colorImage,'circle',...
               circle,label,'AnnotationColor','red');
    step(videoPlayer,colorImage);
    pause(0.1);
  end
```



Release resources.

```
release(videoPlayer);
```



Input Arguments

`MotionModel` — Motion model

`'ConstantVelocity'` | `'ConstantAcceleration'`

Motion model, specified as `'ConstantVelocity'` or `'ConstantAcceleration'`. The motion model you select applies to all dimensions. For example, for the 2-D Cartesian coordinate system. This mode applies to both X and Y directions.

Data Types: `char`

`InitialLocation` — Initial location of object

vector

Initial location of object, specified as a numeric vector. This argument also determines the number of dimensions for the coordinate system. For example, if you specify the initial location as a two-element vector, $[x_0, y_0]$, then a 2-D coordinate system is assumed.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`InitialEstimateError` — Initial estimate uncertainty variance

2-element vector | 3-element vector

Initial estimate uncertainty variance, specified as a two- or three-element vector. The initial estimate error specifies the variance of the initial estimates of location, velocity, and acceleration of the tracked object. The function assumes a zero initial velocity and acceleration for the object, at the location you set with the `InitialLocation` property. You can set the `InitialEstimateError` to an approximated value when you have the ability to measure location, velocity, or acceleration: (*assumed values – actual values*)^2 + *the variance of the values*

In many visual object tracking applications, it is common to only have a measurement of the object location from an object detector. In this case, you can set the first element (*LocationVariance*) of `InitialEstimationError` to the same value as `MeasurementNoise`. The velocity and acceleration variance values should be set to large values to indicate that the initial state of these are unknown. For example, if you detect the 2-D location of an object with variance of `1` and use a constant velocity motion model, then you can set *LocationVariance* to `1` and the second element *VelocityVariance* to a high value such as `1e4`.

The value of this property affects the Kalman filter for the first few detections. Later, the estimate error is determined by the noise and the input data. A larger value for the initial estimate error helps the Kalman filter to adapt to the detection results faster. However, a larger value also prevents the Kalman filter from removing noise from the first few detections.

Specify the initial estimate error as a two-element vector for constant velocity or a three-element vector for constant acceleration:

MotionModel	InitialEstimateError
`ConstantVelocity`	[*LocationVariance*, *VelocityVariance*]
`ConstantAcceleration`	[*LocationVariance*, *VelocityVariance*, *AccelerationVariance*]

Data Types: `double` | `single`

`MotionNoise` — Deviation of selected and actual model
2-element vector | 3-element vector

Deviation of selected and actual model, specified as a two- or three-element vector. The motion noise specifies the tolerance of the Kalman filter for the deviation from the chosen model. This tolerance compensates for the difference between the object's actual motion and that of the model you choose whether the difference is from random deviation or just a small change in the motion (e.g. direction or speed). Increasing this value may cause the Kalman filter to change its state to fit the detections. Such an increase may prevent the Kalman filter from removing enough noise from the detections. The values of this property stay constant and therefore may affect the long-term performance of the Kalman filter.

MotionModel	MotionNoise
`ConstantVelocity`	[*LocationVariance*, *VelocityVariance*]
`ConstantAcceleration`	[*LocationVariance*, *VelocityVariance*, *AccelerationVariance*]

Data Types: `double` | `single`

`MeasurementNoise` — Variance inaccuracy of detected location
scalar

Variance inaccuracy of detected location, specified as a scalar. It is directly related to the technique used to detect the physical objects. For instance, if the detected location error is approximately 5 pixels, then set the measurement noise to `5^2`. Increasing the `MeasurementNoise` value enables the Kalman filter to remove more noise from the detections. However, it may also cause the Kalman filter to adhere too closely to the motion model you chose, putting less emphasis on the detections. The values of this property stay constant, and therefore may affect the long-term performance of the Kalman filter.

Data Types: `double` | `single`

Output Arguments

`kalmanFilter` — Configured Kalman filter tracking

object

Configured Kalman filter, returned as a `vision.KalmanFilter` object for tracking.

Algorithms

This function provides a simple approach for configuring the vision.KalmanFilter object for tracking. The Kalman filter implements a discrete time, linear State-Space System. The `configureKalmanFilter` function sets the `vision.KalmanFilter` object properties.

The `InitialLocation` property corresponds to the measurement vector used in the Kalman filter state-space model. This table relates the measurement vector, M, to the state-space model for the Kalman filter.		
State transition model, *A*, and Measurement model, *H*		
The state transition model, A, and the measurement model, H of the state-space model, are set to block diagonal matrices made from M identical submatrices As and Hs, respectively: A = `blkdiag`(As _1, As _2, ..., As _M) H = `blkdiag`(Hs _1, Hs _2, ..., Hs _M)		
The submatrices As and Hs are described below:		
MotionModel	***A*s**	***H*s**
`'ConstantVelocity'`	[1 1; 0 1]	[1 0]
`'ConstantAcceleration'`	[1 1 0.5; 0 1 1; 0 0 1]	[1 0 0]
The Initial State, *x*:		
MotionModel	**Initial state, *x***	
`'ConstantVelocity'`	[`InitialLocation`(1), 0, ..., `InitialLocation`(M), 0]	
`'ConstantAcceleration'`	[`InitialLocation`(1), 0, 0, ..., `InitialLocation`(M), 0, 0]	
The initial state estimation error covariance matrix, *P*:		
P = `diag`(`repmat`(`InitialError`, [1, M]))		

The process noise covariance, *Q*:
Q = `diag(repmat(MotionNoise, [1, M]))`
The measurement noise covariance, *R*:
R = `diag(repmat(MeasurementNoise, [1, M]))`.

Version History

Introduced in R2012b

See Also

`vision.BlobAnalysis` | `vision.ForegroundDetector` | `vision.KalmanFilter`

Topics

“Use Kalman Filter for Object Tracking”
“Tuning a Multi-Object Tracker” (Sensor Fusion and Tracking Toolbox)
“Automatically Tune Tracking Filter for Multi-Object Tracker” (Sensor Fusion and Tracking Toolbox)
“Multiple Object Tracking”

detectBRISKFeatures

Detect BRISK features

Syntax

```
points = detectBRISKFeatures(I)
points = detectBRISKFeatures(I,Name,Value)
```

Description

`points = detectBRISKFeatures(I)` returns a `BRISKPoints` object, `points`. The object contains information about BRISK features detected in a 2-D grayscale or binary input image, `I`. The `detectBRISKFeatures` function uses a Binary Robust Invariant Scalable Keypoints (BRISK) algorithm to detect multiscale corner features.

`points = detectBRISKFeatures(I,Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Detect BRISK Points in an Image and Mark Their Locations

Read the image.

```
I = imread('cameraman.tif');
```

Find the BRISK points.

```
points = detectBRISKFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(points.selectStrongest(20));
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinQuality',0.1,'ROI', [50,150,100,200]` specifies that the detector must use a 10% minimum accepted quality of corners within the designated region of interest. This region of interest is located at *x*=`50`, *y*=`150`. The ROI has a width of `100` pixels and a height of `200` pixels.

`MinContrast` — Minimum intensity difference

0.2 (default) | scalar

Minimum intensity difference between a corner and its surrounding region, specified as the comma-separated pair consisting of '`MinContrast`' and a scalar in the range (0 1). The minimum contrast value represents a fraction of the maximum value of the image class. Increase this value to reduce the number of detected corners.

`MinQuality` — Minimum accepted quality of corners
0.1 (default) | scalar

Minimum accepted quality of corners, specified as the comma-separated pair consisting of '`MinQuality`' and a scalar value in the range [0,1]. The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Increase this value to remove erroneous corners.

`NumOctaves` — Number of octaves
4 (default) | scalar

Number of octaves to implement, specified as a comma-separated pair consisting of '`NumOctaves`' and an integer scalar, greater than or equal to `0`. Increase this value to detect larger blobs. Recommended values are between `1` and `4`. When you set `NumOctaves` to `0`, the function disables multiscale detection. It performs the detection at the scale of the input image, `I`.

`ROI` — Rectangular region
[1 1 `size`(`I`,2) `size`(`I`,1)] (default) | vector

Rectangular region for corner detection, specified as a comma-separated pair consisting of '`ROI`' and a vector of the format [*x y width height*]. The first two integer values [*x y*] represent the location of the upper-left corner of the region of interest. The last two integer values represent the width and height.

Output Arguments

`points` — Brisk points
`BRISKPoints` object

Brisk points, returned as a `BRISKPoints` object. The object contains information about the point features detected in the 2-D grayscale or binary input image.

Version History

Introduced in R2014a

References

[1] Leutenegger, S., M. Chli and R. Siegwart. “BRISK: Binary Robust Invariant Scalable Keypoints”, *Proceedings of the IEEE International Conference*, ICCV, 2011.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

```
locations = points.Location;
```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectFASTFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `extractFeatures` | `extractHOGFeatures` | `detectFASTFeatures` | `matchFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

Topics
“Point Feature Types”

detectCheckerboardPoints

Detect checkerboard pattern in image

Syntax

```
[imagePoints,patternDims] = detectCheckerboardPoints(I)
[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(
imageFileNames)
[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(images)

[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints(
imageFileNames1,imageFileNames2)
[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints(images1,
images2)

[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints( ___ ,
Name,Value)
```

Description

Single Image Checkerboard Detection

`[imagePoints,patternDims] = detectCheckerboardPoints(I)` detects the keypoints of a checkerboard calibration pattern in a single image, a set of images, or stereo image pairs. The function returns the detected points `imagePoints`, and dimensions of the checkerboard `patternDims`.

`[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(imageFileNames)` detects a checkerboard pattern in a set of input images, provided as an array of file names.

`[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(images)` detects a checkerboard pattern in a set of input images, provided as an array of grayscale or truecolor images.

Stereo Pair Checkerboard Detection

`[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints(imageFileNames1,imageFileNames2)` detects a checkerboard pattern in stereo pairs of images, provided as cell arrays of file names.

`[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints(images1, images2)` detects a checkerboard pattern in stereo pairs of images, provided as arrays of grayscale or truecolor images.

Optional Arguments

`[imagePoints,patternDims,pairsUsed] = detectCheckerboardPoints( ___ , Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. Unspecified properties have default values.

Examples

Detect Checkerboard in Images with High Distortion

Create an `imageDatastore` containing calibration images from a GoPro camera.

```
imds = imageDatastore(fullfile(toolboxdir("vision"),"visiondata","calibration","gopro"));
```

Detect calibration pattern using the `HighDistortion` option, which is good to use with fisheye lens images.

```
[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(imds.Files(1:4),HighDistortion=t
```

Display detected points.

```
for i = 1:4
  % Read image
  I = readimage(imds,i);

  % Insert markers at detected point locations
  I = insertMarker(I,imagePoints(:,:,i),"o",MarkerColor="red",Size=10);

  % Display image
  subplot(2,2,i);
  imshow(I);
end
```



Detect Checkerboard in Image

Load an image containing a checkerboard pattern.

```
imageFileName = fullfile(toolboxdir("vision"),"visiondata","calibration","dslr","image01.jpg");
I = imread(imageFileName);
```

Detect the checkerboard points.

```
[imagePoints,patternDims] = detectCheckerboardPoints(I);
```

Display detected points.

```
J = insertText(I,imagePoints,1:size(imagePoints,1));
J = insertMarker(J,imagePoints,"o",MarkerColor="red",Size=5);
imshow(J);
title(sprintf("Detected a %d x %d Checkerboard",patternDims));
```



Detect Checkerboard in a Set of Image Files

Create a cell array of file names of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata",...
    "calibration","dslr"));
imageFileNames = images.Files;
```

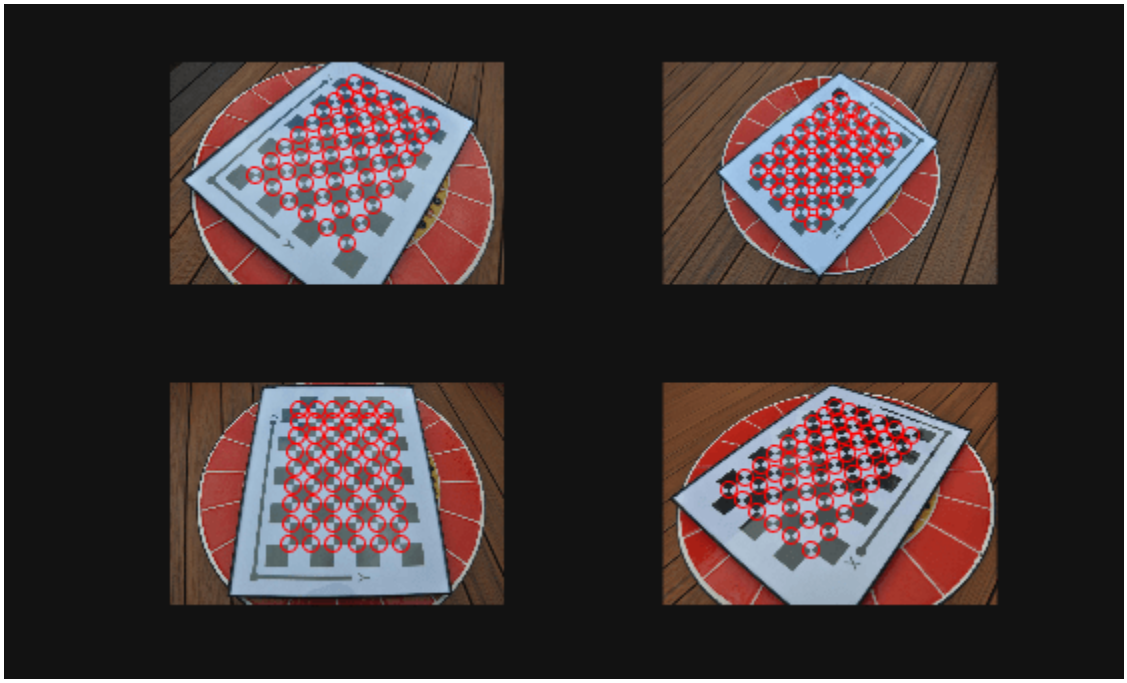
Detect calibration pattern in the images.

```
[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(imageFileNames,PartialDetections=
```

Display the detected points.

```
imageFileNames = imageFileNames(imagesUsed);
for i = 1:4
  I = imread(imageFileNames{i});
```

```
subplot(2, 2, i);
  imshow(I);
  hold on;
  plot(imagePoints(:,1,i),imagePoints(:,2,i),"ro");
end
```



Detect Checkerboard in Stereo Images

Read in stereo images.

```
numImages = 4;
images1 = cell(1,numImages);
images2 = cell(1,numImages);
for i = 1:numImages
    images1{i} = fullfile(matlabroot,'toolbox','vision',...
        'visiondata','calibration','stereo','left',sprintf('left%02d.png',i));
    images2{i} = fullfile(matlabroot,'toolbox','vision',...
        'visiondata','calibration','stereo','right',sprintf('right%02d.png',i));
end
```

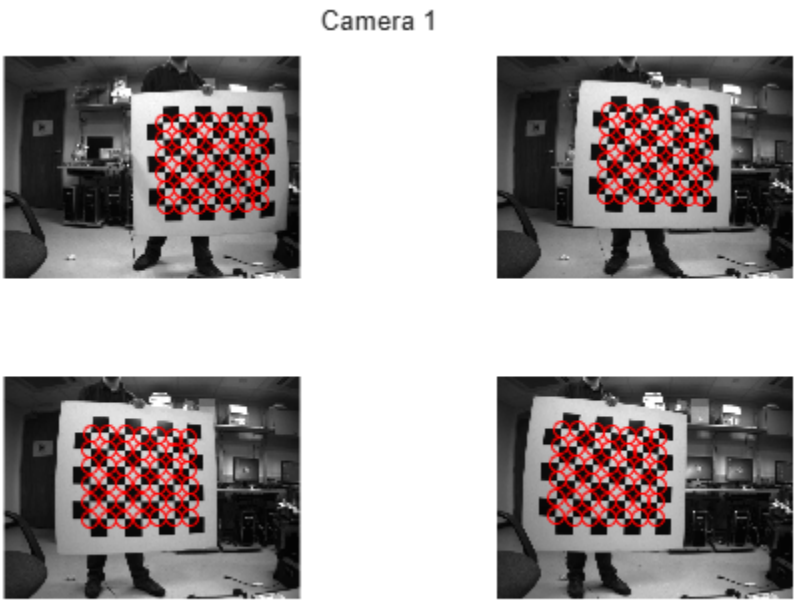
Detect the checkerboards in the images.

```
[imagePoints,patternDims,pairsUsed] = ...
    detectCheckerboardPoints(images1,images2);
```

Display points from `images1`.

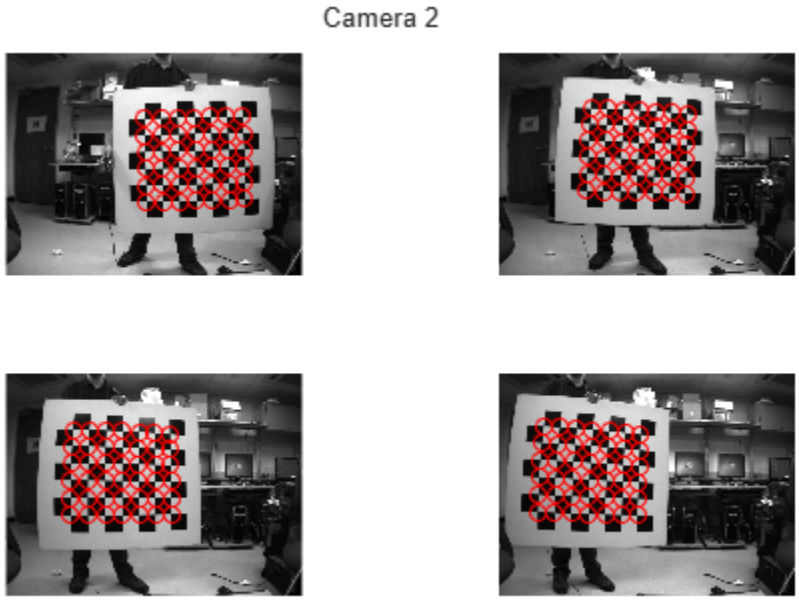
```
images1 = images1(pairsUsed);
figure;
for i = 1:numel(images1)
      I = imread(images1{i});
```

```
      subplot(2,2,i);
      imshow(I);
      hold on;
      plot(imagePoints(:,1,i,1),imagePoints(:,2,i,1),'ro');
end
annotation('textbox',[0 0.9 1 0.1],'String','Camera 1',...
    'EdgeColor','none','HorizontalAlignment','center')
```



Display points from `images2`.

```
images2 = images2(pairsUsed);
figure;
for i = 1:numel(images2)
      I = imread(images2{i});
      subplot(2, 2, i);
      imshow(I);
      hold on;
      plot(imagePoints(:,1,i,2),imagePoints(:,2,i,2),'ro');
end
annotation('textbox',[0 0.9 1 0.1],'String','Camera 2',...
    'EdgeColor','none','HorizontalAlignment','center')
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image, specified as either an *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale. The input image must be real and nonsparse. The function can detect checkerboards with a minimum size of 4-by-4 squares.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`imageFileNames` — Image file names

N-element cell array of character vectors | *N*-element array of strings

Image file names, specified as an *N*-element cell array of *N* file names.

`imageFileNames1` — File names for camera 1 images

N-element cell array of character vectors | *N*-element array of strings

File names for camera 1 images, specified as an *N*-element cell array of *N* file names. The images contained in this array must be in the same order as images contained in `imageFileNames2`, forming stereo pairs.

`imageFileNames2` — File names for camera 2 images

N-element cell array of character vectors | *N*-element array of strings

File names for camera 2 images, specified as an *N*-element cell array of *N* file names. The images contained in this array must be in the same order as images contained in `imageFileNames1`, forming stereo pairs.

`images` — Images
height-by-*width*-by-*color channel*-by-*number of frames* array

Images, specified as an *H*-by-*W*-by-*B*-by-*F* array containing a set of grayscale or truecolor images. The input dimensions are:
H represents the image height.
W represents the image width.
B represents the color channel. A value of `1` indicates a grayscale image, and a value of `3` indicates a truecolor image.
F represents the number of image frames.

`images1` — Stereo pair images from camera 1
height-by-*width*-by-*color channel*-by-*number of frames* array

Stereo pair images from camera 1, specified as an *H*-by-*W*-by-*B*-by-*F* array containing a set of grayscale or truecolor images. The input dimensions are:
H represents the image height.
W represents the image width.
B represents the color channel. A value of `1` indicates a grayscale image, and a value of `3` indicates a truecolor image.
F represents the number of image frames.

`images2` — Stereo pair images from camera 2
height-by-*width*-by-*color channel*-by-*number of frames* array

Stereo pair images from camera 2, specified as an *H*-by-*W*-by-*B*-by-*F* array containing a set of grayscale or truecolor images. The input dimensions are:
H represents the image height.
W represents the image width.
B represents the color channel. A value of `1` indicates a grayscale image, and a value of `3` indicates a truecolor image.
F represents the number of image frames.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinCornerMetric','0.15'`

`MinCornerMetric` — Minimum corner metric threshold
`0.15` | `0.12` | nonnegative scalar

Minimum corner metric threshold, specified as a nonnegative scalar. When the image is noisy or highly textured, increase this value to reduce the number of false corner detections. When you set the `'HighDistortion'` property to `false` the function sets the default value to `0.15`. When you set the `'HighDistortion'` property to `true` the function sets the default value to `0.12`. Decreasing the value results in an increase of corner detections.

`HighDistortion` — High distortion
`false` (default) | `true`

High distortion, specified as `false` or `true`. Set `'HighDistortion'` to `true` when the images contain a high level of distortion, which is typical of a wide field of view camera, such as a fisheye camera. Set `'HighDistortion'` to `false` when the images do not contain a high level of distortion. Setting `'HighDistortion'` to `true` can increase the resiliency to image distortion but decreases the processing speed.

`PartialDetections` — Partial detections
`true` (default) | `false`

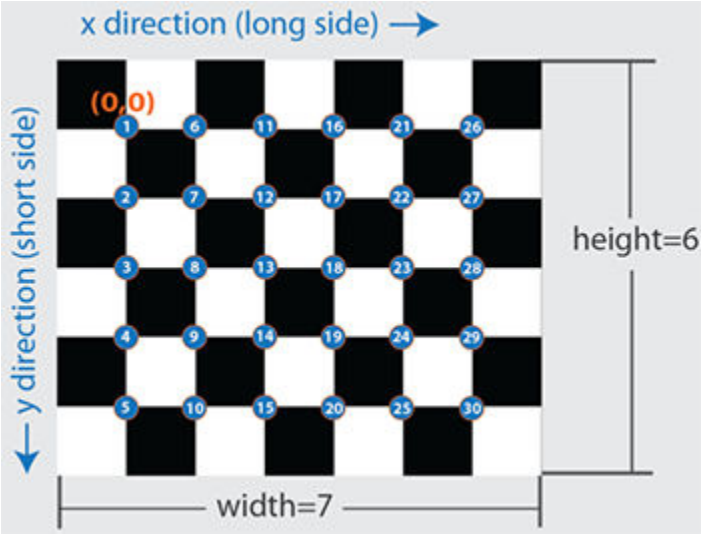
Partial detections, specified as `true` or `false`. Set `'PartialDetections'` to `true` to return partially detected checkerboards. The function fills missing keypoint detections with `[NaN,NaN]` coordinates. Set `'PartialDetections'` to `false` to discard partially detected checkerboards. This property is ignored for stereo image pairs.

Output Arguments

`imagePoints` — Detected checkerboard corner coordinates
M-by-2 matrix | *M*-by-2-by- *number of images* array | *M*-by-2-by-*number of pairs of images*-by-*number of cameras* array

Detected checkerboard corner coordinates, returned as an *M*-by-2 matrix for one image. For multiple images, points are returned as an *M*-by-2-by-*number of images* array, and for stereo pairs of images, the function returns points as an *M*-by-2-by-*number of pairs*-by-*number of cameras* array.

For stereo pairs, `imagePoints`(:,:,:,1) are the points from the first set of images, and `imagePoints`(:,:,:,2) are the points from the second set of images. The output contains *M* number of [*x y*] coordinates. Each coordinate represents a point where square corners are detected on the checkerboard. The number of points the function returns depends on the value of `patternDims`, which indicates the number of squares detected. The function detects the points with sub-pixel accuracy.



The function calculates the number of points, *M*, as follows:
M = `prod`(`patternDims`-1).
If the checkerboard cannot be detected:

```
imagePoints = []
patternDims = [0,0]
```

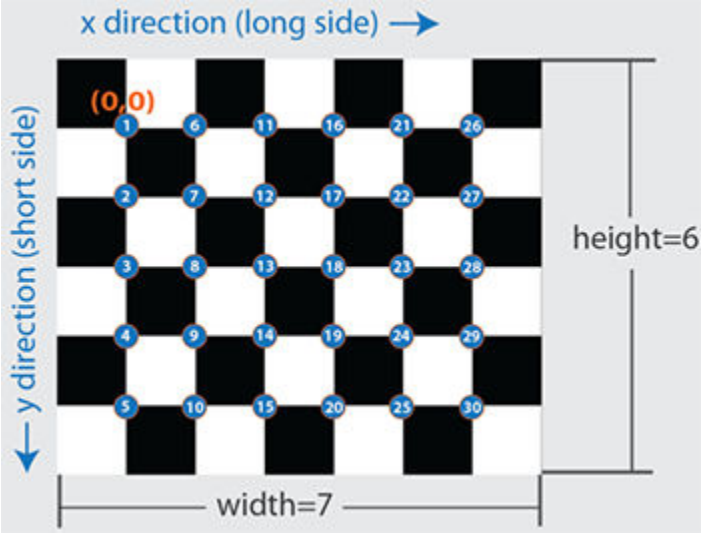
When you specify the `imageFileNames` input, the function can return `imagePoints` as an *M*-by-2-by-*N* array. In this array, *N* represents the number of images in which a checkerboard is detected. If a checkerboard cannot be detected, the function returns `imagePoints` as one of these two options:

For single camera images only:

- If the complete checkerboard cannot be detected, the function returns a partially detected checkerboard with `[NaN,NaN]` as the *x*-*y* coordinates for missing corners in `imagePoints`. This default behavior can be modified using the `'PartialDetections'` name-value argument.
- When possible, the function orients the partially detected checkerboard such that the location of the origin and the arrangement of the corners is consistent with the completely visible checkerboard. If the function cannot detect a complete checkerboard in any of the input images, the largest detected checkerboard is used as the reference checkerboard and the dimensions of this board is returned in `patternDims`.

`patternDims` — Pattern dimensions
2-element [*height*, *width*] vector

Pattern dimensions, returned as a 2-element [*height*, *width*] vector. The dimensions of the pattern are expressed in terms of the number of squares.



If a checkerboard cannot be detected, the function sets `patternDims` to `[0,0]`.

`imagesUsed` — Pattern detection
N-by-1 logical vector

Pattern detection, returned as an *N*-by-1 logical vector, where *N* is the number of images. The function returns `1` (true) if corners are successfully detected in an image, and `0` (false) if no corners are detected.

The function estimates the checkerboard corners based on a successful detection of the keypoints. If any step in the detection process fails, the function does not return corners for that image, and the corresponding value in the logical vector is false.

`pairsUsed` — Stereo pair pattern detection
N-by-1 logical vector

Stereo pair pattern detection, returned as an *N*-by-1 logical vector, where *N* is the number of images. The function returns `1` (true) if corners are successfully detected in a stereo image pair, and `0` (false) if no corners are detected.

For stereo pair pattern detection, the checkerboard needs to be fully visible in both images for it to be detected. Unlike single camera calibration, partially detected checkerboards are rejected for stereo image pairs.

Version History

Introduced in R2014a

References

[1] Geiger, A., F. Moosmann, O. Car, and B. Schuster. "Automatic Camera and Range Sensor Calibration using a Single Shot," *International Conference on Robotics and Automation (ICRA)*, St. Paul, USA, May 2012.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation will not support specifying images as file names or cell arrays of file names. It supports only checkerboard detection in a single image or stereo pair of images. For example, these syntaxes are supported:
  - `detectCheckerboardPoints(I1)`
  - `detectCheckerboardPoints(I1,I2)`

  `I1` and `I2` are single grayscale or RGB images.
- Images must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

The `detectCheckerboardPoints` function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Objects
`cameraParameters` | `stereoParameters` | `fisheyeParameters`

Functions
`estimateFisheyeParameters` | `estimateCameraParameters` | `generateCheckerboardPoints` | `detectCharucoBoardPoints` | `detectAprilGridPoints` | `patternWorldPoints`

Topics
“Using the Single Camera Calibrator App”

detectFASTFeatures

Detect corners using FAST algorithm

Syntax

```
points = detectFASTFeatures(I)
points = detectFASTFeatures(I,Name,Value)
```

Description

`points = detectFASTFeatures(I)` returns a `cornerPoints` object `points` that contains information about corner features detected in the 2-D grayscale or binary input using the Features from Accelerated Segment Test (FAST) algorithm.

`points = detectFASTFeatures(I,Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Find Corner Points in an Image Using the FAST Algorithm

Read the image.

```
I = imread('cameraman.tif');
```

Find the corners.

```
corners = detectFASTFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinQuality','0.01','ROI', [50,150,100,200]` specifies that the detector must use a 1% minimum accepted quality of corners within the designated region of interest. This region of interest is located at *x*=`50`, *y*=`150`. The ROI has a width of `100` pixels, and a height of `200` pixels.

`MinQuality` — Minimum accepted quality of corners

0.1 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of '`MinQuality`' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MinContrast` — Minimum intensity
0.2 (default)

Minimum intensity difference between corner and surrounding region, specified as the comma-separated pair consisting of '`MinContrast`' and a scalar value in the range (0,1).

The minimum intensity represents a fraction of the maximum value of the image class. Increasing the value reduces the number of detected corners.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ROI` — Rectangular region
[1 1 `size`(I,2) `size`(I,1)] (default) | vector

Rectangular region for corner detection, specified as a comma-separated pair consisting of '`ROI`' and a vector of the format [*x y width height*]. The first two integer values [*x y*] represent the location of the upper-left corner of the region of interest. The last two integer values represent the width and height.

Example: `'ROI', [50,150,100,200]`

Output Arguments

`points` — Corner points
`cornerPoints` object

Corner points object, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D grayscale or binary input image.

Version History

Introduced in R2013a

R2022a: GPU Use Being Removed for `detectFASTFeatures` function
Errors starting in R2022a

The `detectFASTFeatures` function no longer supports GPU. You can use the `detectHarrisFeatures` on the GPU to detect corner points instead. Although, it won't provide identical results, it might be suitable based on the application.

References

[1] Rosten, E., and T. Drummond. "Fusing Points and Lines for High Performance Tracking," *Proceedings of the IEEE International Conference on Computer Vision*, Vol. 2 (October 2005): pp. 1508–1511.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectBRISKFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `extractFeatures` | `extractHOGFeatures` | `matchFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

Topics
“Find Corner Points Using the Eigenvalue Algorithm” on page 3-599
“Find Corner Points Using the Harris-Stephens Algorithm” on page 3-595
“Point Feature Types”

detectHarrisFeatures

Detect corners using Harris–Stephens algorithm

Syntax

```
points = detectHarrisFeatures(I)
points = detectHarrisFeatures(I,Name,Value)
```

Description

`points = detectHarrisFeatures(I)` returns a `cornerPoints` object `points` that contains information about corner features detected in the 2-D grayscale or binary input using the Harris-Stephens algorithm.

`points = detectHarrisFeatures(I,Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Find Corner Points Using the Harris-Stephens Algorithm

Read the image.

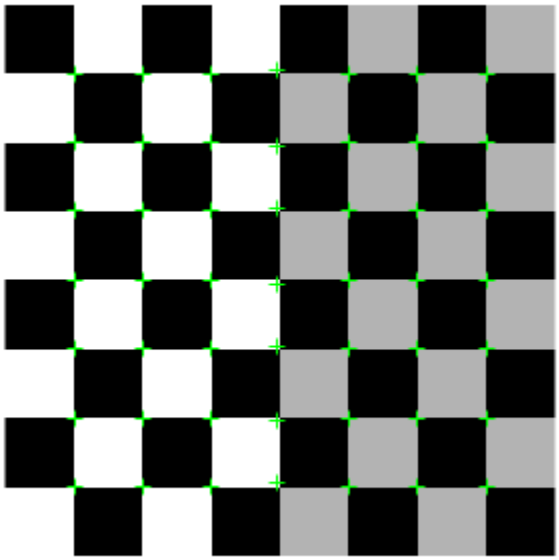
```
I = checkerboard;
```

Find the corners.

```
corners = detectHarrisFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinQuality','0.01','ROI', [50,150,100,200]` specifies that the detector must use a 1% minimum accepted quality of corners within the designated region of interest. This region of interest is located at x=`50`, y=`150`. The ROI has a width of `100` pixels and a height of `200` pixels.

`MinQuality` — Minimum accepted quality of corners

0.01 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of '`MinQuality`' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Example: `'MinQuality',0.01`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Gaussian filter dimension
5 (default)

Gaussian filter dimension, specified as the comma-separated pair consisting of '`FilterSize`' and an odd integer value in the range [3, `min(size(I))`].

The Gaussian filter smooths the gradient of the input image.

The function uses the `FilterSize` value to calculate the filter's dimensions, `FilterSize`-by-`FilterSize`. It also defines the standard deviation of the Gaussian filter as `FilterSize`/3.

Example: `'FilterSize',5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ROI` — Rectangular region
[1 1 `size`(I,2) `size`(I,1)] (default) | vector

Rectangular region for corner detection, specified as a comma-separated pair consisting of '`ROI`' and a vector of the format [*x y width height*]. The first two integer values [*x y*] represent the location of the upper-left corner of the region of interest. The last two integer values represent the width and height.

Example: `'ROI',[50,150,100,200]`

Output Arguments

`points` — Corner points
`cornerPoints` object

Corner points object, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D input image.

Version History

Introduced in R2013a

References

[1] Harris, C., and M. Stephens, "A Combined Corner and Edge Detector," *Proceedings of the 4th Alvey Vision Conference*, August 1988, pp. 147-151.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

- `'FilterSize'` must be a compile-time constant.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

```
locations = points.Location;
```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `extractFeatures` | `extractHOGFeatures` | `detectFASTFeatures` | `matchFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

Topics
“Find Corner Points Using the Eigenvalue Algorithm” on page 3-599
“Find Corner Points in an Image Using the FAST Algorithm” on page 3-591
“Point Feature Types”

detectMinEigenFeatures

Detect corners using minimum eigenvalue algorithm

Syntax

```
points = detectMinEigenFeatures(I)
points = detectMinEigenFeatures(I,Name,Value)
```

Description

`points = detectMinEigenFeatures(I)` returns a `cornerPoints` object `points` that contains information about corner features detected in the 2-D grayscale or binary input using the minimum eigenvalue algorithm developed by Shi and Tomasi.

`points = detectMinEigenFeatures(I,Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Find Corner Points Using the Eigenvalue Algorithm

Read the image.

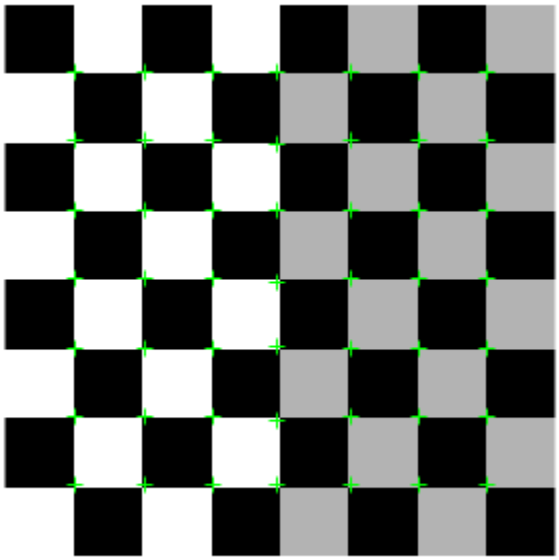
```
I = checkerboard;
```

Find the corners.

```
corners = detectMinEigenFeatures(I);
```

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinQuality','0.01','ROI', [50,150,100,200]` specifies that the detector must use a 1% minimum accepted quality of corners within the designated region of interest. This region of interest is located at x=`50`, y=`150`. The ROI has a width of `100` pixels, and a height of `200` pixels.

`MinQuality` — Minimum accepted quality of corners

0.01 (default)

Minimum accepted quality of corners, specified as the comma-separated pair consisting of '`MinQuality`' and a scalar value in the range [0,1].

The minimum accepted quality of corners represents a fraction of the maximum corner metric value in the image. Larger values can be used to remove erroneous corners.

Example: `'MinQuality', 0.01`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FilterSize` — Gaussian filter dimension
5 (default)

Gaussian filter dimension, specified as the comma-separated pair consisting of '`FilterSize`' and an odd integer value in the range [3, inf).

The Gaussian filter smooths the gradient of the input image.

The function uses the `FilterSize` value to calculate the filter's dimensions, `FilterSize`-by-`FilterSize`. It also defines the standard deviation as `FilterSize`/3.

Example: `'FilterSize', 5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ROI` — Rectangular region
[1 1 `size`(I,2) `size`(I,1)] (default) | vector

Rectangular region for corner detection, specified as a comma-separated pair consisting of '`ROI`' and a vector of the format [*x y width height*]. The first two integer values [*x y*] represent the location of the upper-left corner of the region of interest. The last two integer values represent the width and height.

Example: `'ROI', [50,150,100,200]`

Output Arguments

`points` — Corner points
`cornerPoints` object

Corner points, returned as a `cornerPoints` object. The object contains information about the point features detected in the 2-D grayscale or binary input image.

Version History

Introduced in R2013a

References

[1] Shi, J., and C. Tomasi, "Good Features to Track," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1994, pp. 593–600.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

- `'FilterSize'` must be a compile-time constant.
- Generated code for this function uses a precompiled platform-specific shared library.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectSURFFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `detectKAZEFeatures` | `detectORBFeatures` | `extractFeatures` | `extractHOGFeatures` | `detectFASTFeatures` | `matchFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

Topics
“Find Corner Points Using the Harris-Stephens Algorithm” on page 3-595
“Find Corner Points in an Image Using the FAST Algorithm” on page 3-591
“Point Feature Types”

detectMSERFeatures

Detect MSER features

Syntax

```
regions = detectMSERFeatures(I)
[regions,cc] = detectMSERFeatures(I)
[___] = detectMSERFeatures(I,Name,Value)
```

Description

`regions = detectMSERFeatures(I)` returns an `MSERRegions` object, `regions`, containing information about MSER features detected in the 2-D grayscale or binary input image, `I`. This object uses Maximally Stable Extremal Regions (MSER) algorithm to find regions.

Note For Simulink support using this function, you must enable the model configuration settings. To display the dialog box, in the Simulink Editor, select the **Modeling** tab and then select **Model Settings > Model Properties**. In the **Configuration Parameters** dialog box, select **Simulation Target** > `Advanced Parameters` > `Dynamic Memory allocation in MATLAB functions`.

`[regions,cc] = detectMSERFeatures(I)` optionally returns MSER regions in a connected component structure.

`[___] = detectMSERFeatures(I,Name,Value)` sets additional options specified by one or more name-value arguments.

Examples

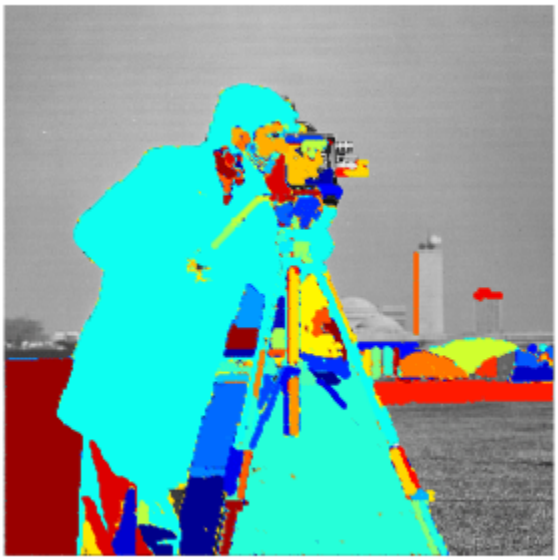
Find MSER Regions in an Image

Read image and detect MSER regions.

```
I = imread('cameraman.tif');
regions = detectMSERFeatures(I);
```

Visualize MSER regions which are described by pixel lists stored inside the returned 'regions' object.

```
figure; imshow(I); hold on;
plot(regions,'showPixelList',true,'showEllipses',false);
```



Display ellipses and centroids fit into the regions. By default, plot displays ellipses and centroids.

```
figure; imshow(I);
hold on;
plot(regions);
```



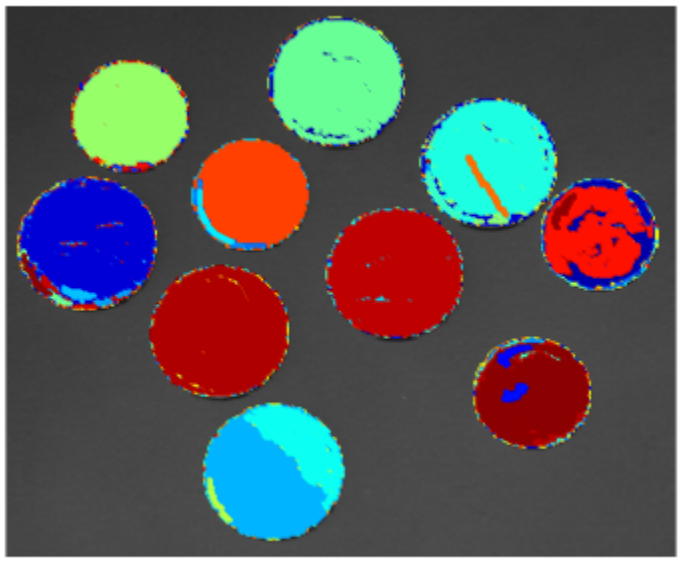
Find Circular MSER Regions

Detect MSER regions.

```
I = imread('coins.png');
[regions,mserCC] = detectMSERFeatures(I);
```

Show all detected MSER Regions.

```
figure
imshow(I)
hold on
plot(regions,'showPixelList',true,'showEllipses',false)
```



Measure the MSER region eccentricity to gauge region circularity.

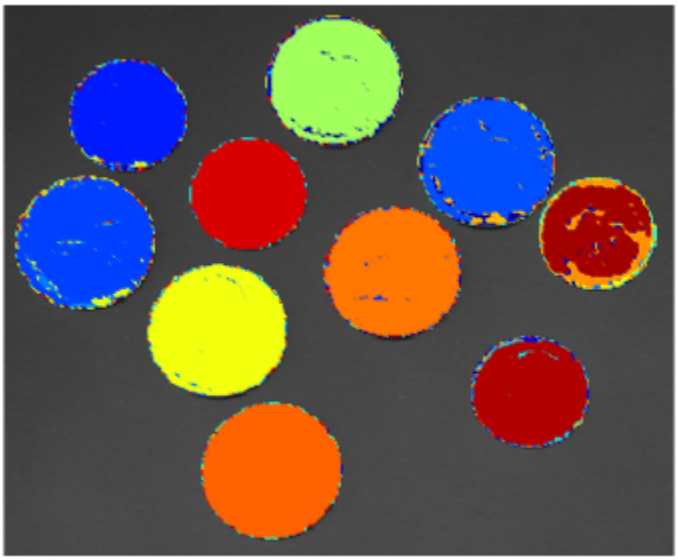
```
stats = regionprops('table',mserCC,'Eccentricity');
```

Threshold eccentricity values to only keep the circular regions. (Circular regions have low eccentricity.)

```
eccentricityIdx = stats.Eccentricity < 0.55;
circularRegions = regions(eccentricityIdx);
```

Show the circular regions.

```
figure
imshow(I)
hold on
plot(circularRegions,'showPixelList',true,'showEllipses',false)
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. It must be real and nonsparse. The function internally converts input images to `uint8` that are not `uint8` before looking for MSER regions.

Data Types: `uint8` | `int16` | `uint16` | `single` | `double` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'RegionAreaRange',[30 14000]`, specifies the size of the region in pixels.

`ThresholdDelta` — Step size between intensity threshold levels

2 (default) | percent numeric value

Step size between intensity threshold levels, specified as the comma-separated pair consisting of '`ThresholdDelta`' and a numeric value in the range (0,100]. This value is expressed as a percentage of the input data type range used in selecting extremal regions while testing for their stability. Decrease this value to return more regions. Typical values range from 0.8 to 4.

`RegionAreaRange` — Size of the region
`[30 14000]` (default) | two-element vector

Size of the region in pixels, specified as the comma-separated pair consisting of '`RegionAreaRange`' and a two-element vector. The vector, [*minArea maxArea*], allows the selection of regions containing pixels to be between *minArea* and *maxArea*, inclusive.

`MaxAreaVariation` — Maximum area variation between extremal regions
`0.25` (default) | positive scalar

Maximum area variation between extremal regions at varying intensity thresholds, specified as the comma-separated pair consisting of '`MaxAreaVariation`' and a positive scalar value. Increasing this value returns a greater number of regions, but they may be less stable. Stable regions are very similar in size over varying intensity thresholds. Typical values range from 0.1 to 1.0.

`ROI` — Rectangular region of interest
[1 1 `size`(`I`,2) `size`(`I`,1)] (default) | vector

Rectangular region of interest, specified as a vector. The vector must be in the format [*x y width height*]. When you specify an `ROI`, the function detects corners within the area located at [*x y*] of size specified by [*width height*]. The [*x y*] elements specify the upper left corner of the region.

Output Arguments

`regions` — MSER regions object
`MSERRegions` object (default)

MSER regions object, returned as a `MSERRegions` object. The object contains information about MSER features detected in the grayscale or binary input image.

`cc` — Connected component structure
structure

Connected component structure, returned as a structure with four fields. The connected component structure is useful for measuring region properties by using the `regionprops` function. You can also use the “properties” input of the `regionprops` function that provides shape measurements.

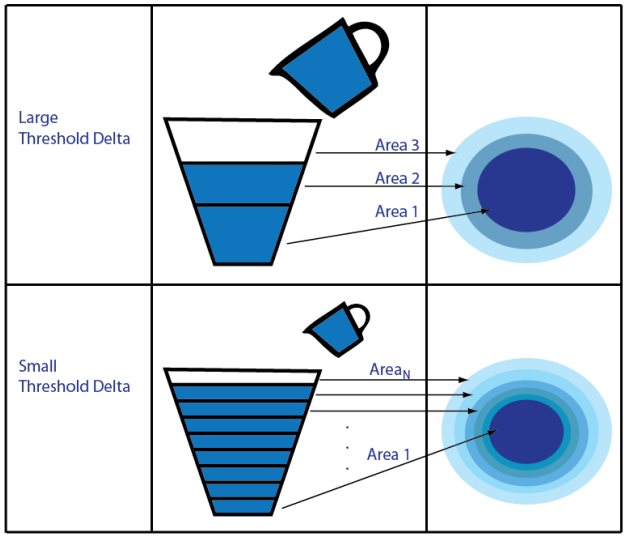
The four fields returned in `cc` are:

Field	Description
`Connectivity`	Connectivity of the MSER regions. Default: 8
`ImageSize`	Size of `I`.
`NumObjects`	Number of MSER regions in `I`.
`PixelIdxList`	1-by-`NumObjects` cell array containing `NumObjects` vectors. Each vector represents the linear indices of the pixels in the element’s corresponding MSER region.

Algorithms

Intensity Threshold Levels

The MSER detector incrementally steps through the intensity range of the input image to detect stable regions. The `ThresholdDelta` parameter determines the number of increments the detector tests for stability. You can think of the threshold delta value as the size of a cup to fill a bucket with water. The smaller the cup, the more number of increments it takes to fill up the bucket. The bucket can be thought of as the intensity profile of the region.



The MSER object checks the variation of the region area size between different intensity thresholds. The variation must be less than the value of the `MaxAreaVariation` parameter to be considered stable.

Note MSER feature detection is not suitable for use in images with extreme intensity value changes.

At a high level, MSER can be explained, by thinking of the intensity profile of an image representing a series of buckets. Imagine the tops of the buckets flush with the ground, and a hose turned on at one of the buckets. As the water fills into the bucket, it overflows and the next bucket starts filling up. Smaller regions of water join and become bigger bodies of water, and finally the whole area gets filled. As water is filling up into a bucket, it is checked against the MSER stability criterion. Regions appear, grow and merge at different intensity thresholds.

Version History

Introduced in R2012a

References

[1] Nister, D., and H. Stewenius, "Linear Time Maximally Stable Extremal Regions", *Lecture Notes in Computer Science*. 10th European Conference on Computer Vision, Marseille, France: 2008, no. 5303, pp. 183–196.

[2] Matas, J., O. Chum, M. Urba, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions." *Proceedings of British Machine Vision Conference*, pages 384-396, 2002.

[3] Obdrzalek D., S. Basovnik, L. Mach, and A. Mikulik. "Detecting Scene Elements Using Maximally Stable Colour Regions," *Communications in Computer and Information Science*, La Ferte-Bernard, France; 2009, vol. 82 CCIS (2010 12 01), pp 107–115.

[4] Mikolajczyk, K., T. Tuytelaars, C. Schmid, A. Zisserman, T. Kadir, and L. Van Gool, "*A Comparison of Affine Region Detectors*"; International Journal of Computer Vision, Volume 65, Numbers 1–2 / November, 2005, pp 43–72.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- For code generation, the function outputs `regions.PixelList` as an array. The region sizes are defined in `regions.Lengths`.
- For code generation, the `MSERRegions` object is not supported. To access the properties of the `MSERRegions` object for code generation, you can use one of these methods:
  - Read each object property and store it in a variable. For example, to access the `Location` property of the `regions` object, read it to the `regionsLoc` variable.

    ```
    function regionsLoc = codeGenTest(inputImg)
        regions = detectMSERFeatures(inputImg);
        regionsLoc = regions.Location;
    end
    codegen codeGenTest -args {inputImg}
    ```

  - Read all properties and store them in a structure. For example, to access all the properties of the `regions` object, read them to the `regionsStruct` structure.

    ```
    function regionsStruct = codeGenTestAll(inputImg)
        regions = detectMSERFeatures(inputImg);
        regionsStruct = struct('Count',regions.Count, ...
                               'Axes',regions.Axes, ...
                               'Location',regions.Location, ...
                               'Orientation',regions.Orientation, ...
                               'PixelList',regions.PixelList);
    end
    codegen codeGenTestAll -args {inputImg}
    ```

- For embedded real-time code generation support, you must enable the variable-size signals. **Code Generation > Interface** from the Configuration Parameters dialog box.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`detectSURFFeatures` | `detectMinEigenFeatures` | `detectBRISKFeatures` | `detectFASTFeatures` | `detectHarrisFeatures` | `detectKAZEFeatures` | `extractFeatures` | `extractHOGFeatures` | `detectFASTFeatures` | `matchFeatures`

Objects
`MSERRegions` | `SURFPoints` | `BRISKPoints` | `cornerPoints` | `binaryFeatures`

Topics
“Segment and Read Text in Image”
“Point Feature Types”

detectPeopleACF

Detect people using aggregate channel features (ACF)

Note `detectPeopleACF` will be removed in a future release. Use `peopleDetectorACF` instead.

Syntax

```
bboxes = detectPeopleACF(I)
[bboxes,scores] = detectPeopleACF(I)
[___] = detectPeopleACF(I,roi)
[___] = detectPeopleACF(Name,Value)
```

Description

`bboxes = detectPeopleACF(I)` returns a matrix, `bboxes`, that contains the locations of detected upright people in the input image, `I`. The locations are represented as bounding boxes. The function uses the aggregate channel features (ACF) algorithm.

`[bboxes,scores] = detectPeopleACF(I)` also returns the detection scores for each bounding box.

`[___] = detectPeopleACF(I,roi)` detects people within the rectangular search region specified by `roi`, using either of the previous syntaxes.

`[___] = detectPeopleACF(Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. Unspecified properties have default values.

Code Generation Support:
Supports Code Generation: No
Supports MATLAB Function block: No
Code Generation Support, Usage Notes, and Limitations

Examples

Detect People Using Aggregated Channel Features

Read an image.

```
I = imread('visionteam1.jpg');
```

Detect people in the image and store results as bounding boxes and score.

```
[bboxes,scores] = detectPeopleACF(I);
```

Annotate the detected upright people in the image.

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
```

Display the results with annotation.

```
figure
imshow(I)
title('Detected people and detection scores')
```



Input Arguments

`I` — Input image

truecolor image

Input image, specified as a truecolor image. The image must be real and nonsparse.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`roi` — Rectangular search region

four-element vector

Rectangular search region, specified as a four-element vector, [*x*,*y*,*width*,*height*]. The `roi` must be fully contained in `I`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Threshold',-1`

`Model` — ACF classification model

`'inria-100x41'` (default) | `'caltech-50x21'`

ACF classification model, specified as the comma-separated pair consisting of 'Model' and either 'inria-100x41' or 'caltech-50x21'. The 'inria-100x41' model was trained using the INRIA Person dataset. The 'caltech-50x21' model was trained using the Caltech Pedestrian dataset.

NumScaleLevels — Number of scale levels per octave
8 (default) | integer

Number of scale levels per octave, specified as the comma-separated pair consisting of 'NumScaleLevels', and an integer. Each octave is a power-of-two downscaling of the image. Increase this number to detect people at finer scale increments. Recommended values are in the range [4,8].

WindowStride — Window stride for sliding window
4 (default) | integer

Window stride for sliding window, specified as the comma-separated pair consisting of 'WindowStride', and an integer. Set this value to the amount you want to move the window, in the *x* and *y* directions. The sliding window scans the images for object detection. The function uses the same stride for the *x* and *y* directions.

SelectStrongest — Select strongest bounding box
true (default) | false

Select strongest bounding box, specified as the comma-separated pair consisting of 'SelectStrongest' and either true or false. The process, often referred to as nonmaximum suppression, eliminates overlapping bounding boxes based on their scores. Set this property to true to use the selectStrongestBbox function to select the strongest bounding box. Set this property to false, to perform a custom selection operation. Setting this property to false returns detected bounding boxes.

MinSize — Minimum region size
two-element vector [*height width*] | [50 21] | [100 41]

Minimum region size in pixels, specified as the comma-separated pair consisting of 'MinSize', and a two-element vector [*height width*]. You can set this property to [50 21] for the 'caltech-50x21' model or [100 41] for the 'inria-100x41' model. You can reduce computation time by setting this value to the known minimum region size for detecting a person. By default, MinSize is set to the smallest region size possible to detect an upright person for the classification model selected.

MaxSize — Maximum region size
size(I) (default) | two-element vector [*height width*]

Maximum region size in pixels, specified as the comma-separated pair consisting of 'MaxSize', and a two-element vector, [*height width*]. You can reduce computation time by setting this value to the known region size for detecting a person. If you do not set this value, by default the function determines the height and width of the image using the size of I.

Threshold — Classification accuracy threshold
–1 (default) | numeric value

Classification accuracy threshold, specified as the comma-separated pair consisting of 'Threshold' and a numerical value. Typical values are in the range [–1,1]. During multiscale object detection, the threshold value controls the person or nonperson classification accuracy and speed. Increase this threshold to speed up the performance at the risk of missing true detections.

Output Arguments

bboxes — Locations of detected people

M-by-4 matrix

Locations of people detected using the aggregate channel features (ACF) algorithm, returned as an *M*-by-4 matrix. The locations are represented as bounding boxes. Each row in `bboxes` contains a four-element vector, [*x*,*y*,*width*,*height*]. This vector specifies the upper-left corner and size of a bounding box, in pixels, for a detected person.

scores — Confidence value

M-by-1 vector

Confidence value for the detections, returned as an *M*-by-1 vector. The vector contains a value for each bounding box in `bboxes`. The score for each detection is the output of a soft-cascade classifier. The range of score values is `[-inf inf]`. Greater scores indicate a higher confidence in the detection.

Version History

Introduced in R2016a

References

[1] Dollar, P., R. Appel, S. Belongie, and P. Perona. "Fast feature pyramids for object detection." *Pattern Analysis and Machine Intelligence, IEEE Transactions*. Vol. 36, Issue 8, 2014, pp. 1532–1545.

[2] Dollar, C. Wojeck, B. Shiele, and P. Perona. "Pedestrian detection: An evaluation of the state of the art." *Pattern Analysis and Machine Intelligence, IEEE Transactions*.Vol. 34, Issue 4, 2012, pp. 743–761.

[3] Dollar, C., Wojeck, B. Shiele, and P. Perona. "Pedestrian detection: A benchmark." *IEEE Conference on Computer Vision and Pattern Recognition*. 2009.

See Also

Objects

`vision.CascadeObjectDetector` | `vision.PeopleDetector`

Functions

`selectStrongestBbox` | `peopleDetectorACF`

Topics

“Tracking Pedestrians from a Moving Car”
“Point Feature Types”

detectSURFFeatures

Detect SURF features

Syntax

```
points = detectSURFFeatures(I)
points = detectSURFFeatures(I,Name,Value)
```

Description

`points = detectSURFFeatures(I)` returns a `SURFPoints` object, `points`, containing information about SURF features detected in the 2-D grayscale or binary input image `I`. The `detectSURFFeatures` function implements the Speeded-Up Robust Features (SURF) algorithm to find blob features.

`points = detectSURFFeatures(I,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

Examples

Detect SURF Interest Points in a Grayscale Image

Read image and detect interest points.

```
I = imread('cameraman.tif');
points = detectSURFFeatures(I);
```

Display locations of interest in image.

```
imshow(I); hold on;
plot(points.selectStrongest(10));
```



Input Arguments

`I` — Input image

M-by-*N* 2-D grayscale or binary image

Input image, specified as an *M*-by-*N* 2-D grayscale or binary image. The input image must be a real non-sparse value.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `detectSURFFeatures(I,'MetricThreshold',100)`

`MetricThreshold` — Strongest feature threshold

`1000.0` (default) | non-negative scalar

Strongest feature threshold, specified as a non-negative scalar. To return more blobs, decrease the value of this threshold.

`NumOctaves` — Number of octaves

3 (default) | scalar (greater than or equal to 1)

Number of octaves, specified as an integer scalar, greater than or equal to 1. Increase this value to detect larger blobs. Recommended values are between 1 and 4.

Each octave spans a number of scales that are analyzed using varying size filters:

Octave	Filter Sizes
1	9-by-9, 15-by-15, 21-by-21, 27-by-27, ...
2	15-by-15, 27-by-27, 39-by-39, 51-by-51, ...
3	27-by-27, 51-by-51, 75-by-75, 99-by-99, ...
4	....

Higher octaves use larger filters and subsample the image data. Larger number of octaves result in finding larger size blobs. Set the `NumOctaves` parameter appropriately for the image size. For example, a 50-by-50 image require you to set the `NumOctaves` parameter, less than or equal to `2`. The `NumScaleLevels` parameter controls the number of filters used per octave. At least three levels are required to analyze the data in a single octave.

`NumScaleLevels` — Number of scale levels per octave
`4` (default) | integer scalar, greater than or equal to `3`

Number of scale levels per octave to compute, specified as an integer scalar, greater than or equal to `3`. Increase this number to detect more blobs at finer scale increments. Recommended values are between `3` and `6`.

`ROI` — Rectangular region of interest
`[1 1 size(I,2) size(I,1)]` (default) | vector

Rectangular region of interest, specified as a vector. The vector must be in the format [*x y width height*]. When you specify an `ROI`, the function detects blobs within the area at [*x y*] of size specified by [*width height*]. The [*x y*] elements specify the upper left corner of the region.

Output Arguments

`points` — SURF features
`SURFPoints` object

SURF features, returned as a `SURFPoints` object. This object contains information about SURF features detected in a grayscale or binary image.

Version History

Introduced in R2011b

References

[1] Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool. “SURF:Speeded Up Robust Features.” *Computer Vision and Image Understanding (CVIU)*.Vol. 110, No. 3, pp. 346–359, 2008.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates portable C code using a C++ compiler that links to OpenCV libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`detectBRISKFeatures` | `detectFASTFeatures` | `detectORBFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `detectMSERFeatures` | `extractFeatures` | `matchFeatures` | `SURFPoints` | `ORBPoints`

Topics
“Create Panorama”

detectORBFeatures

Detect ORB keypoints

Syntax

```
points = detectORBFeatures(I)
points = detectORBFeatures(I,Name,Value)
```

Description

`points = detectORBFeatures(I)` returns an `ORBPoints` object that contains information about ORB keypoints. The ORB keypoints are detected from the input image by using the Oriented FAST and rotated BRIEF (ORB) feature detection method.

`points = detectORBFeatures(I,Name,Value)` specifies options using one or more name-value arguments.

Examples

Detect ORB Keypoints in Grayscale Image

Read an image into the workspace.

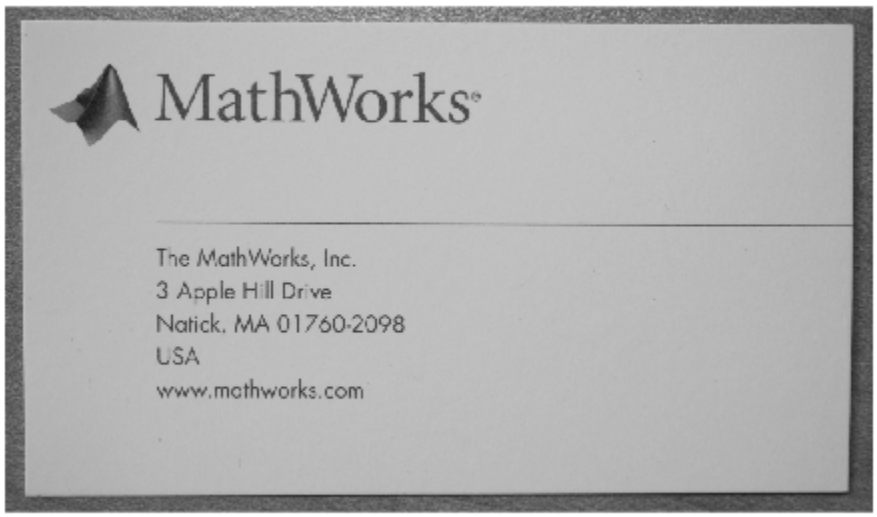
```
I = imread('businessCard.png');
```

Convert the image into a grayscale image.

```
I = im2gray(I);
```

Display the grayscale image.

```
figure
imshow(I)
```

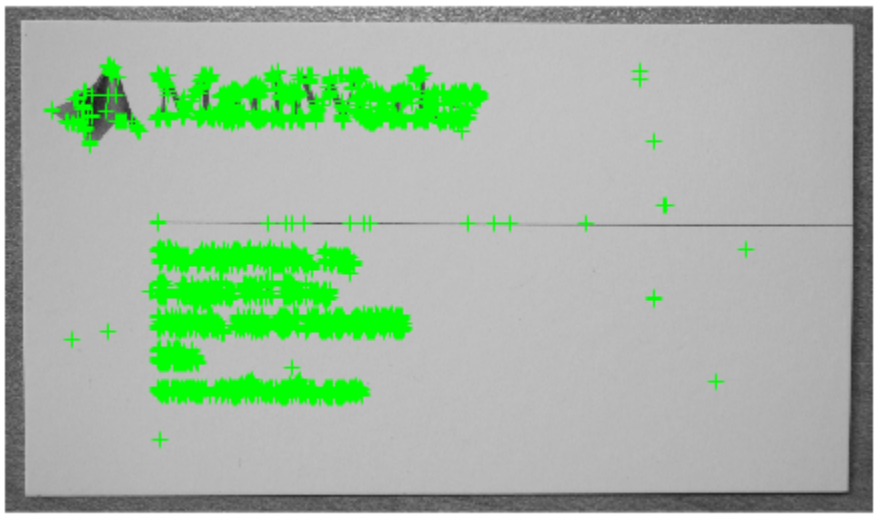


Detect and store ORB keypoints.

```
points = detectORBFeatures(I);
```

Display the grayscale image and plot the detected ORB keypoints. Suppress the display of circles around the detected keypoints. The ORB keypoints are detected in regions with high intensity variance.

```
figure
imshow(I)
hold on
plot(points,'ShowScale',false)
hold off
```



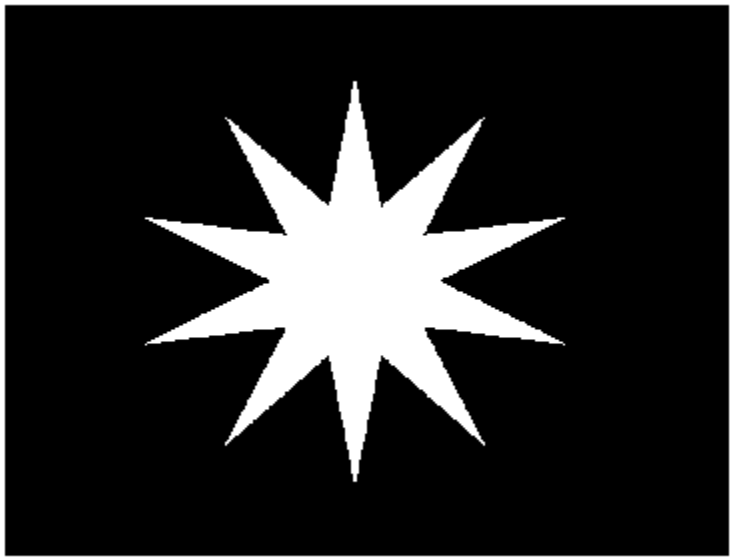
Detect ORB Keypoints in Binary Shape Image

Read a binary image into the workspace.

```
I = imread('star.png');
```

Display the image.

```
figure
imshow(I)
```

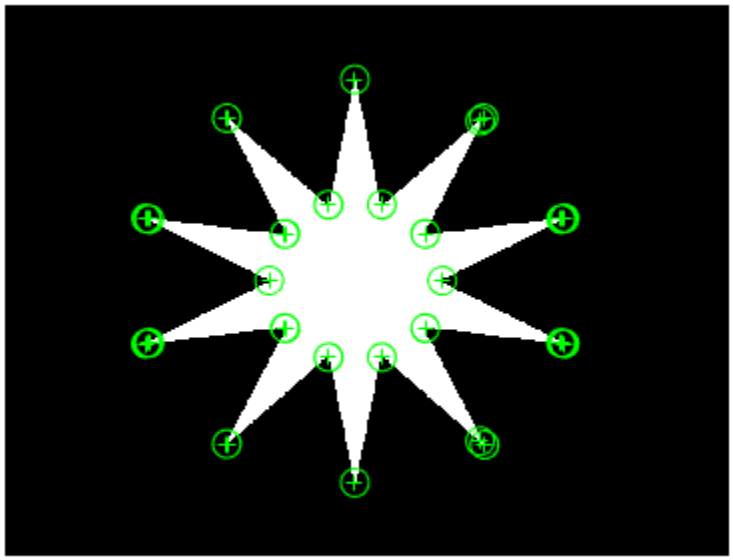


Detect and store ORB keypoints. Specify the scale factor for image decomposition as 1.01 and the number of decomposition levels as 3.

```
points = detectORBFeatures(I,'ScaleFactor',1.01,'NumLevels',3);
```

Display the image and plot the detected ORB keypoints. The inflection points in the binary shape image are detected as the ORB keypoints.

```
figure
imshow(I)
hold on
plot(points)
hold off
```



Input Arguments

`I` — Input image

M-by-*N* grayscale or binary image

Input image, specified as an *M*-by-*N* grayscale or binary image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `detectORBFeatures(I,'NumLevels',4)`

`ScaleFactor` — Scale factor for image decomposition

`1.2` (default) | scalar greater than 1

Scale factor for image decomposition, specified as the comma-separated pair consisting of `'ScaleFactor'` and a scalar greater than 1. The scale value at each level of decomposition is $\texttt{ScaleFactor}^{(level\text{-}1)}$, where *level* is any value in the range [0, `Numlevels`-1]. Given the input image of size *M*-by-*N*, the image size at each level of decomposition is

$$M_{level} = \frac{M}{ScaleFactor^{(level\ -\ 1)}}$$

$$N_{level} = \frac{N}{ScaleFactor^{(level\ -\ 1)}}$$

.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint64` | `uint32`

`NumLevels` — Number of decomposition levels
8 (default) | scalar greater than or equal to 1

Number of decomposition levels, specified as the comma-separated pair consisting of `'NumLevels'` and a scalar greater than or equal to 1. Increase this value to extract keypoints from the image at more levels of decomposition.

The number of decomposition levels for extracting keypoints is limited by the image size at that level. The image size at a level of decomposition must be at least 63-by-63 for detecting keypoints. The maximum level of decomposition is calculated as

$$level_{\max} = \text{floor}\left(\frac{\log(\min(M,N)) - \log(63)}{\log(\text{ScaleFactor})}\right) + 1$$

If either the default value or the specified value of `'NumLevels'` is greater than $level_{max}$, the function modifies `NumLevels` to $level_{max}$ and returns a warning.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint64` | `uint32`

`ROI` — Region of interest
`[1 1 M N]` (default) | four-element vector

Region of interest for keypoint detection, specified as the comma-separated pair consisting of `'ROI'` and a vector of the format [*x y width height*]. The first two elements represent the location of the upper left corner of the region of interest. The last two elements represent the width and the height of the region of interest. The width and height of the region of interest must each be a value greater than or equal to 63.

Output Arguments

`points` — ORB keypoints
`ORBPoints` object

ORB keypoints, returned as an `ORBPoints` object. The object contains information about keypoints detected in the input image.

Algorithms

The function detects keypoints from the input image by using the ORB feature detection method in [1].

Version History

Introduced in R2019a

References

[1] Rublee, E., V. Rabaud, K. Konolige, and G. Bradski. "ORB: An Efficient Alternative to SIFT or SURF." In *Proceedings of the 2011 International Conference on Computer Vision*, 2564–2571. Barcelona, Spain: IEEE, 2011.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = points.Location;
  ```

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`detectBRISKFeatures` | `detectHarrisFeatures` | `detectMinEigenFeatures` | `detectFASTFeatures` | `detectSURFFeatures` | `detectMSERFeatures` | `detectKAZEFeatures` | `ORBPoints` | `extractFeatures` | `matchFeatures`

Topics

“Point Feature Types”
“Local Feature Detection and Extraction”

disparity

(Not recommended) Disparity map between stereo images

Note `disparity` is not recommended. Use `disparityBM` or `disparitySGM` instead. For more information, see “Version History”

Syntax

```
disparityMap = disparity(I1,I2)
disparityMap = disparity(I1,I2,Name,Value)
```

Description

`disparityMap = disparity(I1,I2)` returns the disparity map, `disparityMap`, for a pair of stereo images, `I1` and `I2`.

`disparityMap = disparity(I1,I2,Name,Value)` provides additional control for the disparity algorithm by using one or more `Name,Value` pair arguments.

Examples

Compute Disparity Map for a Pair of Stereo Images

Load the images and convert them to grayscale.

```
I1 = imread('scene_left.png');
I2 = imread('scene_right.png');
```

Show stereo anaglyph. Use red-cyan stereo glasses to view image in 3-D.

```
figure
imshow(stereoAnaglyph(I1,I2));
title('Red-cyan composite view of the stereo images');
```

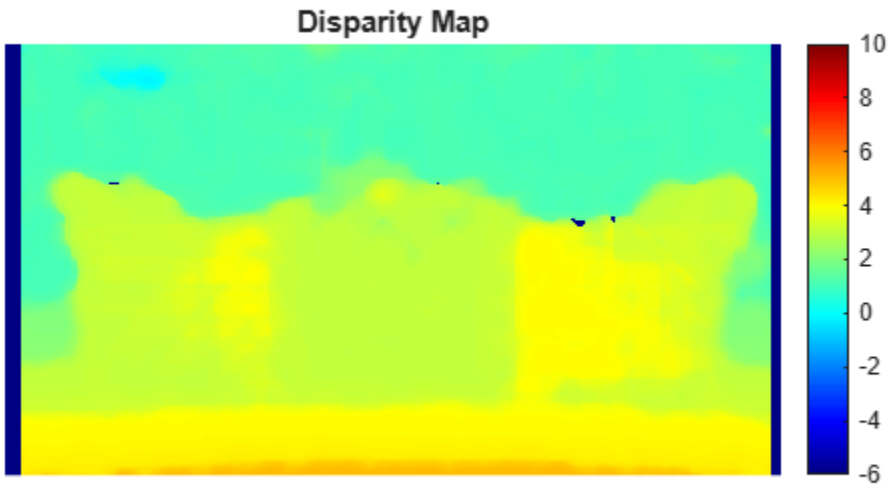


Compute the disparity map.

```
disparityRange = [-6 10];
disparityMap = disparity(rgb2gray(I1),rgb2gray(I2),'BlockSize',...
    15,'DisparityRange',disparityRange);
```

Display the disparity map. For better visualization, use the disparity range as the display range for imshow.

```
figure
imshow(disparityMap,disparityRange);
title('Disparity Map');
colormap(gca,jet)
colorbar
```



Input Arguments

`I1` — Input image 1

M-by-*N* 2-D grayscale image

Input image referenced as `I1` corresponding to camera 1, specified in 2-D grayscale. The stereo images, `I1` and `I2`, must be rectified such that the corresponding points are located on the same rows. You can perform this rectification with the `rectifyStereoImages` function.

You can improve the speed of the function by setting the class of `I1` and `I2` to `uint8`, and the number of columns to be divisible by 4. Input images `I1` and `I2` must be real, finite, and nonsparse. They must be the same class.

Data Types: `uint8` | `uint16` | `int16` | `single` | `double`

`I2` — Input image 2

M-by-*N* 2-D grayscale image

Input image referenced as `I2` corresponding to camera 2, specified in 2-D grayscale. The input images must be rectified such that the corresponding points are located on the same rows. You can improve the speed of the function by setting the class of `I1` and `I2` to `uint8`, and the number of columns to be divisible by 4. Input images `I1` and `I2` must be real, finite, and nonsparse. They must be the same class.

Data Types: `uint8` | `uint16` | `int16` | `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Method','BlockMatching'`, specifies the `'Method'` property be set to `'BlockMatching'`.

`Method` — Disparity estimation algorithm

`'SemiGlobal'` (default) | `'BlockMatching'`

Disparity estimation algorithm, specified as the comma-separated pair consisting of '`Method`' and either `'BlockMatching'` or `'SemiGlobal'`. The disparity function implements the basic Block Matching [1] and the Semi-Global Block Matching [3] algorithms. In the `'BlockMatching'` method, the function computes disparity by comparing the sum of absolute differences (SAD) of each block of pixels in the image. In the `'SemiGlobal'` matching method, the function additionally forces similar disparity on neighboring blocks. This additional constraint results in a more complete disparity estimate than in the `'BlockMatching'` method.

The algorithms perform these steps:

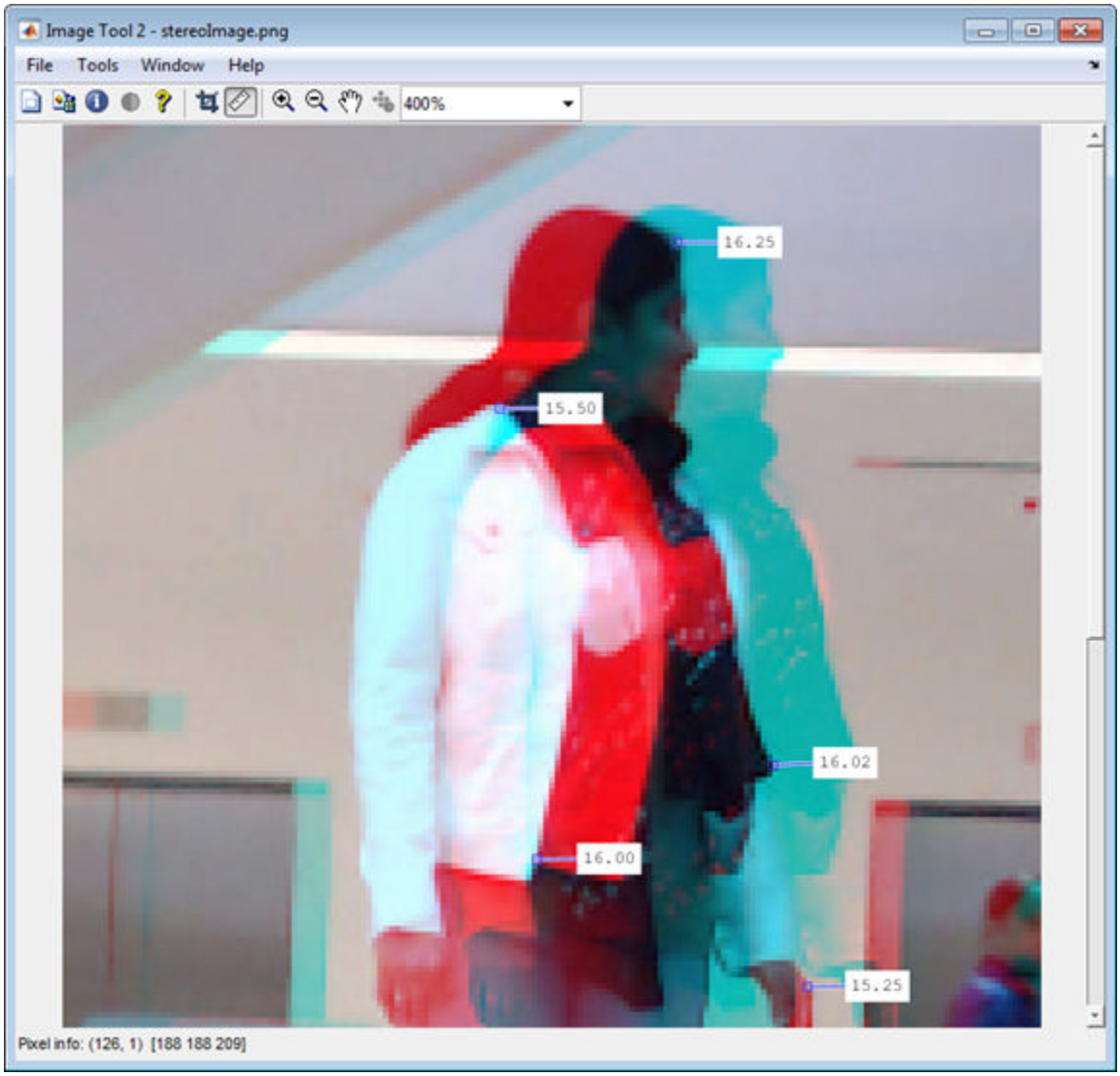
1. Compute a measure of contrast of the image by using the Sobel filter.
2. Compute the disparity for each pixel in `I1`.
3. Mark elements of the disparity map, `disparityMap`, that were not computed reliably. The function uses –`realmax`(`'single'`) to mark these elements.

`DisparityRange` — Range of disparity

`[0 64]` (default) | two-element vector

Range of disparity, specified as the comma-separated pair consisting of '`DisparityRange`' and a two-element vector. The two-element vector must be in the format [*MinDisparity*, *MaxDisparity*]. Both elements must be an integer and can be negative. *MinDisparity* and *MaxDisparity* must be in the range [-*image width*, *image width*]. The difference between *MaxDisparity* and *MinDisparity* must be divisible by `16`. `DisparityRange` must be real, finite, and nonsparse. If the camera used to take `I1` was to the right of the camera used to take `I2`, then *MinDisparity* must be negative.

The disparity range depends on the distance between the two cameras and the distance between the cameras and the object of interest. Increase the `DisparityRange` when the cameras are far apart or the objects are close to the cameras. To determine a reasonable disparity for your configuration, display the stereo anaglyph of the input images in the Image Viewer app and use the Distance tool to measure distances between pairs of corresponding points. Modify the *MaxDisparity* to correspond to the measurement.



`BlockSize` — Square block size
15 (default) | odd integer

Square block size, specified as the comma-separated pair consisting of '`BlockSize`' and an odd integer in the range [5,255]. This value sets the width for the square block size. The function uses the square block of pixels for comparisons between `I1` and I2. `BlockSize` must be real, finite, and nonsparse.

`ContrastThreshold` — Contrast threshold range
`0.5` (default) | scalar value

Contrast threshold range, specified as the comma-separated pair consisting of '`ContrastThreshold`' and a scalar value in the range (0,1]. The contrast threshold defines an acceptable range of contrast values. Increasing this parameter results in fewer pixels being marked as unreliable.`ContrastThreshold` must be real, finite, and nonsparse.

`UniquenessThreshold` — Minimum value of uniqueness
15 (default) | non-negative integer

Minimum value of uniqueness, specified as the comma-separated pair consisting of 'UniquenessThreshold' and a nonnegative integer. Increasing this parameter results in the function marking more pixels unreliable. When the uniqueness value for a pixel is low, the disparity computed for it is less reliable. Setting the threshold to 0 disables uniqueness thresholding. UniquenessThreshold must be real, finite, and nonsparse.

The function defines uniqueness as a ratio of the optimal disparity estimation and the less optimal disparity estimation. For example:
Let K be the best estimated disparity, and let V be the corresponding SAD (Sum of Absolute Difference) value.
Consider V as the smallest SAD value over the whole disparity range, and v as the smallest SAD value over the whole disparity range, excluding K, K-1, and K+1.
If $v < V * (1+0.01*\text{UniquenessThreshold})$, then the function marks the disparity for the pixel as unreliable.

DistanceThreshold — Maximum distance for left-to-right image checking
[] (disabled) (default) | non-negative integer

Maximum distance for left-to-right image checking between two points, specified as the comma-separated pair consisting of 'DistanceThreshold' and a nonnegative integer. Increasing this parameter results in fewer pixels being marked as unreliable. Conversely, when you decrease the value of the distance threshold, you increase the reliability of the disparity map. You can set this parameter to an empty matrix [] to disable it. DistanceThreshold must be real, finite, and nonsparse.

The distance threshold specifies the maximum distance between a point in I1 and the same point found from I2. The function finds the distance and marks the pixel in the following way:
Let p_1 be a point in image I_1.
Step 1: The function searches for point p_1's best match in image I_2 (left-to-right check) and finds point p_2.
Step 2: The function searches for p_2's best match in image I_1 (right-to-left check) and finds point p_3.
If the search returns a distance between p_1 and p_3 greater than DistanceThreshold, the function marks the disparity for the point p_1 as unreliable.

TextureThreshold — Minimum texture threshold
0.0002 (default) | scalar value

Minimum texture threshold, specified as the comma-separated pair consisting of 'TextureThreshold' and a scalar value in the range [0, 1). The texture threshold defines the minimum texture value for a pixel to be reliable. The lower the texture for a block of pixels, the less reliable the computed disparity is for the pixels. Increasing this parameter results in more pixels being marked as unreliable. You can set this parameter to 0 to disable it. This parameter applies only when you set Method to 'BlockMatching'.

The texture of a pixel is defined as the sum of the saturated contrast computed over the BlockSize-by-BlockSize window around the pixel. The function considers the disparity computed for the pixel unreliable and marks it, when the texture falls below the value defined by:

$$\textit{Texture} < X * \text{TextureThreshold} * \text{BlockSize}^2$$

X represents the maximum value supported by the class of the input images, I1 and I2.

TextureThreshold must be real, finite, and nonsparse.

Output Arguments

`disparityMap` — Disparity map

M-by-*N* 2-D grayscale image

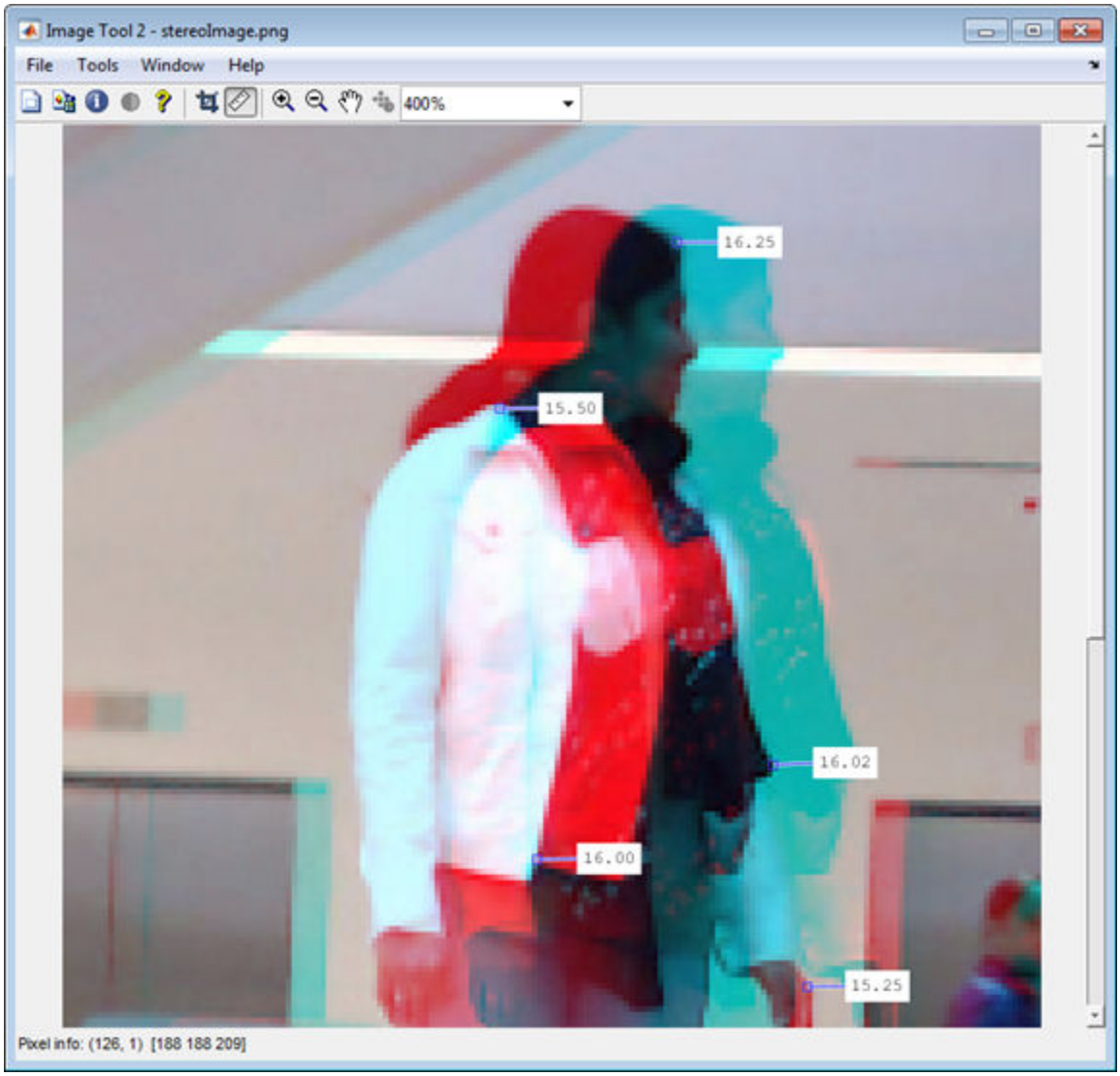
Disparity map for a pair of stereo images, returned as an *M*-by-*N* 2-D grayscale image. The function returns the disparity map with the same size as the input images, `I1` and `I2`. Each element of the output specifies the disparity for the corresponding pixel in the image references as `I1`. The returned disparity values are rounded to $1/16$th pixel.

The function computes the disparity map in three steps:

1. Compute a measure of contrast of the image by using the Sobel filter.
2. Compute the disparity for each of the pixels by using block matching and the sum of absolute differences (SAD).
3. Optionally, mark the pixels which contain unreliable disparity values. The function sets the pixel to the value returned by -`realmax`('`single`').

Tips

If your resulting disparity map looks noisy, try modifying the `DisparityRange`. The disparity range depends on the distance between the two cameras and the distance between the cameras and the object of interest. Increase the `DisparityRange` when the cameras are far apart or the objects are close to the cameras. To determine a reasonable disparity for your configuration, display the stereo anaglyph of the input images in the Image Viewer app and use the Distance tool to measure distances between pairs of corresponding points. Modify the *MaxDisparity* to correspond to the measurement.



Version History

Introduced in R2011b

R2019a: `disparity` function will be removed

Not recommended starting in R2019a

The `disparity` function will be removed in a future release. Use `disparityBM` or `disparitySGM` instead. Use `disparityBM` to compute disparity map using block matching method. Use `disparitySGM` to compute disparity map using semi-global matching method.

References

[1] Konolige, K., *Small Vision Systems: Hardware and Implementation*, Proceedings of the 8th International Symposium in Robotic Research, pages 203-212, 1997.

[2] Bradski, G. and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*, O'Reilly, Sebastopol, CA, 2008.

[3] Hirschmuller, H., *Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information*, International Conference on Computer Vision and Pattern Recognition, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `'Method'` must be a compile-time constant.
- Generates portable C code using a C++ compiler that links to OpenCV (Version 3.4.0) libraries. See “Portable C Code Generation for Functions That Use OpenCV Library”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Stereo Camera Calibrator | `reconstructScene` | `estimateUncalibratedRectification` | `rectifyStereoImages` | `estimateCameraParameters`

Topics

“Depth Estimation from Stereo Video”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Uncalibrated Stereo Image Rectification”

disparityBM

Compute disparity map using block matching

Syntax

```
disparityMap = disparityBM(I1,I2)
disparityMap = disparityBM(I1,I2,Name,Value)
```

Description

`disparityMap = disparityBM(I1,I2)` computes disparity map from a pair of rectified stereo images `I1` and `I2`, by using the block matching method. To know more about rectifying stereo images, see “Image Rectification” on page 3-640.

`disparityMap = disparityBM(I1,I2,Name,Value)` specifies options using one or more name-value pair arguments.

Examples

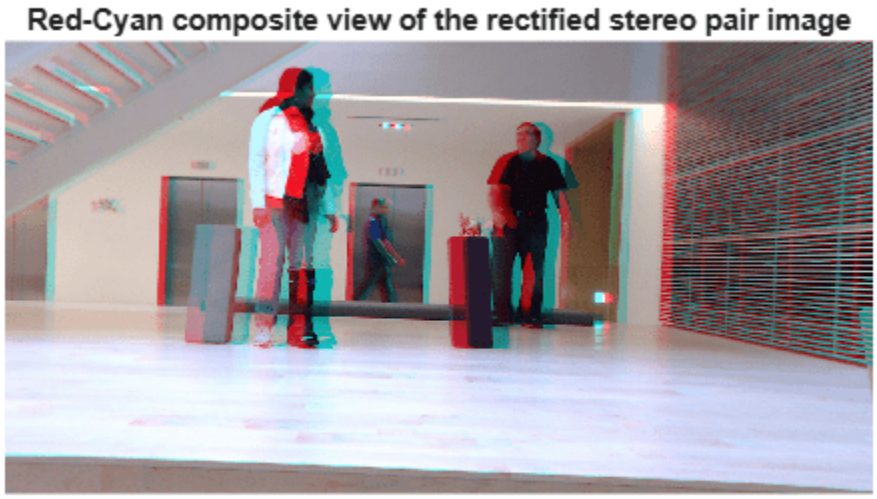
Compute Disparity Map by Using Block Matching Method

Load a rectified stereo pair image.

```
I1 = imread('rectified_left.png');
I2 = imread('rectified_right.png');
```

Create the stereo anaglyph of the rectified stereo pair image and display it. You can view the image in 3-D by using red-cyan stereo glasses.

```
A = stereoAnaglyph(I1,I2);
figure
imshow(A)
title('Red-Cyan composite view of the rectified stereo pair image')
```



Convert the rectified input color images to grayscale images.

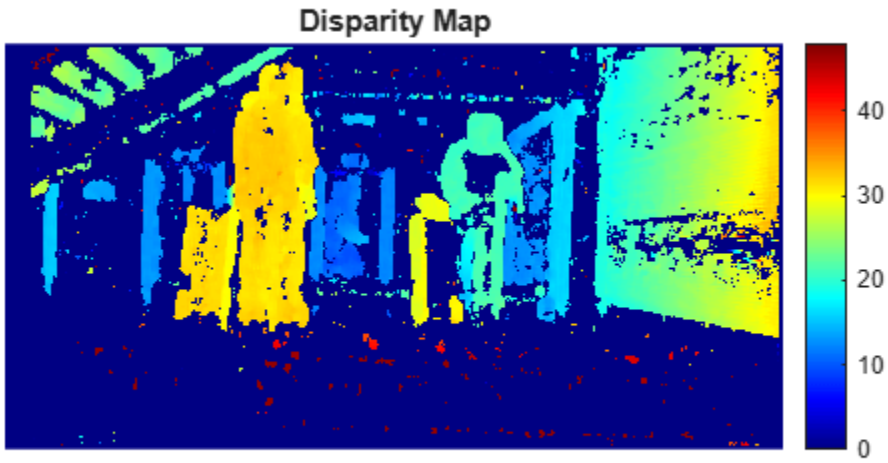
```
J1 = im2gray(I1);
J2 = im2gray(I2);
```

Compute the disparity map. Specify the range of disparity as [0, 48], and the minimum value of uniqueness as 20.

```
disparityRange = [0 48];
disparityMap = disparityBM(J1,J2,'DisparityRange',disparityRange,'UniquenessThreshold',20);
```

Display the disparity map. Set the display range to the same value as the disparity range.

```
figure
imshow(disparityMap,disparityRange)
title('Disparity Map')
colormap jet
colorbar
```



Input Arguments

`I1` — Input image 1

2-D grayscale image | `gpuArray` object

Input image referenced as `I1` corresponding to camera 1, specified as a 2-D grayscale image or a `gpuArray` object. The function uses this image as the reference image for computing the disparity map. The input images `I1` and `I2` must be real, finite, and nonsparse. Also, `I1` and `I2` must be of the same size and same data type.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`I2` — Input image 2

2-D grayscale image | `gpuArray` object

Input image referenced as `I2` corresponding to camera 2, specified as a 2-D grayscale image or a `gpuArray` object. The input images `I1` and `I2` must be real, finite, and nonsparse. `I1` and `I2` must be of the same size and same data type.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `disparityBM(I1,I2,'DisparityRange',[0 64])`

`DisparityRange` — Range of disparity
`[0 64]` (default) | two-element vector

Range of disparity, specified as the comma-separated pair consisting of `'DisparityRange'` and a two-element vector of the form [*MinDisparity MaxDisparity*]. *MinDisparity* is the minimum disparity and *MaxDisparity* is the maximum disparity. The conditions this range must satisfy depend on the type of input images.

If the input images are grayscale images of width *N*, then:

- *MinDisparity* and *MaxDisparity* must be integers in the range (–*N*, *N*).
- The difference between the *MaxDisparity* and *MinDisparity* values must be divisible by 16 and less than the width of the input images.

If the input images are `gpuArray` objects of width *N*, then:

- The value of *MinDisparity* must be equal to zero.

  The value of *MaxDisparity* must be in the range (16, *N*). If *N* is greater than 256, then the *MaxDisparity* must be chosen as less than or equal to 256.

  The difference between the *MaxDisparity* and *MinDisparity* values must be divisible by 16.

The default value for the range of disparity is `[0 64]`. For more information on choosing the range of disparity, see “Choosing Range of Disparity” on page 3-640.

`BlockSize` — Size of squared block
`15` (default) | odd integer

Size of the squared block, specified as the comma-separated pair consisting of `'BlockSize'` and an odd integer. This value specifies the width of the search window used for block matching pixels in the rectified stereo pair image. The range for the size squared block depend on the type of input images.

If the input images are grayscale images, the `'BlockSize'` value must be an odd integer in the range [5, 255].

If the input images are `gpuArray` objects, the `'BlockSize'` value must be an odd integer in the range [5, 51].

The default value is 15.

`ContrastThreshold` — Range of contrast threshold
`0.5` (default) | scalar in the range (0, 1]

Range of contrast threshold, specified as the comma-separated pair consisting of `'ContrastThreshold'` and a scalar value in the range (0, 1]. The contrast threshold defines an acceptable range of contrast values. If the contrast value of a pixel in the reference image is below the contrast threshold, then the disparity computed for that pixel is considered unreliable. Increasing this parameter results in disparity for fewer pixels being marked as unreliable.

`UniquenessThreshold` — Minimum value of uniqueness
`15` (default) | nonnegative integer

Minimum value of uniqueness, specified as the comma-separated pair consisting of `'UniquenessThreshold'` and a nonnegative integer.

The function marks the estimated disparity value K for a pixel as unreliable, if:

$v < V \times (1 + 0.01 \times \texttt{UniquenessThreshold})$,

where V is the sum of absolute difference (SAD) corresponding to the disparity value K. v is the smallest SAD value over the whole disparity range, excluding K, K–1, and K+1.

Increasing the value of `UniquenessThreshold` results in disparity values for more pixels being marked as unreliable. To disable the use of uniqueness threshold, set this value to `[]`.

Note If the input images are `gpuArray` objects, then the `'UniquenessThreshold'` name-value pair argument does not apply.

`DistanceThreshold` — Maximum distance between conjugate pixels in stereo pair image

`[]` (disabled) (default) | nonnegative integer

Maximum distance between conjugate pixels in stereo pair image, specified as the comma-separated pair consisting of `'DistanceThreshold'` and a nonnegative integer. The distance threshold specifies the maximum distance between a pixel in `I1` and the same pixel found in `I2`.

For pixel p_1 in the reference image `I1`, the function performs a left-to-right check to find its best matching pixel p_2 in image `I2`. For pixel p_2, the function performs a right-to-left check to find its best matching pixel p_3 in the reference image `I1`. If the distance between p_1 and p_3 is greater than the `DistanceThreshold`, the function marks the disparity for the pixel p_1 in the reference image `I1` as unreliable.

Increasing the value of `DistanceThreshold` results in disparity values for fewer pixels being marked as unreliable. Conversely, decreasing the value of `DistanceThreshold`, increases the reliability of the disparity map. To disable the use of distance threshold, set this value to `[]`.

Note If the input images are `gpuArray` objects, then the `'DistanceThreshold'` name-value pair argument does not apply.

`TextureThreshold` — Minimum texture threshold

`0.0002` (default) | scalar value in the range [0, 1)

Minimum texture threshold, specified as the comma-separated pair consisting of `'TextureThreshold'` and a scalar value in the range [0, 1).

The texture of a pixel is defined as the sum of the saturated contrast computed over the `BlockSize`-by-`BlockSize` window around the pixel. The texture threshold defines the minimum texture value for a pixel to have reliable disparity value. The function considers the disparity for a pixel as unreliable, if:

$texture < intensity_{max} \times \texttt{TextureThreshold} \times \texttt{BlockSize}^2$,

where *texture* is the texture of a pixel. $intensity_{max}$ is the maximum value supported by the data type of input images `I1` and `I2`.

Increasing the value of `'TextureThreshold'` results in disparity values for more pixels being marked as unreliable. To disable the use of texture threshold, set this value to `[]`.

Output Arguments

`disparityMap` — Disparity map

2-D grayscale image | `gpuArray` object

Disparity map for rectified stereo pair image, returned as a 2-D grayscale image or a `gpuArray` object. The function returns the disparity map with the same size as input images `I1` and `I2`. Each value in this output refers to the displacement between conjugate pixels in the stereo pair image. For details about computing the disparity map, see “Computing Disparity Map Using Block Matching” on page 3-641.

Data Types: `single`

More About

Image Rectification

The input images `I1` and `I2` must be rectified before computing the disparity map. The rectification ensures that the corresponding points in the stereo pair image are on the same rows. You can rectify the input stereo pair image by using the `rectifyStereoImages` function. The reference image must be the same for rectification and disparity map computation.

Algorithms

Choosing Range of Disparity

The range of disparity must be chosen to cover the minimum and the maximum amount of horizontal shift between the corresponding pixels in the rectified stereo pair image. You can determine the approximate horizontal shift values from the stereo anaglyph of the stereo pair image. Compute the stereo anaglyph of the rectified images by using the `stereoAnaglyph` function. Display the stereo anaglyph in the Image Viewer app. To measure the amount of horizontal shift between the corresponding points in the stereo pair image, select **Measure Distance** from the **Tools** menu in Image Viewer. Choose the minimum and maximum disparity values for the range of disparity based on this measurement.

For example, this figure displays the stereo anaglyph of a rectified stereo pair image and the horizontal shift values measured between the corresponding points in the stereo pair image. The minimum and maximum shift values are computed as 12.3101 and 32.3757 respectively. Based on these values, the range of disparity can be chosen as [0, 48].



Computing Disparity Map Using Block Matching

The function computes the disparity map by following these steps:

1. Compute the disparity for each pixel by using the block matching method given in [1].
   - For CPU — The sum of absolute difference (SAD) is used as the cost function in block matching. The cost function estimates the displacement between pixels in the rectified stereo pair image.
2. Optionally, mark the pixels for unreliability based on the `ContrastThreshold`, `UniquenessThreshold`, `DistanceThreshold`, and `TextureThreshold` name-value pairs. The function sets the unreliable disparity values to `NaN`.

Version History

Introduced in R2019a

R2022a: GPU Use Being Removed for `disparityBM` function
Behavior changed in R2022a

The `disparityBM` function no longer supports GPU. You can use the enhanced `disparitySGM` on the GPU instead. Results will not be identical using the `disparitySGM` function, but the semi-global algorithm used in the `disparitySGM` is normally recommended over the block matching algorithm in the `disparityBM` function.

References

[1] Konolige, K. "Small Vision Systems: Hardware and Implementation." In *Proceedings of the 8th International Symposium in Robotic Research*, pp. 203–212. 1997.

[2] Bradski, G. and A. Kaehler. *Learning OpenCV : Computer Vision with the OpenCV Library*. Sebastopol, CA: O'Reilly Media, Inc. 2008.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

- Supports code generation only in generic `MATLAB Host Computer` target platform.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Stereo Camera Calibrator

Functions
`disparitySGM` | `reconstructScene` | `estimateUncalibratedRectification` | `rectifyStereoImages` | `estimateCameraParameters`

disparitySGM

Compute disparity map through semi-global matching

Syntax

```
disparityMap = disparitySGM(I1,I2)
[disparityMap,disparityRangeUsed] = disparitySGM(I1,I2)
___ = disparitySGM(I1,I2,Name,Value)
```

Description

`disparityMap = disparitySGM(I1,I2)` computes disparity map from a pair of rectified stereo images `I1` and `I2`, by using semi-global matching (SGM) method. To know more about rectifying stereo images, see “Image Rectification” on page 3-647.

`[disparityMap,disparityRangeUsed] = disparitySGM(I1,I2)` additionally returns the actual values of the disparity range used to estimate disparity, which may differ from those provided using the `DisparityRange` name-value argument.

`___ = disparitySGM(I1,I2,Name,Value)` specifies additional options using one or more name-value pair arguments along with any of the previous syntaxes.

Examples

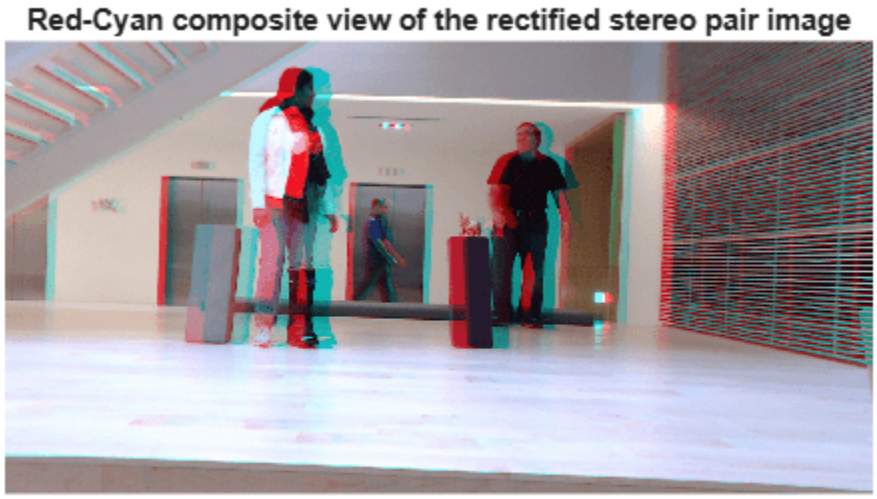
Compute Disparity Map by Using Semi-Global Matching Method

Load a rectified stereo pair image.

```
I1 = imread("rectified_left.png");
I2 = imread("rectified_right.png");
```

Create the stereo anaglyph of the rectified stereo pair image and display it. You can view the image in 3-D by using red-cyan stereo glasses.

```
A = stereoAnaglyph(I1,I2);
figure
imshow(A)
title("Red-Cyan composite view of the rectified stereo pair image")
```



Convert the rectified input color images to grayscale images.

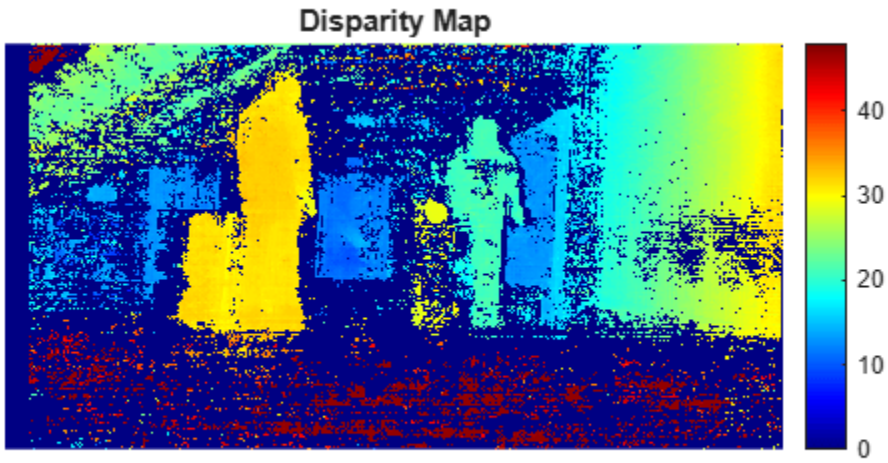
```
J1 = im2gray(I1);
J2 = im2gray(I2);
```

Compute the disparity map through semi-global matching. Specify the range of disparity as [0, 48], and the minimum value of uniqueness as 20.

```
disparityRange = [0 48];
disparityMap = disparitySGM(J1,J2,"DisparityRange",disparityRange,"UniquenessThreshold",20);
```

Display the disparity map. Set the display range to the same value as the disparity range.

```
figure
imshow(disparityMap,disparityRange)
title("Disparity Map")
colormap jet
colorbar
```



Input Arguments

`I1` — Input image 1

2-D grayscale image | `gpuArray` object

Input image referenced as `I1` corresponding to camera 1, specified as a 2-D grayscale image or a `gpuArray` object. The function uses this image as the reference image for computing the disparity map. The input images `I1` and `I2` must be real, finite, and nonsparse. Also, `I1` and `I2` must be of the same size and same data type.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`I2` — Input image 2

2-D grayscale image | `gpuArray` object

Input image referenced as `I2` corresponding to camera 2, specified as a 2-D grayscale image or a `gpuArray` object. The input images `I1` and `I2` must be real, finite, and nonsparse. `I1` and `I2` must be of the same size and same data type.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `disparitySGM(I1,I2,'DisparityRange',[0 64])`

`DisparityRange` — Range of disparity
`[0 128]` (default) | two-element vector

Range of disparity, specified as the comma-separated pair consisting of `'DisparityRange'` and a two-element vector of the form [*MinDisparity MaxDisparity*]. *MinDisparity* is the minimum disparity and *MaxDisparity* is the maximum disparity.

For input images of width *N*, the *MinDisparity* and *MaxDisparity* must be integers in the range (–*N*, *N*). The difference between the *MaxDisparity* and *MinDisparity* values must be less than or equal to 128. If the difference is not divisible by 8, the function internally rounds up the disparity range to the nearest multiple of 8. The function returns the actual value used in the disparity map computation in the `disparityRangeUsed` output argument.

The default value for the range of disparity is `[0 128]`. For more information on choosing the range of disparity, see “Choosing Range of Disparity” on page 3-647.

Data Types: `integers`

`UniquenessThreshold` — Minimum value of uniqueness
15 (default) | non-negative integer

Minimum value of uniqueness, specified as the comma-separated pair consisting of `'UniquenessThreshold'` and a nonnegative integer.

The function marks the estimated disparity value *K* for a pixel as unreliable, if:

$$v < V \times (1 + 0.01 \times \texttt{UniquenessThreshold}),$$

where *V* is the Hamming distance corresponding to the disparity value *K*. *v* is the smallest Hamming distance value over the whole disparity range, excluding *K*, *K*–1, and *K*+1.

Increasing the value of `UniquenessThreshold` results in disparity values for more pixels being marked as unreliable. To disable the use of uniqueness threshold, set this value to 0.

Output Arguments

`disparityMap` — Disparity map
2-D grayscale image | `gpuArray` object

Disparity map for rectified stereo pair image, returned as a 2-D grayscale image or a `gpuArray` object. The function returns the disparity map with the same size as input images `I1` and `I2`. Each value in this output refers to the displacement between conjugate pixels in the stereo pair image. For details about computing the disparity map, see “Computing Disparity Map Using Semi-Global Matching” on page 3-648.

Data Types: `single`

`disparityRangeUsed` — Range of disparity actually used in the disparity map computation
2 element vector

Range of disparity actually used in the disparity map computation, returned as a two-element vector of the form [*MinDisparity MaxDisparity*]. *MinDisparity* is the minimum disparity and *MaxDisparity* is the maximum disparity. This value can differ from the value specified using the `DisparityRange` name-value argument in order to satisfy internal numerical constraints, where the difference between the *MaxDisparity* and *MinDisparity* values must be divisible by 8.

Data Types: `double`

More About

Image Rectification

The input images `I1` and `I2` must be rectified before computing the disparity map. The rectification ensures that the corresponding points in the stereo pair image are on the same rows. You can rectify the input stereo pair image by using the `rectifyStereoImages` function. The reference image must be the same for rectification and disparity map computation.

Tips

- Ensure accurate rectification — Imperfect rectification can significantly degrade the quality of the disparitySGM function results. Use the `rectifyStereoImages` function to rectify the input stereo image pair `I1` and `I2`, so that the corresponding points are on the same rows. Use the `stereoAnaglyph` function to inspect the rectification results by creating an anaglyph image.
- Apply post-processing — For better results, improve disparity maps by applying image processing operations such as median filtering (`medfilt2`), hole filling (`imfill`), or bilateral filtering (`imbilatfilt`).
- Explore deep learning-based alternatives — For well-rectified, narrow-baseline stereo setups where classical SGM struggles with texture-less regions or repeated patterns, try `opticalFlowRAFT` as a deep learning-based alternative. For more information on comparison between classical SGM and RAFT deep learning model, see “Compare RAFT Optical Flow and Semi-Global Matching for Stereo Reconstruction”.

Algorithms

Choosing Range of Disparity

The range of disparity must be chosen to cover the minimum and the maximum amount of horizontal shift between the corresponding pixels in the rectified stereo pair image. You can determine the approximate horizontal shift values from the stereo anaglyph of the stereo pair image. Compute the stereo anaglyph of the rectified images by using the `stereoAnaglyph` function. Display the stereo anaglyph in the Image Viewer app. To measure the amount of horizontal shift between the corresponding points in the stereo pair image, select **Measure Distance** from the **Tools** menu in Image Viewer. Choose the minimum and maximum disparity values for the range of disparity based on this measurement.

For example, this figure displays the stereo anaglyph of a rectified stereo pair image and the horizontal shift values measured between the corresponding points in the stereo pair image. The minimum and maximum shift values are computed as 12.3101 and 32.3757 respectively. Based on these values, the range of disparity can be chosen as [0, 48].



Computing Disparity Map Using Semi-Global Matching

1. Compute Census transform of the rectified stereo pair image.
2. Compute Hamming distance between pixels in the census-transformed image to obtain the matching cost matrix.
3. Compute the pixel-wise disparity from matching cost matrix by using the semi-global matching method given in [1].
4. Optionally, mark the pixels for unreliability based on the `UniquenessThreshold` name-value pair. The function sets the disparity values of the unreliable pixels to `NaN`.

Version History

Introduced in R2019a

R2026a: Disparity Range No Longer Required to Be Divisible by 8
Behavior changed in R2026a

When you specify the `DisparityRange` name-value argument, the difference between the *MaxDisparity* and *MinDisparity* values is no longer required to be divisible by 8. If the difference is not divisible by 8, the function internally rounds up the disparity range to the nearest multiple of 8. To see the actual value used in the disparity map computation, use the `disparityRangeUsed` output argument.

References

[1] Hirschmuller, H. "Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 807-814. San Diego, CA: IEEE, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.
Usage notes and limitations:

- The name-value pair arguments, `'DisparityRange'` and `'UniquenessThreshold'` must be compile-time constants.
- Supports code generation only in generic `MATLAB Host Computer` target platform.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input images `I1` and `I2` must be rectified, same size, and of same data type.
- GPU code generation supports the `'UniquenessThreshold'` and `'disparityMap'` name-value pairs.
- For very large inputs, the memory requirements of the algorithm may exceed the GPU device limits. In such cases, consider reducing the input size to proceed with code generation.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

The `disparitySGM` function fully supports GPU arrays. To run the function on a GPU, specify the input data as a `gpuArray`. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Apps

Stereo Camera Calibrator

Functions

`disparityBM` | `reconstructScene` | `estimateUncalibratedRectification` | `rectifyStereoImages` | `estimateCameraParameters`

dist

Calculate distance between transformations

Syntax

```
distance = dist(transformation1,transformation2)
distance = dist(transformation1,transformation2,weights)
distance = dist(rotation1,rotation2)
```

Description

`distance = dist(transformation1,transformation2)` returns the distance `distance` between the poses represented by transformation `transformation1` and transformation `transformation2`.

For the homogeneous transformation object `se3`, the `dist` function calculates translational and rotational distance independently and combines them in a weighted sum. Translational distance is the Euclidean distance, and rotational distance is the angular difference between the rotation quaternions of `transformation1` and `transformation2`.

`distance = dist(transformation1,transformation2,weights)` specifies the weights `weights` for the translational and rotational distances for calculating the weighted sum of two homogeneous transformations.

`distance = dist(rotation1,rotation2)` returns the distance `distance` between the poses represented by transformation `rotation1` and transformation `rotation2`.

For the homogeneous transformation object `se3`, the `dist` function calculates translational and rotational distance independently and combines them in a weighted sum. Translational distance is the Euclidean distance, and rotational distance is the angular difference between the rotation quaternions of `rotation1` and `rotation2`.

For rotation object `so3`, the `dist` function calculates the rotational distance as the angular difference between the rotation quaternions of `rotation1` and `rotation2`.

Input Arguments

`transformation1` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects, where *N* is the total number of transformations. If you specify `transformation1` as an array, each element must be of the same type.

Either `transformation1` or `transformation2` must be a scalar transformation object of the same type. For example, if `transformation1` is an array of `se3` objects, `transformation2` must be a scalar `se3` object.

`transformation2` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects, where *N* is the total number of transformations. If you specify `transformation2` as an array, each element must be of the same type.

Either `transformation1` or `transformation2` must be a scalar transformation object of the same type. For example, if `transformation1` is an array of `se3` objects, `transformation2` must be a scalar `se3` object.

`rotation1` — First rotation
`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object, or as an *N*-element array of rotation objects, where *N* is the total number of rotations. If you specify `rotation1` as an array, each element must be of the same type.

Either `rotation1` or `rotation2` must be a scalar rotation object of the same type. For example, if `rotation1` is an array of `so3` objects, `rotation2` must be a scalar `so3` object.

`rotation2` — Last rotation
`so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object, or as an *N*-element array of rotation objects, where *N* is the total number of rotations. If you specify `rotation2` as an array, each element must be of the same type.

Either `rotation1` or `rotation2` must be a scalar rotation object of the same type. For example, if `rotation1` is an array of `se3` objects, `rotation2` must be a scalar `se3` object.

`weights` — Weights of translation and rotation in distance sum
`[1.0 0.1]` (default) | two-element row vector

Weights of the translation and rotation in the distance sum, specified as a two-element row vector in the form [*WeightXYZ WeightQ*]. *WeightXYZ* is the translational weight and *WeightQ* is the rotational weight. Both weights must be nonnegative numeric values.

Data Types: `single` | `double`

Output Arguments

`distance` — Distance between transformations or rotations
nonnegative numeric scalar

Distance between transformations, returned as a nonnegative numeric scalar. The distance calculate changes depending on the transformation object type of `transformation1` and `transformation2` or `rotation1` and `rotation2`:

- `se3` — The `dist` function calculates translational and rotational distance independently and combines them in a weighted sum specified by the `weights` argument. The translational distance is the Euclidean distance between `transformation1` and `transformation2`. The rotational distance is the angular difference between the rotations of `transformation1` and `transformation2`.
- `so3` — The `dist` function calculates the rotational distance as the angular difference between the rotations of `rotation1` and `rotation2`.

To calculate the rotational distance, the `dist` function converts the rotation matrix of `transformation1` and `transformation2` or `rotation1` and `rotation2` into `quaternion` objects and uses the `quaternion` `dist` function to calculate the angular distance.

See Also

Functions
`normalize` | `interp` | `transform`

Objects
`se3` | `so3` | `quaternion`

epipolarLine

Compute epipolar lines for stereo images

Syntax

```
lines = epipolarLine(F,points)
lines = epipolarLine(F',points)
```

Description

`lines = epipolarLine(F,points)` returns an *M*-by-3 matrix, `lines`. The matrix represents the computed epipolar lines in image `I2` corresponding to the `points` in image `I1`. The input `F` represents the fundamental matrix that maps points in `I1` to epipolar lines in image `I2`.

`lines = epipolarLine(F',points)` The matrix represents the computed epipolar lines in image `I1` corresponding to the `points` in image `I2`.

Examples

Compute Fundamental Matrix

This example shows you how to compute the fundamental matrix. It uses the least median of squares method to find the inliers.

The points, matched_points1 and matched_points2, have been putatively matched.

```
load stereoPointPairs
[fLMedS,inliers] = estimateFundamentalMatrix(matchedPoints1,...
    matchedPoints2,'NumTrials',4000);
```

Show the inliers in the first image.

```
I1 = imread('viprectification_deskLeft.png');
figure;
subplot(121);
imshow(I1);
title('Inliers and Epipolar Lines in First Image'); hold on;
plot(matchedPoints1(inliers,1),matchedPoints1(inliers,2),'go')
```



Compute the epipolar lines in the first image.

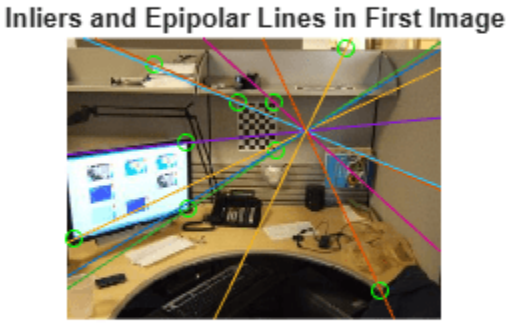
```
epiLines = epipolarLine(fLMedS',matchedPoints2(inliers,:));
```

Compute the intersection points of the lines and the image border.

```
points = lineToBorderPoints(epiLines,size(I1));
```

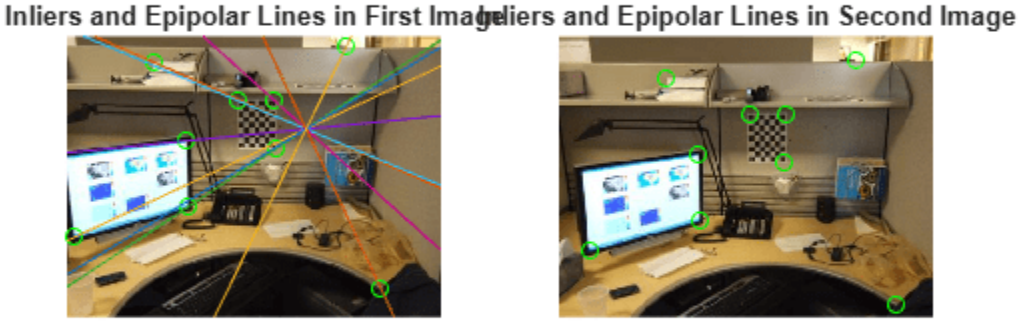
Show the epipolar lines in the first image

```
line(points(:,[1,3])',points(:,[2,4])');
```



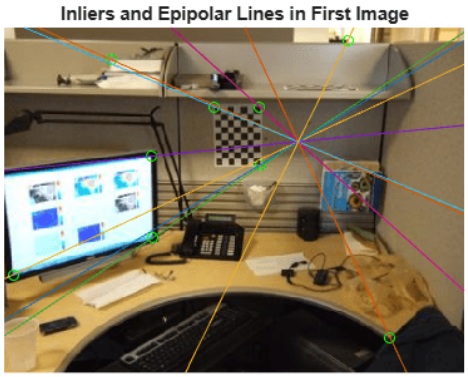
Show the inliers in the second image.

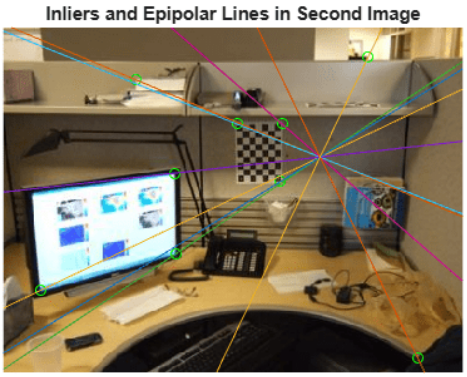
```
I2 = imread('viprectification_deskRight.png');
subplot(122);
imshow(I2);
title('Inliers and Epipolar Lines in Second Image'); hold on;
plot(matchedPoints2(inliers,1),matchedPoints2(inliers,2),'go')
```



Compute and show the epipolar lines in the second image.

```
epiLines = epipolarLine(fLMedS,matchedPoints1(inliers,:));
points = lineToBorderPoints(epiLines,size(I2));
line(points(:,[1,3])',points(:,[2,4])');
truesize;
```





Input Arguments

F — Fundamental matrix

3-by-3 matrix (default)

Fundamental matrix, specified as a 3-by-3 matrix. `F` must be double or single. If $P1$ represents a point in the first image $I1$ that corresponds to $P2$, a point in the second image $I2$, then:
$[P_2,1] * F * [P_1,1]' = 0$

In computer vision, the fundamental matrix is a 3-by-3 matrix which relates corresponding points in stereo images. When two cameras view a 3-D scene from two distinct positions, there are a number of geometric relations between the 3-D points and their projections onto the 2-D images that lead to constraints between the image points. Two images of the same scene are related by epipolar geometry.

Data Types: `single` | `double`

F' — Fundamental matrix

3-by-3 matrix (default)

Fundamental matrix, specified as a 3-by-3 matrix. The `F'` fundamental matrix maps points in image `I2` to epipolar lines in image `I1`.

Data Types: `single` | `double`

points — Coordinates of points

M-by-2 | point feature object

Coordinates of points, specified as an *M*-by-2 matrix, where *M* is the number of points, or a point feature object. The matrix contains the (x,y) coordinates for each point. For details on supported point feature objects, see “Point Feature Types”.

Data Types: `single` | `double`

Output Arguments

`lines` — Lines

M-by-3 matrix

An *M*-by-3 matrix, where *M* represents the number of lines. Each row of the matrix must be of the form, [*A*,*B*,*C*]. These values corresponds to the definition of the line:
$A * x + B * y + C = 0$.

Version History

Introduced in R2011a

R2022a: Epipolar Line: `epipolarLine` function accepts feature point objects

Behavior changed in R2022a

Support added to specify feature point objects described in “Point Feature Types”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`insertShape` | `size` | `line` | `isEpipoleInImage` | `lineToBorderPoints` | `estimateFundamentalMatrix`

Topics

“Point Feature Types”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

estgeotform2d

Estimate 2-D geometric transformation from matching point pairs

Syntax

```
tform = estgeotform2d(matchedPoints1,matchedPoints2,transformType)
[tform,inlierIndex] = estgeotform2d( ___ )
[tform,inlierIndex,status] = estgeotform2d( ___ )
[ ___ ] = estgeotform2d( ___ ,Name=Value)
```

Description

`tform = estgeotform2d(matchedPoints1,matchedPoints2,transformType)` estimates a 2-D geometric transformation between two images by mapping the inliers in the matched points from one image `matchedPoints1` to the inliers in the matched points from another image `matchedPoints2`.

`[tform,inlierIndex] = estgeotform2d( ___ )` additionally returns a vector specifying each matched point pair as either an inlier or an outlier using the input arguments from the previous syntax.

`[tform,inlierIndex,status] = estgeotform2d( ___ )` additionally returns a status code indicating whether or not the function could estimate a transformation and, if not, why it failed. If you do not specify the `status` output, the function instead returns an error for conditions that cannot produce results.

`[ ___ ] = estgeotform2d( ___ ,Name=Value)` specifies additional options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Confidence=99` sets the confidence value for finding the maximum number of inliers to 99.

Examples

Fit 2-D Geometric Transformation to SURF Feature Points

Read an image and display it.

```
original = imread("cameraman.tif");
imshow(original)
title("Original Image")
```



Distort and display the transformed image.

```
distorted = imresize(original,0.7);
distorted = imrotate(distorted,31);

imshow(distorted)
title("Transformed Image")
```



Detect and extract features from the original and the transformed images.

```
ptsOriginal  = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);
[featuresOriginal,validPtsOriginal] = extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = extractFeatures(distorted,ptsDistorted);
```

Match and display features between the images.

```
index_pairs = matchFeatures(featuresOriginal,featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));

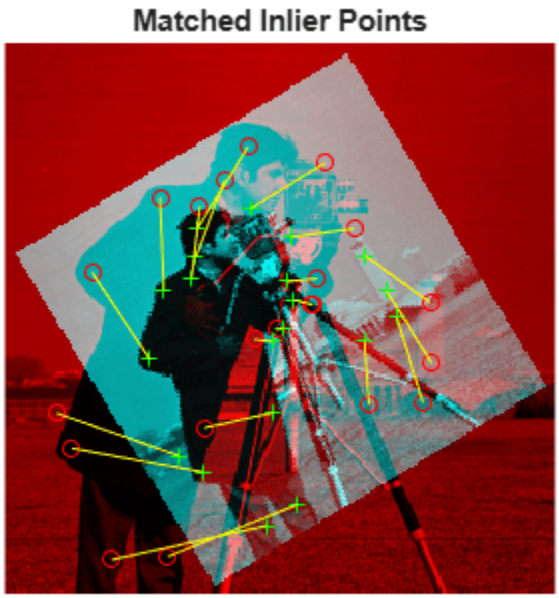
showMatchedFeatures(original,distorted,matchedPtsOriginal,matchedPtsDistorted)
title("Matched SURF Points With Outliers")
```



Exclude the outliers, estimate the transformation matrix, and display the results.

```
[tform,inlierIdx] = estgeotform2d(matchedPtsDistorted,matchedPtsOriginal,"similarity");
inlierPtsDistorted = matchedPtsDistorted(inlierIdx,:);
inlierPtsOriginal  = matchedPtsOriginal(inlierIdx,:);

showMatchedFeatures(original,distorted,inlierPtsOriginal,inlierPtsDistorted)
title("Matched Inlier Points")
```



Use the estimated transformation to recover and display the original image from the distorted image.

```
outputView = imref2d(size(original));
Ir = imwarp(distorted,tform,"OutputView",outputView);

imshow(Ir)
title("Recovered Image")
```



Input Arguments

`matchedPoints1` — Matched points from first image

M-by-2 matrix | point feature object

Matched points from the first image, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`matchedPoints2` — Matched points from second image

M-by-2 matrix | point feature object

Matched points from the second image, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`transformType` — Transformation type

`"rigid"` | `"similarity"` | `"affine"` | `"projective"` | `"translation"`

Transformation type, specified as `"rigid"`, `"similarity"`, `"affine"`, `"projective"`, or `"translation"`. Each transform type requires a minimum number of matched pairs of points to estimate a transformation. You can generally improve the accuracy of a transformation by using a larger number of matched pairs of points. This table shows the type of object associated with each transformation type and the minimum number of matched pairs of points the transformation requires.

Transformation Type	`tform` Object	Minimum Number of Matched Pairs of Points
`"rigid"`	`rigidtform2d`	2
`"similarity"`	`simtform2d`	2
`"affine"`	`affinetform2d`	3
`"projective"`	`projtform2d`	4
`"translation"`	`transltform2d`	1

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Confidence=99` sets the confidence value for finding the maximum number of inliers to `99`.

`MaxNumTrials` — Maximum random trials

`1000` (default) | positive integer

Maximum number of random trials, specified as a positive integer. This value specifies the number of randomized attempts the function makes to find matching point pairs. Specifying a higher value causes the function to perform additional computations, which increases the likelihood of finding inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Confidence` — Confidence of finding maximum number of inliers
99 (default) | positive numeric scalar

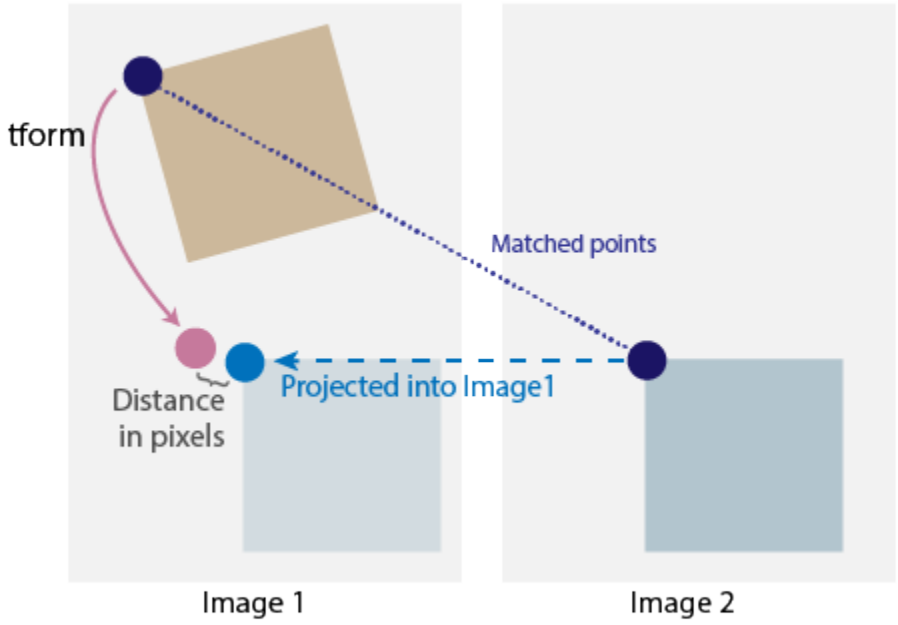
Confidence of finding the maximum number of inliers, specified as a positive numeric scalar in the range (0, 100). Increasing this value causes the function to perform additional computations, which increases the likelihood of finding a greater number of inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxDistance` — Maximum distance from point to projection
`1.5` (default) | positive numeric scalar

Maximum distance from a point to the projection of the corresponding point, specified as a positive numeric scalar. `MaxDistance` specifies the maximum distance, in pixels, that a point can differ from the projected location of its corresponding point to be considered an inlier. The corresponding projection is based on the estimated transform.

The function checks for a transformation from `matchedPoints1` to `matchedPoints2`, and then calculates the distance between the matched points in each pair after applying the transformation. If the distance between the matched points in a pair is greater than the `MaxDistance` value, then the pair is considered an outlier for that transformation. If the distance is less than `MaxDistance`, then the pair is considered an inlier.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`tform` — Geometric transformation
`rigidtform2d` object | `simtform2d` object | `affinetform2d` object | `projtform2d` object | `transltform2d` object

Geometric transformation, returned as a `rigidtform2d`, `simtform2d`, `affinetform2d`, `projtform2d`, or `transltform2d` object.

The returned geometric transformation matrix maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`. The function returns an object specific to the transformation type specified by the `transformType` input argument.

`transformType`	Geometric Transformation Object
`"rigid"`	`rigidtform2d`
`"similarity"`	`simtform2d`
`"affine"`	`affinetform2d`
`"projective"`	`projtform2d`
`"translation"`	`transltform2d`

`inlierIndex` — Index of inliers
M-by-1 logical vector

Index of inliers, returned as an *M*-by-1 logical vector, where *M* is the number of point pairs. Each element contains either a logical `1` (`true`), indicating that the corresponding point pair is an inlier, or a logical `0` (`false`), indicating that the corresponding point pair is an outlier.

Data Types: `logical`

`status` — Status code
`0` | `1` | `2`

Status code, returned as `0`, `1`, or `2`. The status code indicates whether or not the function could estimate the transformation and, if not, why it failed.

Value	Description
`0`	No error
`1`	`matchedPoints1` and `matchedPoints2` inputs do not contain enough points
`2`	Not enough inliers found

If you do not specify the `status` code output, the function returns an error if it cannot produce results.

Data Types: `int32`

Algorithms

The function excludes outliers using the M-estimator sample consensus (MSAC) algorithm. The MSAC algorithm is a variant of the random sample consensus (RANSAC) algorithm. Results may not be identical between runs due to the randomized nature of the MSAC algorithm.

Version History

Introduced in R2022b

R2022b: Recommended over `estimateGeometricTransform2D`

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateGeometricTransform2D` function estimates geometric transformations using the old postmultiply convention. Although there are no plans to remove `estimateGeometricTransform2D` at this time, you can streamline your geometric transformation workflows by switching to the `estgeotform2d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- You cannot use a `SURFPoints` object as an input to `estgeotform2d`. See the “Introduction to Code Generation with Feature Matching and Registration” example for details on how to use `SURFPoints` with `estgeotform2d` in code generation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`estgeotform3d` | `estimateFundamentalMatrix` | `detectSURFFeatures` | `detectMinEigenFeatures` | `detectORBFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `extractFeatures` | `matchFeatures` | `fitgeotform2d`

Objects

`KAZEPoints` | `cornerPoints` | `MSERRegions` | `SURFPoints` | `ORBPoints` | `BRISKPoints` | `transltform2d` | `rigidtform2d` | `simtform2d` | `affinetform2d` | `projtform2d`

Topics

“Create Panorama”
“Point Feature Types”
“Coordinate Systems”
“2-D and 3-D Geometric Transformation Process Overview”
“Introduction to Code Generation with Feature Matching and Registration”

estgeotform3d

Estimate 3-D geometric transformation from matching point pairs

Syntax

```
tform = estgeotform3d(matchedPoints1,matchedPoints2,transformType)
[tform,inlierIndex] = estgeotform3d( ___ )
[tform,inlierIndex,status] = estgeotform3d( ___ )
[ ___ ] = estgeotform3d( ___ ,Name=Value)
```

Description

`tform = estgeotform3d(matchedPoints1,matchedPoints2,transformType)` estimates a 3-D geometric transformation between two sets of 3-D points by mapping the inliers in the matched points from one set of 3-D points `matchedPoints1` to the inliers in the matched points from the other set of 3-D points `matchedPoints2`.

`[tform,inlierIndex] = estgeotform3d( ___ )` additionally returns a vector specifying each matched point pair as either an inlier or an outlier using the input arguments from the previous syntax.

`[tform,inlierIndex,status] = estgeotform3d( ___ )` additionally returns a status code indicating whether or not the function could estimate a transformation and, if not, why it failed. If you do not specify the `status` output, the function instead returns an error for conditions that cannot produce results.

`[ ___ ] = estgeotform3d( ___ ,Name=Value)` specifies additional options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Confidence=99` sets the confidence value for finding the maximum number of inliers to 99.

Examples

Fit 3-D Geometric Transformation to Point Clouds

Load a point cloud file into the workspace.

```
ptCloud1 = pcread("teapot.ply")
```

```
ptCloud1 =
  pointCloud with properties:

     Location: [41472×3 single]
        Count: 41472
      XLimits: [-3 3.4340]
      YLimits: [-2 2]
      ZLimits: [0 3.1500]
        Color: []
       Normal: []
    Intensity: []
```

```
ptCloud1 = pcdownsample(ptCloud1,"random",0.25);
```

Create a rigid 3-D transformation object with a 30-degree rotation about the *z*-axis.

```
eulerAngles = [0 0 30];
trans = [0 0 0];
tform = rigidtform3d(eulerAngles,trans);
```

Transform the point cloud using the transformation object.

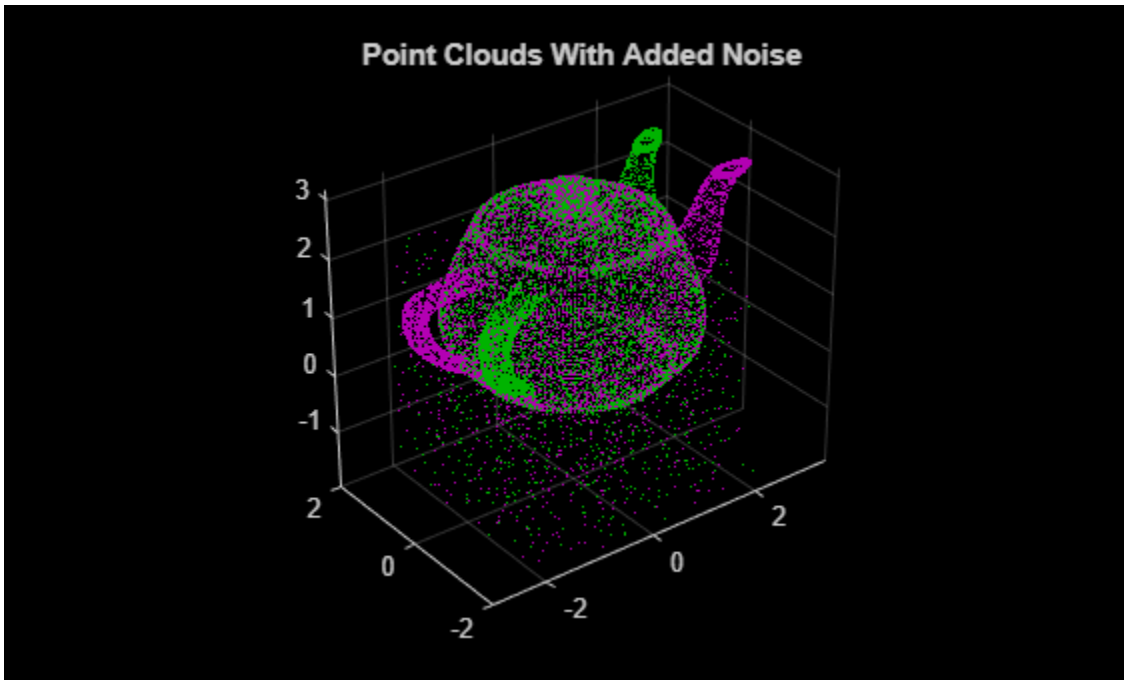
```
ptCloud2 = pctransform(ptCloud1,tform);
```

To introduce noise, add random points to both point clouds.

```
noise1 = rescale(rand(1000,3),-2,2);
ptCloud1 = pointCloud([ptCloud1.Location;noise1]);
noise2 = rescale(rand(1000,3),-2,2);
ptCloud2 = pointCloud([ptCloud2.Location;noise2]);
```

Visualize the noisy point clouds.

```
figure
pcshowpair(ptCloud1,ptCloud2)
title("Point Clouds With Added Noise")
```



Extract matched points from the point clouds.

```
matchedPoints1 = ptCloud1.Location;
matchedPoints2 = ptCloud2.Location;
```

Estimate the rigid transformation between the point clouds.

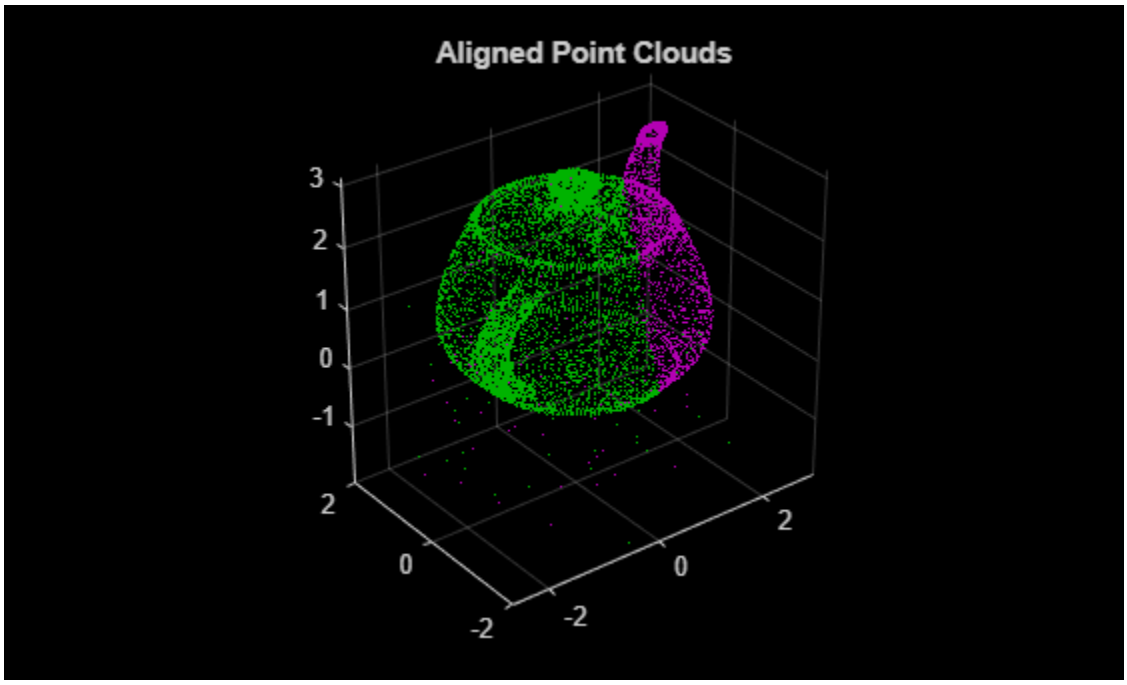
```
[tformEst,inlierIndex] = estgeotform3d(matchedPoints1, ...
    matchedPoints2,"rigid");
```

Extract the inlier points.

```
inliersPtCloud1 = transformPointsForward(tformEst, ...
    matchedPoints1(inlierIndex,:));
inliersPtCloud2 = matchedPoints2(inlierIndex,:);
```

Visualize the inliers of the aligned point clouds.

```
figure
firstPtCloud = pointCloud(inliersPtCloud1);
secondPtCloud = pointCloud(inliersPtCloud2);
pcshowpair(firstPtCloud,secondPtCloud)
title("Aligned Point Clouds")
```



Input Arguments

`matchedPoints1` — Matched points from first image

M-by-3 matrix of [*x*,*y*,*z*] coordinates

Matched points from the first image, specified as an *M*-by-3 matrix in which each row is a set of (*x*,*y*,*z*) coordinates and *M* is the number of matched points.

`matchedPoints2` — Matched points from second image

M-by-3 matrix of [*x*,*y*,*z*] coordinates

Matched points from the second image, specified as an *M*-by-3 matrix in which each row is a set of (*x*,*y*,*z*) coordinates and *M* is the number of matched points.

`transformType` — Transformation type

`"rigid"` | `"similarity"` | `"translation"`

Transformation type, specified as `"rigid"`, `"similarity"`, or `"translation"`. Each transform type requires a minimum number of matched pairs of points to estimate a transformation. You can generally improve the accuracy of a transformation by using a larger number of matched pairs of points. This table shows the type of object associated with each transformation type and the minimum number of matched pairs of points the transformation requires.

transformType	**tform Object**	**Minimum Number of Matched Pairs of Points**
`"rigid"`	`rigidtform3d`	3
`"similarity"`	`simtform3d`	3
`"translation"`	`transltform3d`	1

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Confidence=99` sets the confidence value for finding the maximum number of inliers to 99.

`MaxNumTrials` — Maximum random trials

`1000` (default) | positive integer

Maximum number of random trials, specified as a positive integer. This value specifies the number of randomized attempts the function makes to find matching point pairs. Specifying a higher value causes the function to perform additional computations, which increases the likelihood of finding inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Confidence` — Confidence of finding maximum number of inliers

`99` (default) | positive numeric scalar

Confidence of finding the maximum number of inliers, specified as a positive numeric scalar in the range (0, 100). Increasing this value causes the function to perform additional computations, which increases the likelihood of finding a greater number of inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

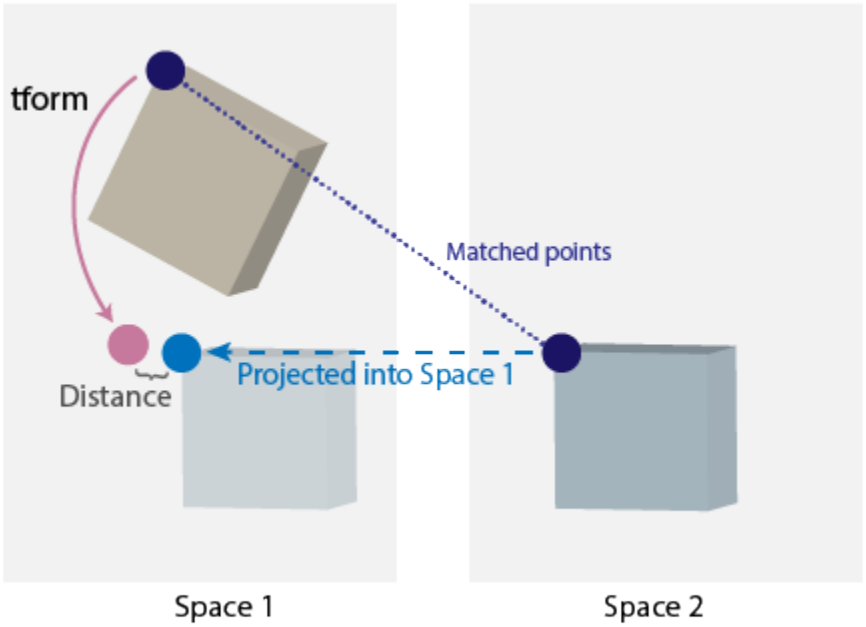
`MaxDistance` — Maximum distance from point to projection

`1.5` (default) | positive numeric scalar

Maximum distance from a point to the projection of the corresponding point, specified as a positive numeric scalar. `MaxDistance` specifies the maximum distance, in pixels, that a point can differ from the projected location of its corresponding point to be considered an inlier. The corresponding projection is based on the estimated transform.

The function checks for a transformation from `matchedPoints1` to `matchedPoints2`, and then calculates the distance between the matched points in each pair after applying the transformation. If the distance between the matched points in a pair is greater than the `MaxDistance` value, then the

pair is considered an outlier for that transformation. If the distance is less than `MaxDistance`, then the pair is considered an inlier.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`tform` — Geometric transformation

`rigidtform3d` object | `simtform3d` object | `transltform3d` object

Geometric transformation, returned as a `rigidtform3d`, `simtform3d`, or `transltform3d` object.

The returned geometric transformation matrix maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`. The function returns an object specific to the transformation type specified by the `transformType` input argument.

`transformType`	`tform`
`"rigid"`	`rigidtform3d`
`"similarity"`	`simtform3d`
`"translation"`	`transltform3d`

`inlierIndex` — Inliers

M-by-1 logical vector

Inliers, returned as an *M*-by-1 logical vector of *M* point pairs. Each element contains either a logical true, `1`, to indicate the point pair is an inlier, or a logical false, `0`, to indicate the point pair is an outlier.

`status` — Status code

`0` | `1` | `2`

Status code, returned as `0`, `1`, or `2`. The status code indicates whether or not the function could estimate the transformation and, if not, why it failed.

Value	Description
`0`	No error
`1`	`matchedPoints1` and `matchedPoints2` inputs do not contain enough points
`2`	Not enough inliers found

If you do not specify the `status` code output, the function returns an error if it cannot produce results.

Data Types: `int32`

Algorithms

The function excludes outliers using the M-estimator sample consensus (MSAC) algorithm. The MSAC algorithm is a variant of the random sample consensus (RANSAC) algorithm. Results may not be identical between runs due to the randomized nature of the MSAC algorithm.

Version History

Introduced in R2022b

R2022b: Recommended over `estimateGeometricTransform3D`

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateGeometricTransform3D` function estimates geometric transformations using the old postmultiply convention. Although there are no plans to remove `estimateGeometricTransform3D` at this time, you can streamline your geometric transformation workflows by switching to the `estgeotform3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding*. 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`estgeotform2d` | `pcregistericp` | `ransac`

Objects
`rigidtform3d` | `simtform3d` | `transltform3d`

Topics
“Coordinate Systems”
“2-D and 3-D Geometric Transformation Process Overview”

estimateCameraParameters

Calibrate a single or stereo camera

Syntax

```
[cameraParams,imagesUsed,estimationErrors] = estimateCameraParameters(
imagePoints,worldPoints)
[stereoParams,pairsUsed,estimationErrors] = estimateCameraParameters(
imagePoints,worldPoints)
cameraParams = estimateCameraParameters( ___ ,Name=Value)
```

Description

`[cameraParams,imagesUsed,estimationErrors] = estimateCameraParameters( imagePoints,worldPoints)` returns an object `cameraParams`, containing estimates for the intrinsic and extrinsic parameters and the distortion coefficients of a single and stereo camera. The function also returns the images you used to estimate the camera parameters and the standard estimation errors for the single camera calibration. The `estimateCameraParameters` function estimates extrinsic and intrinsic parameters.

`[stereoParams,pairsUsed,estimationErrors] = estimateCameraParameters( imagePoints,worldPoints)` returns `stereoParams`, a `stereoParameters` object containing the parameters of the stereo camera. The function also returns the images you used to estimate the stereo parameters and the standard estimation errors for the stereo camera calibration.

`cameraParams = estimateCameraParameters( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `WorldUnits="mm"` sets the world units to millimeters.

Examples

Single Camera Calibration

Create a set of calibration images.

```
imds = imageDatastore(fullfile(toolboxdir("vision"),"visiondata",...
            "calibration","mono"));
imageFileNames = imds.Files;
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate the world coordinates of the corners of the squares.

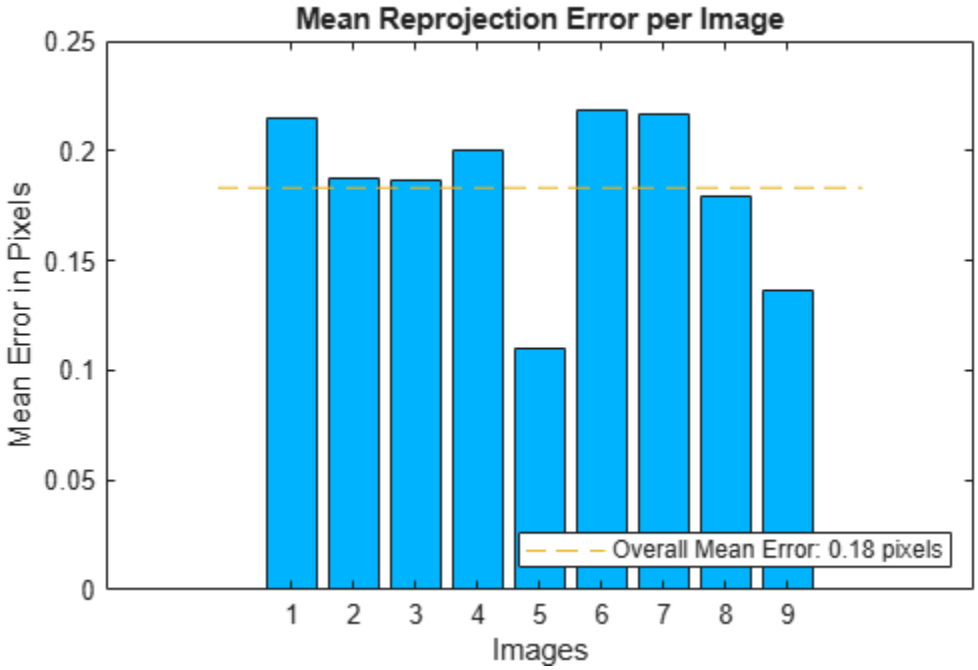
```
squareSizeInMM = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSizeInMM);
```

Calibrate the camera.

```
I = preview(imds);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

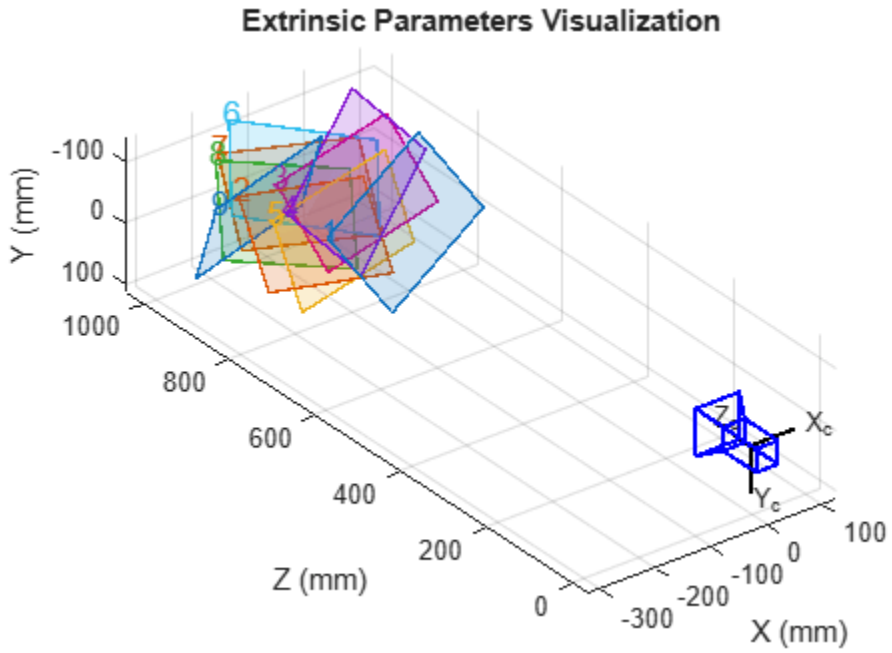
Visualize the calibration accuracy.

```
showReprojectionErrors(params);
```



Visualize camera extrinsics.

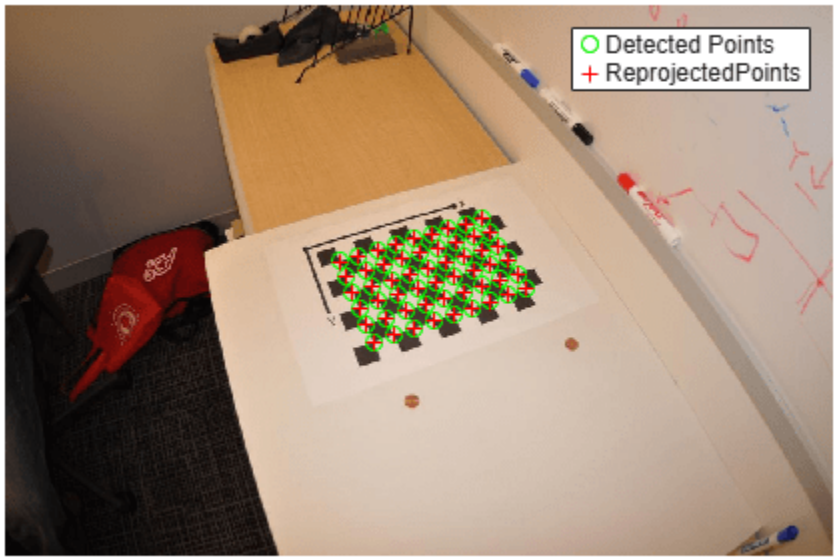
```
figure;
showExtrinsics(params);
```




```
drawnow;
```

Plot detected and reprojected points.

```
figure;
imshow(imageFileNames{1});
hold on;
plot(imagePoints(:,1,1), imagePoints(:,2,1),"go");
plot(params.ReprojectedPoints(:,1,1),params.ReprojectedPoints(:,2,1),"r+");
legend("Detected Points","ReprojectedPoints");
hold off;
```



Perform Stereo Camera Calibration

Specify calibration images.

```
leftImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","left"));
rightImages = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo","right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = ...
  detectCheckerboardPoints(leftImages.Files,rightImages.Files);
```

Specify the world coordinates of the checkerboard keypoints. Square size is in millimeters.

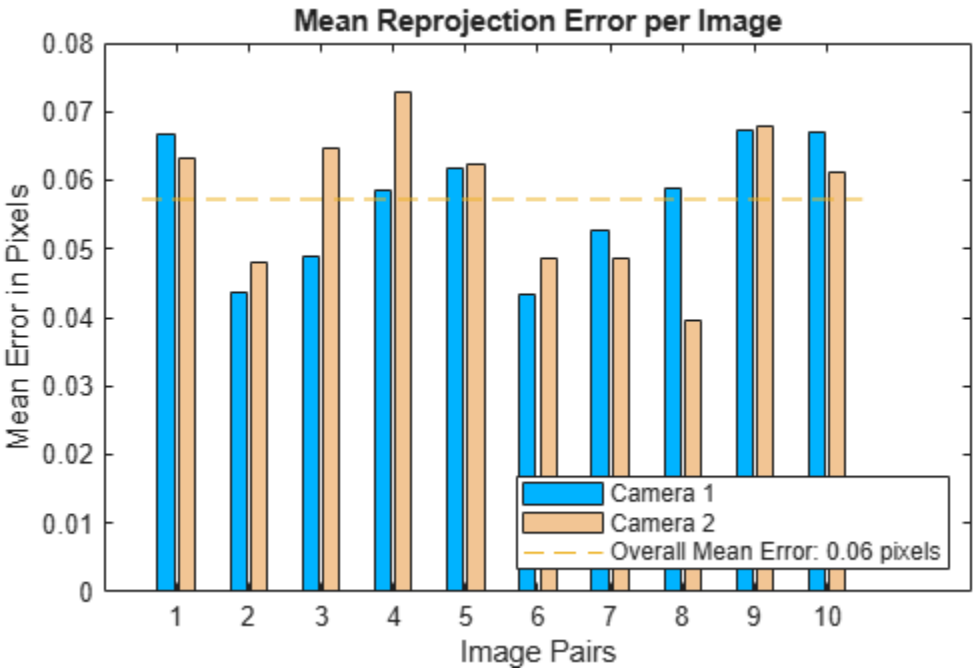
```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

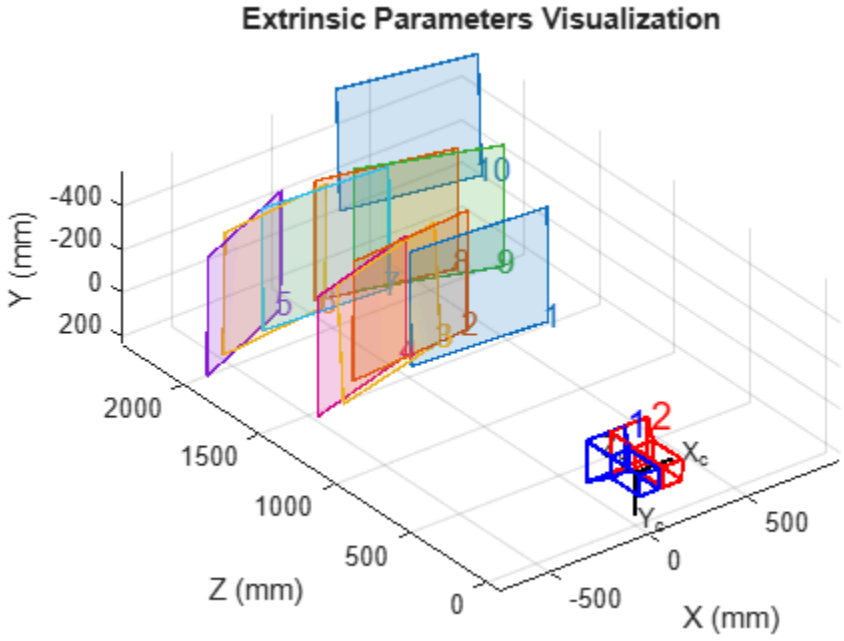
Visualize the calibration accuracy.

```
showReprojectionErrors(params)
```



Visualize camera extrinsics.

```
figure
showExtrinsics(params)
```



Convert to structure.

```
paramsStruct = toStruct(params)
```

```
paramsStruct = struct with fields:
       CameraParameters1: [1×1 struct]
       CameraParameters2: [1×1 struct]
       RotationOfCamera2: [3×3 double]
    TranslationOfCamera2: [-119.8720 -0.4005 -0.0258]
                 Version: [1×1 struct]
     RectificationParams: [1×1 struct]
```

Estimate Camera Parameters Using Circle Grid

Create a set of calibration images.

```
imds = imageDatastore(fullfile(toolboxdir("vision"),"visiondata",...
                        "calibration","circleGrid","mono"));
calibrationImages = readall(imds);
calibrationImages = cat(4,calibrationImages{:});
```

Define the circle grid pattern dimensions.

```
patternDims = [8 11];
```

Detect the pattern in the calibration images.

```
imagePoints = detectCircleGridPoints(calibrationImages, patternDims,...
                        PatternType="symmetric");
```

Specify the world coordinates for the circle grid keypoints. Center distance is in millimeters.

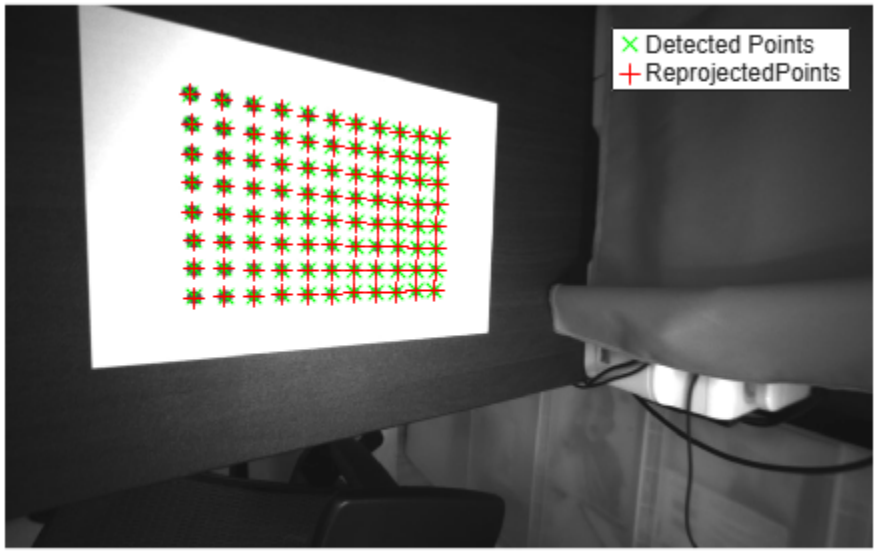
```
centerDistance = 18; % millimeters
worldPoints = patternWorldPoints("circle-grid-symmetric",patternDims,centerDistance);
```

Calibrate the camera using the calibration images.

```
imageSize = size(calibrationImages,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Plot the detected pattern grid and the reprojected points.

```
figure
imshow(calibrationImages(:,:,:,1))
hold on
plot(imagePoints(:,1,1), imagePoints(:,2,1),"gx",MarkerSize=8)
plot(params.ReprojectedPoints(:,1,1),params.ReprojectedPoints(:,2,1),"r+",MarkerSize=8)
legend("Detected Points","ReprojectedPoints")
hold off
```



Input Arguments

`imagePoints` — Key points of calibration pattern

M-by-2-by-*numImages* | *M*-by-2-by-*numPairs*-by-2 array

Key points of calibration pattern, specified as an array of [*x*,*y*] intrinsic image coordinates.

Calibration	Input Array of [*x*,*y*] Key Points
Single Camera	*M*-by-2-by-*numImages* array of [*x*,*y*] points. • The number of images, *numImages*, must be greater than or equal to 2. • The number of keypoint coordinates in each pattern, *M*, must be greater than 3. Partially detected patterns are only supported for single camera calibration. To include partially detected patterns in the estimate, use `[NaN,NaN]` as *x*-*y* coordinates for missing keypoints.
Stereo Camera	*M*-by-2-by-*numPairs*-by-2 array of [*x*,*y*] points. • *numPairs* is the number of stereo image pairs containing the calibration pattern. Must be greater than 2. • The number of keypoint coordinates in each pattern, *M*, must be greater than 3. • `imagePoints(:,:,:,1)` are the points from camera 1. • `imagePoints(:,:,:,2)` are the points from camera 2.

Data Types: `single` | `double`

`worldPoints` — Key points of calibration pattern in world coordinates
M-by-2 array

Key points of calibration pattern in world coordinates, specified as an *M*-by-2 array of *M* number of [*x*,*y*] world coordinates. The pattern must be planar; therefore, *z*-coordinates are zero. Calibration images must not be coplanar. Use only one calibration image per spatial plane.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"WorldUnits","mm"` sets the world point units to millimeters.

`WorldUnits` — World points units
`"mm"` (default) | character vector | string scalar

World points units, specified as a character vector or string scalar.

`EstimateSkew` — Estimate skew
`false` (default) | `true`

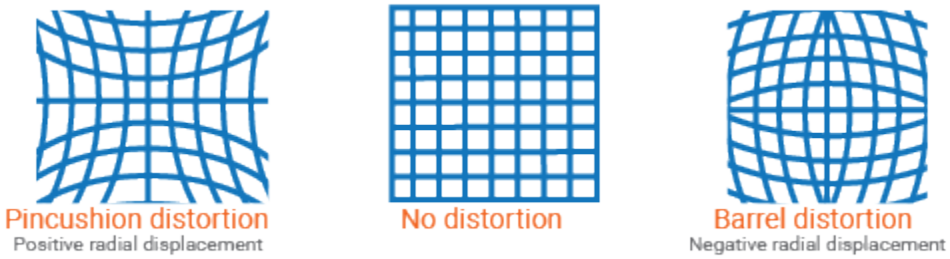
Estimate skew, specified as a logical scalar. When you set this property to `true`, the function estimates the image axes skew. When set to `false`, the image axes are exactly perpendicular and the function sets the skew to zero.

`NumRadialDistortionCoefficients` — Number of radial distortion coefficients
2 (default) | 3

Number of radial distortion coefficients to estimate, specified as the value 2 or 3.

Radial distortion is the displacement of image points along radial lines extending from the principal point.

- As image points move away from the principal point (positive radial displacement), image magnification decreases and a pincushion-shaped distortion occurs on the image.
- As image points move toward the principal point (negative radial displacement), image magnification increases and a barrel-shaped distortion occurs on the image.



The radial distortion coefficients model this type of distortion. The distorted points are denoted as $(x_{\text{distorted}}, y_{\text{distorted}})$:

$x_{distorted} = x(1 + k_1*r^2 + k_2*r^4 + k_3*r^6)$

$y_{distorted} = y(1 + k_1*r^2 + k_2*r^4 + k_3*r^6)$

- x, y — Undistorted pixel locations. x and y are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, x and y are dimensionless.
- k_1, k_2, and k_3 — Radial distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

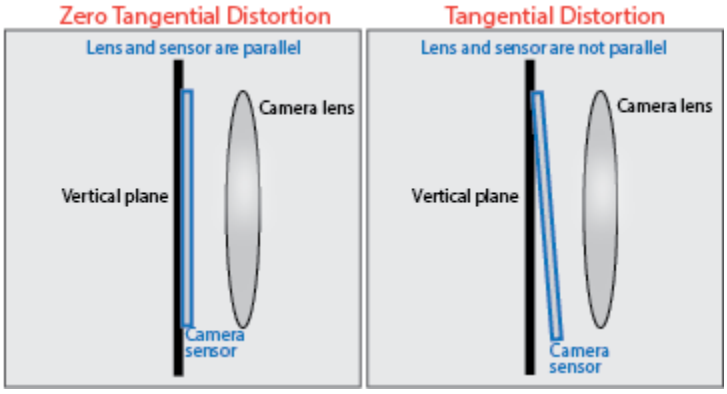
Typically, two coefficients are sufficient for calibration. For severe distortion, such as in wide-angle lenses, you can select three coefficients to include k_3.

`EstimateTangentialDistortion` — Tangential distortion flag

`false` (default) | `true`

Tangential distortion flag, specified as a logical scalar. When you set this property to `true`, the function estimates the tangential distortion. When you set it to `false`, the tangential distortion is negligible.

Tangential distortion occurs when the lens and the image plane are not parallel. The tangential distortion coefficients model this type of distortion.



The distorted points are denoted as $(x_{distorted}, y_{distorted})$:

$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$

$y_{distorted} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$

- x, y — Undistorted pixel locations. x and y are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, x and y are dimensionless.
- p_1 and p_2 — Tangential distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

`InitialK` — Initial guess for camera intrinsics

`[ ]` (default) | 3-by-3 matrix

Initial guess for camera intrinsics, specified as a 3-by-3 matrix. The matrix has this format:

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The coordinates $[c_x\ c_y]$ represent the optical center (the principal point), in pixels. When the *x*- and *y*-axes are exactly perpendicular, the skew parameter *s* equals `0`.

$f_x = F*s_x$
$f_y = F*s_y$

- *F* is the focal length in world units, typically expressed in millimeters.
- s_x and s_y are the number of pixels per world unit in the *x*- and *y*-direction respectively.
- f_x and f_y are expressed in pixels.

If you do not provide an initial value, the function computes the initial intrinsic matrix using linear least squares.

`InitialRadialDistortion` — Initial guess for radial distortion coefficients
`[ ]` (default) | 2-element vector | 3-element vector

Initial guess for radial distortion coefficients, specified as a 2- or 3-element vector. If you do not provide an initial value, the function uses `0` as the initial value for all the coefficients.

`ImageSize` — Image size produced by camera
1-by-2 [*mrows*, *ncols*] vector | []

Image size produced by camera, specified as a 1-by-2 [*mrows*, *ncols*] vector. To return the camera `Intrinsics` property values, you must specify the `ImageSize` name-value argument.

Output Arguments

`cameraParams` — Camera parameters
`cameraParameters` object

Camera parameters, returned as a `cameraParameters` object. If the `ImageSize` name-value argument was not specified with the input, the `cameraParams` object will not contain the `Intrinsics` property values.

`imagesUsed` — Images used to estimate camera parameters
numImages-by-1 logical array

Images you use to estimate camera parameters, returned as a *numImages*-by-1 logical array. *numImages* corresponds to the number of images. The array indicates which images you used to estimate the camera parameters. A logical `true` value in the array indicates which images you used to estimate the camera parameters.

The function computes a homography between the world points and the points detected in each image. If the homography computation fails for an image, the function issues a warning. The points for that image are not used for estimating the camera parameters. The function also sets the corresponding element of `imagesUsed` to `false`.

`estimationErrors` — Standard errors of estimated parameters
`cameraCalibrationErrors` object | `stereoCalibrationErrors` object

Standard errors of estimated parameters, returned as a `cameraCalibrationErrors` object or a `stereoCalibrationErrors` object.

`stereoParams` — Camera parameters for stereo system
`stereoParameters` object

Camera parameters for stereo system, returned as a `stereoParameters` object. The object contains the intrinsic, extrinsic, and lens distortion parameters of the stereo camera system.

`pairsUsed` — Image pairs used to estimate stereo camera parameters
numPairs-by-1 logical array

Image pairs used to estimate camera parameters, returned as a *numPairs*-by-1 logical array. *numPairs* corresponds to the number of image pairs. A logical `true` value in the array indicates which image pairs you used to estimate the camera parameters.

Algorithms

Calibration Algorithm

You can use the Camera Calibrator app with cameras up to a field of view (FOV) of 95 degrees. The calibration algorithm assumes a pinhole camera model.

This equation provides the transformation that relates a world coordinate $[X\ Y\ Z]$ and the corresponding image point $[x\ y]$:

$$w\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R\ t]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- w: arbitrary scale factor
- K: camera intrinsic matrix
- R: matrix representing the 3-D rotation of the camera
- t: translation of the camera relative to the world coordinate system

Camera calibration estimates the values of the intrinsic parameters, the extrinsic parameters, and the distortion coefficients. There are two steps involved in camera calibration:

1. Solve for the intrinsics and extrinsics in closed form, assuming that lens distortion is zero. [1]
2. Estimate all parameters simultaneously including the distortion coefficients using nonlinear least-squares minimization (Levenberg–Marquardt algorithm). Use the closed form solution from the preceding step as the initial estimate of the intrinsics and extrinsics. Then set the initial estimate of the distortion coefficients to zero. [1][2]

Version History

Introduced in R2014b

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler™ will be removed in a future release.

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, to specify an initial guess for camera intrinsics in the premultiply convention, use the new `InitialK` name-value argument.

Although you can still specify an initial guess for camera intrinsics using the `InitialIntrinsicMatrix` name-value argument, this argument is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see "Migrate Geometric Transformations to Premultiply Convention".

References

[1] Zhang, Z. "A Flexible New Technique for Camera Calibration." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, no. 11 (November 2000): 1330–34. https://doi.org/10.1109/34.888718.

[2] Heikkila, J., and O. Silven. "A Four-Step Camera Calibration Procedure with Implicit Image Correction." In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1106–12. San Juan, Puerto Rico: IEEE Comput. Soc, 1997. https://doi.org/10.1109/CVPR.1997.609468.

[3] Bouguet, J.Y. "Camera Calibration Toolbox for Matlab", Computational Vision at the California Institute of Technology.

[4] Bradski, G., and A. Kaehler. *Learning OpenCV : Computer Vision with the OpenCV Library*. Sebastopol, CA: O'Reilly, 2008.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Objects
`cameraCalibrationErrors` | `stereoCalibrationErrors` | `stereoParameters` | `cameraParameters`

Functions
`showReprojectionErrors` | `showExtrinsics` | `undistortImage` | `undistortPoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `patternWorldPoints` | `reconstructScene` | `rectifyStereoImages` | `disparityBM` | `disparitySGM` | `estimateStereoRectification` | `estimateFundamentalMatrix` | `estimateStereoBaseline`

Topics
"What Is Camera Calibration?"
"Using the Single Camera Calibrator App"
"Coordinate Systems"
"Evaluating the Accuracy of Single Camera Calibration"

estimateFundamentalMatrix

Estimate fundamental matrix from corresponding points in stereo images

Syntax

```
estimateFundamentalMatrix
F = estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
[F,inliersIndex] = estimateFundamentalMatrix(matchedPoints1,matchedPoints2)
[F,inliersIndex,status] = estimateFundamentalMatrix(matchedPoints1,
matchedPoints2)
[F,inliersIndex,status] = estimateFundamentalMatrix(matchedPoints1,
matchedPoints2,Name=Value)
```

Description

`estimateFundamentalMatrix` estimates the fundamental matrix from corresponding points in stereo images. This function can be configured to use all corresponding points or to exclude outliers. You can exclude outliers by using a robust estimation technique such as random-sample consensus (RANSAC). When you use robust estimation, results may not be identical between runs because of the randomized nature of the algorithm.

`F = estimateFundamentalMatrix(matchedPoints1,matchedPoints2)` returns the 3-by-3 fundamental matrix, `F`, using the least median of squares (LMedS) method from matched feature points in stereo images.

`[F,inliersIndex] = estimateFundamentalMatrix(matchedPoints1,matchedPoints2)` additionally returns logical indices, `inliersIndex`, for the inliers used to compute the fundamental matrix. The `inliersIndex` output is an *M*-by-1 vector. The function sets the elements of the vector to `true` when the corresponding point was used to compute the fundamental matrix. The elements are set to `false` if they are not used.

`[F,inliersIndex,status] = estimateFundamentalMatrix(matchedPoints1, matchedPoints2)` additionally returns a status code.

`[F,inliersIndex,status] = estimateFundamentalMatrix(matchedPoints1, matchedPoints2,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `estimateFundamentalMatrix(matchedPoints1,matchedPoints2,Method="MSAC")` specifies MSAC as the method to compute the fundamental matrix.

Examples

Compute Fundamental Matrix

Use the random sample consensus (RANSAC) method to compute the fundamental matrix. The RANSAC method requires that the input points are putatively matched. You can use the matchFeatures function to return these matched points. Outlier points which may still be contained within putatively matched points are further eliminated by using the RANSAC algorithm.

Load stereo points into the workspace.

```
load stereoPointPairs
```

Estimate the fundamental matrix.

```
fRANSAC = estimateFundamentalMatrix(matchedPoints1, ...
    matchedPoints2,Method="RANSAC", ...
    NumTrials=2000,DistanceThreshold=1e-4)
```

```
fRANSAC = 3×3

    0.0000   -0.0004    0.0348
    0.0004    0.0000   -0.0937
   -0.0426    0.0993    0.9892
```

Use Least Median of Squares Method to Find Inliers

Load the putatively matched points into the workspace.

```
load stereoPointPairs
[fLMedS,inliers] = estimateFundamentalMatrix(matchedPoints1,matchedPoints2,NumTrials=2000)
```

```
fLMedS = 3×3

    0.0000   -0.0004    0.0349
    0.0004    0.0000   -0.0938
   -0.0426    0.0994    0.9892


inliers = 18×1 logical array

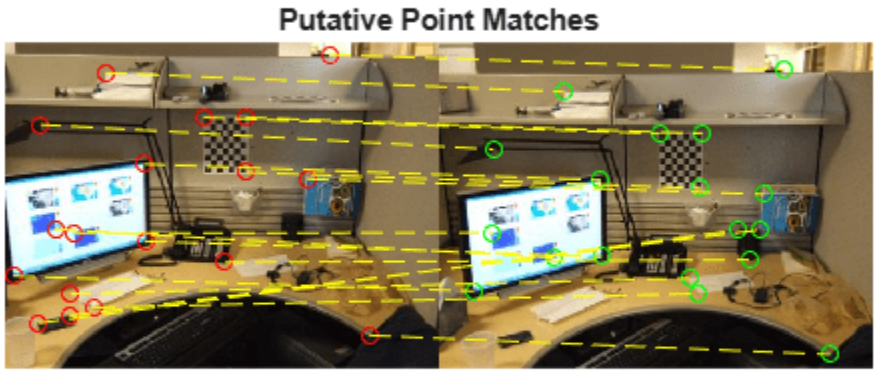
   1
   1
   1
   1
   1
   1
   0
   1
   0
   0
   0
   1
   0
   0
   0
      ⋮
```

Load the stereo images.

```
I1 = imread("viprectification_deskLeft.png");
I2 = imread("viprectification_deskRight.png");
```

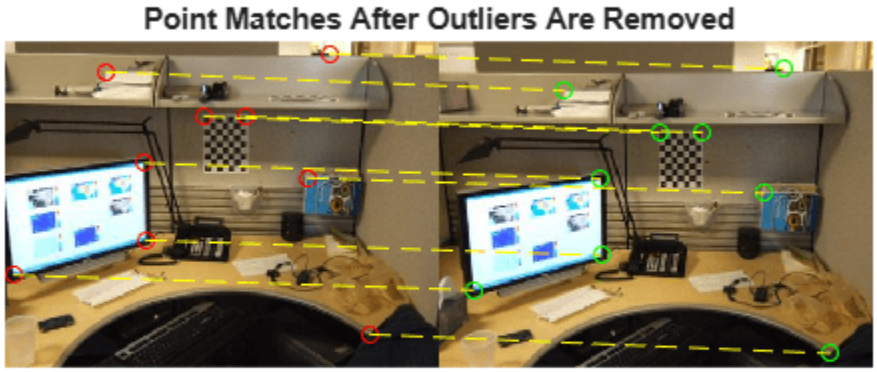
Show the putatively matched points.

```
figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,"montage",PlotOptions=["ro","go","y--"]);
title("Putative Point Matches");
```



Show the inlier points.

```
figure;
showMatchedFeatures(I1,I2,matchedPoints1(inliers,:),matchedPoints2(inliers,:),"montage",PlotOptio
title("Point Matches After Outliers Are Removed");
```



Use Normalized Eight-Point Algorithm to Compute Fundamental Matrix

Load the stereo point pairs into the workspace.

```
load stereoPointPairs
```

Compute the fundamental matrix for input points which do not contain any outliers.

```
inlierPts1 = matchedPoints1(knownInliers,:);
inlierPts2 = matchedPoints2(knownInliers,:);
fNorm8Point = estimateFundamentalMatrix(inlierPts1,inlierPts2,Method="Norm8Point")
```

```
fNorm8Point = 3×3

    0.0000   -0.0004    0.0348
    0.0004    0.0000   -0.0937
   -0.0426    0.0993    0.9892
```

Input Arguments

`matchedPoints1` — Coordinates of corresponding points

M-by-2 matrix | point feature object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The `matchedPoints1` input must contain points which do not lie on a single planar surface, (e.g., a wall, table, or book) and are putatively matched by using a function such as `matchFeatures`.

`matchedPoints2` — Coordinates of corresponding points

M-by-2 matrix | point feature object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The `matchedPoints1` input must contain points which do not lie on a single planar surface, (e.g., a wall, table, or book) and are putatively matched by using a function such as `matchFeatures`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `estimateFundamentalMatrix(matchedPoints1,matchedPoints2,Method="MSAC")` specifies MSAC as the method to compute the fundamental matrix.

`Method` — Method used to compute the fundamental matrix

`LMedS` (default) | `MSAC` | `Norm8Point`

Method used to compute the fundamental matrix, specified as one of the following:

`"LMedS"`	Least Median of Squares. Use this method if at least 50% of the points in `matchedPoints1` and `matchedPoints2` are inliers. The function stops the search for inliers and the estimation of the fundamental matrix after 50% of the inliers are found.
`"MSAC"`	M-estimator SAmple Consensus. Select the M-estimator SAmple Consensus method if you would like to set the distance threshold for the inliers. Generally, the MSAC method converges more quickly than the RANSAC method.
`"Norm8Point"`	Normalized eight-point algorithm [1]. To produce reliable results, the inputs, `matchedPoints1` and `matchedPoints2` must match precisely.

To produce reliable results using the `Norm8Point` algorithm, the inputs, `matchedPoints1` and `matchedPoints2`, must match precisely. The other methods can tolerate outliers and therefore only require putatively matched input points. You can obtain putatively matched points by using the `matchFeatures` function.

`OutputClass` — Fundamental matrix class
`"double"` (default) | `"single"`

Fundamental matrix class, specified as `"double"` or `"single"`. Use this property to specify the class for the fundamental matrix and for the function's internal computations.

`NumTrials` — Number of random trials for finding outliers
`500` (default) | integer

Number of random trials for finding the outliers, specified as an integer. This argument applies when you set the `Method` to `LMedS` or `MSAC`.

- `LMedS` — The function uses the actual number of trials as the parameter value.
- `MSAC` — The function uses the maximum number of trials as the parameter value.

The actual number of trials depends on `matchedPoints1`, `matchedPoints2`, and the value of the `Confidence` parameter. Select the number of random trials to optimize speed and accuracy.

`DistanceThreshold` — Distance threshold for finding outliers
`0.01` (default)

Distance threshold for finding outliers, specified as a positive scalar. This parameter applies when you set the `Method` to `MSAC`.

`Confidence` — Desired confidence for finding maximum number of inliers
`99` (default) | scalar

Desired confidence for finding maximum number of inliers, specified as a percentage scalar in the range `(0 100)`. This argument applies when you set `Method` to `MSAC`.

`ReportRuntimeError` — Report runtime error
`true` (default) | `false`

Report runtime error, specified as a logical value. Set this parameter to `true` to report run-time errors when the function cannot compute the fundamental matrix from `matchedPoints1` and `matchedPoints2`. When you set this parameter to `false`, you can check the `status` output to verify validity of the fundamental matrix.

Output Arguments

F — Fundamental matrix

3-by-3 matrix

Fundamental matrix, returned as a 3-by-3 matrix that is computed from the points in the inputs `matchedPoints1` and `matchedPoints2`.

$$[P_2\ 1]*FundamentalMatrix*[P_1\ 1]' = 0$$

P_1, the point in `matchedPoints1` of image 1 in pixels, corresponds to the point, P_2, the point in `matchedPoints2` in image 2.

In computer vision, the fundamental matrix is a 3-by-3 matrix which relates corresponding points in stereo images. When two cameras view a 3-D scene from two distinct positions, there are a number of geometric relations between the 3-D points and their projections onto the 2-D images that lead to constraints between the image points. Two images of the same scene are related by epipolar geometry.

inliersIndex — Inliers index

M-by-1 logical vector

Inliers index, returned as an *M*-by-1 logical index vector. An element set to `true` means that the corresponding indexed matched points in `matchedPoints1` and `matchedPoints2` were used to compute the fundamental matrix. An element set to `false` means the indexed points were not used for the computation.

Data Types: `logical`

status — Status code

`0` | `1` | `2`

Status code, returned as one of the following possible values:

status	Value
0:	No error.
1:	`matchedPoints1` and `matchedPoints2` do not contain enough points. The `Norm8Point` and `MSAC` methods require at least 8 points, and the `LMedS` method requires 16 points.
2:	Not enough inliers found.

Data Types: `int32`

Tips

Use `estimateEssentialMatrix` when you know the camera intrinsics. You can obtain the intrinsics using the Camera Calibrator app. Otherwise, you can use the `estimateFundamentalMatrix` function that does not require camera intrinsics. Note that the fundamental matrix cannot be estimated from coplanar world points.

Algorithms

Computing the Fundamental Matrix

When you choose the `Norm8Point` method, the function uses all points in `matchedPoints1` and `matchedPoints2` to compute the fundamental matrix. When you choose any other method, the function uses the following algorithm to exclude outliers and compute the fundamental matrix from inliers:

1. Initialize the fundamental matrix, F, to a 3-by-3 matrix of zeros.
2. Set the loop counter n, to zero, and the number of loops N, to the number of random trials specified.
3. Loop through the following steps while $n < N$:
   a. Randomly select 8 pairs of points from `matchedPoints1` and `matchedPoints2`.
   b. Use the selected 8 points to compute a fundamental matrix, f, by using the normalized 8-point algorithm.
   c. Compute the fitness of f for all points in `matchedPoints1` and `matchedPoints2`.
   d. If the fitness of f is better than F, replace F with f.

      For the `MSAC` method, update N.
   e. $n = n + 1$

Number of Random Samplings for the MSAC Method

The MSAC method updates the number of random trials N for every iteration in the algorithm loop. The function resets N, according to the following:

$N = \min(N, \frac{\log(1-p)}{\log(1-r^8)})$.

Where, p represents the confidence parameter you specified, and r is calculated as follows:

$\sum_{i}^{N} \text{sgn}(d(u_i, v_i), t)/N$, where $\text{sgn}(a, b) = 1$ if $a \le b$ and 0 otherwise.

When you use the `MSAC` method, results may not be identical between runs because of the randomized nature of the algorithm.

Sampson Distance

The function uses the Sampson distance to measure the distance of a pair of points according to a fundamental matrix. The equation below can be used to calculate the Sampson distance. In the equation, u represents `matchedPoints1` and v represents `matchedPoints2`.

$$d(u_i, v_i) = (v_i F u_i^T)^2 \left[\frac{1}{(Fu_i^T)_1^2 + (Fu_i^T)_2^2} + \frac{1}{(v_i F)_1^2 + (v_i F)_2^2} \right]$$

where i represents the index of the corresponding points, and $\left(Fu_i^T\right)_j^2$, the square of the j-th entry of the vector Fu_i^T.

Fitness of Fundamental Matrix for Corresponding Points

The following table summarizes how each method determines the fitness of the computed fundamental matrix:

Method	Measure of Fitness
LMedS	$median(d(u_i, v_i); i = 1:N)$, the number of input points. The smaller the value, the better the fitness.
MSAC	$\sum_{i}^{N} \min(d(u_i, v_i), t)$. The smaller the value, the better the fitness.

Version History

Introduced in R2012b

References

[1] Hartley, R., A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2003.

[2] Rousseeuw, P., A. Leroy, *Robust Regression and Outlier Detection*, John Wiley & Sons, 1987.

[3] Torr, P. H. S., and A. Zisserman, *MLESAC: A New Robust Estimator with Application to Estimating Image Geometry*, Computer Vision and Image Understanding, 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `'Method'`, `'OutputClass'`, `'DistanceType'`, and `'ReportRuntimeError'` must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`epipolarLine` | `extractFeatures` | `matchFeatures` | `estimateUncalibratedRectification` | `relativeCameraPose`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Uncalibrated Stereo Image Rectification”

“Point Feature Types”
“What Is Structure from Motion?”
“Coordinate Systems”

estimateEssentialMatrix

Estimate essential matrix from corresponding points in a pair of images

Syntax

```
E = estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics)
E = estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics1,
intrinsics2)

[E,inliersIndex] = estimateEssentialMatrix( ___ )
[E,inliersIndex,status] = estimateEssentialMatrix( ___ )

[ ___ ] = estimateEssentialMatrix( ___ ,Name=Value)
```

Description

`E = estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics)` returns the 3-by-3 essential matrix, `E`, using the M-estimator sample consensus (MSAC) algorithm. The input points can be *M*-by-2 matrices of *M* number of [*x*,*y*] coordinates, or a `KAZEPoints` , `SIFTPoints`,`SURFPoints`, `MSERRegions`, `BRISKPoints`, or `cornerPoints` object. The `instinsics` object contains the intrinsic parameters of the camera used to take the images.

`E = estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics1, intrinsics2)` returns the essential matrix relating two images taken by different cameras. `intrinsics1` and `intrinsics2` are `cameraIntrinsics` objects containing the intrinsic parameters of camera 1 and camera 2 respectively.

`[E,inliersIndex] = estimateEssentialMatrix( ___ )` additionally returns an *M*-by-1 logical vector, `inliersIndex`, used to compute the essential matrix. The function sets the elements of the vector to `true` when the corresponding point was used to compute the fundamental matrix. The elements are set to `false` if they are not used.

`[E,inliersIndex,status] = estimateEssentialMatrix( ___ )` additionally returns a status code to indicate the validity of points.

`[ ___ ] = estimateEssentialMatrix( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `(MaxNumTrials=500)` sets the maximum number of random trials for finding outliers to `500`.

Examples

Estimate Essential Matrix from a Pair of Images

Load a MAT file containing precomputed camera parameters into the workspace.

```
load upToScaleReconstructionCameraParameters.mat
```

Read and undistort two images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata","upToScaleReconstructionImages");
images = imageDatastore(imageDir);
I1 = undistortImage(readimage(images,1),cameraParams);
I2 = undistortImage(readimage(images,2),cameraParams);
I1gray = im2gray(I1);
I2gray = im2gray(I2);
```

Detect feature points each image.

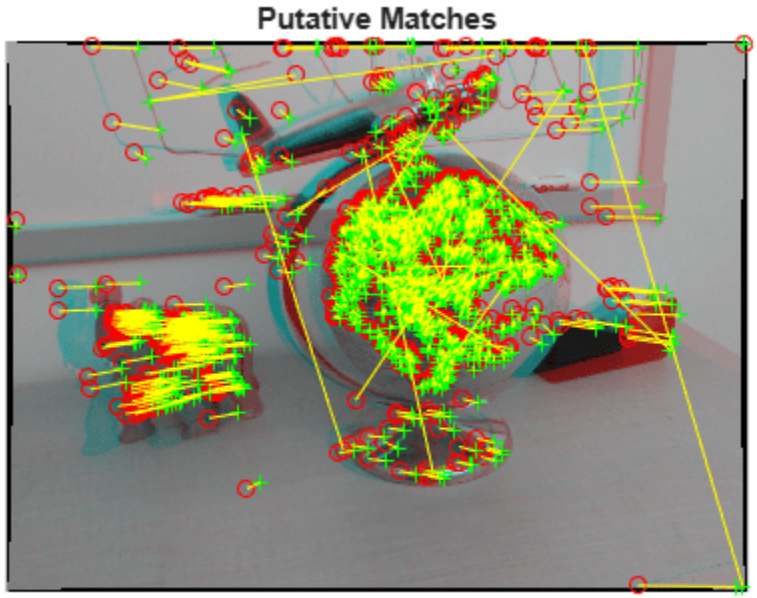
```
imagePoints1 = detectSIFTFeatures(I1gray);
imagePoints2 = detectSIFTFeatures(I2gray);
```

Extract feature descriptors from each image.

```
[features1, imagePoints1] = extractFeatures(I1gray,imagePoints1);
[features2, imagePoints2] = extractFeatures(I2gray,imagePoints2);
```

Match features across the images.

```
indexPairs = matchFeatures(features1,features2,MatchThreshold=10,Unique=true);
matchedPoints1 = imagePoints1(indexPairs(:,1));
matchedPoints2 = imagePoints2(indexPairs(:,2));
figure
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
title("Putative Matches")
```



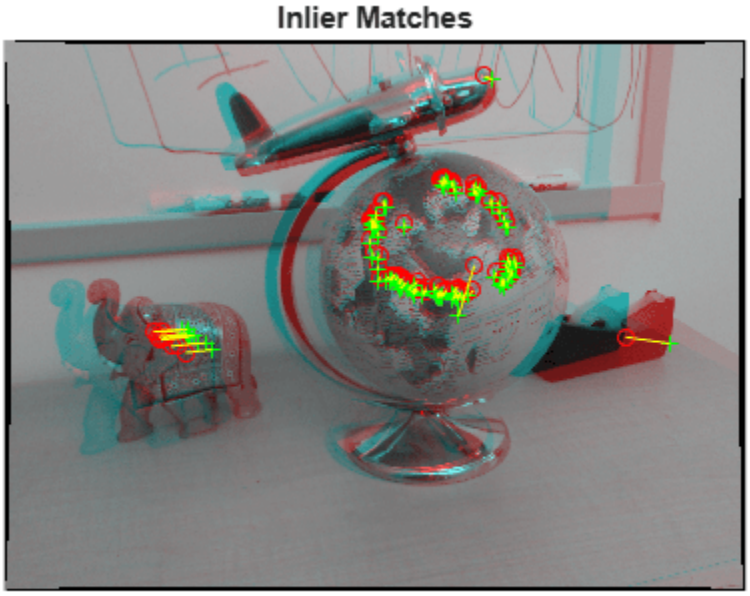
Estimate the essential matrix.

```
[E,inliers] = estimateEssentialMatrix(matchedPoints1,matchedPoints2,cameraParams);
```

Display the inlier matches.

```
inlierPoints1 = matchedPoints1(inliers);
inlierPoints2 = matchedPoints2(inliers);
```

```
figure
showMatchedFeatures(I1,I2,inlierPoints1,inlierPoints2);
title("Inlier Matches")
```



Input Arguments

`matchedPoints1` — Coordinates of corresponding points
M-by-2 matrix of [*x*,*y*] coordinates | `KAZEPoints` | `SIFTPoints` | `SURFPoints` | `BRISKPoints` | `MSERRegions` | `cornerPoints`

Coordinates of corresponding points in image 1, specified as an *M*-by-2 matrix of *M* of [*x*,*y*] coordinates, or as a `KAZEPoints`, `SIFTPoints`, `SURFPoints`, `BRISKPoints`, `MSERRegions`, or `cornerPoints` object. The `matchedPoints1` input must contain at least five points, which are putatively matched by using a function such as `matchFeatures`.

`matchedPoints2` — Coordinates of corresponding points
`SIFTPoints` | `SURFPoints` | `cornerPoints` | `MSERRegions` | `BRISKPoints` | *M*-by-2 matrix of [*x*,*y*] coordinates

Coordinates of corresponding points in image 1, specified as an *M*-by-2 matrix of *M* of [*x*,*y*] coordinates, or as a `KAZEPoints`, `SIFTPoints`, `SURFPoints`, `MSERRegions`, `BRISKPoints`,or `cornerPoints` object. The `matchedPoints2` input must contain at least five points, which are putatively matched by using a function such as `matchFeatures`.

`intrinsics` — Object for storing intrinsic parameters
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

`intrinsics1` — Intrinsic parameters for camera 1
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

`intrinsics2` — Intrinsic parameters for camera 2
`cameraIntrinsics` object

Camera intrinsic parameters, specified as a `cameraIntrinsics` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics,MaxNumTrials=500)` sets the maximum number of random trials for finding outliers to `500`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `estimateEssentialMatrix(matchedPoints1,matchedPoints2,intrinsics,"MaxNumTrials",500)` sets the maximum number of random trials for finding outliers to `500`.

`MaxNumTrials` — Maximum number of random trials for finding the outliers
`500` (default) | positive integer

Maximum number of random trials for finding outliers, specified as the comma-separated pair consisting of '`MaxNumTrials`' and a positive integer. The actual number of trials depends on `matchedPoints1`, `matchedPoints2`, and the value of the `Confidence` parameter. To optimize speed and accuracy, select the number of random trials.

`Confidence` — Desired confidence for finding maximum number of inliers
`99` (default) | scalar

Desired confidence for finding the maximum number of inliers, specified as the comma-separated pair consisting of '`Confidence`' and a percentage scalar value in the range (0,100). Increasing this value improves the robustness of the output but increases the amount of computations.

`MaxDistance` — Sampson distance threshold
`0.1` (default) | scalar

Sampson distance threshold, specified as the comma-separated pair consisting of '`MaxDistance`' and a scalar value. The function uses the threshold to find outliers returned in pixels squared. The Sampson distance is a first-order approximation of the squared geometric distance between a point and the epipolar line. Increase this value to make the algorithm converge faster, but this can also adversely affect the accuracy of the result.

Output Arguments

`E` — Essential matrix
3-by-3 matrix

Essential matrix, returned as a 3-by-3 matrix that is computed from the points in the `matchedPoints1` and `matchedPoints2` inputs with known camera intrinsics.

$$[P_2\ 1]*EssentialMatrix*[P_1\ 1]' = 0$$

The P_1 point in image 1, in normalized image coordinates, corresponds to the , P_2 point in image 2.

In computer vision, the essential matrix is a 3-by-3 matrix which relates corresponding points in stereo images which are in normalized image coordinates. When two cameras view a 3-D scene from two distinct positions, the geometric relations between the 3-D points and their projections onto the 2-D images lead to constraints between image points. The two images of the same scene are related by epipolar geometry.

Data Types: `double`

`inliersIndex` — Inliers index
M-by-1 logical vector

Inliers index, returned as an *M*-by-1 logical index vector. An element set to `true` indicates that the corresponding indexed matched points in `matchedPoints1` and `matchedPoints2` were used to compute the essential matrix. An element set to `false` means the indexed points were not used for the computation.

Data Types: `logical`

`status` — Status code
`0` | `1` | `2`

Status code, returned as one of the following possible values:

status	Value
0:	No error.
1:	`matchedPoints1` and `matchedPoints2` do not contain enough points. At least five points are required.
2:	Not enough inliers found. A least five inliers are required.

Data Types: `int32`

Tips

Use `estimateEssentialMatrix` when you know the camera intrinsics. You can obtain the intrinsics using the Camera Calibrator app. Otherwise, you can use the `estimateFundamentalMatrix` function that does not require camera intrinsics. The fundamental matrix cannot be estimated from coplanar world points.

Version History

Introduced in R2016b

R2019b: Intrinsic Parameters Object Replaces Camera Parameters Object
Behavior changed in R2019b

The intrinsic objects `intrinsics`, `intrinsics1`, and `intrinsics2`, which are specified as `cameraIntrinsics` objects that contain the intrinsic parameters of the camera used to take the images replaces the camera parameters objects `cameraParams`, `cameraParams1`, and `cameraParams2`.

References

[1] Kukelova, Z., M. Bujnak, and T. Pajdla *Polynomial Eigenvalue Solutions to the 5-pt and 6-pt Relative Pose Problems.* Leeds, UK: BMVC, 2008.

[2] Nister, D.. "An Efficient Solution to the Five-Point Relative Pose Problem." *IEEE Transactions on Pattern Analysis and Machine Intelligence*.Volume 26, Issue 6, June 2004.

[3] Torr, P. H. S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* Volume 78, Issue 1, April 2000, pp. 138-156.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator

Functions

`estimateFundamentalMatrix` | `relativeCameraPose` | `estimateCameraParameters` | `estimateWorldCameraPose`

Topics

"Structure from Motion from Two Views"
"Structure from Motion from Multiple Views"
"Point Feature Types"
"What Is Structure from Motion?"
"Coordinate Systems"

estimateStereoBaseline

Estimate baseline of stereo camera

Syntax

```
[stereoParams,pairsUsed,estimationErrors] = estimateStereoBaseline(
imagePoints,worldPoints,intrinsics1,intrinsics2)
[stereoParams,pairsUsed,estimationErrors] = estimateStereoBaseline(
imagePoints,worldPoints,intrinsics1,intrinsics2,'WorldUnits',units)
```

Description

`[stereoParams,pairsUsed,estimationErrors] = estimateStereoBaseline( imagePoints,worldPoints,intrinsics1,intrinsics2)` estimates the translation and orientation between two cameras of the same resolution.

The `estimateStereoBaseline` function is best used for a stereo system with a wide baseline when the cameras are so far apart that you cannot capture the complete calibration pattern in the field-of-view of each camera. This function estimates extrinsic parameters using fixed camera intrinsics that have been estimated independently for each camera. Calibrating each camera independently results in more precise estimates of the intrinsics.

`[stereoParams,pairsUsed,estimationErrors] = estimateStereoBaseline( imagePoints,worldPoints,intrinsics1,intrinsics2,'WorldUnits',units)` specifies the units in which `worldPoints` are measured.

Examples

Estimate Baseline of a Stereo System

Load camera intrinsic parameters.

```
ld = load('wideBaselineStereo');
```

Specify stereo calibration images.

```
leftImages = imageDatastore(fullfile(toolboxdir('vision'),'visiondata', ...
    'calibration','wideBaseline','left'));
rightImages = imageDatastore(fullfile(toolboxdir('vision'),'visiondata', ...
    'calibration','wideBaseline','right'));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = ...
    detectCheckerboardPoints(leftImages.Files,rightImages.Files);
```

Specify world coordinates of checkerboard keypoints.

```
squareSize = 29; % in millimeters
worldPoints = patternWorldPoints('checkerboard',patternDims,squareSize);
```

Estimate the baseline and extrinsic parameters.

```
params = estimateStereoBaseline(imagePoints,worldPoints, ...
    ld.intrinsics1,ld.intrinsics2)
```

```
params =
  stereoParameters with properties:

   Parameters of Two Cameras
        CameraParameters1: [1×1 cameraParameters]
        CameraParameters2: [1×1 cameraParameters]

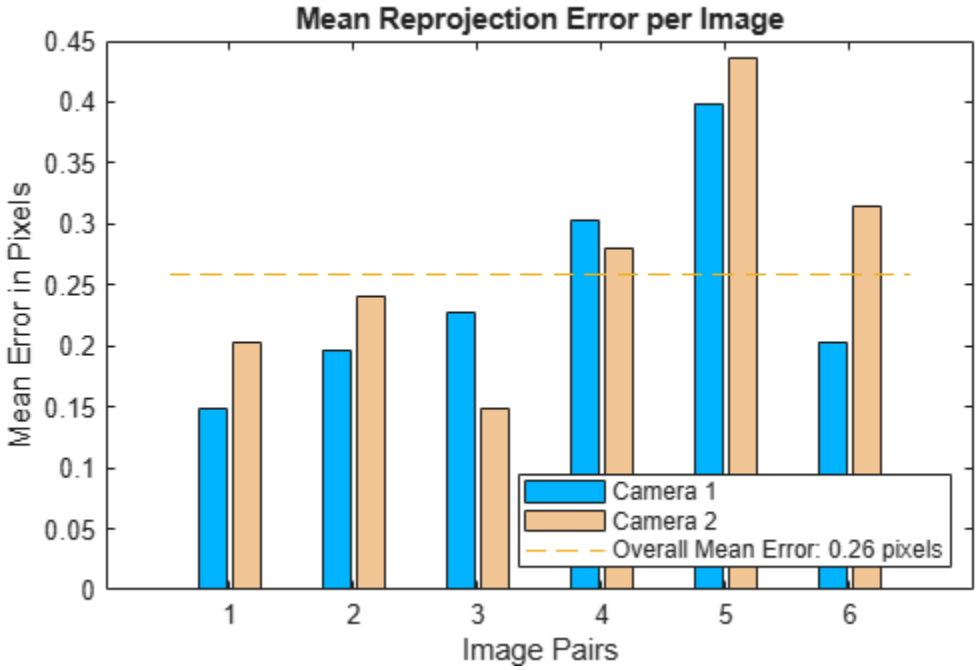
   Inter-camera Geometry
              PoseCamera2: [1×1 rigidtform3d]
        FundamentalMatrix: [3×3 double]
          EssentialMatrix: [3×3 double]

   Accuracy of Estimation
    MeanReprojectionError: 0.2586

   Calibration Settings
              NumPatterns: 6
              WorldPoints: [54×2 double]
               WorldUnits: 'mm'
```

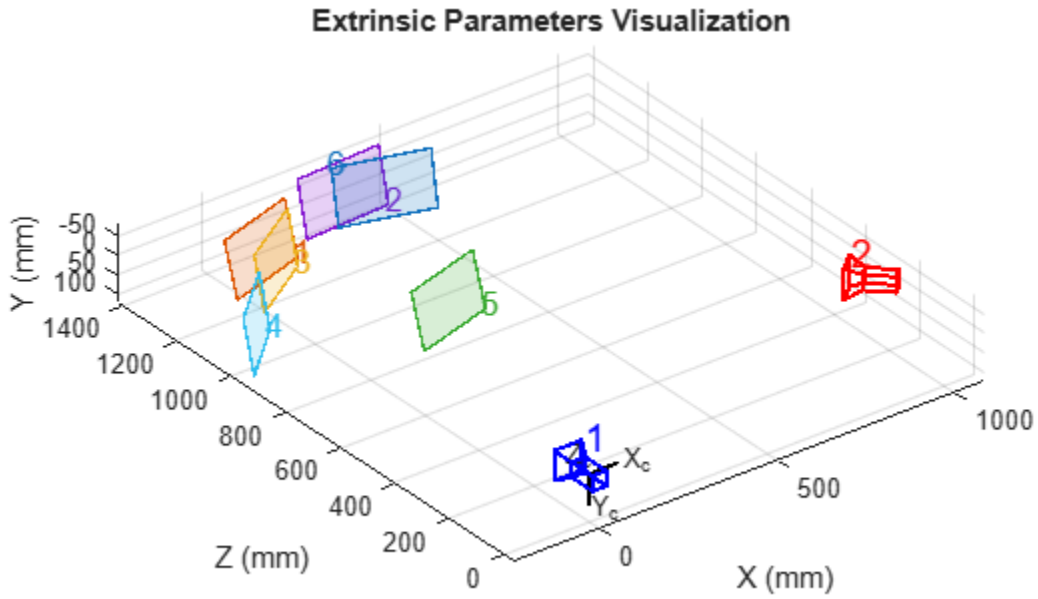
Visualize calibration accuracy.

```
figure
showReprojectionErrors(params)
```



Visualize camera extrinsics.

```
figure
showExtrinsics(params)
```



Input Arguments

`imagePoints` — Key points of calibration pattern

M-by-2-by-*numPairs*-by-2 array

Key points of calibration pattern, specified as an *M*-by-2-by-*numPairs*-by-2 array. The array contains the [*x*,*y*] intrinsic image coordinates of key points in the calibration pattern.

- *numPairs* is the number of stereo image pairs containing the calibration pattern. Must be greater than 2.
- The number of keypoint coordinates in each pattern, *M*, must be greater than 3.
- `imagePoints(:,:,:,1)` are the points from camera 1.
- `imagePoints(:,:,:,2)` are the points from camera 2.

Data Types: `single` | `double`

`worldPoints` — Key points of calibration pattern in world coordinates

M-by-2 array

Key points of calibration pattern in world coordinates, specified as an *M*-by-2 array of *M* number of [*x*,*y*] world coordinates. The pattern must be planar; therefore, *z*-coordinates are zero. Calibration images must not be coplanar. Use only one calibration image per spatial plane.

Data Types: `single` | `double`

`intrinsics1` — Intrinsic parameters of camera 1

`cameraIntrinsics` object

Intrinsic parameters of camera 1, specified as a `cameraIntrinsics` object. The intrinsics object stores information about a camera's intrinsic calibration parameters, including lens distortion.

`intrinsics2` — Intrinsic parameters of camera 2
`cameraIntrinsics` object

Intrinsic parameters of camera 2, specified as a `cameraIntrinsics` object. The intrinsics object stores information about a camera's intrinsic calibration parameters, including lens distortion.

`units` — World points units
`'mm'` (default) | string

World points units, specified as a string representing units.

Output Arguments

`stereoParams` — Camera parameters for stereo system
`stereoParameters` object

Camera parameters for stereo system, returned as a `stereoParameters` object. The object contains the intrinsic, extrinsic, and lens distortion parameters of the stereo camera system.

`pairsUsed` — Image pairs used to estimate camera parameters
numPairs-by-1 logical vector

Image pairs used to estimate camera parameters, returned as a *numPairs*-by-1 logical array. A logical `true` value in the vector indicates which that the image pairs was used to estimate the camera parameters. An image pair will not be used for estimation if the algorithm fails to estimate a homography between the world points and the points detected in that pair of images.

`estimationErrors` — Standard errors of estimated parameters
`stereoCalibrationErrors` object

Standard errors of estimated parameters, returned as a `stereoCalibrationErrors` object.

Version History

Introduced in R2018a

See Also

Apps
Stereo Camera Calibrator

Classes
`stereoParameters` | `cameraParameters` | `cameraIntrinsics` | `cameraCalibrationErrors` | `stereoCalibrationErrors`

Functions
`estimateCameraParameters` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `showExtrinsics` | `showReprojectionErrors` | `undistortImage`

estimateStereoRectification

Uncalibrated stereo rectification

Syntax

```
[tform1,tform2] = estimateStereoRectification(F,inlierPoints1,inlierPoints2,
imageSize)
```

Description

`[tform1,tform2] = estimateStereoRectification(F,inlierPoints1,inlierPoints2, imageSize)` returns projective transformations for rectifying stereo images. This function does not require intrinsic or extrinsic camera parameters.

Examples

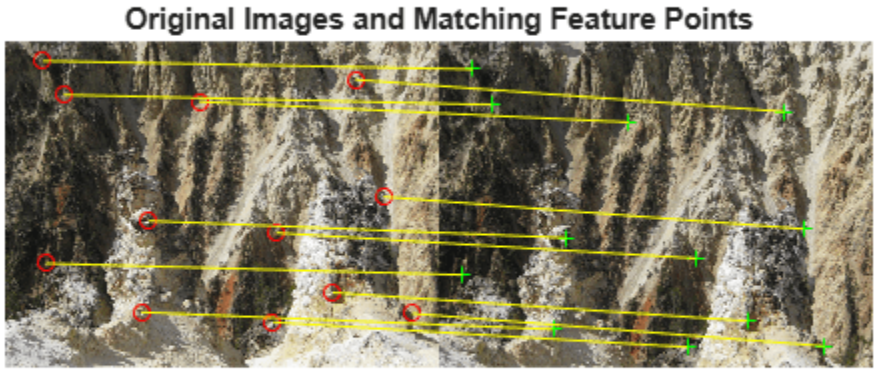
Estimate and Apply Stereo Rectification

Load the stereo images and feature points which are already matched.

```
I1 = imread("yellowstone_left.png");
I2 = imread("yellowstone_right.png");
load yellowstone_inlier_points;
```

Display point correspondences. Notice that the matching points are in different rows, indicating that the stereo pair is not rectified.

```
showMatchedFeatures(I1,I2,inlier_points1,inlier_points2,"montage");
title("Original Images and Matching Feature Points");
```



Calculate the fundamental matrix from the corresponding points.

```
f = estimateFundamentalMatrix(inlier_points1,inlier_points2, ...
    "Method","Norm8Point");
```

Calculate the rectification transformations.

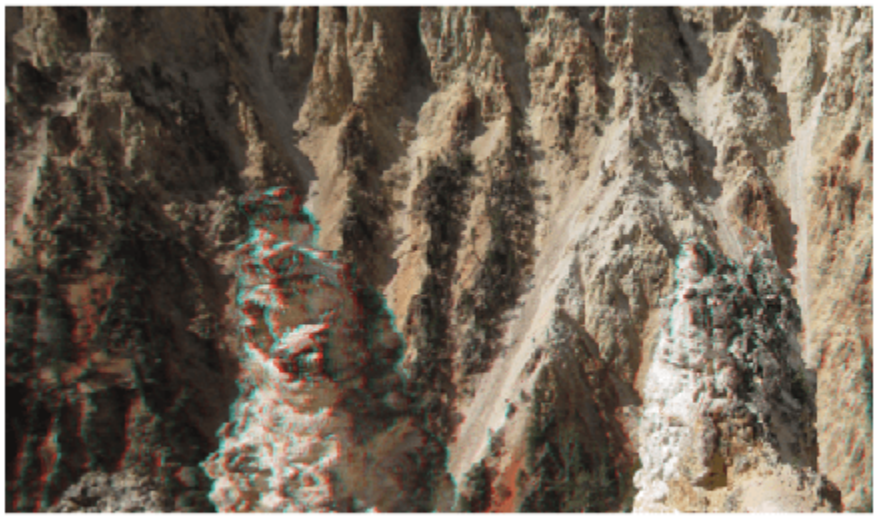
```
[tform1,tform2] = estimateStereoRectification(f,inlier_points1,...
    inlier_points2,size(I2));
```

Rectify the stereo images using projective transformations `tform1` and `tform2`.

```
[I1Rect,I2Rect] = rectifyStereoImages(I1,I2,tform1,tform2);
```

Display the stereo anaglyph, which can also be viewed with 3-D glasses.

```
figure
imshow(stereoAnaglyph(I1Rect,I2Rect))
```



Input Arguments

F — Fundamental matrix

3-by-3 matrix

Fundamental matrix for the stereo images, specified as a 3-by-3 matrix. The fundamental matrix satisfies this criteria for P_1, a point in image 1, and P_2, a corresponding point in image 2:

$[P_2,1] * F * [P_1,1]' = 0$

Data Types: `single` | `double`

`inlierPoints1` — Coordinates of points in image 1

M-by-2 matrix | point feature object

Coordinates of points in image 1, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`inlierPoints2` — Coordinates of corresponding points in image 2

M-by-2 matrix | point feature object

Coordinates of corresponding points in image 2, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`imageSize` — Size of image 2

numeric vector

Size of image 2, specified as a numeric vector in the format returned by the `size` function.

Output Arguments

`tform1` — Projective transformation 1

`projtform2d` object

Projective transformation 1 describing the projective transformations for input image 1, returned as a `projtform2d` object.

`tform2` — Projective transformation 2

`projtform2d` object

Projective transformation 2 describing the projective transformations for input image 2, returned as a `projtform2d` object.

Tips

- Applying the output uncalibrated rectification of `tform1` (or `tform2`) to image 1 (or image 2) can result in an undesired distortion if an epipole exists within the image. You can check for an epipole within an image by applying the `isEpipoleInImage` function.

Version History

Introduced in R2022b

R2022b: Recommended over `estimateUncalibratedRectification`

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateUncalibratedRectification` function uses the postmultiply convention. Although there are no plans to remove `estimateUncalibratedRectification` at this time, you can streamline your geometric transformation workflows by switching to the `estimateStereoRectification` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Pollefeys, M., Koch, R., and Van Gool, L.. *A Simple and Efficient Rectification Method for General Motion*. Proceedings of the Seventh IEEE International Conference on Computer Vision. Volume 1, pages 496-501. 1999. DOI:10.1109/ICCV.1999.791262.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps
Stereo Camera Calibrator | Camera Calibrator

Functions
`estimateFundamentalMatrix` | `rectifyStereoImages` | `extractFeatures` | `matchFeatures`

Topics
“Uncalibrated Stereo Image Rectification”
“Point Feature Types”
“Coordinate Systems”

estimateWorldCameraPose

(Not recommended) Estimate camera pose from 3-D to 2-D point correspondences

Note `estimateWorldCameraPose` is not recommended. Use the `estworldpose` function instead. For more information, see “Version History”.

Syntax

```
[worldOrientation,worldLocation] = estimateWorldCameraPose(imagePoints,
worldPoints,cameraParams)
[___,inlierIdx] = estimateWorldCameraPose(imagePoints,worldPoints,
cameraParams)
[___,status] = estimateWorldCameraPose(imagePoints,worldPoints,cameraParams)
[___] = estimateWorldCameraPose(___,Name,Value)
```

Description

`[worldOrientation,worldLocation] = estimateWorldCameraPose(imagePoints, worldPoints,cameraParams)` returns the orientation and location of a calibrated camera in a world coordinate system. The input `worldPoints` must be defined in the world coordinate system.

This function solves the perspective-*n*-point (PnP) problem using the perspective-three-point (P3P) algorithm [1]. The function eliminates spurious outlier correspondences using the M-estimator sample consensus (MSAC) algorithm. The inliers are the correspondences between image points and world points that are used to compute the camera pose.

`[___,inlierIdx] = estimateWorldCameraPose(imagePoints,worldPoints, cameraParams)` returns the indices of the inliers used to compute the camera pose, in addition to the arguments from the previous syntax.

`[___,status] = estimateWorldCameraPose(imagePoints,worldPoints,cameraParams)` additionally returns a status code to indicate whether there were enough points.

`[___] = estimateWorldCameraPose(___,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

Examples

Determine Camera Pose from World-to-Image Correspondences

Load previously calculated world-to-image correspondences.

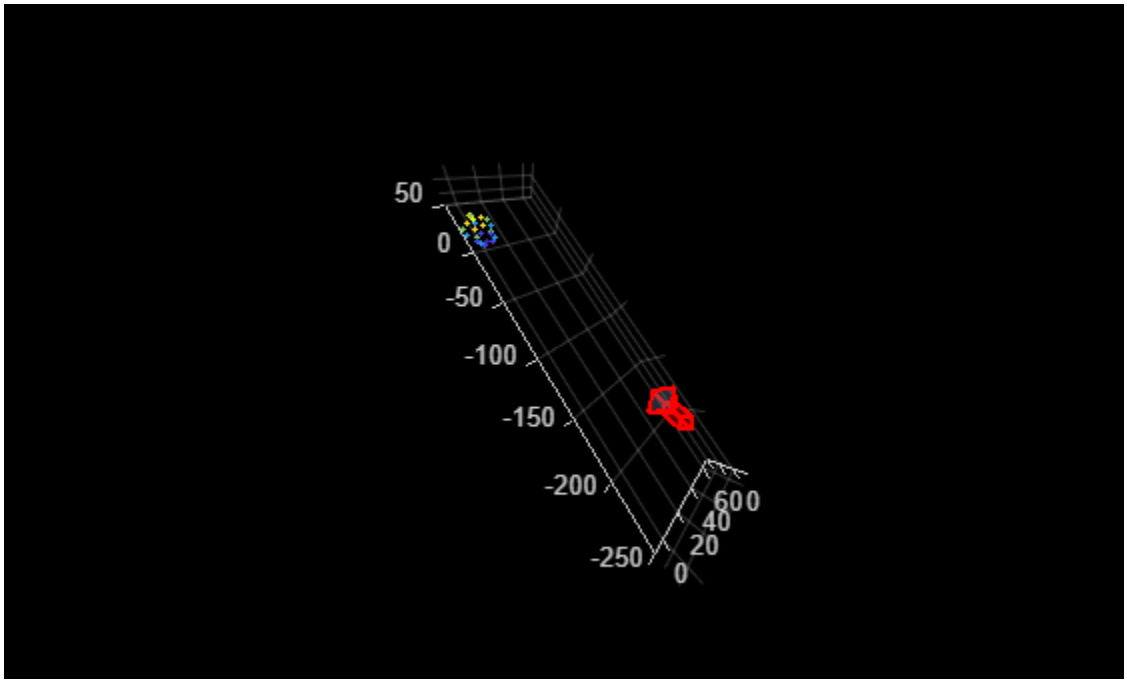
```
data = load('worldToImageCorrespondences.mat');
```

Estimate the world camera pose.

```
[worldOrientation,worldLocation] = estimateWorldCameraPose(...
     data.imagePoints,data.worldPoints,data.cameraParams);
```

Plot the world points.

```
pcshow(data.worldPoints,'VerticalAxis','Y','VerticalAxisDir','down', ...
    'MarkerSize',30);
hold on
plotCamera('Size',10,'Orientation',worldOrientation,'Location',...
    worldLocation);
hold off
```



Input Arguments

`imagePoints` — Coordinates of undistorted image points

M-by-2 array

Coordinates of undistorted image points, specified as an *M*-by-2 array of [*x*,*y*] coordinates. The number of image points, *M*, must be at least four.

The function does not account for lens distortion. You can either undistort the images using the `undistortImage` function before detecting the image points, or you can undistort the image points themselves using the `undistortPoints` function.

Data Types: `single` | `double`

`worldPoints` — Coordinates of world points

M-by-3 array

Coordinates of world points, specified as an *M*-by-3 array of [*x*,*y*,*z*] coordinates.

Data Types: `single` | `double`

`cameraParams` — Object for storing camera parameters

`cameraParameters` object | `cameraIntrinsics` object

Camera parameters, specified as a `cameraParameters` or `cameraIntrinsics` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MaxNumTrials',1000`

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | positive integer scalar

Maximum number of random trials, specified as a positive integer scalar. The actual number of trials depends on the number of image and world points, and the values for the `MaxReprojectionError` and `Confidence` properties. Increasing the number of trials improves the robustness of the output at the expense of additional computation.

`Confidence` — Confidence for finding maximum number of inliers
`99` (default) | scalar in the range (0,100)

Confidence for finding maximum number of inliers, specified as a scalar in the range (0,100). Increasing this value improves the robustness of the output at the expense of additional computation.

`MaxReprojectionError` — Reprojection error threshold
`1` (default) | positive numeric scalar

Reprojection error threshold for finding outliers, specified as a positive numeric scalar in pixels. Increasing this value makes the algorithm converge faster, but can reduce the accuracy of the result. Correspondences with a reprojection error larger than the `MaxReprojectionError` are considered outliers, and are not used to compute the camera pose.

Output Arguments

`worldOrientation` — Orientation of camera in world coordinates
3-by-3 matrix

Orientation of camera in world coordinates, returned as a 3-by-3 matrix.

Data Types: `double`

`worldLocation` — Location of camera
1-by-3 vector

Location of camera, returned as a 1-by-3 unit vector.

Data Types: `double`

`inlierIdx` — Indices of inlier points
M-by-1 logical vector

Indices of inlier points, returned as an *M*-by-1 logical vector. A logical true value in the vector corresponds to inliers represented in `imagePoints` and `worldPoints`.

`status` — Status code
integer value

Status code, returned as `0`, `1`, or `2`.

Status code	Status
`0`	No error
`1`	`imagePoints` and `worldPoints` do not contain enough points. A minimum of four points are required.
`2`	Not enough inliers found. A minimum of 4 inliers are required.

Version History

Introduced in R2016b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateWorldCameraPose` function uses the postmultiply convention. Although there are no plans to remove `estimateWorldCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `estworldpose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `estimateWorldCameraPose` to `estworldpose`.
- Specify the `cameraParams` argument as a `cameraIntrinsics` object. If you have a `cameraParameters` object, then you can get a `cameraIntrinsics` object by querying the `Intrinsics` property. If the `Intrinsics` property is empty according to the `isempty` function, then set the `ImageSize` property of the `cameraParameters` object to an arbitrary vector before querying the `Intrinsics` property. For example:

  ```
  load worldToImageCorrespondences.mat
  if(isempty(cameraParams.Intrinsics))
      cameraParams.ImageSize = [128 128];
  end
  intrinsics = cameraParams.Intrinsics;
  ```
- Replace the two output arguments `worldOrientation` and `worldLocation` with a single output argument, `worldPose`. If you need to obtain the orientation matrix and location vector, then you can query the `R` and the `Translation` properties of the `rigidtform3d` object returned by the `worldPose` argument. Note that the value of `R` is the transpose of `worldOrientation`.

The table shows examples of how to update your code.

Discouraged Usage	Recommended Replacement

This example estimates a camera pose using the `estimateWorldCameraPose` function with the `cameraParams` argument specified as a `cameraParameters` object.
 	
`[worldOrientation,worldLocation] = estimateWorldCameraPose( ...`
`  imagePoints,worldPoints,cameraParams);`
	This example gets the camera intrinsics using the `Intrinsics` property of the `cameraParameters` object, then estimates a camera pose using the `estworldpose` function.
	
	`intrinsics = cameraParams.Intrinsics;`
	`worldPose = estworldpose( ...`
	`  imagePoints,worldPoints,intrinsics);`
	
	If you need to obtain the camera orientation and location, then you can query properties of `worldPose`.
	
	`worldOrientation = worldPose.R;`
	`worldLocation = worldPose.Translation;`
	
	If you want the orientation in the postmultiply convention, take the transpose of `worldPose.R`.
	
	`worldOrientation = worldPose.R';`

References

[1] Gao, X.-S., X.-R. Hou, J. Tang, and H.F. Cheng. "Complete Solution Classification for the Perspective-Three-Point Problem." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Volume 25,Issue 8, pp. 930–943, August 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding*. 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`estworldpose` | `estrelpose` | `triangulateMultiview` | `bundleAdjustment` | `plotCamera` | `pcshow` | `estimateExtrinsics` | `extr2pose` | `pose2extr` | `img2world2d` | `world2img`

Topics
“What Is Structure from Motion?”

estrelpose

Calculate relative rotation and translation between camera poses

Syntax

```
relativePose = estrelpose(M,intrinsics,inlierPoints1,inlierPoints2)
relativePose = estrelpose(M,intrinsics1,intrinsics2,inlierPoints1,
inlierPoints2)
[relativePose,validPointsFraction] = estrelpose( ___ )
```

Description

`relativePose = estrelpose(M,intrinsics,inlierPoints1,inlierPoints2)` returns the pose of a calibrated camera relative to its previous pose. The two poses are related by the fundamental, essential, or homography matrix `M`. The function calculates the camera location up to scale.

`relativePose = estrelpose(M,intrinsics1,intrinsics2,inlierPoints1, inlierPoints2)` returns the pose of the second camera relative to the first one.

`[relativePose,validPointsFraction] = estrelpose( ___ )` additionally returns the fraction of the inlier points that project in front of both cameras.

Examples

Estimate Camera Pose Relative to Prior Pose

Load a previously calculated fundamental matrix, camera parameters, and image points for a single camera.

```
load("relCameraPoseData.mat")
intrinsics = cameraParams.Intrinsics;
```

Calculate the camera pose relative to its previous pose.

```
relPose = estrelpose(M,intrinsics,inlierPoints1,inlierPoints2)
```

```
relPose =
  rigidtform3d with properties:

    Dimensionality: 3
       Translation: [0.9952 -0.0887 -0.0412]
                 R: [3×3 double]

                 A: [ 0.9911    0.0203   -0.1314    0.9952
                     -0.0196    0.9998    0.0065   -0.0887
                      0.1315   -0.0039    0.9913   -0.0412
                           0         0         0    1.0000]
```

Input Arguments

`M` — Fundamental, essential, or homography matrix

3-by-3 matrix | `affinetform2d` object | `projtform2d` object | `simtform2d` object

Fundamental, essential, or homography matrix, specified as a 3-by-3 matrix, an `affinetform2d` object, `projtform2d` object, or a `simtform2d` object containing a homography matrix. You can obtain the 3-by-3 matrix using one of these functions:

- `estimateFundamentalMatrix` for the fundamental matrix.
- `estimateEssentialMatrix` for the essential matrix
- `estgeotform2d` for the projective, affine, or similarity 2-D object.

Data Types: `single` | `double`

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object

Camera intrinsics, specified as a `cameraIntrinsics` object.

`intrinsics1` — Camera intrinsics for camera 1

`cameraParameters` object | `cameraIntrinsics` object

Camera intrinsics for camera 1, specified as a `cameraIntrinsics` object.

`intrinsics2` — Camera parameters for camera 2

`cameraParameters` object | `cameraIntrinsics` object

Camera intrinsics for camera 2, specified as a `cameraIntrinsics` object.

`inlierPoints1` — Coordinates of corresponding points in view 1

M-by-2 matrix | point feature object

Coordinates of corresponding points in view 1, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. You can obtain these points using the `estimateFundamentalMatrix` function or the `estimateEssentialMatrix`.

`inlierPoints2` — Coordinates of corresponding points in view 2

M-by-2 matrix | point feature object

Coordinates of corresponding points in view 2, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. You can obtain these points using the `estimateFundamentalMatrix` function or the `estimateEssentialMatrix`.

Output Arguments

`relativePose` — Relative camera pose

`rigidtform3d` object

Relative camera pose in world coordinates, returned as a `rigidtform3d` object. The `"R"` and the `"Translation"` properties of the object represent the orientation and location of the camera. If you

use only one camera, the properties describe the orientation and location of the second camera pose relative to the first camera pose. If you use two cameras, the matrix describes the orientation and location of camera 2 relative to camera 1.

`validPointsFraction` — Fraction of valid inlier points
scalar

Fraction of valid inlier points that project in front of both cameras, returned as a scalar. If `validPointsFraction` is too small, typically less than 0.9, it can indicate that the fundamental matrix is incorrect.

Tips

- You can calculate the camera extrinsics according to:

```
relativeOrientation = relativePose.R;
relativeLocation = relativePose.Translation;
camPose = rigidtform3d(relativeOrientation,relativeLocation);
extrinsics = pose2extr(camPose)
```

- The `estrelpose` function uses the `inlierPoints1` and `inlierPoints2` arguments to determine which of the multiple possible solutions is physically realizable. If the input `M` is a `projtform2d` object, there could be up to two solutions that are equally realizable.

Version History

Introduced in R2022b

R2022b: Recommended over `relativeCameraPose`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `relativeCameraPose` function uses the postmultiply convention. Although there are no plans to remove `relativeCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `estrelpose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding*. 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

[3] Faugeras, O., & Lustman, F. (1988). Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence* 2(3), 485–508.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator

Functions

`estworldpose` | `estimateCameraParameters` | `estimateEssentialMatrix` | `estimateFundamentalMatrix` | `cameraProjection` | `plotCamera` | `triangulate` | `triangulateMultiview` | `pose2extr`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Point Feature Types”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”
“What Is Structure from Motion?”

estworldpose

Estimate camera pose from 3-D to 2-D point correspondences

Syntax

```
worldPose = estworldpose(imagePoints,worldPoints,intrinsics)
[worldPose,inlierIdx] = estworldpose(imagePoints,worldPoints,intrinsics)
[worldPose,inlierIdx,status] = estworldpose(imagePoints,worldPoints,
intrinsics)
[___] = estworldpose(___,Name=Value)
```

Description

`worldPose = estworldpose(imagePoints,worldPoints,intrinsics)` returns the pose of a calibrated camera in a world coordinate system. The input `worldPoints` must be defined in the world coordinate system.

This function solves the perspective-*n*-point (PnP) problem using the perspective-three-point (P3P) algorithm [1]. The function eliminates spurious outlier correspondences using the M-estimator sample consensus (MSAC) algorithm. The inliers are the correspondences between image points and world points that are used to compute the camera pose.

`[worldPose,inlierIdx] = estworldpose(imagePoints,worldPoints,intrinsics)` returns the indices of the inliers used to compute the camera pose, in addition to the arguments from the previous syntax.

`[worldPose,inlierIdx,status] = estworldpose(imagePoints,worldPoints, intrinsics)` additionally returns a status code to indicate whether there were enough points.

`[___] = estworldpose(___,Name=Value)` uses additional options specified by one or more name-value arguments, using any of the preceding syntaxes.

Examples

Determine Camera Pose from World-to-Image Correspondences

Load previously calculated world-to-image correspondences.

```
load("worldToImageCorrespondences.mat");
cameraParams.ImageSize = [128 128];
intrinsics = cameraParams.Intrinsics;
```

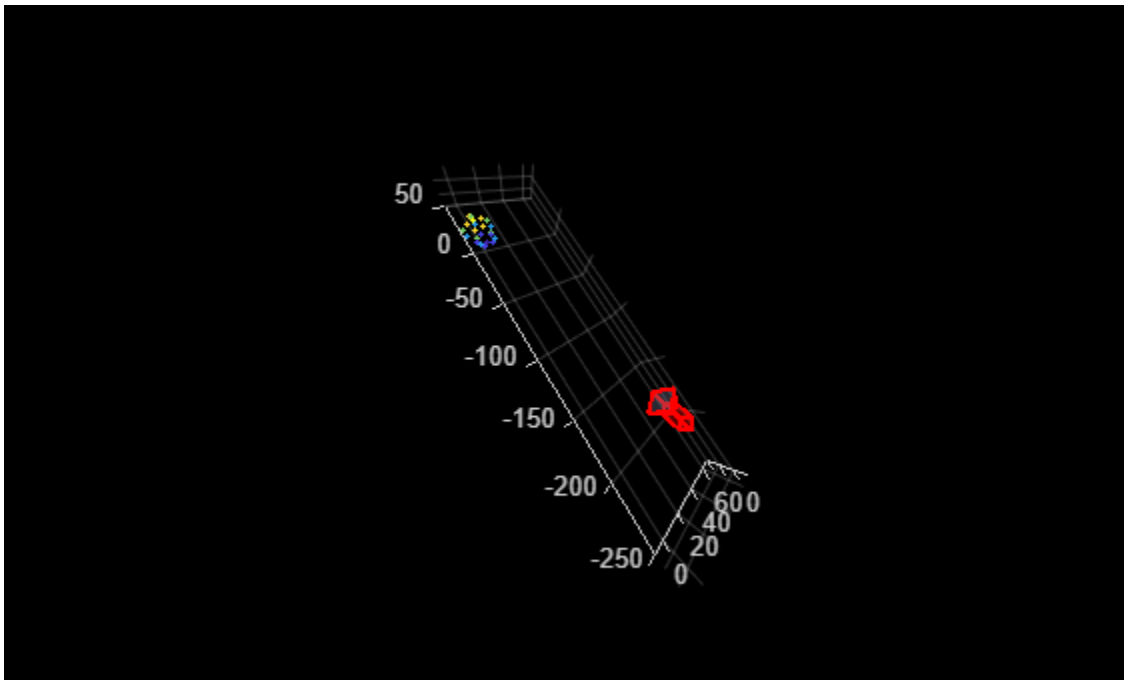
Estimate the world camera pose.

```
worldPose = estworldpose(imagePoints,worldPoints,intrinsics);
```

Plot the world points.

```
pcshow(worldPoints,VerticalAxis="Y",VerticalAxisDir="down", ...
    MarkerSize=30);
```

```
hold on
plotCamera(Size=10,Orientation=worldPose.R', ...
    Location=worldPose.Translation);
hold off
```



Input Arguments

`imagePoints` — Coordinates of undistorted image points

M-by-2 array

Coordinates of undistorted image points, specified as an *M*-by-2 array of [*x*, *y*] coordinates. The number of image points, *M*, must be at least four.

The function does not account for lens distortion. You can either undistort the images using the `undistortImage` function before detecting the image points, or you can undistort the image points themselves using the `undistortPoints` function.

Data Types: `single` | `double`

`worldPoints` — Coordinates of world points

M-by-3 array

Coordinates of world points, specified as an *M*-by-3 array of [*x*, *y*, *z*] coordinates.

Data Types: `single` | `double`

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object

Camera intrinsics, specified as a `cameraIntrinsics` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxNumTrials=1000`

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | positive integer scalar

Maximum number of random trials, specified as a positive integer scalar. The actual number of trials depends on the number of image and world points, and the values for the `MaxReprojectionError` and `Confidence` properties. Increasing the number of trials improves the robustness of the output at the expense of additional computation.

`Confidence` — Confidence for finding maximum number of inliers
`99` (default) | scalar in the range (0, 100)

Confidence for finding maximum number of inliers, specified as a scalar in the range (0, 100). Increasing this value improves the robustness of the output at the expense of additional computation.

`MaxReprojectionError` — Reprojection error threshold
`1` (default) | positive numeric scalar

Reprojection error threshold for finding outliers, specified as a positive numeric scalar in pixels. Increasing this value makes the algorithm converge faster, but can reduce the accuracy of the result. Correspondences with a reprojection error larger than the `MaxReprojectionError` are considered outliers, and are not used to compute the camera pose.

Output Arguments

`worldPose` — Camera pose
`rigidtform3d` object

Camera pose in world coordinates, returned as a `rigidtform3d` object. The `R` and the `Translation` properties of the object represent the orientation and location of the camera.

`inlierIdx` — Indices of inlier points
M-by-1 logical vector

Indices of inlier points, returned as an *M*-by-1 logical vector. A logical true value in the vector corresponds to inliers represented in `imagePoints` and `worldPoints`.

`status` — Status code
integer value

Status code, returned as `0`, `1`, or `2`.

Status code	Status
`0`	No error
`1`	`imagePoints` and `worldPoints` do not contain enough points. A minimum of four points are required.

Status code	Status
2	Not enough inliers found. A minimum of 4 inliers are required.

Tips

- This function does not account for lens distortion. You can undistort the images using the `undistortImage` function before detecting the image points. You can undistort the image points themselves using the `undistortPoints` function.

Version History

Introduced in R2022b

R2024a: Support for MATLAB Function block

Support added to use this function in a MATLAB Function block

R2022b: Recommended over `estimateWorldCameraPose`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateWorldCameraPose` function uses the postmultiply convention. Although there are no plans to remove `estimateWorldCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `estworldpose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Gao, X.-S., X.-R. Hou, J. Tang, and H.F. Cheng. "Complete Solution Classification for the Perspective-Three-Point Problem." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Volume 25,Issue 8, pp. 930–943, August 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`estrelpose` | `triangulateMultiview` | `triangulate` | `bundleAdjustment` | `plotCamera` | `pcshow` | `estimateExtrinsics` | `extr2pose` | `pose2extr` | `viewSet`

Topics
“What Is Structure from Motion?”

cameraPoseToExtrinsics

(Not recommended) Convert camera pose to extrinsics

Note `cameraPoseToExtrinsics` is not recommended. Use the `pose2extr` function instead. For more information, see “Version History”.

Syntax

```
tform = cameraPoseToExtrinsics(cameraPose)
[rotationMatrix,translationVector] = cameraPoseToExtrinsics(orientation,
location)
```

Description

`tform = cameraPoseToExtrinsics(cameraPose)` returns a `rigid3d` object that contains the transformation from world coordinates to camera coordinates. `cameraPose` is the orientation and location of the camera in world coordinates, specified as a `rigid3d` object.

`[rotationMatrix,translationVector] = cameraPoseToExtrinsics(orientation, location)` returns the camera extrinsics, `rotationMatrix` and `translationVector`, which represent the coordinate system transformation from world coordinates to camera coordinates. The inputs, `orientation` and `location`, represent the 3-D camera pose in the world coordinates.

Examples

Convert World Coordinates to Camera Coordinates

Create the 3-D orientation matrix and the location vector.

```
orientation = eye(3);
location = [0 0 10];
```

Compute the rotation matrix and translation vector to transform points from world coordinates to camera coordinates.

```
[rotationMatrix,translationVector] = cameraPoseToExtrinsics(orientation,location)
```

```
rotationMatrix = 3×3

     1     0     0
     0     1     0
     0     0     1


translationVector = 1×3

     0     0   -10
```

Compute the transformation from world coordinates to camera coordinates as a rigid 3-D object.

```
cameraPose = rigid3d(orientation,location);
tform = cameraPoseToExtrinsics(cameraPose)
```

```
tform =
  rigid3d with properties:

       Rotation: [3×3 double]
    Translation: [0 0 -10]
```

Input Arguments

`cameraPose` — Orientation and location of the camera

`rigid3d` object

Orientation and location of the camera in world coordinates, specified as a `rigid3d` object.

`orientation` — 3-D orientation

3-by-3 matrix

3-D orientation of the camera in world coordinates, specified as a 3-by-3 matrix. The `orientation` and `location` inputs must be the same data type.

Data Types: `double` | `single`

`location` — 3-D location

three-element vector

3-D location of the camera in world coordinates, specified as a three-element vector. The `orientation` and `location` inputs must be the same data type.

Data Types: `double` | `single`

Output Arguments

`tform` — Transformation

`rigid3d`

Transformation from world coordinates to camera coordinates, returned as a `rigid3d` object. The transformation allows you to transform points from the world coordinate system to the camera coordinate system. `tform` is computed as:

```
tform.Rotation = cameraPose.Rotation'
tform.Translation = -cameraPose.Translation * cameraPose.Rotation'
```

`rotationMatrix` — 3-D rotation

3-by-3 matrix

3-D rotation, returned as a 3-by-3 matrix. The rotation matrix, together with the translation vector allows you to transform points from the world coordinate system to the camera coordinate system.



The relationship between the rotation matrix and the input orientation matrix is:
`rotationMatrix = orientation'`

`translationVector` — 3-D translation
1-by-3 vector

3-D translation, returned as a 1-by-3 vector. The translation vector together with the rotation matrix, enables you to transform points from the world coordinate system to the camera coordinate system.



The relationship between the translation vector and the input orientation matrix is :
`translationVector = -location*orientation'`

Version History

Introduced in R2016b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `cameraPoseToExtrinsics` function uses the postmultiply convention. Although there are no plans to remove `cameraPoseToExtrinsics` at this time, you can streamline your geometric transformation workflows by switching to the `pose2extr` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `cameraPoseToExtrinsics` to `pose2extr`.
- Specify the camera pose as a `rigidtform3d` object using the `cameraPose` argument. `pose2extr` does not support specifying the camera pose using the `orientation` and `translation` input arguments. Note that you create the `rigidtform3d` object using the transpose of `orientation` or the transpose of the transformation matrix in the `T` property of `tform`.

Discouraged Usage	Recommended Replacement

This example specifies the camera pose as a rotation matrix `orientation` in the premultiply convention and a translation vector `location`, then converts the camera pose to extrinsics using the `cameraPoseToExtrinsics` function, using the transpose of `orientation`.
 	
`orientation = eye(3);`
`location = [0 0 10];`
`[rotationMatrixOld,translationVector] = cameraPoseToExtrinsics( ...`
`    orientation',location)`
	This example specifies the camera pose as a `rigidtform3d` object, then converts the camera pose to extrinsics using the `pose2extr` function.
	
	`orientation = eye(3);`
	`location = [0 0 10];`
	`cameraPose = rigidtform3d(orientation,location);`
	`extrinsics = pose2extr(cameraPose);`
	
	If you need to obtain a rotation matrix and translation vector, then you can query properties of `extrinsics`.
	
	`rotationMatrix = extrinsics.R;`
	`translationVector = extrinsics.Translation;`
	
	If you want the rotation matrix in the postmultiply convention, take the transpose of `extrinsics.R`.
	
	`rotationMatrixOld = extrinsics.R';`

This example specifies a camera pose as a `rigid3d` object using the transpose of the geometric transformation matrix `A` in the premultiply convention, then converts the camera pose to extrinsics using the `cameraPoseToExtrinsics` function.
 	
`A = [1 0 0 0; 0 1 0 0; 0 0 1 10; 0 0 0 1];`
`cameraPoseOld = rigid3d(A');`
`extrinsicsOld = cameraPoseToExtrinsics(cameraPoseOld);`
	This example specifies the camera pose as a `rigidtform3d` object using the geometric transformation matrix `A`, then converts the camera pose to extrinsics using the `pose2extr` function.
	
	`A = [1 0 0 0; 0 1 0 0; 0 0 1 10; 0 0 0 1];`
	`cameraPose = rigidtform3d(A);`
	`extrinsics = pose2extr(cameraPose);`
	If instead you start with an existing `rigid3d` object `cameraPoseOld`, then you can get the geometric transformation matrix in the postmultiply convention by querying the property `T` of `cameraPoseOld`.
	
	`T = cameraPoseOld.T;`
	`cameraPose = rigidtform3d(T');`
	`extrinsics = pose2extr(cameraPose);`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`pose2extr` | `extr2pose` | `estimateExtrinsics` | `estrelpose` | `estworldpose`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

extr2pose

Convert extrinsics to camera pose

Syntax

```
cameraPose = extr2pose(camExtrinsics)
```

Description

`cameraPose = extr2pose(camExtrinsics)` returns the camera pose, `cameraPose`, in world coordinates, given the camera extrinsics.

Examples

Convert Camera Coordinates to World Coordinates

Create a 3-D rigid geometric transformation from a rotation matrix and a translation vector.

```
rotationMatrix = eye(3);
translationVector = [0 0 -10];
tform = rigidtform3d(rotationMatrix,translationVector);
```

Compute the camera pose in world coordinates.

```
cameraPose = extr2pose(tform)
```

```
cameraPose =
  rigidtform3d with properties:

    Dimensionality: 3
       Translation: [0 0 10]
                 R: [3×3 double]

                 A: [1     0     0     0
                     0     1     0     0
                     0     0     1    10
                     0     0     0     1]
```

Input Arguments

`camExtrinsics` — Camera extrinsics

`rigidtform3d` object | `se3` object

Camera extrinsics, specified as a `rigidtform3d` or `se3` object. The extrinsics are a transformation from world coordinates to camera coordinates that enables you to transform points from the world coordinate system to the camera coordinate system.

Output Arguments

cameraPose — Camera pose
`rigidtform3d` object | `se3` object

Camera pose in world coordinates, returned as a `rigidtform3d` or `se3` object. `cameraPose` is the inverse of `camExtrinsics` as calculated using the `invert` function.

Version History

Introduced in R2022b

R2022b: Recommended over `extrinsicsToCameraPose`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `extrinsicsToCameraPose` function uses the postmultiply convention. Although there are no plans to remove `extrinsicsToCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `extr2pose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`se3` | `rigidtform3d`

Functions
`pose2extr` | `estimateExtrinsics` | `estrelpose` | `estworldpose` | `img2world2d`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

extrinsicsToCameraPose

(Not recommended) Convert extrinsics to camera pose

Note `extrinsicsToCameraPose` is not recommended. Use the `extr2pose` function instead. For more information, see “Version History”.

Syntax

```
cameraPose = extrinsicsToCameraPose(tform)
[orientation,location] = extrinsicsToCameraPose(rotationMatrix,
translationVector)
```

Description

`cameraPose = extrinsicsToCameraPose(tform)` returns a `rigid3d` object that contains the camera pose, `cameraPose`, in world coordinates. `tform` is a transformation from world coordinates to camera coordinates, specified as a `rigid3d` object.

`[orientation,location] = extrinsicsToCameraPose(rotationMatrix, translationVector)` returns 3-D camera pose orientation and location in world coordinates. The inputs, `rotationMatrix` and `translationVector`, represent the transformation from world coordinates to camera coordinates.

Examples

Convert Camera Coordinates to World Coordinates

Create a rotation matrix and a translation vector.

```
rotationMatrix = eye(3);
translationVector = [0 0 -10];
```

Compute the camera orientation matrix and location vector in world coordinates.

```
[orientation,location] = extrinsicsToCameraPose(rotationMatrix,translationVector)
```

```
orientation = 3×3

     1     0     0
     0     1     0
     0     0     1


location = 1×3

     0     0    10
```

Compute the camera pose in world coordinates as a rigid 3-D object.

```
tform = rigid3d(rotationMatrix,translationVector);
cameraPose = extrinsicsToCameraPose(tform)
```

```
cameraPose =
  rigid3d with properties:

      Rotation: [3×3 double]
    Translation: [0 0 10]
```

Input Arguments

`tform` — Transformation

`rigid3d` object

Transformation from world coordinates to camera coordinates, specified as a `rigid3d` object. The transformation allows you to transform points from the world coordinate system to the camera coordinate system.

`rotationMatrix` — 3-D rotation

3-by-3 matrix

3-D rotation, specified as a 3-by-3 matrix. The rotation matrix, together with the translation vector allows you to transform points from the world coordinate system to the camera coordinate system.

Data Types: `double` | `single`

`translationVector` — 3-D translation

1-by-3 vector

3-D translation, specified as a 1-by-3 vector. The translation vector together with the rotation matrix, enables you to transform points from the world coordinate system to the camera coordinate system.

Data Types: `double` | `single`

Output Arguments

`cameraPose` — Camera pose

`rigid3d` object

Camera pose in world coordinates, returned as a `rigid3d` object. `cameraPose` is computed as:

```
cameraPose.Rotation = tform.Rotation'
cameraPose.Translation = -tform.Translation * tform.Rotation'
```

`orientation` — 3-D orientation

3-by-3 matrix

3-D orientation of the camera in world coordinates, returned as a 3-by-3 matrix.



The relationship between the rotation matrix and the output orientation matrix is:
`orientation = rotationMatrix'`

`location` — 3-D location
3-element vector

3-D location of the camera in world coordinates, specified as a three-element vector.



The relationship between the translation vector and the output orientation matrix is:

`location = -translationVector*rotationMatrix'`

Version History

Introduced in R2016b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `extrinsicsToCameraPose` function uses the postmultiply convention. Although there are no plans to remove `extrinsicsToCameraPose` at this time, you can streamline your geometric transformation workflows by switching to the `extr2pose` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

- Change instances of the function name `cameraPoseToExtrinsics` to `pose2extr`.
- Specify the extrinsics as a `rigidtform3d` object using the `extrinsics` argument. `extr2pose` does not support specifying extrinsics using the `rotationMatrix` and `translationVector` input arguments. Note that you create the `rigidtform3d` object using the transpose of `rotationMatrix` or the transpose of the transformation matrix in the `T` property of `tform`.

Discouraged Usage	Recommended Replacement

This example specifies the extrinsics as a rotation matrix `rotationMatrix` in the premultiply convention and a translation vector `translationVector`, then converts the extrinsics to a camera pose using the `extrinsicsToCameraPose` function, using the transpose of `rotationMatrix`.
 	
`rotationMatrix = eye(3);`
`translationVector = [0 0 -10];`
`[orientationOld,location] = ...`
`  extrinsicsToCameraPose(rotationMatrix',translationVector);`
	This example specifies extrinsics as a `rigidtform3d` object, then converts the extrinsics to a camera pose using the `extr2pose` function.
	
	`rotationMatrix = eye(3);`
	`translationVector = [0 0 -10];`
	`extrinsics = rigidtform3d(orientation,location);`
	`cameraPose = extr2pose(extrinsics);`
	
	If you need to obtain the camera orientation and location, then you can query properties of `cameraPose`.
	
	`orientation = cameraPose.R;`
	`location = cameraPose.Translation;`
	
	If you want the orientation in the postmultiply convention, take the transpose of `cameraPose.R`.
	
	`orientationOld = cameraPose.R;`

This example specifies extrinsics as a `rigid3d` object using the transpose of the geometric transformation matrix `T` in the premultiply convention, then converts the extrinsics to a camera pose using the `extrinsicsToCameraPose` function.
 	
`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
`extrinsicsOld = rigid3d(A');`
`cameraPoseOld = extrinsicsToCameraPose(extrinsicsOld);`
	This example specifies extrinsics as a `rigidtform3d` object using the geometric transformation matrix `T`, then converts the extrinsics to a camera pose using the `extr2pose` function.
	
	`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
	`extrinsics = rigidtform3d(A);`
	`cameraPose = extr2pose(extrinsics);`
	If instead you start with an existing `rigid3d` object `extrinsicsOld`, then you can get the geometric transformation matrix in the postmultiply convention by querying the property `T` of `extrinsicsOld`.
	
	`T = extrinsicsOld.T;`
	`extrinsics = rigidtform3d(T');`
	`cameraPose = extr2pose(extrinsics);`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`extr2pose` | `pose2extr` | `estimateExtrinsics` | `estrelpose` | `estworldpose` | `img2world2d` | `world2img`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

trainACFObjectDetector

Train ACF object detector

Syntax

```
detector = trainACFObjectDetector(trainingData)
detector = trainACFObjectDetector(trainingData,Name=Value)
```

Description

`detector = trainACFObjectDetector(trainingData)` returns a trained aggregate channel features (ACF) object detector. The function uses positive instances of objects in images stored in a table or a datastore, and specified by `trainingData`. The function automatically collects negative instances from the images during training. To create a ground truth table, use the Image Labeler or Video Labeler app.

`detector = trainACFObjectDetector(trainingData,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ObjectTrainingSize=[100,100]` sets the height and width of objects during training.

Examples

Train Stop Sign Detector Using ACF Object Detector

Use the `trainACFObjectDetector` with training images to create an ACF object detector that can detect stop signs. Test the detector with a separate image.

Load the training data.

```
load('stopSignsAndCars.mat')
```

Prefix the full path to the stop sign images.

```
stopSigns = fullfile(toolboxdir('vision'),'visiondata',stopSignsAndCars{:,1});
```

Create datastores to load the ground truth data for stop signs.

```
imds = imageDatastore(stopSigns);
blds = boxLabelDatastore(stopSignsAndCars(:,2));
```

Combine the image and box label datastores.

```
ds = combine(imds,blds);
```

Train the ACF detector. Set the number of negative samples to use at each stage to 2. You can turn off the training progress output by specifying `Verbose=false`, as a `Name-Value` argument.

```
acfDetector = trainACFObjectDetector(ds,NegativeSamplesFactor=2);
```

```
ACF Object Detector Training
The training will take 4 stages. The model size is 34x31.
```

```
Sample positive examples(~100% Completed)
Compute approximation coefficients...Completed.
Compute aggregated channel features...Completed.
--------------------------------------------
Stage 1:
Sample negative examples(~100% Completed)
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 19 weak learners.
--------------------------------------------
Stage 2:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 20 weak learners.
--------------------------------------------
Stage 3:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 54 weak learners.
--------------------------------------------
Stage 4:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 61 weak learners.
--------------------------------------------
ACF object detector training is completed. Elapsed time is 17.9982 seconds.
```

Test the ACF detector on a test image.

```
img = imread('stopSignTest.jpg');
[bboxes,scores] = detect(acfDetector,img);
```

Display the detection results and insert the bounding boxes for objects into the image.

```
for i = 1:length(scores)
   annotation = sprintf('Confidence = %.1f',scores(i));
   img = insertObjectAnnotation(img,'rectangle',bboxes(i,:),annotation);
end

figure
imshow(img)
```



Input Arguments

`trainingData` — Labeled ground truth

datastore | table

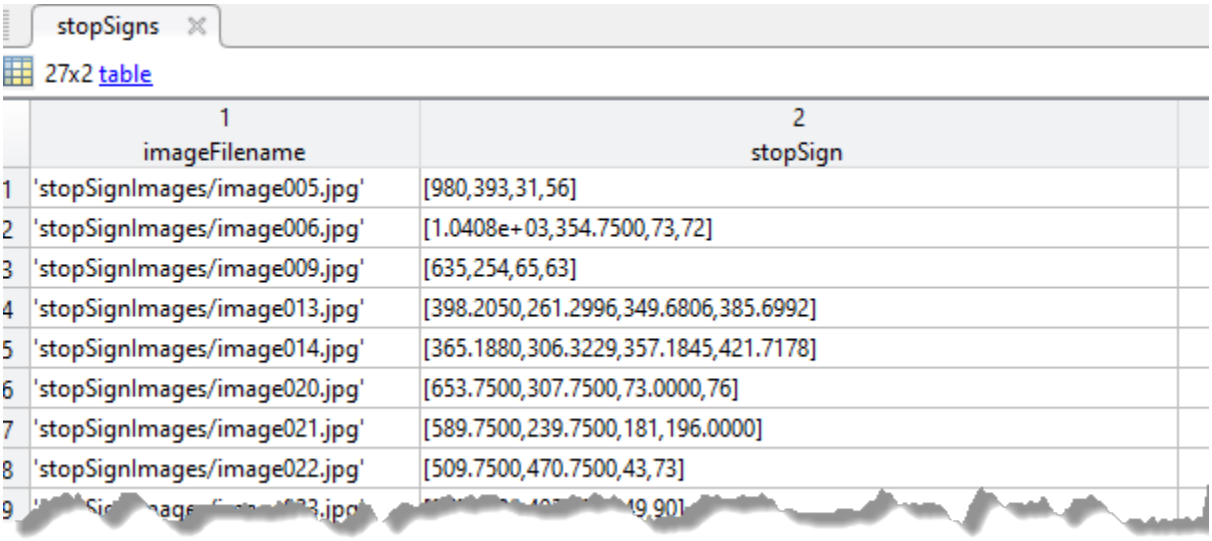
Labeled ground truth, specified as a datastore or a table.

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with at least two columns. The table describes the data contained in the columns:

Images	***boxes***	***labels (optional)***
Cell vector of grayscale or RGB images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	Cell array that contains an *M*-element categorical vector containing object class names. All categorical data returned by the datastore must contain the same categories. When you provide this data, the function uses the class label to fill the `ModelName` property of the trained detector, specified as an `acfObjectDetector` object. Otherwise, the class labels are not required for training because the ACF object detector is a single class detector.

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they

can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.


stopSigns

27x2 table

	1 imageFilename	2 stopSign
1	'stopSignImages/image005.jpg'	[980,393,31,56]
2	'stopSignImages/image006.jpg'	[1.0408e+03,354.7500,73,72]
3	'stopSignImages/image009.jpg'	[635,254,65,63]
4	'stopSignImages/image013.jpg'	[398.2050,261.2996,349.6806,385.6992]
5	'stopSignImages/image014.jpg'	[365.1880,306.3229,357.1845,421.7178]
6	'stopSignImages/image020.jpg'	[653.7500,307.7500,73.0000,76]
7	'stopSignImages/image021.jpg'	[589.7500,239.7500,181,196.0000]
8	'stopSignImages/image022.jpg'	[509.7500,470.7500,43,73]
9		

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ObjectTrainingSize=[100,100]` sets the height and width of objects during training.

`ObjectTrainingSize` — Size of objects during training

`'Auto'` (default) | 2-element vector

Size of objects during training, specified as a 2-element vector of the form [*height width*] in pixels. The minimum training size is `[8 8]`. During the training process, objects are resized to the height and width specified by `'ObjectTrainingSize'`. Increasing the size can improve detection accuracy, but also increases training and detection times.

When you specify `'Auto'`, the size is set based on the median width-to-height ratio of the positive instances.

Example: `[100,100]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumStages` — Number of training stages

4 (default) | positive integer

Number of training stages for the iterative training process, specified as a positive integer. Increasing this number can improve the detector and reduce training errors, at the expense of longer training time.

Data Types: `double`

`NegativeSamplesFactor` — Negative sample factor
5 (default) | real-valued scalar

Negative sample factor, specified as a real-valued scalar. The number of negative samples to use at each stage is equal to

`NegativeSamplesFactor` × *number of positive samples used at each stage*

Data Types: `double`

`MaxWeakLearners` — Maximum number of weak learners
`2048` (default) | positive integer scalar | vector of positive integers

Maximum number of weak learners for the last stage, specified as a positive integer scalar or vector of positive integers. If the input is a scalar, `MaxWeakLearners` specifies the maximum number for the last stage. If the input is a vector, `MaxWeakLearners` specifies the maximum number for each of the stages and must have a length equal to '`NumStages`'. These values typically increase throughout the stages. The ACF object detector uses the boosting algorithm to create an ensemble of weaker learners. You can use higher values to improve the detection accuracy, at the expense of reduced detection performance speeds. Recommended values range from 300 to 5000.

Data Types: `double`

`Verbose` — Display progress information
`true` (default) | `false`

Option to display progress information for the training process, specified as `true` or `false`.

Data Types: `logical`

Output Arguments

`detector` — Trained ACF-based object detector
`acfObjectDetector` object

Trained ACF-based object detector, returned as an `acfObjectDetector` object.

Version History

Introduced in R2017a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

References

[1] Dollar, Piotr, Ron Appel, Serge Belongie, and Pietro Perona. “Fast Feature Pyramids for Object Detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 36, no. 8 (August 2014): 1532–45. *DOI.org (Crossref)*, https://doi.org/10.1109/TPAMI.2014.2300479.

See Also

Apps
Image Labeler | Video Labeler

Functions
`acfObjectDetector` | `trainCascadeObjectDetector` | `detectPeopleACF`

trainRCNNObjectDetector

(Not recommended) Train R-CNN deep learning object detector

Note `trainRCNNObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Syntax

```
detector = trainRCNNObjectDetector(trainingData,network,options)
detector = trainRCNNObjectDetector( ___ ,Name,Value)
detector = trainRCNNObjectDetector( ___ ,'RegionProposalFcn',proposalFcn)
[detector,info] = trainRCNNObjectDetector( ___ )

detector = trainRCNNObjectDetector( ___ ,Name=Value)
```

Description

`detector = trainRCNNObjectDetector(trainingData,network,options)` trains an R-CNN (regions with convolutional neural networks) based object detector. The function uses deep learning to train the detector to detect multiple object classes.

This implementation of R-CNN does not train an SVM classifier for each object class.

This function requires that you have Deep Learning Toolbox and Statistics and Machine Learning Toolbox. It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`detector = trainRCNNObjectDetector( ___ ,Name,Value)` returns a `detector` object with optional input properties specified by one or more `Name,Value` pair arguments.

`detector = trainRCNNObjectDetector( ___ ,'RegionProposalFcn',proposalFcn)` optionally trains an R-CNN detector using a custom region proposal function.

`[detector,info] = trainRCNNObjectDetector( ___ )` also returns information on the training progress, such as training loss and accuracy, for each iteration.

`detector = trainRCNNObjectDetector( ___ ,Name=Value)` uses additional options specified by one or more name-value pair arguments and any of the previous inputs.

Examples

Train R-CNN Stop Sign Detector

Load training data and network layers.

```
load('rcnnStopSigns.mat', 'stopSigns', 'layers')
```

Add the image directory to the MATLAB path.

```
imDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondata',...
  'stopSignImages');
addpath(imDir);
```

Set network training options to use mini-batch size of 32 to reduce GPU memory usage. Lower the InitialLearnRate to reduce the rate at which network parameters are changed. This is beneficial when fine-tuning a pre-trained network and prevents the network from changing too rapidly.

```
options = trainingOptions('sgdm', ...
  'MiniBatchSize', 32, ...
  'InitialLearnRate', 1e-6, ...
  'MaxEpochs', 10);
```

Train the R-CNN detector. Training can take a few minutes to complete.

```
rcnn = trainRCNNObjectDetector(stopSigns, layers, options, 'NegativeOverlapRange', [0 0.3]);
```

```
*********************************************************************
Training an R-CNN Object Detector for the following object classes:

* stopSign

--> Extracting region proposals from 27 training images...done.

--> Training a neural network to classify objects in training data...

Training on single CPU.
Initializing input data normalization.
|========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|========================================================================================|
|       1 |           1 |       00:00:01 |       96.88% |       0.1651 |      1.0000e-06 |
|       2 |          50 |       00:00:15 |       96.88% |       0.0807 |      1.0000e-06 |
|       3 |         100 |       00:00:27 |       96.88% |       0.1340 |      1.0000e-06 |
|       5 |         150 |       00:00:38 |       96.88% |       0.0225 |      1.0000e-06 |
|       6 |         200 |       00:00:49 |       93.75% |       0.6584 |      1.0000e-06 |
|       8 |         250 |       00:01:00 |       93.75% |       0.5233 |      1.0000e-06 |
|       9 |         300 |       00:01:11 |      100.00% |   2.9456e-05 |      1.0000e-06 |
|      10 |         350 |       00:01:23 |      100.00% |       0.0009 |      1.0000e-06 |
|========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*********************************************************************
```

Test the R-CNN detector on a test image.

```
img = imread('stopSignTest.jpg');

[bbox, score, label] = detect(rcnn, img, MiniBatchSize=32);
```

Display strongest detection result.

```
[score, idx] = max(score);

bbox = bbox(idx, :);
annotation = sprintf('%s: (Confidence = %f)', label(idx), score);

detectedImg = insertObjectAnnotation(img, 'rectangle', bbox, annotation);

figure
imshow(detectedImg)
```



Remove the image directory from the path.

```
rmpath(imDir);
```

Resume Training an R-CNN Object Detector

Resume training an R-CNN object detector using additional data. To illustrate this procedure, half the ground truth data will be used to initially train the detector. Then, training is resumed using all the data.

Load training data and initialize training options.

```
load('rcnnStopSigns.mat', 'stopSigns', 'layers')

stopSigns.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
    stopSigns.imageFilename);
```

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize', 32, ...
    'InitialLearnRate', 1e-6, ...
    'MaxEpochs', 10, ...
    'Verbose', false);
```

Train the R-CNN detector with a portion of the ground truth.

```
rcnn = trainRCNNObjectDetector(stopSigns(1:10,:), layers, options, 'NegativeOverlapRange', [0 0.3
```

Get the trained network layers from the detector. When you pass in an array of network layers to `trainRCNNObjectDetector`, they are used as-is to continue training.

```
network = rcnn.Network;
layers = network.Layers;
```

Resume training using all the training data.

```
rcnnFinal = trainRCNNObjectDetector(stopSigns, layers, options);
```

Create a Network for Multiclass R-CNN Object Detection

Create an R-CNN object detector for two object classes: dogs and cats.

```
objectClasses = {'dogs','cats'};
```

The network must be able to classify both dogs, cats, and a "background" class in order to be trained using `trainRCNNObjectDetector`. In this example, a one is added to include the background.

```
numClassesPlusBackground = numel(objectClasses) + 1;
```

The final fully connected layer of a network defines the number of classes that the network can classify. Set the final fully connected layer to have an output size equal to the number of classes plus a background class.

```
layers = [ ...
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    fullyConnectedLayer(numClassesPlusBackground);
    softmaxLayer()
    classificationLayer()];
```

These network layers can now be used to train an R-CNN two-class object detector.

Use A Saved Network In R-CNN Object Detector

Create an R-CNN object detector and set it up to use a saved network checkpoint. A network checkpoint is saved every epoch during network training when the `trainingOptions` 'CheckpointPath' parameter is set. Network checkpoints are useful in case your training session terminates unexpectedly.

Load the stop sign training data.

```
load('rcnnStopSigns.mat','stopSigns','layers')
```

Add full path to image files.

```
stopSigns.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
      stopSigns.imageFilename);
```

Set the 'CheckpointPath' using the `trainingOptions` function.

```
checkpointLocation = tempdir;
options = trainingOptions('sgdm','Verbose',false, ...
    'CheckpointPath',checkpointLocation);
```

Train the R-CNN object detector with a few images.

```
rcnn = trainRCNNObjectDetector(stopSigns(1:3,:),layers,options);
```

Load a saved network checkpoint.

```
wildcardFilePath = fullfile(checkpointLocation,'convnet_checkpoint__*.mat');
contents = dir(wildcardFilePath);
```

Load one of the checkpoint networks.

```
filepath = fullfile(contents(1).folder,contents(1).name);
checkpoint = load(filepath);

checkpoint.net
```

```
ans =

  SeriesNetwork with properties:

    Layers: [15×1 nnet.cnn.layer.Layer]
```

Create a new R-CNN object detector and set it up to use the saved network.

```
rcnnCheckPoint = rcnnObjectDetector();
rcnnCheckPoint.RegionProposalFcn = @rcnnObjectDetector.proposeRegions;
```

Set the Network to the saved network checkpoint.

```
rcnnCheckPoint.Network = checkpoint.net
```

```
rcnnCheckPoint =

  rcnnObjectDetector with properties:

            Network: [1×1 SeriesNetwork]
         ClassNames: {'stopSign'  'Background'}
```

```
    RegionProposalFcn: @rcnnObjectDetector.proposeRegions
```

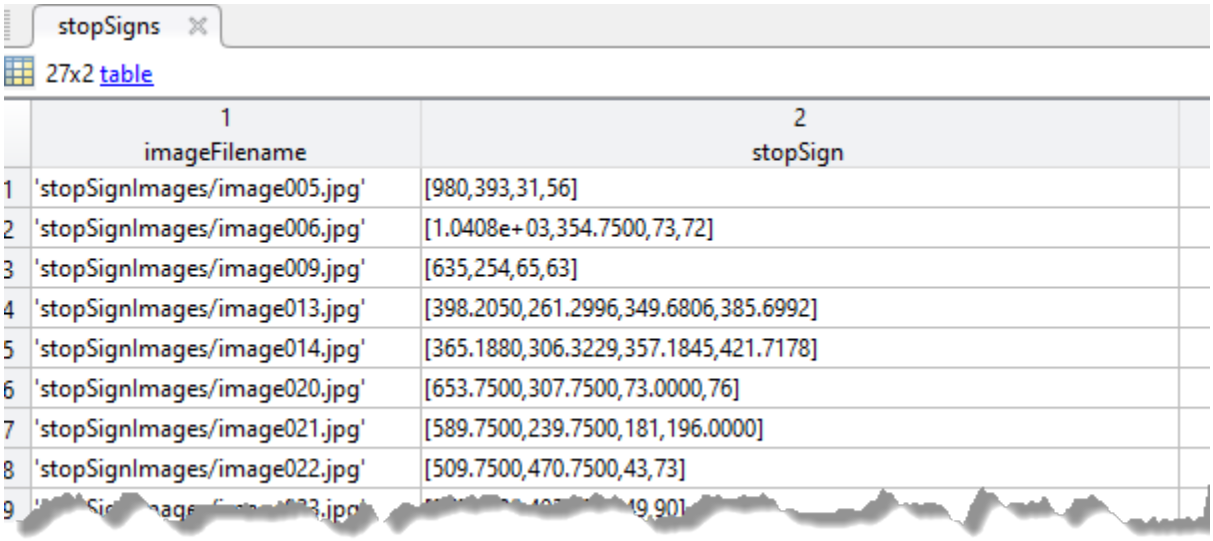
Input Arguments

`trainingData` — Labeled ground truth images

table

Labeled ground truth images, specified as a table with two or more columns.

If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.



stopSigns

27x2 table

	1 imageFilename	2 stopSign
1	'stopSignImages/image005.jpg'	[980,393,31,56]
2	'stopSignImages/image006.jpg'	[1.0408e+03,354.7500,73,72]
3	'stopSignImages/image009.jpg'	[635,254,65,63]
4	'stopSignImages/image013.jpg'	[398.2050,261.2996,349.6806,385.6992]
5	'stopSignImages/image014.jpg'	[365.1880,306.3229,357.1845,421.7178]
6	'stopSignImages/image020.jpg'	[653.7500,307.7500,73.0000,76]
7	'stopSignImages/image021.jpg'	[589.7500,239.7500,181,196.0000]
8	'stopSignImages/image022.jpg'	[509.7500,470.7500,43,73]
9		

The table variable name defines the object class name. To create the ground truth table, use the Image Labeler app. Boxes smaller than 32-by-32 are not used for training.

`network` — Network

`SeriesNetwork` object | array of `Layer` objects | `LayerGraph` object | network name

Network, specified as a `SeriesNetwork`, an array of `Layer` objects, a `layerGraph` object, or by the network name. The network is trained to classify the object classes defined in the `trainingData` table. The `SeriesNetwork`, `Layer`, and `layerGraph` objects are available in the Deep Learning Toolbox.

- When you specify the network as a `SeriesNetwork`, an array of `Layer` objects, or by the network name, the network is automatically transformed into a R-CNN network by adding new classification and regression layers to support object detection.

- The array of `Layer` objects must contain a classification layer that supports the number of object classes, plus a background class. Use this input type to customize the learning rates of each layer. An example of an array of `Layer` objects:

  ```
  layers = [imageInputLayer([28 28 3])
          convolution2dLayer([5 5],10)
          reluLayer()
          fullyConnectedLayer(10)
          softmaxLayer()
          classificationLayer()];
  ```

- When you specify the network as `SeriesNetwork`, `Layer` array, or network by name, the weights for convolution and fully-connected layers are initialized to `'narrow-normal'`.
- The network name must be one of the following valid networks names. You must also install the corresponding Add-on.
  - `'alexnet'`
  - `'vgg16'`
  - `'vgg19'`
  - `'resnet18'`
  - `'resnet50'`
  - `'resnet101'`
  - `'inceptionv3'`
  - `'googlenet'`
  - `'inceptionresnetv2'`
  - `'squeezenet'`
  - `'mobilenetv2'`
- The `LayerGraph` object must be a valid R-CNN object detection network. You can also use a `LayerGraph` object to train a custom R-CNN network.

See “Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN” to learn more about how to create a R-CNN network.

`options` — Training options
`trainingOptions` output

Training options, returned by the `trainingOptions` function from the Deep Learning Toolbox. To specify solver and other options for network training, use `trainingOptions`.

Note `trainRCNNObjectDetector` does not support these training options:

- The `ValidationData`, `ValidationFrequency`, or `ValidationPatience` options

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `PositiveOverlapRange=[0.5 1]`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"PositiveOverlapRange",[0.5 1]`

`PositiveOverlapRange` — Positive training sample ratios for range of bounding box overlap
`[0.5 1]` (default) | two-element vector

Positive training sample ratios for range of bounding box overlap, specified as a two-element vector. The vector contains values in the range [0,1]. Region proposals that overlap with ground truth bounding boxes within the specified range are used as positive training samples.

The overlap ratio used for both the `PositiveOverlapRange` and `NegativeOverlapRange` is defined as:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

A and B are bounding boxes.

`NegativeOverlapRange` — Negative training sample ratios for range of bounding box overlap
`[0.1 0.5]` (default) | two-element vector

Negative training sample ratios for range of bounding box overlap, specified as a two-element vector. The vector contains values in the range [0,1]. Region proposals that overlap with the ground truth bounding boxes within the specified range are used as negative training samples.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | integer

Maximum number of strongest region proposals to use for generating training samples, specified as an integer. Reduce this value to speed up processing time, although doing so decreases training accuracy. To use all region proposals, set this value to `inf`.

`RegionProposalFcn` — Custom region proposal
function handle

Custom region proposal function handle, specified as a function handle. If you do not specify a custom region proposal function, the default variant of the Edge Boxes algorithm [3], set in `rcnnObjectDetector`, is used. A custom `proposalFcn` must have the following functional form:

```
[bboxes,scores] = proposalFcn(I)
```

The input, `I`, is an image defined in the `groundTruth` table. The function must return rectangular bounding boxes in an *M*-by-4 array. Each row of `bboxes` contains a four-element vector, [*x*,*y*,*width*,*height*], that specifies the upper–left corner and size of a bounding box in pixels. The function must also return a score for each bounding box in an *M*-by-1 vector. Higher scores indicate that the bounding box is more likely to contain an object. The scores are used to select the strongest regions, which you can specify in `NumStrongestRegions`.

`BoxRegressionLayer` — Box regression layer name
`'auto'` (default) | character vector

Box regression layer name, specified as a character vector. Valid values are `'auto'` or the name of a layer in the input network. The output activations of this layer are used as features to train a regression model for refining the detected bounding boxes.

If the name is `'auto'`, then `trainRCNNObjectDetector` automatically selects a layer from the input network based on the type of input network:

- If the input `network` is a `SeriesNetwork` or an array of `Layer` objects, then the function selects the last convolution layer.
- If the input `network` is a `LayerGraph`, then the function selects the source of the last fully connected layer.

`ExperimentManager` — Detector training experiment monitoring
`'none'` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots.

Information monitored during training:

- Training loss at each iteration.
- Training accuracy at each iteration.
- Training root mean square error (RMSE) for the box regression layer.
- Learning rate at each iteration.

Output Arguments

`detector` — Trained R-CNN-based object detector
`rcnnObjectDetector` object

Trained R-CNN-based object detector, returned as an `rcnnObjectDetector` object. You can train an R-CNN detector to detect multiple object classes.

`info` — Training information
structure

Training information, returned as a structure with the following fields. Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are represented by `NaN`.

- `TrainingLoss` — Training loss at each iteration. This is the combination of the classification and regression loss used to train the R-CNN network.
- `TrainingAccuracy` — Training set accuracy at each iteration
- `BaseLearnRate` — Learning rate at each iteration

Limitations

- This implementation of R-CNN does not train an SVM classifier for each object class.

Tips

- To accelerate data preprocessing for training, `trainRCNNObjectDetector` automatically creates and uses a parallel pool based on your parallel preference settings. This requires Parallel Computing Toolbox.
- VGG-16, VGG-19, ResNet-101, and Inception-ResNet-v2 are large models. Training with large images may produce "Out of Memory" errors. To mitigate these errors, manually resize the images along with the bounding box ground truth data before calling `trainRCNNObjectDetector`.
- This function supports transfer learning. When a network is input by name, such as `'resnet50'`, then the software automatically transforms the network into a valid R-CNN network model based on the pretrained `resnet50` model. Alternatively, manually specify a custom R-CNN network using the `LayerGraph` extracted from a pretrained DAG network. See “Create R-CNN Object Detection Network”.
- Use the `trainingOptions` function to enable or disable verbose printing.

Version History

Introduced in R2016b

R2025a: MATLAB Compiler support will be removed

Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024b: Not Recommended

Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

References

[1] Girshick, R., J. Donahue, T. Darrell, and J. Malik. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.”*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587.

[2] Girshick, R. “Fast R-CNN.” *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1440–1448.

[3] Zitnick, C. Lawrence, and P. Dollar. “Edge Boxes: Locating Object Proposals from Edges.” *Computer Vision-ECCV, Springer International Publishing.* 2014, pp. 391–405.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainingOptions` | `trainFastRCNNObjectDetector` | `trainYOLOv2ObjectDetector` | `objectDetectorTrainingData`

Objects
`Layer` | `rcnnObjectDetector` | `imageCategoryClassifier`

Topics
“Image Category Classification Using Deep Learning”
“Object Detection Using Faster R-CNN Deep Learning”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”

estimateGeometricTransform2D

(Not recommended) Estimate 2-D geometric transformation from matching point pairs

Note `estimateGeometricTransform2D` is not recommended. Use the `estgeotform2d` function instead. For more information, see “Version History”.

Syntax

```
tform = estimateGeometricTransform2D(matchedPoints1,matchedPoints2,
transformType)
[tform,inlierIndex] = estimateGeometricTransform2D( ___ )
[tform,inlierIndex,status] = estimateGeometricTransform2D( ___ )
[ ___ ] = estimateGeometricTransform2D( ___ , Name,Value)
```

Description

`tform = estimateGeometricTransform2D(matchedPoints1,matchedPoints2, transformType)` estimates a 2-D geometric transformation between two images by mapping the inliers in the matched points from one image `matchedPoints1` to the inliers in the matched points from another image `matchedPoints2`.

`[tform,inlierIndex] = estimateGeometricTransform2D( ___ )` additionally returns a vector specifying each matched point pair as either an inlier or an outlier using the input arguments from the previous syntax.

`[tform,inlierIndex,status] = estimateGeometricTransform2D( ___ )` additionally returns a status code indicating whether or not the function could estimate a transformation and, if not, why it failed. If you do not specify the `status` output, the function instead returns an error for conditions that cannot produce results.

`[ ___ ] = estimateGeometricTransform2D( ___ , Name,Value)` specifies additional options using one or more name-value pair arguments in addition to any combination of arguments from previous syntaxes. For example, `"Confidence",99` sets the confidence value for finding the maximum number of inliers to 99.

Examples

Recover Transformed Image Using SURF Feature Points

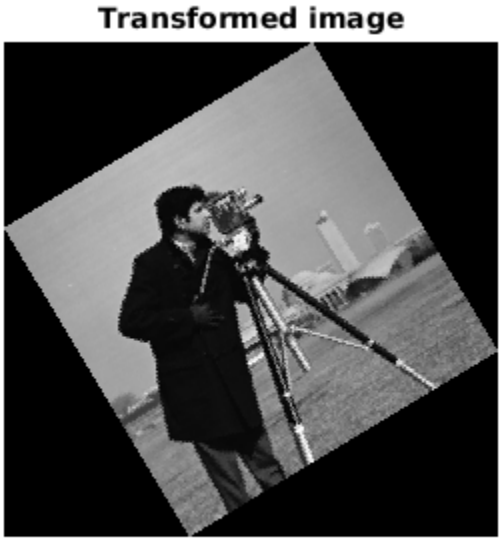
Read an image and display it.

```
original = imread('cameraman.tif');
imshow(original)
title('Base Image')
```



Distort and display the transformed image.

```
distorted = imresize(original,0.7);
distorted = imrotate(distorted,31);
figure
imshow(distorted)
title('Transformed Image')
```

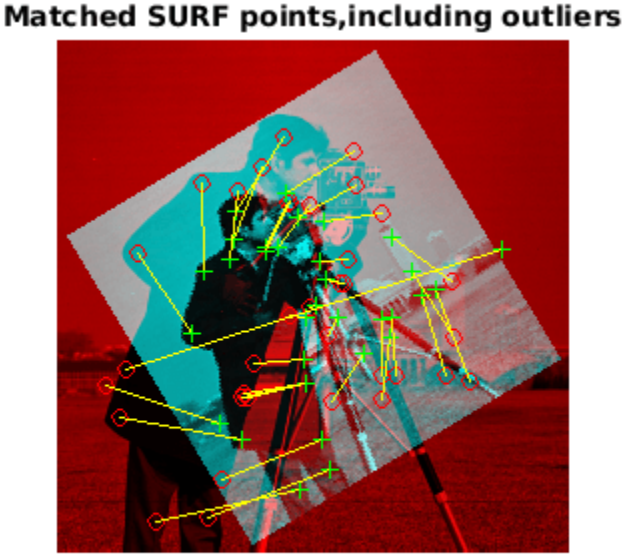


Detect and extract features from the original and the transformed images.

```
ptsOriginal  = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);
[featuresOriginal,validPtsOriginal] = extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = extractFeatures(distorted,ptsDistorted);
```

Match and display features between the images.

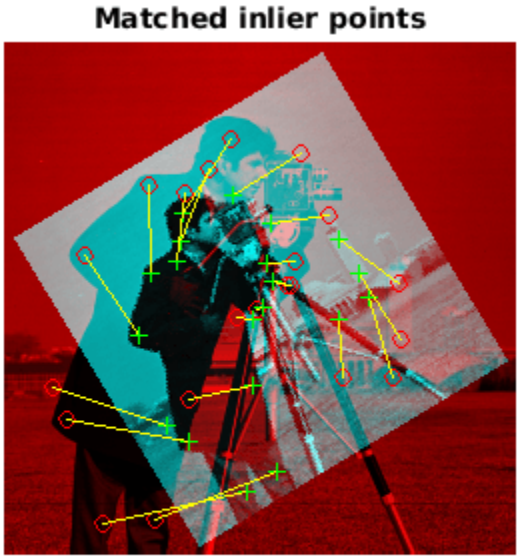
```
index_pairs = matchFeatures(featuresOriginal,featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));
figure
showMatchedFeatures(original,distorted,matchedPtsOriginal,matchedPtsDistorted)
title('Matched SURF Points With Outliers');
```



Exclude the outliers, estimate the transformation matrix, and display the results.

```
[tform,inlierIdx] = estimateGeometricTransform2D(matchedPtsDistorted,matchedPtsOriginal,'similar
inlierPtsDistorted = matchedPtsDistorted(inlierIdx,:);
inlierPtsOriginal  = matchedPtsOriginal(inlierIdx,:);

figure
showMatchedFeatures(original,distorted,inlierPtsOriginal,inlierPtsDistorted)
title('Matched Inlier Points')
```



Use the estimated transformation to recover and display the original image from the distorted image.

```
outputView = imref2d(size(original));
Ir = imwarp(distorted,tform,'OutputView',outputView);
figure
imshow(Ir);
title('Recovered Image');
```



Input Arguments

`matchedPoints1` — Matched points from first image

`cornerPoints` object | `SURFPoints` object | `MSERRegions` object | `ORBPoints` object | `BRISKPoints` object | `KAZEPoints` object | *M*-by-2 matrix of [*x*,*y*] coordinates

Matched points from the first image, specified as either a `KAZEPoints` object, `cornerPoints` object, `SURFPoints` object, `MSERRegions` object, `ORBPoints` object, `BRISKPoints` object, or an *M*-by-2 matrix in which each row is a pair of [*x*,*y*] coordinates and *M* is the number of matched points.

`matchedPoints2` — Matched points from second image

`cornerPoints` object | `SURFPoints` object | `MSERRegions` object | `ORBPoints` object | `BRISKPoints` object | `KAZEPoints` object | *M*-by-2 matrix of [*x*,*y*] coordinates

Matched points from the second image, specified as either a `KAZEPoints` object, `cornerPoints` object, `SURFPoints` object, `MSERRegions` object, `ORBPoints` object, `BRISKPoints` object, or an *M*-by-2 matrix in which each row is a pair of [*x*,*y*] coordinates and *M* is the number of matched points.

`transformType` — Transformation type

`"rigid"` | `"similarity"` | `"affine"` | `"projective"`

Transformation type, specified as a character string. You can set the transform type to `"rigid"`, `"similarity"`, `"affine"`, or `"projective"`. Each transform type requires a minimum number of matched pairs of points to estimate a transformation. You can generally improve the accuracy of a transformation by using a larger number of matched pairs of points. This table shows the type of object associated with each transformation type and the minimum number of matched pairs of points the transformation requires.

Transformation Type	`tform` Object	Minimum Number of Matched Pairs of Points
`"rigid"`	`rigid2d`	2
`"similarity"`	`affine2d`	2
`"affine"`	`affine2d`	3
`"projective"`	`projective2d`	4

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Confidence",99` sets the confidence value for finding the maximum number of inliers to `99`.

`MaxNumTrials` — Maximum random trials

`1000` (default) | positive integer

Maximum number of random trials, specified as the comma-separated pair consisting of `"MaxNumTrials"` and a positive integer. This value specifies the number of randomized attempts the function makes to find matching point pairs. Specifying a higher value causes the function to perform additional computations, which increases the likelihood of finding inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Confidence` — Confidence of finding maximum number of inliers
99 (default) | positive numeric scalar

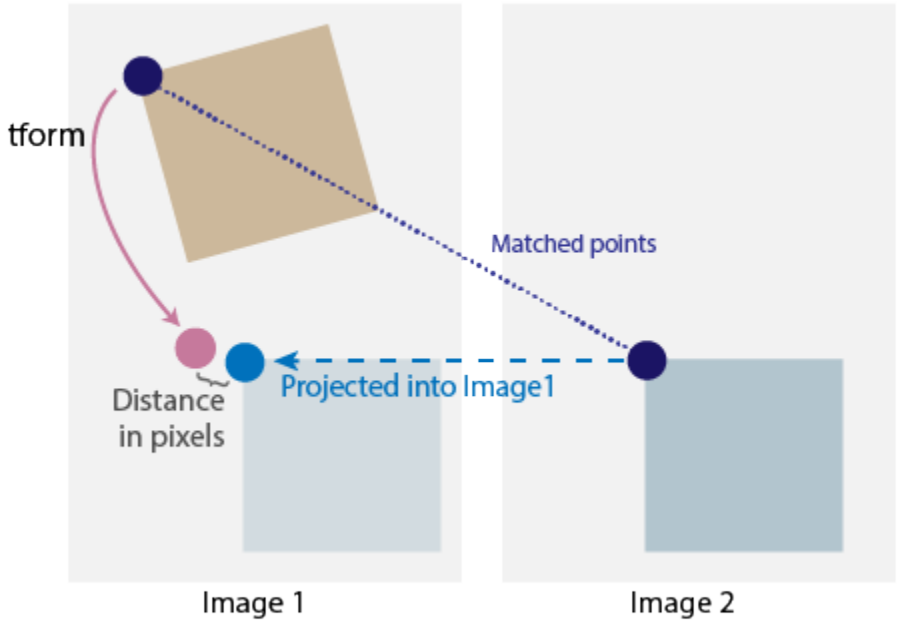
Confidence of finding the maximum number of inliers, specified as the comma-separated pair consisting of `"Confidence"` and a positive numeric scalar in the range (0, 100). Increasing this value causes the function to perform additional computations, which increases the likelihood of finding a greater number of inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxDistance` — Maximum distance from point to projection
`1.5` (default) | positive numeric scalar

Maximum distance from a point to the projection of the corresponding point, specified as the comma-separated pair consisting of `"MaxDistance"` and a positive numeric scalar. `"MaxDistance"` specifies the maximum distance, in pixels, that a point can differ from the projected location of its corresponding point to be considered an inlier. The corresponding projection is based on the estimated transform.

The function checks for a transformation from `matchedPoints1` to `matchedPoints2`, and then calculates the distance between the matched points in each pair after applying the transformation. If the distance between the matched points in a pair is greater than the `"MaxDistance"` value, then the pair is considered an outlier for that transformation. If the distance is less than `"MaxDistance"`, then the pair is considered an inlier.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`tform` — Geometric transformation

`rigid2d` object | `affine2d` object | `projective2d` object

Geometric transformation, returned as a `rigid2d`, `affine2d`, or `projective2d` object.

The returned geometric transformation matrix maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`. The function returns an object specific to the transformation type specified by the `transformType` input argument.

`transformType`	Geometric Transformation Object
`"rigid"`	`rigid2d`
`"similarity"`	`affine2d`
`"affine"`	`affine2d`
`"projective"`	`projective2d`

`inlierIndex` — Index of inliers

M-by-1 logical vector

Index of inliers, returned as an *M*-by-1 logical vector, where *M* is the number of point pairs. Each element contains either a logical `1` (`true`), indicating that the corresponding point pair is an inlier, or a logical `0` (`false`), indicating that the corresponding point pair is an outlier.

Data Types: `logical`

`status` — Status code

`0` | `1` | `2`

Status code, returned as `0`, `1`, or `2`. The status code indicates whether or not the function could estimate the transformation and, if not, why it failed.

Value	Description
`0`	No error
`1`	`matchedPoints1` and `matchedPoints2` inputs do not contain enough points
`2`	Not enough inliers found

If you do not specify the `status` code output, the function returns an error if it cannot produce results.

Data Types: `int32`

Algorithms

The function excludes outliers using the M-estimator sample consensus (MSAC) algorithm. The MSAC algorithm is a variant of the random sample consensus (RANSAC) algorithm. Results may not be identical between runs due to the randomized nature of the MSAC algorithm.

Version History

Introduced in R2020b

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateGeometricTransform2d` function estimates geometric transformations using the postmultiply convention. Although there are no plans to remove `estimateGeometricTransform2d` at this time, you can streamline your geometric transformation workflows by switching to the `estgeotform2d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code, change instances of the function name `estimateGeometricTransform2d` to `estgeotform2d`. You do not need to change the other arguments.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- You cannot use `SURFPoints` object as an input to `estimateGeometricTransform2D`. See the “Introduction to Code Generation with Feature Matching and Registration” example for details on how to use `SURFPoints` with `estimateGeometricTransform2D` in codegen.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`estgeotform2d` | `fitgeotform2d` | `estgeotform3d` | `estimateFundamentalMatrix` | `detectSURFFeatures` | `detectMinEigenFeatures` | `detectORBFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `extractFeatures` | `matchFeatures`

Objects

`KAZEPoints` | `cornerPoints` | `MSERRegions` | `SURFPoints` | `ORBPoints` | `BRISKPoints`

Topics

“Create Panorama”

“Point Feature Types”
“Coordinate Systems”
“2-D and 3-D Geometric Transformation Process Overview”
“Introduction to Code Generation with Feature Matching and Registration”

estimateGeometricTransform3D

(Not recommended) Estimate 3-D geometric transformation from matching point pairs

Note `estimateGeometricTransform3D` is not recommended. Use the `estgeotform3d` function instead. For more information, see “Version History”.

Syntax

```
tform = estimateGeometricTransform3D(matchedPoints1,matchedPoints2,
transformType)
[tform,inlierIndex] = estimateGeometricTransform3D( ___ )
[tform,inlierIndex,status] = estimateGeometricTransform3D( ___ )
[ ___ ] = estimateGeometricTransform3D( ___ , Name,Value)
```

Description

`tform = estimateGeometricTransform3D(matchedPoints1,matchedPoints2, transformType)` estimates a 3-D geometric transformation between two sets of 3-D points by mapping the inliers in the matched points from one set of 3-D points `matchedPoints1` to the inliers in the matched points from the other set of 3-D points `matchedPoints2`.

`[tform,inlierIndex] = estimateGeometricTransform3D( ___ )` additionally returns a vector specifying each matched point pair as either an inlier or an outlier using the input arguments from the previous syntax.

`[tform,inlierIndex,status] = estimateGeometricTransform3D( ___ )` additionally returns a status code indicating whether or not the function could estimate a transformation and, if not, why it failed. If you do not specify the `status` output, the function instead returns an error for conditions that cannot produce results.

`[ ___ ] = estimateGeometricTransform3D( ___ , Name,Value)` specifies additional options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `"Confidence",99` sets the confidence value for finding the maximum number of inliers to `99`.

Examples

Estimate Transformation Between Two Point Clouds

Load a point cloud file into the workspace.

```
ptCloud1 = pcread('teapot.ply')
```

```
ptCloud1 =
  pointCloud with properties:

     Location: [41472×3 single]
        Count: 41472
```

```
      XLimits: [-3 3.4340]
      YLimits: [-2 2]
      ZLimits: [0 3.1500]
        Color: []
       Normal: []
    Intensity: []
```

```
ptCloud1 = pcdownsample(ptCloud1,'random',0.25);
```

Create a rigid 3-D transformation object with a 30-degree rotation.

```
theta = 30; % degrees
rot = [cosd(theta)  sind(theta) 0; ...
      -sind(theta)  cosd(theta) 0; ...
            0           0       1];
trans = [0 0 0];
tform = rigid3d(rot,trans);
```

Transform the point cloud using the transformation object.

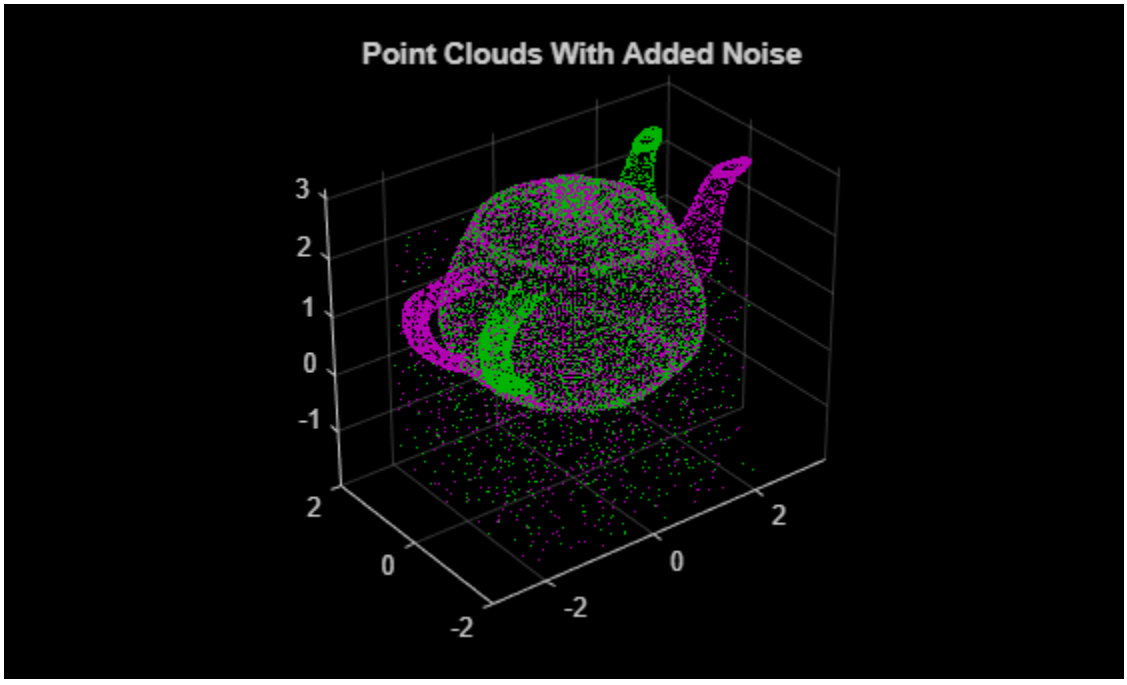
```
ptCloud2 = pctransform(ptCloud1,tform);
```

To introduce noise, add random points to both point clouds.

```
noise1 = rescale(rand(1000,3),-2,2);
ptCloud1 = pointCloud([ptCloud1.Location;noise1]);
noise2 = rescale(rand(1000,3),-2,2);
ptCloud2 = pointCloud([ptCloud2.Location;noise2]);
```

Visualize the noisy point clouds.

```
figure
pcshowpair(ptCloud1,ptCloud2)
title('Point Clouds With Added Noise')
```



Extract matched points from the point clouds.

```
matchedPoints1 = ptCloud1.Location;
matchedPoints2 = ptCloud2.Location;
```

Estimate the rigid transformation between the point clouds.

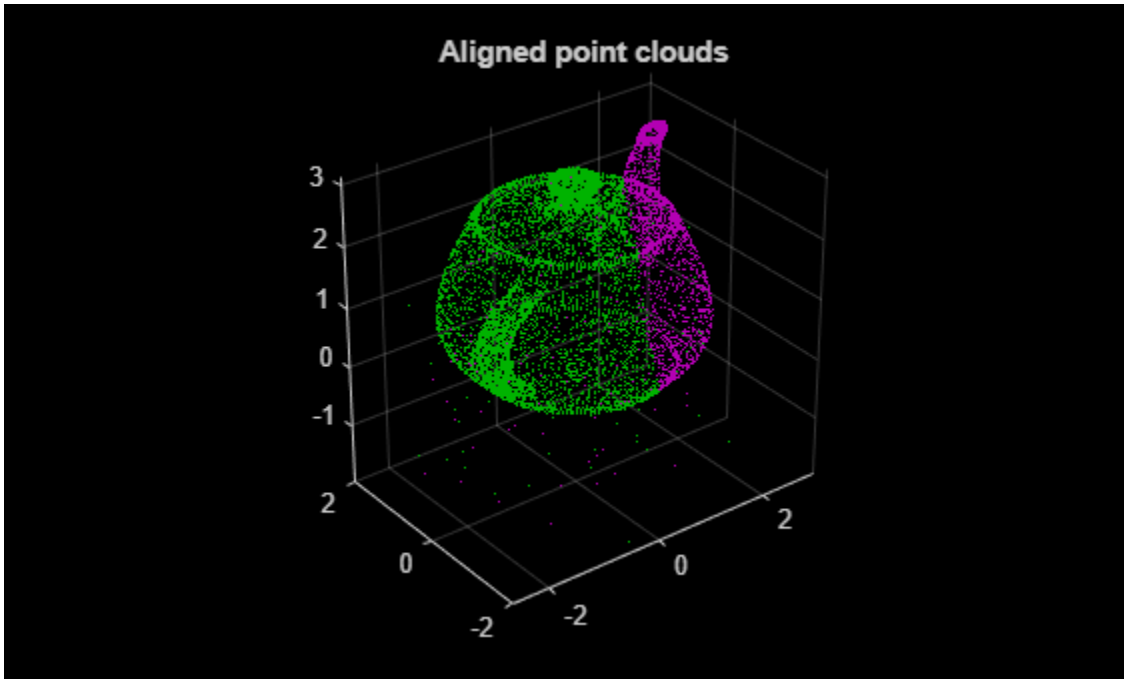
```
[tformEst,inlierIndex] = estimateGeometricTransform3D(matchedPoints1, ...
    matchedPoints2,'rigid');
```

Extract the inlier points.

```
inliersPtCloud1 = transformPointsForward(tformEst,matchedPoints1(inlierIndex,:));
inliersPtCloud2 = matchedPoints2(inlierIndex,:);
```

Visualize the inliers of the aligned point clouds.

```
figure
firstPtCloud = pointCloud(inliersPtCloud1);
secondPtCloud = pointCloud(inliersPtCloud2);
pcshowpair(firstPtCloud,secondPtCloud)
title('Aligned point clouds')
```



Input Arguments

`matchedPoints1` — First set of matched 3-D points

M-by-3 matrix of [*x*,*y*,*z*] coordinates

First set of matched 3-D points, specified as an *M*-by-3 matrix in which each row is a set of (*x*,*y*,*z*) coordinates and *M* is the number of matched points.

`matchedPoints2` — Second set of matched 3-D points
M-by-3 matrix of [*x*,*y*,*z*] coordinates

Second set of matched 3-D points, specified as an *M*-by-3 matrix in which each row is a set of (*x*,*y*,*z*) coordinates and *M* is the number of matched points.

`transformType` — Transformation type
`"rigid"` | `"similarity"`

Transformation type, specified as `"rigid"` or `"similarity"`. Each transform type requires a minimum number of matched pairs of points to estimate a transformation. You can generally improve the accuracy of a transformation by using a larger number of matched pairs of points. This table shows the type of object associated with each transformation type and the minimum number of matched pairs of points the transformation requires.

`transformType`	`tform` Object	Minimum Number of Matched Pairs of Points
`"rigid"`	`rigid3d`	3
`"similarity"`	`affine3d`	3

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Confidence",99` sets the confidence value for finding the maximum number of inliers to `99`.

`MaxNumTrials` — Maximum random trials
`1000` (default) | positive integer

Maximum number of random trials, specified as a positive integer. This value specifies the number of randomized attempts the function makes to find matching point pairs. Specifying a higher value causes the function to perform additional computations, which increases the likelihood of finding inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Confidence` — Confidence of finding maximum number of inliers
`99` (default) | positive numeric scalar

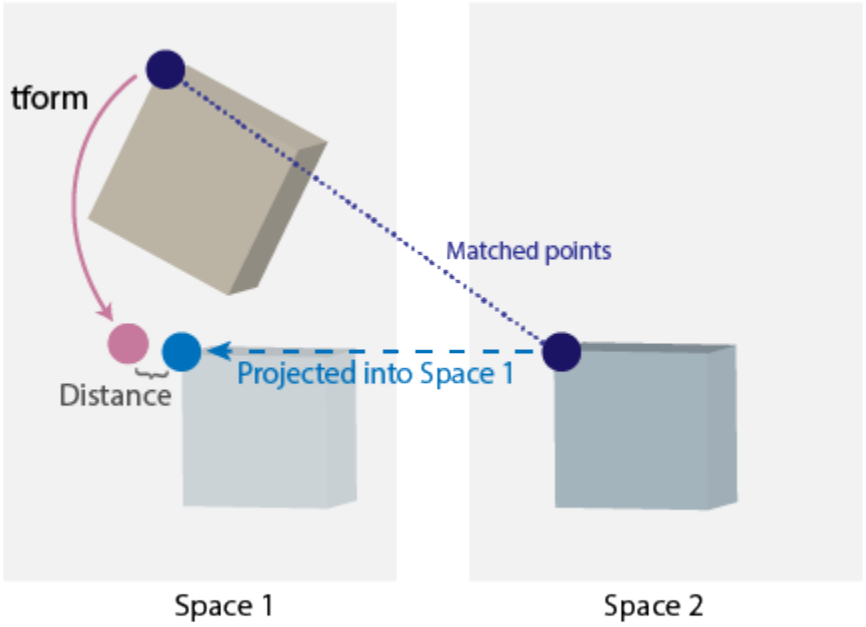
Confidence of finding the maximum number of inliers, specified as a positive numeric scalar in the range (0, 100). Increasing this value causes the function to perform additional computations, which increases the likelihood of finding a greater number of inliers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxDistance` — Maximum distance from point to projection
`1.5` (default) | positive numeric scalar

Maximum distance from a point to the projection of the corresponding point, specified as a positive numeric scalar. `"MaxDistance"` specifies the maximum distance, in pixels, that a point can differ from the projected location of its corresponding point to be considered an inlier. The corresponding projection is based on the estimated transform.

The function checks for a transformation from `matchedPoints1` to `matchedPoints2`, and then calculates the distance between the matched points in each pair after applying the transformation. If the distance between the matched points in a pair is greater than the `"MaxDistance"` value, then the pair is considered an outlier for that transformation. If the distance is less than `"MaxDistance"`, then the pair is considered an inlier.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`tform` — Geometric transformation

`rigid3d` object | `affine3d` object

Geometric transformation, returned as an `affine3d` or a `rigid3d` object.

The returned geometric transformation matrix maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`. The function returns an object specific to the transformation type specified by the `transformType` input argument.

`transformType`	`tform`
`"rigid"`	`rigid3d`
`"similarity"`	`affine3d`

`inlierIndex` — Inliers

M-by-1 logical vector

Inliers, returned as an *M*-by-1 logical vector of *M* point pairs. Each element contains either a logical true, `1`, to indicate the point pair is an inlier, or a logical false, `0`, to indicate the point pair is an outlier.

`status` — Status code
`0` | `1` | `2`

Status code, returned as `0`, `1`, or `2`. The status code indicates whether or not the function could estimate the transformation and, if not, why it failed.

Value	Description
`0`	No error
`1`	`matchedPoints1` and `matchedPoints2` inputs do not contain enough points
`2`	Not enough inliers found

If you do not specify the `status` code output, the function returns an error if it cannot produce results.

Data Types: `int32`

Algorithms

The function excludes outliers using the M-estimator sample consensus (MSAC) algorithm. The MSAC algorithm is a variant of the random sample consensus (RANSAC) algorithm. Results may not be identical between runs due to the randomized nature of the MSAC algorithm.

Version History

Introduced in R2020b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateGeometricTransform3d` function estimates geometric transformations using the postmultiply convention. Although there are no plans to remove `estimateGeometricTransform3d` at this time, you can streamline your geometric transformation workflows by switching to the `estgeotform3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code, change instances of the function name `estimateGeometricTransform3d` to `estgeotform3d`. You do not need to change the other arguments.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions

`estgeotform3d` | `estgeotform2d` | `pcregistericp` | `ransac`

Topics

“Coordinate Systems”
“2-D and 3-D Geometric Transformation Process Overview”

estimateGeometricTransform

(Not recommended) Estimate geometric transform from matching point pairs

Note `estimateGeometricTransform` is not recommended. Use the `estgeotform2d` or `estgeotform3d` function instead. For more information, see “Version History”.

Syntax

```
tform = estimateGeometricTransform(matchedPoints1,matchedPoints2,
transformType)

[tform,inlierpoints1,inlierpoints2] = estimateGeometricTransform(
matchedPoints1,matchedPoints2,transformType)
[___,status] = estimateGeometricTransform(matchedPoints1,matchedPoints2,
transformType)

[___] = estimateGeometricTransform(matchedPoints1,matchedPoints2,
transformType, Name,Value)
```

Description

`tform = estimateGeometricTransform(matchedPoints1,matchedPoints2, transformType)` returns a 2-D geometric transform object, `tform`. The `tform` object maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`.

The function excludes outliers using the M-estimator SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a variant of the Random Sample Consensus (RANSAC) algorithm. Results may not be identical between runs because of the randomized nature of the MSAC algorithm.

`[tform,inlierpoints1,inlierpoints2] = estimateGeometricTransform( matchedPoints1,matchedPoints2,transformType)` returns the corresponding inlier points in `inlierpoints1` and `inlierpoints2`.

`[___,status] = estimateGeometricTransform(matchedPoints1,matchedPoints2, transformType)` returns a status code of `0`, `1`, or `2`. If you do not request the `status` code output, the function returns an error for conditions that cannot produce results.

`[___] = estimateGeometricTransform(matchedPoints1,matchedPoints2, transformType, Name,Value)` uses additional options specified by one or more name-value arguments.

Examples

Recover Transformed Image Using SURF Feature Points

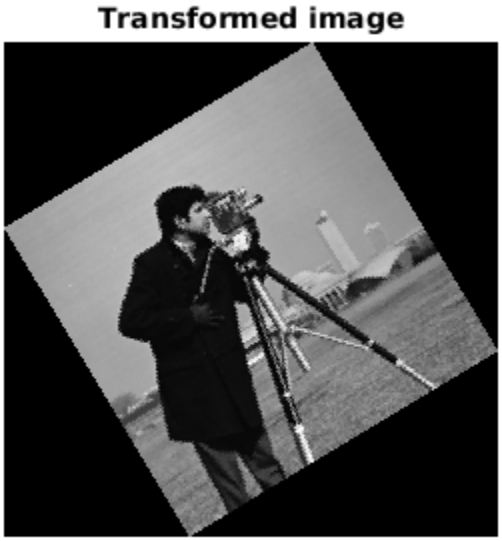
Read an image and display it.

```
original = imread('cameraman.tif');
imshow(original)
title('Base Image')
```



Distort and display the transformed image.

```
distorted = imresize(original,0.7);
distorted = imrotate(distorted,31);
figure
imshow(distorted)
title('Transformed Image')
```

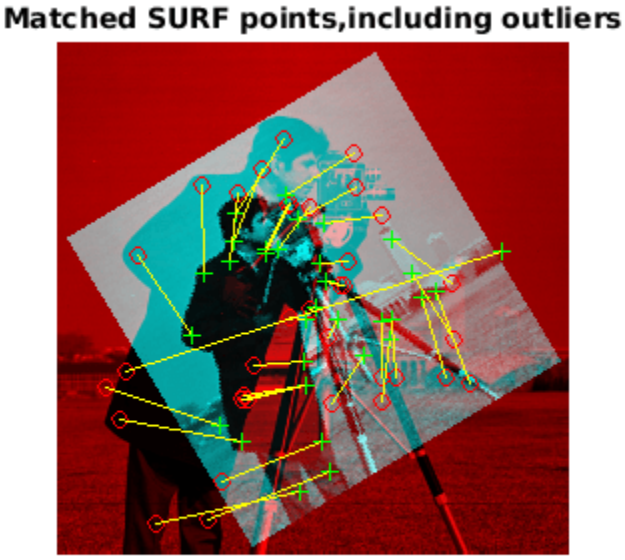


Detect and extract features from the original and the transformed images.

```
ptsOriginal  = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);
[featuresOriginal,validPtsOriginal] = extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = extractFeatures(distorted,ptsDistorted);
```

Match and display features between the images.

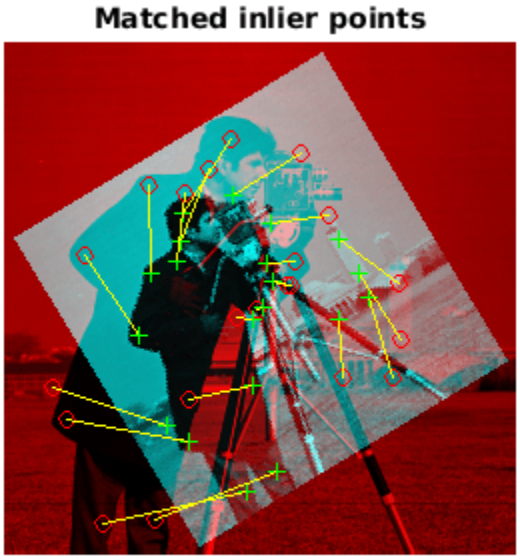
```
index_pairs = matchFeatures(featuresOriginal,featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));
figure
showMatchedFeatures(original,distorted,matchedPtsOriginal,matchedPtsDistorted)
title('Matched SURF Points With Outliers');
```



Exclude the outliers, estimate the transformation matrix, and display the results.

```
[tform,inlierIdx] = estimateGeometricTransform2D(matchedPtsDistorted,matchedPtsOriginal,'similar
inlierPtsDistorted = matchedPtsDistorted(inlierIdx,:);
inlierPtsOriginal  = matchedPtsOriginal(inlierIdx,:);

figure
showMatchedFeatures(original,distorted,inlierPtsOriginal,inlierPtsDistorted)
title('Matched Inlier Points')
```



Use the estimated transformation to recover and display the original image from the distorted image.

```
outputView = imref2d(size(original));
Ir = imwarp(distorted,tform,'OutputView',outputView);
figure
imshow(Ir);
title('Recovered Image');
```



Input Arguments

`matchedPoints1` — Matched points from image 1

`cornerPoints` object | `SIFTPoints` | `SURFPoints` object | `MSERRegions` object | `ORBPoints` object | `BRISKpoints` | `KAZEPoints` | *M*-by-2 matrix of [*x*,*y*] coordinates

Matched points from image 1, specified as either a `KAZEPoints`, `cornerPoints` object, `SIFTPoints`, `SURFPoints` object, `MSERRegions` object, `ORBPoints` object, `BRISKPoints` or an *M*-by-2 matrix of [x,y] coordinates. The function excludes outliers using the M-estimator SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a variant of the Random Sample Consensus (RANSAC) algorithm.

`matchedPoints2` — Matched points from image 2

`cornerPoints` object | `SIFTPoints` | `SURFPoints` object | `MSERRegions` object | `ORBPoints` object | `BRISKpoints` | `KAZEPoints` | *M*-by-2 matrix of [*x*,*y*] coordinates

Matched points from image 2, specified as either a `KAZEPoints`, `cornerPoints` object, `SIFTPoints`, `SURFPoints` object, `MSERRegions` object, `ORBPoints` object, `BRISKPoints` or an *M*-by-2 matrix of [x,y] coordinates. The function excludes outliers using the M-estimator SAmple Consensus (MSAC) algorithm. The MSAC algorithm is a variant of the Random Sample Consensus (RANSAC) algorithm.

`transformType` — Transform type

`"similarity"` | `"affine"` | `"projective"`

Transform type, specified as `"similarity"`, `"affine"`, or `"projective"`. The greater the number of matched pairs of points, the greater the accuracy of the estimated transformation. The minimum number of matched pairs of points for each transform type:

Transform Type	Minimum Number of Matched Pairs of Points
`"similarity"`	2
`"affine"`	3
`"projective"`	4

Data Types: `string` | `char`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Confidence",99` sets the confidence value to 99.

`MaxNumTrials` — Maximum random trials

`1000` (default) | positive integer

Maximum number of random trials for finding the inliers, specified as a positive integer scalar. Increasing this value improves the robustness of the results at the expense of additional computations.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

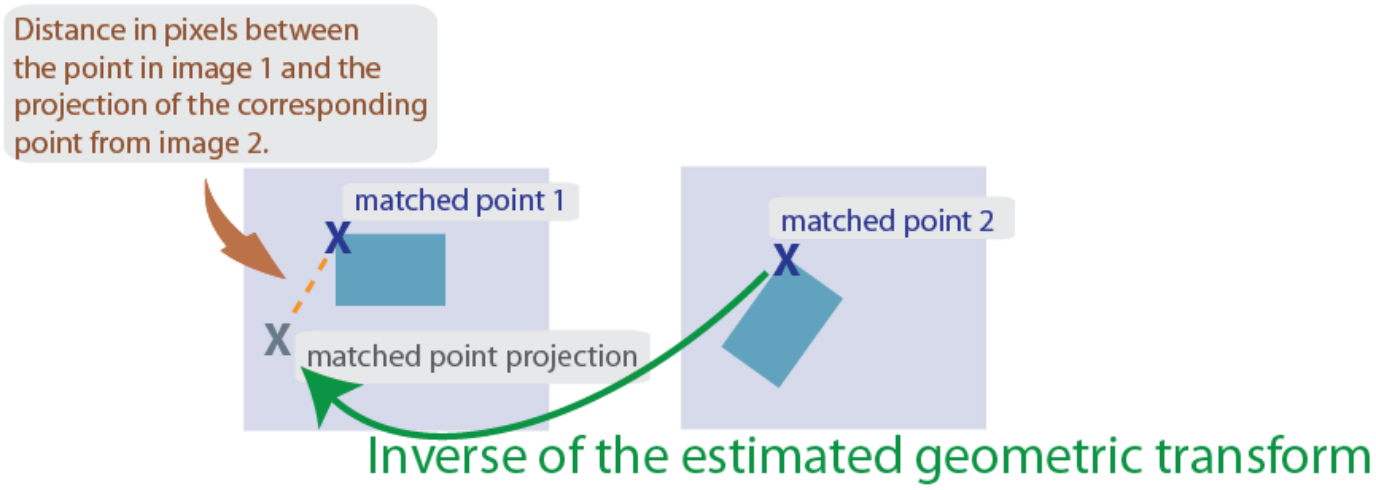
`Confidence` — Confidence of finding maximum number of inliers
99 (default) | positive numeric scalar

Confidence of finding the maximum number of inliers, specified as a percentage numeric scalar in the range (0 100). Increasing this value improves the robustness of the results at the expense of additional computations.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxDistance` — Maximum distance from point to projection
`1.5` (default) | positive numeric scalar

Maximum distance in pixels, from a point to the projection of its corresponding point, specified as a positive numeric scalar. The corresponding projection is based on the estimated transform.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`tform` — Geometric transformation
`affine2d` object | `projective2d` object

Geometric transformation, returned as either an `affine2d` object or a `projective2d` object.

The returned geometric transformation matrix maps the inliers in `matchedPoints1` to the inliers in `matchedPoints2`. When you set the `transformType` input to either `"similarity"` or `"affine"`, the function returns an `affine2d` object. Otherwise, it returns a `projective2d` object.

`inlierpoints1` — Inlier points in image 1
inlier points

Inlier points in image 1, returned as the same type as the input matching points.

`inlierpoints2` — Inlier points in image 2
inlier points

Inlier points in image 2, returned as the same type as the input matching points.

`status` — Status code
`0` | `1` | `2`

Status code, returned as the value `0`, `1`, or `2`.

`status`	Description
`0`	No error.
`1`	`matchedPoints1` and `matchedPoints2` inputs do not contain enough points.
`2`	Not enough inliers found.

If you do not request the `status` code output, the function will throw an error for the two conditions that cannot produce results.

Data Types: `double`

Version History

Introduced in R2013a

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, many Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateGeometricTransform` function estimates geometric transformations using the postmultiply convention. Although there are no plans to remove `estimateGeometricTransform` at this time, you can streamline your geometric transformation workflows by switching to the `estgeotform2d` and `estgeotform3d` functions, which support the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Torr, P.H.S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding.* 78, no. 1 (April 2000): 138–56. https://doi.org/10.1006/cviu.1999.0832.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- You cannot use `SURFPoints` object as an input to `estimateGeometricTransform`. See the “Introduction to Code Generation with Feature Matching and Registration” example for details on how to use `SURFPoints` with `estimateGeometricTransform` in codegen.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Objects

`SIFTPoints` | `KAZEPoints` | `cornerPoints` | `MSERRegions` | `SURFPoints` | `ORBPoints` | `BRISKPoints`

Functions

`estgeotform2d` | `estgeotform3d` | `fitgeotform2d` | `estimateFundamentalMatrix` | `detectSURFFeatures` | `detectMinEigenFeatures` | `detectORBFeatures` | `detectFASTFeatures` | `detectMSERFeatures` | `extractFeatures` | `matchFeatures`

Topics

“Create Panorama”
“Point Feature Types”
“Coordinate Systems”
“2-D and 3-D Geometric Transformation Process Overview”
“Introduction to Code Generation with Feature Matching and Registration”

estimateUncalibratedRectification

(Not recommended) Uncalibrated stereo rectification

Note `estimateUncalibratedRectification` is not recommended. Use the `estimateStereoRectification` function instead. For more information, see “Version History”.

Syntax

```
[T1,T2] = estimateUncalibratedRectification(F,inlierPoints1,inlierPoints2,
imagesize)
```

Description

`[T1,T2] = estimateUncalibratedRectification(F,inlierPoints1,inlierPoints2, imagesize)` returns projective transformations for rectifying stereo images. This function does not require either intrinsic or extrinsic camera parameters.

Examples

Find Fundamental Matrix Describing Epipolar Geometry

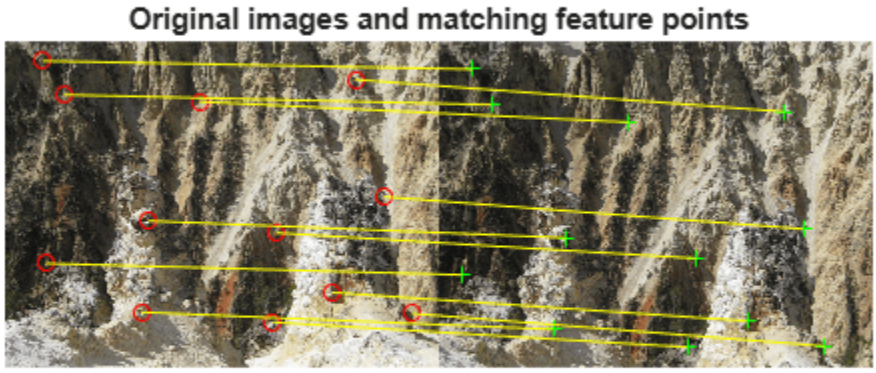
This example shows how to compute the fundamental matrix from corresponding points in a pair of stereo images.

Load the stereo images and feature points which are already matched.

```
I1 = imread('yellowstone_left.png');
I2 = imread('yellowstone_right.png');
load yellowstone_inlier_points;
```

Display point correspondences. Notice that the matching points are in different rows, indicating that the stereo pair is not rectified.

```
showMatchedFeatures(I1, I2,inlier_points1,inlier_points2,'montage');
title('Original images and matching feature points');
```



Compute the fundamental matrix from the corresponding points.

```
f = estimateFundamentalMatrix(inlier_points1,inlier_points2,...
    'Method','Norm8Point');
```

Compute the rectification transformations.

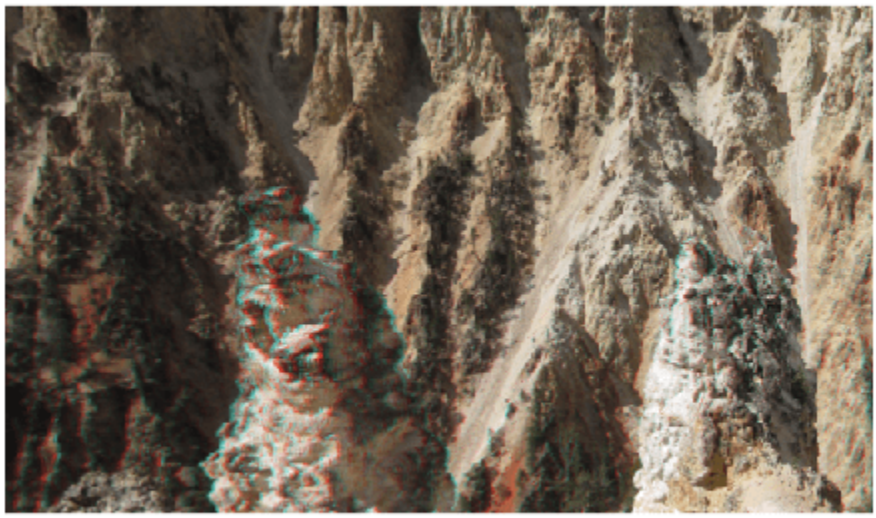
```
[t1, t2] = estimateUncalibratedRectification(f,inlier_points1,...
    inlier_points2,size(I2));
```

Rectify the stereo images using projective transformations t1 and t2.

```
[I1Rect,I2Rect] = rectifyStereoImages(I1,I2,t1,t2);
```

Display the stereo anaglyph, which can also be viewed with 3-D glasses.

```
figure;
imshow(stereoAnaglyph(I1Rect,I2Rect));
```



Input Arguments

F — Fundamental matrix for stereo images

3-by-3 matrix

Fundamental matrix for the stereo images, specified as a 3-by-3 fundamental matrix. The fundamental matrix satisfies the following criteria:
If P_1, a point in image 1, corresponds to P_2, a point in image 2, then:
$[P_2,1] * F * [P_1,1]' = 0$

Data Types: `single` | `double`

inlierPoints1 — Coordinates of corresponding points

`ORBPoints` | `BRISKPoints` | `SIFTPoints` | `SURFPoints` | `cornerPoints` | `MSERRegions` | *M*-by-2 matrix of [*x*,*y*] coordinates

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [x y] coordinates, or as a `ORBPoints`, `BRISKPoints`, `SIFTPoints`, `SURFPoints`, `MSERRegions`, or `cornerPoints` object.

inlierPoints2 — Coordinates of corresponding points

`ORBPoints` | `BRISKPoints` | `SIFTPoints` | `SURFPoints` | `cornerPoints` | `MSERRegions` | *M*-by-2 matrix of [*x*,*y*] coordinates

Coordinates of corresponding points in image two, specified as an *M*-by-2 matrix of *M* number of [x y] coordinates, or as a `ORBPoints`, `BRISKPoints`, `SIFTPoints`, `SURFPoints`, `MSERRegions` , or `cornerPoints` object.

imagesize — Input image size

single | double | integer

Second input image size, specified as a double, single, or integer value and in the format returned by the `size` function. The size of input image 2 corresponds to `inlierPoints2`.

Output Arguments

T1 — Projective transformation one

3-by-3 matrix

Projective transformation, returned as a 3-by-3 matrix describing the projective transformations for input image `T1`.

T2 — Projective transformation two

3-by-3 matrix

Projective transformation, returned as a 3-by-3 matrix describing the projective transformations for input image `T2`.

Tips

- An epipole may be located in the first image or the second image. Applying the output uncalibrated rectification of `T1` (or `T2`) to image `1` (or image `2`) may result in an undesired distortion. You can check for an epipole within an image by applying the `isEpipoleInImage` function.

Version History

Introduced in R2012b

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `estimateUncalibratedRectification` function uses the postmultiply convention. Although there are no plans to remove `estimateUncalibratedRectification` at this time, you can streamline your geometric transformation workflows by switching to the `estimateStereoRectification` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `estimateUncalibratedRectification` to `estimateStereoRectification`.
- The `estimateStereoRectification` function returns the projective transformations as `projtform2d` objects, which use the premultiply convention. If you need to obtain the projective transformation matrices, then you can query the `A` property of the `projtform2d` objects. Note that the values of `A` are the transpose of `T1` and `T2`.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[2] Pollefeys, M., Koch, R., and Van Gool, L.. *A Simple and Efficient Rectification Method for General Motion*. Proceedings of the Seventh IEEE International Conference on Computer Vision. Volume 1, pages 496-501. 1999. DOI:10.1109/ICCV.1999.791262.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps

Stereo Camera Calibrator | Camera Calibrator

Objects

`MSERRegions` | `SIFTPoints` | `SURFPoints` | `BRISKPoints` | `ORBPoints` | `cornerPoints` | `cameraParameters` | `stereoParameters`

Functions

`detectHarrisFeatures` | `detectSIFTFeatures` | `detectMinEigenFeatures` | `detectHarrisFeatures` | `estimateFundamentalMatrix` | `extractFeatures` | `isEpipoleInImage` | `matchFeatures` | `imwarp` | `reconstructScene` | `estimateCameraParameters` | `cameraProjection` | `undistortImage`

Topics

“Uncalibrated Stereo Image Rectification”
“Point Feature Types”
“Coordinate Systems”

eul

Convert transformation or rotation into Euler angles

Syntax

```
angles = eul(transformation)
angles = eul(rotation)
angles = eul( ___ ,sequence)
```

Description

`angles = eul(transformation)` converts the rotation of the transformation `transformation` to the Euler angles `angles`.

`angles = eul(rotation)` converts the rotation `rotation` to the Euler angles `angles`.

`angles = eul( ___ ,sequence)` specifies the sequence of the Euler-angle rotations `sequence` using any of the input arguments in previous syntaxes. For example, a sequence of `"ZYX"` first rotates the *z*-axis, followed by the *y*-axis and *x*-axis.

Examples

Convert SE(3) Transformation to Euler Angles

Create SE(3) transformation with no translation but with a rotation defined by a Euler angles.

```
eul1 = [pi/4 pi/3 pi/8]
```

```
eul1 = 1×3

    0.7854    1.0472    0.3927
```

```
T = se3(eul1,"eul")
```

```
T = se3
    0.3536   -0.4189    0.8364         0
    0.3536    0.8876    0.2952         0
   -0.8660    0.1913    0.4619         0
         0         0         0    1.0000
```

Get the Euler angles from the transformation.

```
eul2 = eul(T)
```

```
eul2 = 1×3

    0.7854    1.0472    0.3927
```

Convert SO(3) Rotation to Euler Angles

Create SO(3) rotation defined by a Euler angles.

```
eul1 = [pi/4 pi/3 pi/8]
```

```
eul1 = 1×3

    0.7854    1.0472    0.3927
```

```
R = so3(eul1,"eul")
```

```
R = so3
    0.3536   -0.4189    0.8364
    0.3536    0.8876    0.2952
   -0.8660    0.1913    0.4619
```

Get the Euler angles from the transformation.

```
eul2 = eul(R)
```

```
eul2 = 1×3

    0.7854    1.0472    0.3927
```

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of `se3` objects

Transformation, specified as an `se3` object or as an *N*-element array of `se3` objects. *N* is the total number of transformations.

`rotation` — Rotation

`so3` object | *N*-element array of `so3` objects

Rotation, specified as an `so3` object or as an *N*-element array of `so3` objects. *N* is the total number of rotations.

`sequence` — Axis-rotation sequence

`"ZYX"` (default) | `"ZYZ"` | `"ZXY"` | `"ZXZ"` | `"YXY"` | `"YZX"` | `"YXZ"` | `"YZY"` | `"XYX"` | `"XYZ"` | `"XZX"` | `"XZY"`

Axis-rotation sequence for the Euler angles, specified as one of these string scalars:

- `"ZYX"` (default)
- `"ZYZ"`
- `"ZXY"`
- `"ZXZ"`

- `"YXY"`
- `"YZX"`
- `"YXZ"`
- `"YZY"`
- `"XYX"`
- `"XYZ"`
- `"XZX"`
- `"XZY"`

Each character indicates the corresponding axis. For example, if the sequence is `"ZYX"`, then the three specified Euler angles are interpreted in order as a rotation around the *z*-axis, a rotation around the *y*-axis, and a rotation around the *x*-axis. When applying this rotation to a point, it will apply the axis rotations in the order *x*, then *y*, then *z*.

Data Types: `string` | `char`

Output Arguments

`angles` — Euler angles

M-by-3 matrix

Euler angles, returned as an *M*-by-3 matrix of Euler rotation angles. Each row represents one Euler angle set.

See Also

`se3` | `so3`

evaluateDetectionMissRate

(To be removed) Evaluate miss rate metric for object detection

Note `evaluateDetectionMissRate` will be removed in a future release. Use `evaluateObjectDetection` instead.

Syntax

```
logAverageMissRate = evaluateDetectionMissRate(detectionResults,
groundTruthData)
[logAverageMissRate,fppi,missRate] = evaluateDetectionMissRate( ___ )
[ ___ ] = evaluateDetectionMissRate( ___ ,threshold)
```

Description

`logAverageMissRate = evaluateDetectionMissRate(detectionResults, groundTruthData)` returns the log-average miss rate of the `detectionResults` compared to `groundTruthData`, which is used to measure the performance of the object detector. For a multiclass detector, the log-average miss rate is a vector of scores for each object class in the order specified by `groundTruthData`.

`[logAverageMissRate,fppi,missRate] = evaluateDetectionMissRate( ___ )` returns data points for plotting the log miss rate–false positives per image (FPPI) curve, using input arguments from the previous syntax.

`[ ___ ] = evaluateDetectionMissRate( ___ ,threshold)` specifies the overlap threshold for assigning a detection to a ground truth box.

Input Arguments

`detectionResults` — Object locations and scores

table

Object locations and scores, specified as a two-column table containing the bounding boxes and scores for each detected object. For multiclass detection, a third column contains the predicted label for each detection. The bounding boxes must be stored in an *M*-by-4 cell array. The scores must be stored in an *M*-by-1 cell array, and the labels must be stored as a categorical vector.

When detecting objects, you can create the detection results table by using `imageDatastore`.

```
ds = imageDatastore(stopSigns.imageFilename);
detectionResults = detect(detector,ds);
```

Data Types: `table`

`groundTruthData` — Labeled ground truth

datastore | table

Labeled ground truth, specified as a datastore or a table.

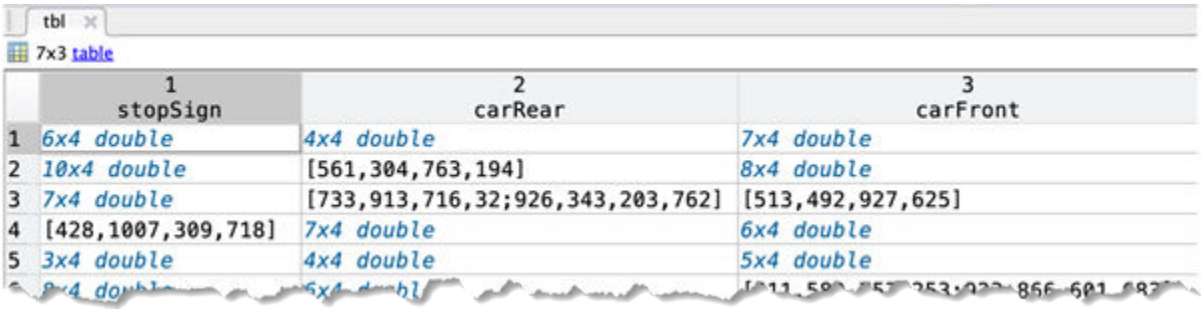
Each bounding box must be in the format [*x y width height*].

- Datastore — A datastore whose `read` and `readall` functions return a cell array or a table with at least two columns of bounding box and labels cell vectors. The bounding boxes must be in a cell array of *M*-by-4 matrices in the format [*x*,*y*,*width*,*height*]. The datastore's `read` and `readall` functions must return one of the formats:

  - {*boxes*,*labels*} — The `boxLabelDatastore` creates this type of datastore.
  - {*images*,*boxes*,*labels*} — A combined datastore. For example, using `combine(imds,blds)`.

  See `boxLabelDatastore`.

- Table — One or more columns. All columns contain bounding boxes. Each column must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *stopSign*, *carRear*, or *carFront*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image.


tbl

7x3 table

	1 stopSign	2 carRear	3 carFront
1	6x4 double	4x4 double	7x4 double
2	10x4 double	[561,304,763,194]	8x4 double
3	7x4 double	[733,913,716,32;926,343,203,762]	[513,492,927,625]
4	[428,1007,309,718]	7x4 double	6x4 double
5	3x4 double	4x4 double	5x4 double

`threshold` — Overlap threshold
0.5 | numeric scalar

Overlap threshold for a detection assigned to a ground truth box, specified as a numeric scalar. The overlap ratio is computed as the intersection over union.

Output Arguments

`logAverageMissRate` — Log-average miss rate metric
numeric scalar | vector

Log-average miss rate metric, returned as either a numeric scalar or vector. For a multiclass detector, the log-average miss rate is returned as a vector of values that correspond to the data points for each class.

`fppi` — False positives per image
vector of numeric scalars | cell array

False positives per image, returned as either a vector of numeric scalars or as a cell array. For a multiclass detector, the FPPI and log miss rate are cell arrays, where each cell contains the data points for each object class.

`missRate` — Log miss rate
vector of numeric scalars | cell array

Log miss rate, returned as either a vector of numeric scalars or as a cell array. For a multiclass detector, the FPPI and log miss rate are cell arrays, where each cell contains the data points for each object class.

Version History

Introduced in R2017a

R2023b: `evaluateDetectionMissRate` will be removed in a future release
Not recommended starting in R2023b

The `evaluateDetectionMissRate` function will be removed in a future release. Use the `evaluateObjectDetection` function to evaluate object detection results with metrics such as the log-average miss rate, instead.

See Also

Apps
Video Labeler | Image Labeler

Functions
`evaluateObjectDetection` | `evaluateDetectionPrecision` | `trainACFObjectDetector`

Objects
`boxLabelDatastore` | `acfObjectDetector`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)

evaluateDetectionPrecision

(To be removed) Evaluate precision metric for object detection

Note Use `evaluateObjectDetection` instead of `evaluateDetectionPrecision`, which will be removed in a future release. The more recent `evaluateObjectDetection` can be used to perform a comprehensive analysis of object detector performance.

Syntax

```
averagePrecision = evaluateDetectionPrecision(detectionResults,
groundTruthData)
[averagePrecision,recall,precision] = evaluateDetectionPrecision( ___ )
[ ___ ] = evaluateDetectionPrecision( ___ ,threshold)
```

Description

`averagePrecision = evaluateDetectionPrecision(detectionResults, groundTruthData)` returns the average precision, of the `detectionResults` compared to the `groundTruthData`. You can use the average precision to measure the performance of an object detector. For a multiclass detector, the function returns `averagePrecision` as a vector of scores for each object class in the order specified by `groundTruthData`.

`[averagePrecision,recall,precision] = evaluateDetectionPrecision( ___ )` returns data points for plotting the precision–recall curve, using input arguments from the previous syntax.

`[ ___ ] = evaluateDetectionPrecision( ___ ,threshold)` specifies the overlap threshold for assigning a detection to a ground truth box.

Input Arguments

`detectionResults` — Object locations and scores

table

Object locations and scores, specified as a two-column table containing the bounding boxes and scores for each detected object. For multiclass detection, a third column contains the predicted label for each detection. The bounding boxes must be stored in an *M*-by-4 cell array. The scores must be stored in an *M*-by-1 cell array, and the labels must be stored as a categorical vector.

When detecting objects, you can create the detection results table by using `imageDatastore`.

```
ds = imageDatastore(stopSigns.imageFilename);
detectionResults = detect(detector,ds);
```

Data Types: `table`

`groundTruthData` — Labeled ground truth

datastore | table

Labeled ground truth, specified as a datastore or a table.

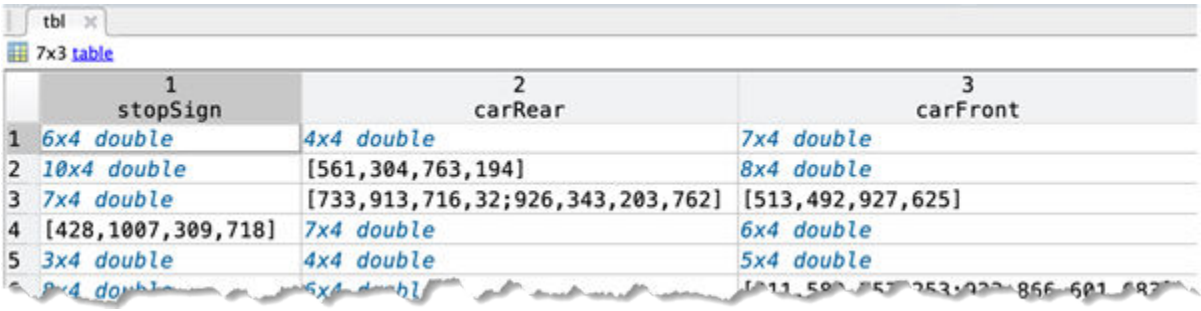
Each bounding box must be in the format [*x y width height*].

- Datastore — A datastore whose `read` and `readall` functions return a cell array or a table with at least two columns of bounding box and labels cell vectors. The bounding boxes must be in a cell array of *M*-by-4 matrices in the format [*x*,*y*,*width*,*height*]. The datastore's `read` and `readall` functions must return one of the formats:

  - {*boxes*,*labels*} — The `boxLabelDatastore` creates this type of datastore.
  - {*images*,*boxes*,*labels*} — A combined datastore. For example, using `combine(imds,blds)`.

  See `boxLabelDatastore`.

- Table — One or more columns. All columns contain bounding boxes. Each column must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *stopSign*, *carRear*, or *carFront*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image.


tbl

7x3 table

	1 stopSign	2 carRear	3 carFront
1	6x4 double	4x4 double	7x4 double
2	10x4 double	[561,304,763,194]	8x4 double
3	7x4 double	[733,913,716,32;926,343,203,762]	[513,492,927,625]
4	[428,1007,309,718]	7x4 double	6x4 double
5	3x4 double	4x4 double	5x4 double

`threshold` — Overlap threshold
`0.5` (default) | numeric scalar

Overlap threshold for assigned a detection to a ground truth box, specified as a numeric scalar. The overlap ratio is computed as the intersection over union.

Output Arguments

`averagePrecision` — Average precision
numeric scalar | vector

Average precision over all the detection results, returned as a numeric scalar or vector. Precision is a ratio of true positive instances to all positive instances of objects in the detector, based on the ground truth. For a multiclass detector, the average precision is a vector of average precision scores for each object class.

`recall` — Recall values from each detection
vector of numeric scalars | cell array

Recall values from each detection, returned as an *M*-by-1 vector of numeric scalars or as a cell array. The length of *M* equals 1 + the number of detections assigned to a class. For example, if your detection results contain 4 detections with class label `'car'`, then `recall` contains 5 elements. The first value of recall is always `0`.

Recall is a ratio of true positive instances to the sum of true positives and false negatives in the detector, based on the ground truth. For a multiclass detector, `recall` and `precision` are cell arrays, where each cell contains the data points for each object class.

`precision` — Precision values from each detection
vector of numeric scalars | cell array

Precision values from each detection, returned as an *M*-by-1 vector of numeric scalars or as a cell array. The length of *M* equals 1 + the number of detections assigned to a class. For example, if your detection results contain 4 detections with class label `'car'`, then `precision` contains 5 elements. The first value of `precision` is always `1`.

Precision is a ratio of true positive instances to all positive instances of objects in the detector, based on the ground truth. For a multi-class detector, `recall` and `precision` are cell arrays, where each cell contains the data points for each object class.

Version History

Introduced in R2017a

R2023b: `evaluateDetectionPrecision` will be removed in a future release
Not recommended starting in R2023b

Use `evaluateObjectDetection` instead of `evaluateDetectionPrecision`, which will be removed in a future release. The more recent `evaluateObjectDetection` can be used to perform a comprehensive analysis of object detector performance.

See Also

Apps
Image Labeler | Video Labeler

Functions
`evaluateDetectionMissRate` | `evaluateObjectDetection` | `trainACFObjectDetector`

Objects
`boxLabelDatastore` | `acfObjectDetector`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)

evaluateImageRetrieval

Evaluate image search results

Syntax

```
averagePrecision = evaluateImageRetrieval(queryImage,imageIndex,expectedIDs)
[averagePrecision,imageIDs,scores] = evaluateImageRetrieval(queryImage,
imageIndex,expectedIDs)
[averagePrecision,imageIDs,scores] = evaluateImageRetrieval( ___ ,Name,Value)
```

Description

`averagePrecision = evaluateImageRetrieval(queryImage,imageIndex,expectedIDs)` returns the average precision metric for measuring the accuracy of image search results for the `queryImage`. The `expectedIDs` input contains the indices of images within `imageIndex` that are known to be similar to the query image.

`[averagePrecision,imageIDs,scores] = evaluateImageRetrieval(queryImage, imageIndex,expectedIDs)` optionally returns the indices corresponding to images within `imageIndex` that are visually similar to the query image. It also returns the corresponding similarity scores.

`[averagePrecision,imageIDs,scores] = evaluateImageRetrieval( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

Examples

Evaluate Image Retrieval Results

Define a set of images.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata','bookCovers');
bookCovers = imageDatastore(dataDir);
```

Display the set of images.

```
thumbnailGallery = [];
for i = 1:length(bookCovers.Files)
    img = readimage(bookCovers,i);
    thumbnail = imresize(img,[300 300]);
    thumbnailGallery = cat(4,thumbnailGallery,thumbnail);
end
figure
montage(thumbnailGallery);
```



Index the images. This will take a few minutes.

```
imageIndex = indexImages(bookCovers);
```

```
Creating an inverted image index using Bag-Of-Features.
-------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 58 images...done. Extracted 29216 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 23373.
** Using the strongest 23373 features from each of the other image categories.

* Creating a 20000 word visual vocabulary.
* Number of levels: 1
* Branching factor: 20000
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 23373
* Number of clusters          : 20000
* Initializing cluster centers...100.00%.
* Clustering...completed 5/100 iterations (~1.23 seconds/iteration)...converged in 5 iterations.
```

```
* Finished creating Bag-Of-Features

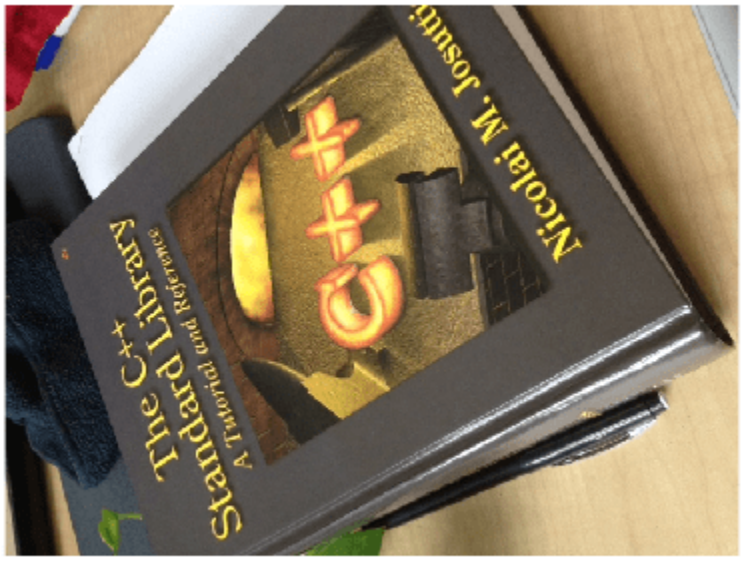

Encoding images using Bag-Of-Features.
--------------------------------------

* Encoding 58 images...done.
Finished creating the image index.
```

Select and display the query image.

```
queryDir = fullfile(dataDir,'queries',filesep);
query = imread([queryDir 'query2.jpg']);

figure
imshow(query)
```



Evaluation requires knowing the expected results. Here, the query image is known to be the 3rd book in the imageIndex.

```
expectedID = 3;
```

Find and report the average precision score.

```
[averagePrecision,actualIDs] = evaluateImageRetrieval(query,...
    imageIndex,expectedID);

fprintf('Average Precision: %f\n\n',averagePrecision)
```

```
Average Precision: 0.125000
```

Show the query and best match side-by-side.

```
bestMatch = actualIDs(1);
bestImage = imread(imageIndex.ImageLocation{bestMatch});
```

```
figure
imshowpair(query,bestImage,'montage')
```



Compute Mean Average Precision (MAP) for Image Retrieval

Create an image set of book covers.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata','bookCovers');
bookCovers = imageDatastore(dataDir);
```

Index the image set. The indexing may take a few minutes.

```
imageIndex = indexImages(bookCovers,'Verbose',false);
```

Create a set of query images.

```
queryDir = fullfile(dataDir,'queries',filesep);
querySet = imageDatastore(queryDir);
```

Specify the expected search results for each query image.

```
expectedIDs = [1 2 3];
```

Evaluate each query image and collect average precision scores.

```
for i = 1:numel(querySet.Files)
    query = readimage(querySet,i);
    averagePrecision(i) = evaluateImageRetrieval(query, imageIndex, expectedIDs(i));
end
```

Compute mean average precision (MAP).

```
  map = mean(averagePrecision)

map =
0.8333
```

Input Arguments

queryImage — Input query image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input query image, specified as either an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-D grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

imageIndex — Image search index

`invertedImageIndex` object

Image search index, specified as an `invertedImageIndex` object. The `indexImages` function creates the `invertedImageIndex` object, which stores the data used for the image search.

expectedIDs — Image indices

row or column vector

Image indices, specified as a row or column vector. The indices correspond to the images within `imageIndex` that are known to be similar to the query image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumResults',25`

NumResults — Maximum number of search results to evaluate

`Inf` (default) | positive integer value

Maximum number of search results to evaluate, specified as the comma-separated pair consisting of '`NumResults`' and a positive integer value. The function evaluates the top `NumResults` and returns the average-precision-at-`NumResults` metric.

ROI — Rectangular search region

`[1 1 size(queryImage,2) size(queryImage,1)]` (default) | [*x y width height*] vector

Rectangular search region within the query image, specified as the comma-separated pair consisting of 'ROI' and a [*x y width height*] formatted vector.

Output Arguments

averagePrecision — Average precision metric

scalar value in the range [0 1]

Average precision metric, returned as a scalar value in the range [0 1]. The average precision metric represents the accuracy of image search results for the query image.

`imageIDs` — Ranked index of retrieved images
M-by-1 vector

Ranked index of retrieved images, returned as an *M*-by-1 vector. The image IDs are returned in ranked order, from the most to least similar matched image.

`scores` — Similarity metric
N-by-1 vector

Similarity metric, returned as an *N*-by-1 vector. This output contains the scores that correspond to the retrieved images in the `imageIDs` output. The scores are computed using the cosine similarity and range from 0 to 1.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`retrieveImages` | `indexImages` | `imageDatastore` | `bagOfFeatures` | `invertedImageIndex`

Topics
“Image Retrieval Using Customized Bag of Features”
“Image Retrieval with Bag of Visual Words”

evaluateInstanceSegmentation

Evaluate instance segmentation data set against ground truth

Syntax

```
metrics = evaluateInstanceSegmentation(dsResults,dsTruth)
metrics = evaluateInstanceSegmentation(dsResults,dsTruth,threshold)
metrics = evaluateInstanceSegmentation( ___ ,Verbose=verbose)
```

Description

`metrics = evaluateInstanceSegmentation(dsResults,dsTruth)` calculates various metrics to evaluate the quality of the instance segmentation results, `dsResults`, against the ground truth segmentation, `dsTruth`.

`metrics = evaluateInstanceSegmentation(dsResults,dsTruth,threshold)` also specifies an overlap threshold for the intersection over union (IoU) metric, or Jaccard Index. When the IoU of the predicted object mask in `dsResults` and the ground truth object mask in `dsTruth` is equal to or greater than the overlap threshold, the prediction is considered a true positive.

`metrics = evaluateInstanceSegmentation( ___ ,Verbose=verbose)` also specifies whether to display evaluation progress information in the command window.

Input Arguments

`dsResults` — Predicted instance segmentation

datastore

Predicted instance segmentation, specified as a datastore. The data must be set up so that calling the datastore with the `read` function returns a cell array with at least three elements. The table describes the format of the three required elements.

masks	*labels*	*scores*
Predicted binary masks, specified as a logical array of size *H*-by-*W*-by-*M*. Each mask is the segmentation of one instance in the image.	Predicted object labels, specified as an *M*-by-1 categorical vector.	Predicted scores, specified as an *M*-by-1 numeric vector.

The order of the elements does not matter. When the datastore returns a cell array with more than three elements, the `evaluateInstanceSegmentation` function assumes that the first element with logical data contains the instance masks, the first element with categorical data contains the label data, and the first element with numeric data contains the scores.

You can get predicted instance segmentations by using the `segmentObjects` function.

`dsTruth` — Ground truth instance segmentation

datastore

Ground truth instance segmentation, specified as a datastore. The data must be set up so that calling the datastore with the `read` function returns a cell array with at least two elements. The table describes the format of the two required elements.

masks	***labels***
Ground truth binary masks, specified as a logical array of size *H*-by-*W*-by-*M*. Each mask is the segmentation of one instance in the image.	Ground truth object labels, specified as an *M*-by-1 categorical vector.

The order of the elements does not matter. When the datastore returns a cell array with more than two elements, the `evaluateInstanceSegmentation` function assumes that the first element with logical data contains the instance masks, and the first element with categorical data contains the label data.

`threshold` — Overlap threshold
`0.5` (default) | numeric scalar | numeric vector

Overlap threshold for a predicted object mask to be considered a true positive, specified as a numeric scalar or numeric vector. When you specify a numeric vector, the `evaluateInstanceSegmentation` function calculates metrics for each threshold.

`verbose` — Display evaluation progress
`1` (default) | `0`

Display evaluation progress information in the command window, specified as a numeric or logical `1` (`true`) or `0` (`false`). The displayed information includes a progress bar, elapsed time, estimated time remaining, and data set metrics.

Data Types: `logical`

Output Arguments

`metrics` — Instance segmentation metrics
`instanceSegmentationMetrics` object

Instance segmentation metrics, returned as an `instanceSegmentationMetrics` object. The `OverlapThreshold` property of the object is set as the value of the `threshold` argument.

Version History

Introduced in R2022b

See Also

`solov2` | `trainSOLOV2` | `instanceSegmentationMetrics` | `maskrcnn`

Topics
“Get Started with SOLOv2 for Instance Segmentation”
“Perform Instance Segmentation Using SOLOv2”
“Getting Started with Mask R-CNN for Instance Segmentation”
“Perform Instance Segmentation Using Mask R-CNN”
“Get Started with Instance Segmentation Using Deep Learning”

External Websites
https://cocodataset.org/#detection-eval

evaluateSemanticSegmentation

Evaluate semantic segmentation data set against ground truth

Syntax

```
ssm = evaluateSemanticSegmentation(dsResults,dsTruth)
ssm = evaluateSemanticSegmentation(imageSetConfusion,classNames)
[ssm,blockMetrics] = evaluateSemanticSegmentation(blockSetConfusion,
classNames)
[___] = evaluateSemanticSegmentation(___,Name,Value)
```

Description

`ssm = evaluateSemanticSegmentation(dsResults,dsTruth)` computes various metrics to evaluate the quality of the semantic segmentation results, `dsResults`, against the ground truth segmentation, `dsTruth`.

`ssm = evaluateSemanticSegmentation(imageSetConfusion,classNames)` computes various metrics to evaluate the quality of the semantic segmentation results from confusion matrices, `imageSetConfusion`, with segmentation classes `classNames`.

`[ssm,blockMetrics] = evaluateSemanticSegmentation(blockSetConfusion, classNames)` computes various metrics to evaluate the quality of the block-based semantic segmentation results from confusion matrices, `blockSetConfusion` with classes `classNames`.

`[___] = evaluateSemanticSegmentation(___,Name,Value)` computes semantic segmentation metrics using one or more name-value arguments to control the evaluation, in addition to output arguments from the previous syntaxes.

Examples

Evaluate Semantic Segmentation Results

The `triangleImages` data set has 100 test images with ground truth labels. Define the location of the data set.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
```

Define the location of the test images and ground truth labels.

```
testImagesDir = fullfile(dataSetDir,"testImages");
testLabelsDir = fullfile(dataSetDir,"testLabels");
```

Create an `imageDatastore` holding the test images.

```
imds = imageDatastore(testImagesDir);
```

Define the class names and their associated label IDs.

```
classNames = ["triangle" "background"];
labelIDs   = [255 0];
```

Create a `pixelLabelDatastore` holding the ground truth pixel labels for the test images.

```
pxdsTruth = pixelLabelDatastore(testLabelsDir,classNames,labelIDs);
```

Load a semantic segmentation network that has been trained on the training images of `triangleImages`.

```
net = load("triangleSegmentationNetwork");
net = net.net;
```

Run the network on the test images. Predicted labels are written to disk in a temporary directory and returned as a `pixelLabelDatastore`.

```
pxdsResults = semanticseg(imds,net,Classes=classNames,WriteLocation=tempdir);
```

```
Running semantic segmentation network
-------------------------------------
* Processed 100 images.
```

Evaluate the prediction results against the ground truth.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth);
```

```
Evaluating semantic segmentation results
----------------------------------------
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 100 images.
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

       0.99074          0.99183       0.91118      0.98299        0.80563
```

Display the properties of the `semanticSegmentationMetrics` object.

```
metrics
```

```
metrics =
  semanticSegmentationMetrics with properties:

             ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
            DataSetMetrics: [1×5 table]
              ClassMetrics: [2×3 table]
              ImageMetrics: [100×5 table]
```

Display the classification accuracy, the intersection over union, and the boundary F-1 score for each class. These values are stored in the `ClassMetrics` property.

```
metrics.ClassMetrics
```

```
ans=2×3 table
                  Accuracy      IoU      MeanBFScore
```

```
                ________    _______    ___________

    triangle      0.99302     0.83206      0.67208
    background    0.99063      0.9903      0.93918
```

Display the normalized confusion matrix that is stored in the `NormalizedConfusionMatrix` property.

```
metrics.ConfusionMatrix
```

```
ans=2×2 table
                  triangle    background
                  ________    __________

    triangle         4697           33
    background        915         96755
```

Input Arguments

`dsResults` — Predicted pixel labels

datastore | cell array of datastore objects

Predicted pixel labels resulting from semantic segmentation, specified as a datastore or a cell array of datastore objects. `dsResults` can be any datastore that returns categorical images, such as `PixelLabelDatastore`. The `read(dsResults)` must return a categorical array, a cell array, or a table. If the `read` function returns a multicolumn cell array or table, the second column must contain categorical arrays.

`dsTruth` — Ground truth pixel labels

datastore | cell array of datastore objects

Ground truth pixel labels, specified as a datastore or a cell array of datastore objects. `dsTruth` can be any datastore that returns categorical images, such as `PixelLabelDatastore`. Using `read(dsTruth)` must return a categorical array, a cell array, or a table. If the `read` function returns a multicolumn cell array or table, the second column must contain categorical arrays.

`imageSetConfusion` — Confusion matrices for segmented images

table | cell array

Confusion matrix for the classes in the segmented images, specified as one of the following, where *F* is the number of images in the data set.

- Table with *F* rows and one variable with the name `ConfusionMatrix`. Each row in the table contains a cell array with the confusion matrix for the corresponding image.
- *F*-by-one cell array. Each element of the cell array contains the confusion matrix for the corresponding image.

`blockSetConfusion` — Confusion matrices for segmented blocks

table

Confusion matrices for segmented blocks, specified as a table with *B* rows and three columns, where *B* is the total number of blocks in all images in the data set. The three columns are the variables

`ImageNumber`, `ConfusionMatrix`, and `BlockInfo`. You can obtain a table of the correct format by using the `segmentationConfusionMatrix` function within a call to the block-based `apply` function. For an example, see “Calculate Segmentation Metrics in Block-Based Workflow”.

`classNames` — Class names
array of strings | cell array of character vectors

Class names, specified as an array of strings or a cell array of character vectors.

Example: `["sky" "grass" "building" "sidewalk"]`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth,Metrics="bfscore")` computes only the mean BF score of each class, each image, and the entire data set.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

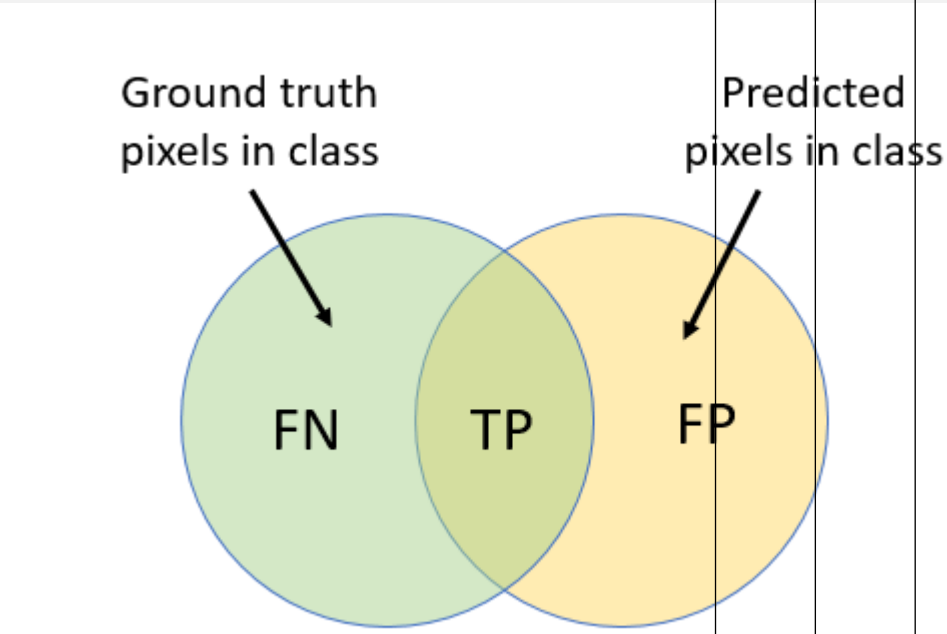
Example: `metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth,"Metrics","bfscore")` computes only the mean BF score of each class, each image, and the entire data set.

`Metrics` — Segmentation metrics
`"all"` (default) | vector of strings

Segmentation metrics in `semanticSegmentationMetrics` to compute, specified as a vector of strings. This argument specifies which variables in the `DataSetMetrics`, `ClassMetrics`, and `ImageMetrics` tables to compute. `ConfusionMatrix` and `NormalizedConfusionMatrix` are computed regardless of the value of `Metrics`.

Value	Description	Aggregate Data Set Metric	Image Metric	Class Metric
`"all"`	Evaluate all semantic segmentation metrics. The function excludes `MeanBFScore` from the semantic segmentation metrics when you specify a confusion matrix (`imageSetConfusion` or `blockSetConfusion`) as input to the function.	All aggregate data set metrics	All image metrics	All class metrics

Value	Description	Aggregate Data Set Metric	Image Metric	Class Metric
`"accuracy"`	Accuracy indicates the percentage of correctly identified pixels for each class. Use the accuracy metric if you want to know how well each class correctly identifies pixels. • For each class, `Accuracy` is the ratio of correctly classified pixels to the total number of pixels in that class, according to the ground truth. In other words, Accuracy score = TP / (TP + FN) TP is the number of true positives and FN is the number of false negatives. • For the aggregate data set, `MeanAccuracy` is the average `Accuracy` of all classes in all images. • For each image, `MeanAccuracy` is the average `Accuracy` of all classes in that particular image. The class accuracy is a simple metric analogous to global accuracy, but it can be misleading. For example, labeling all pixels "car" gives a perfect score for the "car" class (although not for the other classes). Use class accuracy in conjunction with IoU for a more complete evaluation of segmentation results.	`MeanAccuracy`	`MeanAccuracy`	`Accuracy`
`"bfscore"`	The boundary F1 (BF) contour matching score indicates how well the predicted boundary of each class aligns with the true boundary. Use the BF score if you want a metric that tends to correlate better with human qualitative assessment than the IoU metric. • For each class, `MeanBFScore` is the average BF score of that class over all images. • For each image, `MeanBFScore` is the average BF score of all classes in that particular image. • For the aggregate data set, `MeanBFScore` is the average BF score of all classes in all images. For more information, see `bfscore`. This metric is not available when you specify a confusion matrix (`imageSetConfusion` or `blockSetConfusion`) as input to the function.	`MeanBFScore`	`MeanBFScore`	`MeanBFScore`
`"global-accuracy"`	`GlobalAccuracy` is the ratio of correctly classified pixels, regardless of class, to the total number of pixels. Use the global accuracy metric if you want a quick and computationally inexpensive estimate of the percentage of correctly classified pixels.	`GlobalAccuracy`	`GlobalAccuracy`	none

Value	Description	Aggregate Data Set Metric	Image Metric	Class Metric
`"iou"`	Intersection over union (IoU), also known as the Jaccard similarity coefficient, is the most commonly used metric. Use the IoU metric if you want a statistical accuracy measurement that penalizes false positives. • For each class, `IoU` is the ratio of correctly classified pixels to the total number of ground truth and predicted pixels in that class. In other words, IoU score = TP / (TP + FP + FN) The image describes the true positives (TP), false positives (FP), and false negatives (FN).  • For each image, `MeanIoU` is the average IoU score of all classes in that particular image. • For the aggregate data set, `MeanIoU` is the average IoU score of all classes in all images. For more information, see `jaccard`.	`MeanIoU`	`MeanIoU`	`IoU`
`"weighted-iou"`	Average IoU of each class, weighted by the number of pixels in that class. Use this metric if images have disproportionally sized classes, to reduce the impact of errors in the small classes on the aggregate quality score.	`WeightedIoU`	`WeightedIoU`	none

Example: `["global-accuracy","iou"]` calculates the global accuracy and IoU metrics across the data set, images, and classes.

Data Types: `string`

`Verbose` — Flag to display evaluation progress
`1` (default) | `0`

Flag to display evaluation progress information in the command window, specified as `1` (`true`) or `0` (`false`).

The displayed information includes a progress bar, elapsed time, estimated time remaining, and data set metrics. When `Verbose` is `0` (`false`), the `evaluateSemanticSegmentation` function calculates segmentation metrics without displaying progress information.

Data Types: `logical`

Output Arguments

`ssm` — Semantic segmentation metrics
`semanticSegmentationMetrics` object

Semantic segmentation metrics, returned as a `semanticSegmentationMetrics` object.

`blockMetrics` — Block-based semantic segmentation metrics
F-by-1 cell array

Block-based semantic segmentation metrics, returned as an *F*-by-1 cell array, where *F* is the number of images in the data set. Each element in the cell array contains information about all of the metrics calculated for all blocks in the corresponding image, formatted as a table.

Each table has *K*(*f*) rows, where *K*(*f*) is the number of blocks in the *f*th image in the data set. The table has up to five variables:

- The table always includes the `BlockInfo` variable, which corresponds to the `BlockInfo` column specified in the `blockSetConfusion` input argument. The table data in this variable are structs that provide spatial information about the block. The four fields of the struct are `BlockStartWorld`, `BlockEndWorld`, `DataStartWorld`, and `DataEndWorld`. For more information about these fields, see the `IncludeBlockInfo` name-value pair argument of the `apply` function.
- The table includes the metrics in the `DataSetMetrics` property of the `ssm` output argument. By default, the metrics are `GlobalAccuracy`, `MeanAccuracy`, `MeanIoU`, and `WeightedIoU`. However, if you create the `ssm` and specify a subset of the metrics to calculate by using the `Metrics` name-value argument, then the table includes only the specified metrics.

To return block-based semantic segmentation metrics, you must specify the `blockSetConfusion` and `classNames` input arguments.

Tips

- A value of `NaN` in the dataset, class, or image metrics, indicates that one or more classes were missing during the computation of the metrics when using the `evaluateSemanticSegmentation` function. In this case, the software was unable to accurately compute the metrics.

The missing classes can be found by looking at the `ClassMetrics` property, which provides the metrics for each class. To more accurately evaluate your network, augment your ground truth with more data that includes the missing classes.

Version History

Introduced in R2017b

References

[1] Csurka, G., D. Larlus, and F. Perronnin. "What is a good evaluation measure for semantic segmentation?" *Proceedings of the British Machine Vision Conference*, 2013, pp. 32.1–32.11.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

- To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

  For more information, see “Parallel Computing Toolbox Support”.
- Parallel processing is only supported when the input datastores are `pixelLabelDatastore` objects.

See Also

Functions

`semanticseg` | `plotconfusion` | `jaccard` | `bfscore` | `segmentationConfusionMatrix`

Objects

`semanticSegmentationMetrics` | `PixelLabelDatastore`

Topics

“Calculate Segmentation Metrics in Block-Based Workflow”
“Get Started with Semantic Segmentation Using Deep Learning”
“Datastores for Deep Learning” (Deep Learning Toolbox)

extractFeatures

Extract interest point descriptors

Syntax

```
[features,validPoints] = extractFeatures(I,points)
[features,validPoints] = extractFeatures(I,points,Name=Value)
```

Description

`[features,validPoints] = extractFeatures(I,points)` returns extracted feature vectors, also known as descriptors, and their corresponding locations, from a binary or intensity image.

The function derives the descriptors from pixels surrounding an interest point. The pixels represent and match features specified by a single-point location. Each single-point specifies the center location of a neighborhood. The method you use for descriptor extraction depends on the class of the input `points`.

`[features,validPoints] = extractFeatures(I,points,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `extractFeatures(I,points,Method="Block")` sets the method to extract descriptors to `Block`.

Examples

Extract Corner Features from an Image

Read the image.

```
I = imread("cameraman.tif");
```

Find and extract corner features from the image.

```
corners = detectHarrisFeatures(I);
[features,valid_corners] = extractFeatures(I,corners);
```

Display image.

```
figure;
imshow(I);
hold on
```

Plot valid corner points.

```
plot(valid_corners);
```



Extract SURF Features from an Image

Read image.

```
I = imread("cameraman.tif");
```

Find and extract features from the input image.

```
points = detectSURFFeatures(I);
[features,valid_points] = extractFeatures(I,points);
```

Display and plot ten strongest SURF features.

```
figure;
imshow(I);
hold on;
plot(valid_points.selectStrongest(10),showOrientation=true);
```



Extract MSER Features from an Image

Read image.

```
I = imread("cameraman.tif");
```

Find features in the image using MSER feature detection.

```
regions = detectMSERFeatures(I);
[features,valid_points] = extractFeatures(I,regions,Upright=true);
```

Display features corresponding to the MSER ellipse centers.

```
figure;
imshow(I);
hold on;
plot(valid_points,showOrientation=true);
```



Input Arguments

`I` — Input image

binary image | *M*-by-*N* 2-D grayscale image

Input image, specified as either a binary or 2-D grayscale image.

Data Types: `logical` | `single` | `double` | `int16` | `uint8` | `uint16`

`points` — Center location point

M-by-2 matrix | point feature object

Center location point of a square neighborhood,specified as an *M*-by-2 matrix of *M* number of [*x y*] coordinates, or as one of the point feature objects described in “Point Feature Types”. The table lists the possible input classes of points that can be used for extraction.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `extractFeatures(I,points,Method="Block")` sets the method to extract descriptors to `Block`.

`Method` — Descriptor extraction method

`"Auto"` | point feature object | *M*-by-2 matrix

Descriptor extraction method, specified as one of the point feature objects described in “Point Feature Types” or `"Auto"`.

The table describes how the function implements the descriptor extraction methods:

Method	Feature Vector (Descriptor)
`SIFT`	Scale-Invariant Feature Transform (SIFT). The function sets the `Orientation` property of the `validPoints` output object to the orientation of the extracted features, in radians.
`BRISK`	Binary Robust Invariant Scalable Keypoints (BRISK). The function sets the `Orientation` property of the `validPoints` output object to the orientation of the extracted features, in radians.
`FREAK`	Fast Retina Keypoint (FREAK). The function sets the `Orientation` property of the `validPoints` output object to the orientation of the extracted features, in radians.
`SURF`	Speeded-Up Robust Features (SURF). The function sets the `Orientation` property of the `validPoints` output object to the orientation of the extracted features, in radians. When you use an `MSERRegions` object with the `SURF` method, the `Centroid` property of the object extracts SURF descriptors. The `Axes` property of the object selects the scale of the SURF descriptors such that the circle representing the feature has an area proportional to the MSER ellipse area. The scale is calculated as `1/4*sqrt((majorAxes/2).*(minorAxes/2))` and saturated to `1.6`, as required by the `SURFPoints` object.
`ORB`	Oriented FAST and rotated BRIEF (ORB) features. The `Orientation` property of the `validPoints` output object is automatically set to the `Orientation` property of the input `ORBPoints` object `points`.
`KAZE`	Non-linear pyramid-based features. The function sets the `Orientation` property of the `validPoints` output object to the orientation of the extracted features, in radians. When you use an `MSERRegions` object with the `KAZE` method, the `Location` property of the object is used to extract KAZE descriptors. The `Axes` property of the object selects the scale of the KAZE descriptors such that the circle representing the feature has an area proportional to the MSER ellipse area.

Method	Feature Vector (Descriptor)
`Block`	Simple square neighborhood. The `Block` method extracts only the neighborhoods fully contained within the image boundary. Therefore, the output, `validPoints`, can contain fewer points than the input `POINTS`.
`Auto`	The function selects the `Method`, based on the class of the input points and implements: The `FREAK` method for a `cornerPoints` input object. The `SURF` method for a `SURFPoints` or `MSERRegions` input object. The `BRISK` method for a `BRISKPoints` input object. The `ORB` method for a `ORBPoints` input object. For an *M*-by-2 input matrix of [*x y*] coordinates, the function implements the `Block` method.

Note The descriptor extraction method must be `ORB`, if the input `points` is an `ORBPoints` object. Also, `ORB` descriptor extraction method is not supported for any other class of points, except `ORBPoints`.

`BlockSize` — Block size

`11` (default) | odd integer scalar

Block size, specified as an odd integer scalar. This value defines the local square neighborhood `BlockSize`-by-`BlockSize`, centered at each interest point. This option applies only when the function implements the `Block` method.

`Upright` — Rotation invariance flag

`false` | logical scalar

Rotation invariance flag, specified a logical scalar. When you set this property to `true`, the orientation of the feature vectors are not estimated and the feature orientation is set to pi/2. Set this to `true` when you do not need the image descriptors to capture rotation information. When you set this property to `false`, the orientation of the features is estimated and the features are then invariant to rotation.

Note The rotation invariance flag `"Upright"` is not supported if the input `points` is an `ORBPoints` object.

`FeatureSize` — Length of feature vector

`64` (default) | `128`

Length of the feature vector (descriptor), specified as `64` or `128`. This option applies only when the function implements the `SURF` or `KAZE` method. The larger feature size of `128` provides greater accuracy, but decreases the feature matching speed.

Output Arguments

`features` — Feature vectors

M-by-*N* matrix | `binaryFeatures` object

Feature vectors, returned as a `binaryFeatures` object or an *M*-by-*N* matrix of *M* feature vectors, also known as descriptors. Each descriptor is of length *N*.

`validPoints` — Valid points

M-by-2 matrix of [*x*,*y*] coordinates | point feature object

Valid points associated with each output feature vector (descriptor) in `features`, returned in the same format as the input. Valid points can be an *M*-by-2 matrix of [*x*,*y*] coordinates or as one of the point feature objects described in “Point Feature Types”.

The function extracts descriptors from a region around each interest point. If the region lies outside of the image, or if interest point lies too close to the edge of the image, the function cannot compute the feature descriptor. In this case, the function ignores the point. The point is not included in the valid points output.

For `SIFT` features, more than one descriptor can be extracted at a single location, but at different orientations. Therefore, the number of points returned in `validPoints` can be greater than or equal to the number of input points to `extractFeatures`. For input point features other than `SIFT` features, `validPoints` can be less than or equal to the number of input points.

Version History

Introduced in R2011a

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*, O'Reilly, Sebastopol, CA, 2008.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, *SURF: Speeded Up Robust Features*", *Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008

[3] Bay, Herbert, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "SURF: Speeded Up Robust Features", *Computer Vision and Image Understanding (CVIU)*, Vol. 110, No. 3, pp. 346--359, 2008.

[4] Alahi, Alexandre, Ortiz, Raphael, and Pierre Vandergheynst, "FREAK: Fast Retina Keypoint", *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[5] Alcantarilla, P.F., A. Bartoli, and A.J. Davison. "KAZE Features", *ECCV 2012, Part VI, LNCS 7577* pp. 214, 2012

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `'Method'` must be a compile-time constant.
- Supports MATLAB Function block for `BRISK`, `FREAK`, and `SURF` methods only.
- Generates portable C code using a C++ compiler that links to OpenCV libraries for `SIFT`, `BRISK`, `ORB`, `FREAK`, and `SURF` Methods. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- The `KAZE` method and the `detectKAZEFeatures` function do not support code generation.
- Code generation does not support arrays of point feature objects returned by this function. Instead, use the `Location` property to obtain an *M*-by-2 matrix of coordinates:

  ```
  locations = valid_points.Location;
  ```

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Topics
“Point Feature Types”
“Local Feature Detection and Extraction”

extractHOGFeatures

Extract histogram of oriented gradients (HOG) features

Syntax

```
features = extractHOGFeatures(I)
[features,validPoints] = extractHOGFeatures(I,points)
[___, visualization] = extractHOGFeatures(I,points)
[___] = extractHOGFeatures(___,Name,Value)
```

Description

`features = extractHOGFeatures(I)` returns extracted HOG features from a truecolor or grayscale input image, `I`. The features are returned in a 1-by-*N* vector, where *N* is the HOG feature length. The returned features encode local shape information from regions within an image. You can use this information for many tasks including classification, detection, and tracking.

`[features,validPoints] = extractHOGFeatures(I,points)` returns HOG features extracted around specified point locations. The function also returns `validPoints`, which contains the input point locations whose surrounding region is fully contained within `I`. Scale information associated with the points is ignored.

`[___, visualization] = extractHOGFeatures(I,points)` optionally returns a HOG feature visualization. You can display this visualization using `plot(visualization)`.

`[___] = extractHOGFeatures(___,Name,Value)` uses additional options specified by one or more Name,Value pair arguments, using any of the preceding syntaxes.

Examples

Extract and Plot HOG Features

Read the image of interest.

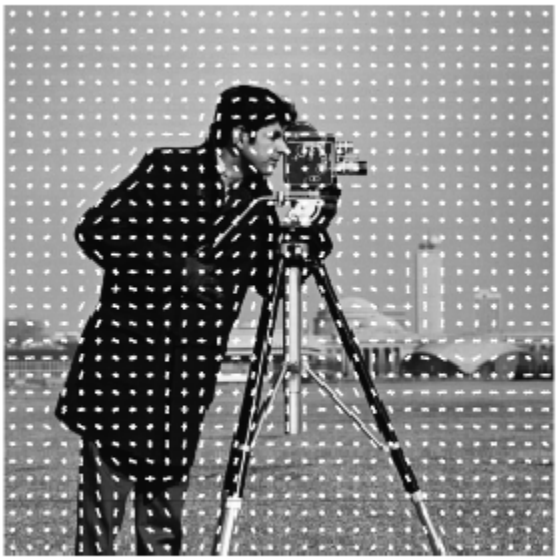
```
img = imread('cameraman.tif');
```

Extract HOG features.

```
[featureVector,hogVisualization] = extractHOGFeatures(img);
```

Plot HOG features over the original image.

```
figure;
imshow(img);
hold on;
plot(hogVisualization);
```



Extract HOG Features Using CellSize

Read the image of interest.

```
I1 = imread('gantrycrane.png');
```

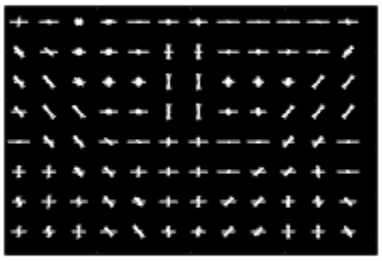
Extract HOG features.

```
[hog1,visualization] = extractHOGFeatures(I1,'CellSize',[32 32]);
```

Display the original image and the HOG features.

```
subplot(1,2,1);
imshow(I1);
subplot(1,2,2);
plot(visualization);
```





Extract HOG Features Around Corner Points

Read in the image of interest.

```
I2 = imread('gantrycrane.png');
```

Detect and select the strongest corners in the image.

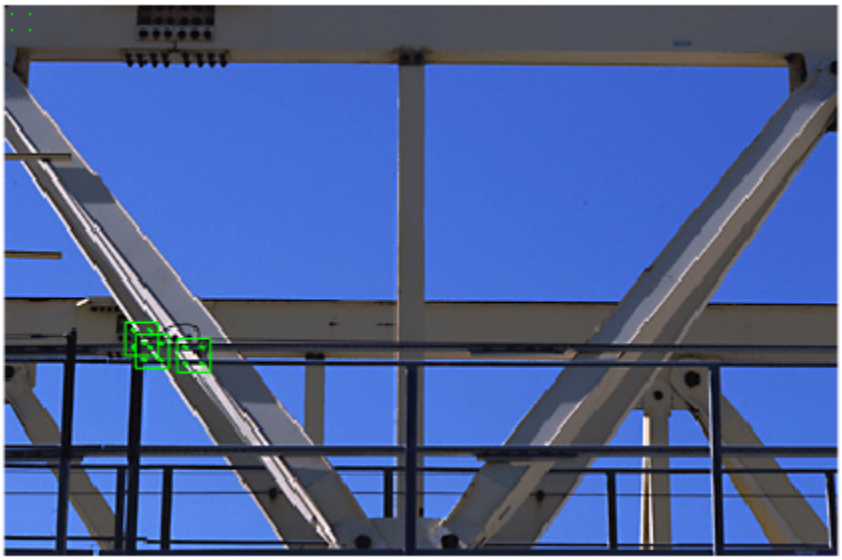
```
corners   = detectFASTFeatures(im2gray(I2));
strongest = selectStrongest(corners,3);
```

Extract HOG features.

```
[hog2,validPoints,ptVis] = extractHOGFeatures(I2,strongest);
```

Display the original image with an overlay of HOG features around the strongest corners.

```
figure;
imshow(I2);
hold on;
plot(ptVis,'Color','green');
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image, specified in either *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale. The input image must be a real, nonsparse value. If you have tightly cropped images, you may lose shape information that the HOG function can encode. You can avoid losing this information by including an extra margin of pixels around the patch that contains background pixels.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`points` — Center location point

`BRISKPoints` object | `cornerPoints` object | `SURFPoints` object | `MSERRegions` object | *M*-by-2 matrix of [*x*, *y*] coordinates

Center location point of a square neighborhood, specified as either a `BRISKPoints`, `SURFPoints`, `MSERRegions`, `ORBPoints` or `cornerPoints` object, or an *M*-by-2 matrix of *M* number of [*x*, *y*] coordinates. The function extracts descriptors from the neighborhoods that are fully contained within the image boundary. You can set the size of the neighborhood with the `BlockSize` parameter. Only neighborhoods fully contained within the image are used to determine the valid output points. The function ignores scale information associated with these points.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'BlockSize',[2 2]` sets the `BlockSize` to be a 2-by-2 square block.

`CellSize` — Size of HOG cell
`[8 8]` (default) | 2-element vector

Size of HOG cell, specified in pixels as a 2-element vector. To capture large-scale spatial information, increase the cell size. When you increase the cell size, you may lose small-scale detail.

`BlockSize` — Number of cells in block
`[2 2]` (default) | 2-element vector

Number of cells in a block, specified as a 2-element vector. A large block size value reduces the ability to suppress local illumination changes. Because of the number of pixels in a large block, these changes may get lost with averaging. Reducing the block size helps to capture the significance of local pixels. Smaller block size can help suppress illumination changes of HOG features.

`BlockOverlap` — Number of overlapping cells between adjacent blocks
`ceil(BlockSize/2)` (default)

Number of overlapping cells between adjacent blocks, specified as a 2-element vector. To ensure adequate contrast normalization, select an overlap of at least half the block size. Large overlap values can capture more information, but they produce larger feature vector size. This property applies only when you are extracting HOG features from regions and not from point locations. When you are extracting HOG features around a point location, only one block is used, and thus, no overlap occurs.

`NumBins` — Number of orientation histogram bins
9 (default) | positive scalar

Number of orientation histogram bins, specified as positive scalar. To encode finer orientation details, increase the number of bins. Increasing this value increases the size of the feature vector, which requires more time to process.

`UseSignedOrientation` — Selection of orientation values
`false` (default) | logical scalar

Selection of orientation values, specified as a logical scalar. When you set this property to `true`, orientation values are evenly spaced in bins between -180 and 180 degrees. When you set this property to `false`, they are evenly spaced from 0 through 180. In this case, values of theta that are less than 0 are placed into a theta + 180 value bin. Using signed orientation can help differentiate light-to-dark versus dark-to-light transitions within an image region.

Output Arguments

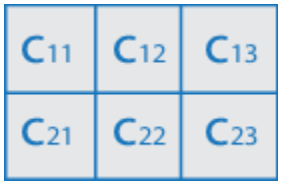
`features` — Extracted HOG features
1-by-*N* vector | *P*-by-*Q* matrix

Extracted HOG features, returned as either a 1-by-*N* vector or a *P*-by-*Q* matrix. The features encode local shape information from regions or from point locations within an image. You can use this information for many tasks including classification, detection, and tracking.

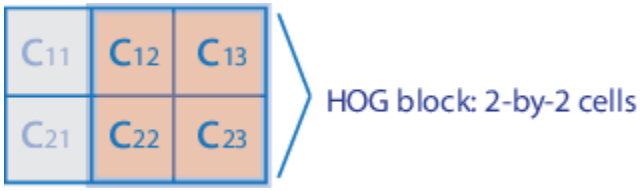
features output	Description
1-by-*N* vector	HOG feature length, *N*, is based on the image size and the function parameter values. *N* = `prod`([*BlocksPerImage*, `BlockSize`, `NumBins`]) *BlocksPerImage* = `floor`((`size`(`I`)./`CellSize` – `BlockSize`)./(`BlockSize` – `BlockOverlap`) + 1)
P-by-*Q* matrix	*P* is the number of valid points whose surrounding region is fully contained within the input image. You provide the `points` input value for extracting point locations. The surrounding region is calculated as: `CellSize`.*`BlockSize`. The feature vector length, *Q*, is calculated as: `prod`([`NumBins`,`BlockSize`]).

Example 3.1. Arrangement of Histograms in HOG Feature Vectors

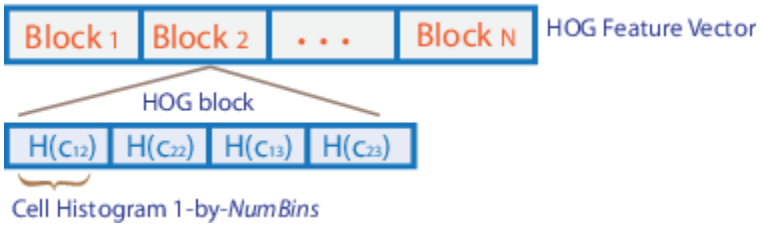
The figure below shows an image with six cells.



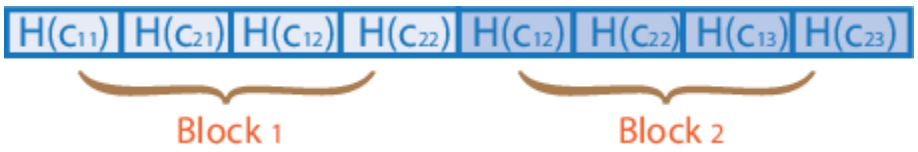
If you set the `BlockSize` to `[2 2]`, it would make the size of each HOG block, 2-by-2 cells. The size of the cells are in pixels. You can set it with the `CellSize` property.



The HOG feature vector is arranged by HOG blocks. The cell histogram, $H(C_{yx})$, is 1-by-`NumBins`.



The figure below shows the HOG feature vector with a 1-by-1 cell overlap between blocks.



`validPoints` — Valid points
`cornerPoints` object | `BRISKPoints` object | `SURFPoints` object | `MSERRegions` object | `ORBPoints` object | *M*-by-2 matrix of [*x*,*y*] coordinates

Valid points associated with each `features` descriptor vector output. This output can be returned as either a `cornerPoints` object, `BRISKPoints`, `SURFPoints` object, `MSERRegions` object, `ORBPoints` object or an *M*-by-2 matrix of [*x*,*y*] coordinates. The function extracts *M* number of descriptors from valid interest points in a region of size equal to [`CellSize`.*`BlockSize`]. The extracted descriptors are returned as the same type of object or matrix as the input. The region must be fully contained within the image.

`visualization` — HOG feature visualization
object

HOG feature visualization, returned as an object. The function outputs this optional argument to visualize the extracted HOG features. You can use the `plot` method with the `visualization` output. See the “Extract and Plot HOG Features” on page 3-817 example.

HOG features are visualized using a grid of uniformly spaced `rose` plots. The cell size and the size of the image determines the grid dimensions. Each rose plot shows the distribution of gradient orientations within a HOG cell. The length of each petal of the rose plot is scaled to indicate the contribution each orientation makes within the cell histogram. The plot displays the edge directions, which are normal to the gradient directions. Viewing the plot with the edge directions allows you to better understand the shape and contours encoded by HOG. Each rose plot displays two times `NumBins` petals.

You can use the following syntax to plot the HOG features:
`plot(visualization)` plots the HOG features as an array of rose plots.
`plot(visualization,AX)` plots HOG features into the axes AX.
`plot(___,'Color',colorValue)` Specifies the color used to plot HOG features, where `colorValue` represents the color as a 1-by-3 RGB vector, a short, or a long color name, described in the Color Value on page 3-823 table.

More About

Color Value

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	

Color Name	Short Name	RGB Triplet	Appearance
"black"	"k"	[0 0 0]	
"white"	"w"	[1 1 1]	

Version History

Introduced in R2013b

References

[1] Dalal, N. and B. Triggs. "Histograms of Oriented Gradients for Human Detection", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 1 (June 2005), pp. 886–893.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`extractLBPFeatures` | `extractFeatures` | `detectMSERFeatures` | `matchFeatures` | `detectSURFFeatures` | `SURFPoints` | `MSERRegions` | `ORBPoints` | `detectORBFeatures` | `detectHarrisFeatures` | `detectFASTFeatures` | `detectMinEigenFeatures` | `binaryFeatures` | `rose`

Topics

“Digit Classification Using HOG Features”
“Local Feature Detection and Extraction”
“Point Feature Types”

fitPolynomialRANSAC

Fit polynomial to points using RANSAC

Syntax

```
P = fitPolynomialRANSAC(xyPoints,N,maxDistance)
[P,inlierIdx] = fitPolynomialRANSAC( ___ )
[ ___ ] = fitPolynomialRANSAC( ___ ,Name=Value)
```

Description

`P = fitPolynomialRANSAC(xyPoints,N,maxDistance)` finds the *N*th-degree polynomial coefficients, `P`, by sampling a small set of points given in `xyPoints` and generating the *N*th polynomial fits. The fit that has the most inliers within `maxDistance` is returned. If a fit cannot be found, then `P` is returned empty. The function uses the M-estimator sample consensus (MSAC) algorithm, a variation of the random sample consensus (RANSAC) algorithm to fit the data.

`[P,inlierIdx] = fitPolynomialRANSAC( ___ )` returns a logical array, `inlierIdx`, that specifies the indices for data points that are inliers to the fit polynomial based on `maxDistance`. Use the input arguments from the previous syntax.

`[ ___ ] = fitPolynomialRANSAC( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `(MaxNumTrials=2000)` sets the maximum number of random trials to `2000`.

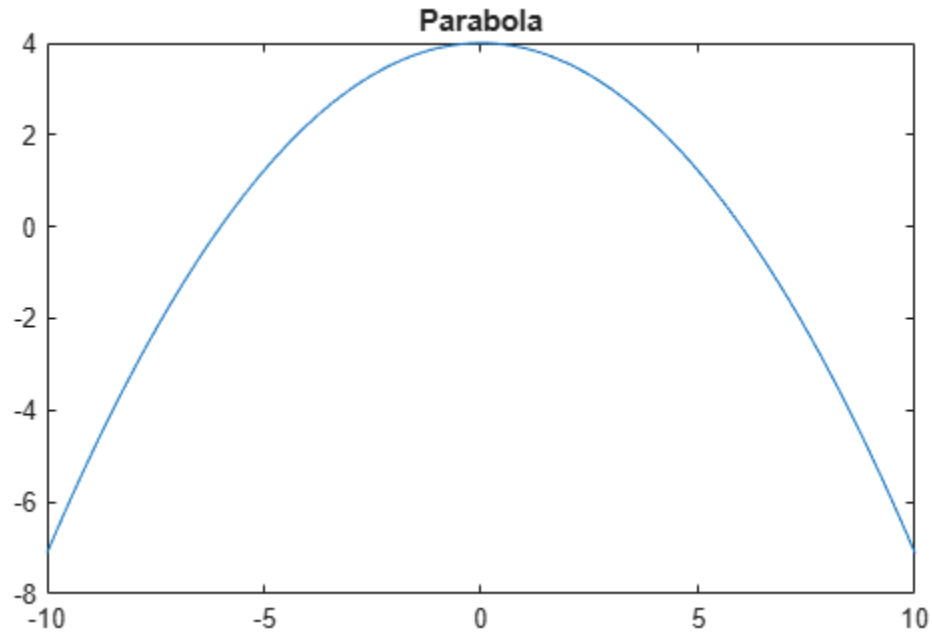
Examples

Fit Parabola to Noisy Data Using RANSAC

Use the RANSAC algorithm to generate a polynomial that fits a set of noisy data. The `fitPolynomialRANSAC` function generates a polynomial by sampling a small set of points from `[x y]` point data and generating polynomial fits. The fit with the most inliers within `maxDistance` is returned.

Construct and plot a parabola with `[x y]` points.

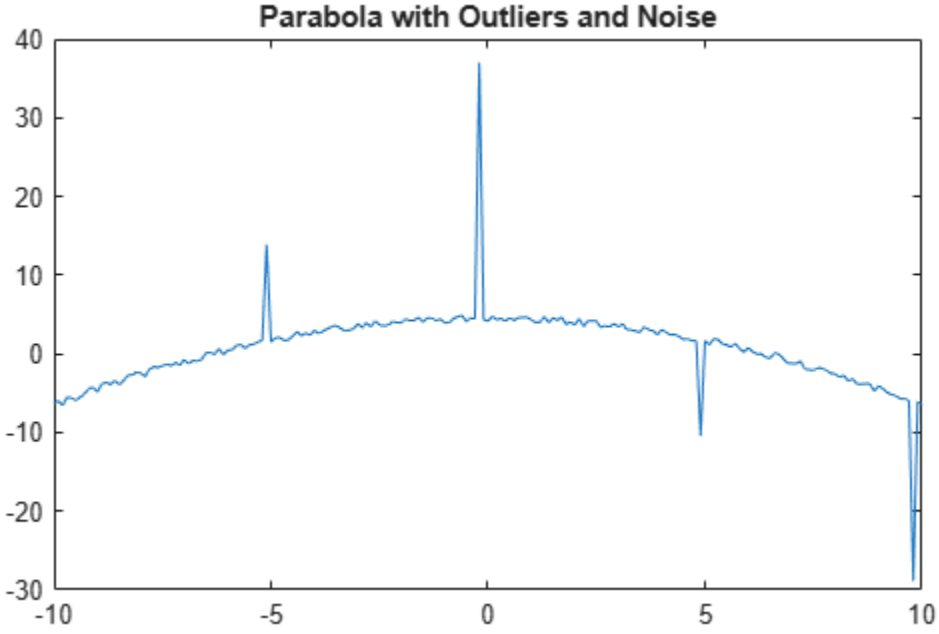
```
x = (-10:0.1:10)';
y = (36-x.^2)/9;
figure
plot(x,y)
title('Parabola')
```



Add noise and outlier points to the points on the parabola.

```
y = y+rand(length(y),1);
y([50,150,99,199]) = [y(50)+12,y(150)-12,y(99)+33,y(199)-23];

plot(x,y)
title('Parabola with Outliers and Noise')
```



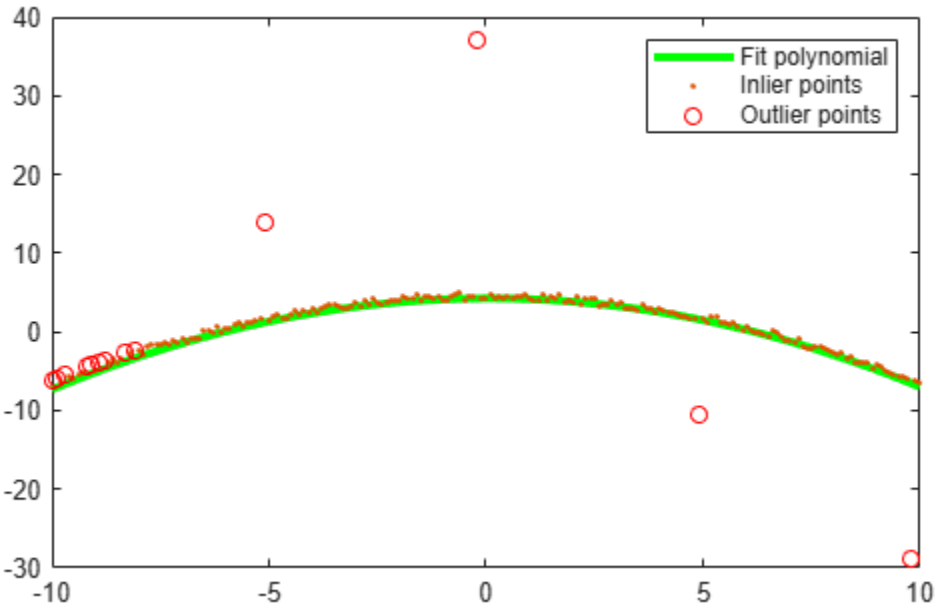
Use `fitPolynomialRANSAC` to generate coefficients for a second-degree polynomial. Also get the inliers identified by the specified `maxDistance` from the polynomial fit.

```
N = 2;           % second-degree polynomial
maxDistance = 1; % maximum allowed distance for a point to be inlier

[P, inlierIdx] = fitPolynomialRANSAC([x,y],N,maxDistance);
```

Evaluate the polynomial using `polyval`. Plot the curve and overlay the `[x y]` points. Mark outliers with a red circle.

```
yRecoveredCurve = polyval(P,x);
figure
plot(x,yRecoveredCurve,'-g','LineWidth',3)
hold on
plot(x(inlierIdx),y(inlierIdx),'.',x(~inlierIdx),y(~inlierIdx),'ro')
legend('Fit polynomial','Inlier points','Outlier points')
hold off
```



Input Arguments

`xyPoints` — `[x y]` coordinate points

m-by-2 matrix

`[x y]` coordinate points, specified as an *m*-by-2 matrix. The polynomial is fit to these points.

Data Types: `double` | `single` | `uint32` | `int32` | `uint16` | `int16`

`N` — Degree of polynomial fit

integer

Degree of polynomial fit, `P`, specified as an integer. The degree of a polynomial is the highest degree of the terms in the equation. For example, a polynomial of degree 2 is:

Ax^2+Bx+C

A, B, and C are constants. In general, higher degree polynomials allow for a better fit, but the fit depends on your data.

`maxDistance` — Maximum distance for inlier points
positive scalar

Maximum distance from the polynomial fit curve to an inlier point, specified as a positive scalar. Any points further away are considered outliers. The RANSAC algorithm creates a fit from a small sample of points but tries to maximize the number of inlier points. Lowering the maximum distance helps to improve the polynomial fit by putting a tighter tolerance on inlier points. For an M-by-2 matrix of points in the form [x y], the distance is defined as `abs`(y - `polyval`(`model`,x)).

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `(MaxNumTrials=2000)` sets the maximum number of random trials to `2000`.

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | integer

Maximum number of random trials, specified as an integer. A single trial uses a minimum number of random points from `xyPoints` to fit a parabolic model. Then, the trial checks the number of inliers within the `maxDistance` from the model. After all trials, the model with the highest number of inliers is selected. Increasing the number of trials improves the robustness of the output at the expense of additional computation.

`Confidence` — Confidence of final solution
`99` (default) | scalar from 0 to 100

Confidence that the final solution finds the maximum number of inliers for the polynomial fit, specified as a scalar from 0 to 100. Increasing this value improves the robustness of the output at the expense of additional computation.

`ValidatePolynomialFcn` — Function to validate polynomial
function handle

Function to validate polynomial, specified as a function handle. The function returns `true` if the polynomial is accepted based on criteria defined in the function. Use this function to reject specific polynomial fits. The function must be of the form:

```
isValid = validatePolynomialFcn(P,varargin)
```

If no function is specified, all polynomials are assumed to be valid.

`MaxSamplingAttempts` — Maximum number of sample attempts
`100` (default) | integer

Maximum number of attempts to find a sample that yields a valid polynomial, specified as an integer.

Output Arguments

`P` — Polynomial coefficients
vector of numeric scalars

Polynomial coefficients, returned as a vector of numeric scalars. Each element corresponds to a constant in the polynomial equation with degree `N`. For example, for a second-degree polynomial, Ax^2+Bx+C:

```
P = [A B C];
```

Data Types: `single` | `double`

`inlierIdx` — Inlier points
logical vector

Inlier points, returned as a logical vector. The vector is the same length as `xyPoints`, and each element indicates if that point is an inlier for the polynomial fit based on `maxDistance`.

Version History

Introduced in R2017a

References

[1] Torr, P. H. S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding*. Vol. 18, Issue 1, April 2000, pp. 138–156.

See Also

`ransac` | `polyfit` | `polyval`

extrinsics

(Not recommended) Compute location of calibrated camera

Note `extrinsics` is not recommended. Use the `estimateExtrinsics` function instead. For more information, see "Version History".

Syntax

```
[rotationMatrix,translationVector] = extrinsics(imagePoints,worldPoints,
cameraParams)
```

Description

`[rotationMatrix,translationVector] = extrinsics(imagePoints,worldPoints, cameraParams)` returns the 3-D rotation matrix and the 3-D translation vector to allow you to transform points from the world coordinate to the camera coordinate system.

Examples

Compute Camera Extrinsics

Create a set of calibration images.

```
  images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration", "slr"));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0). The square size is in millimeters.

```
squareSize = 29;
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

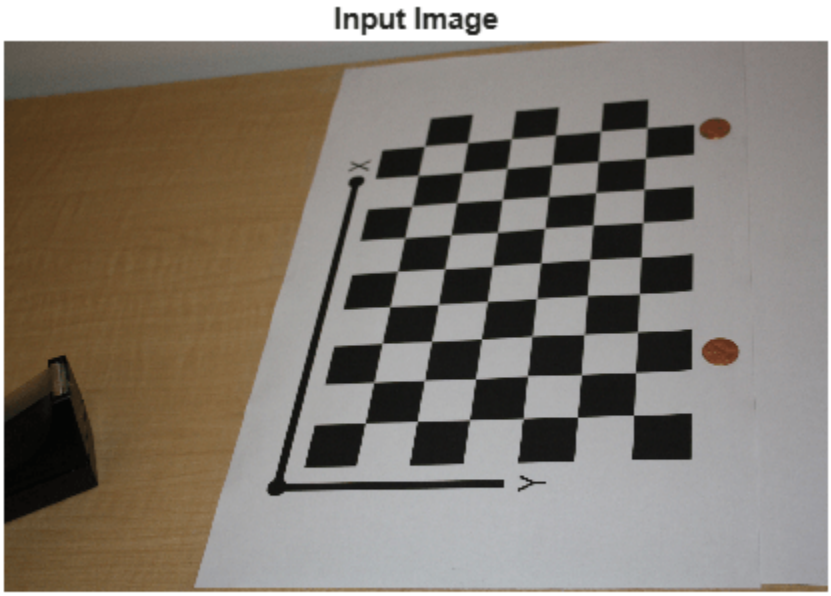
Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I,1), size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints, ...
                                        ImageSize=imageSize);
intrinsics = cameraParams.Intrinsics;
```

Load image at new location.

```
imOrig = imread(fullfile(matlabroot,"toolbox","vision","visiondata", ...
    "calibration","slr","image9.jpg"));
figure
```

```
imshow(imOrig);
title("Input Image");
```



Undistort image.

```
[im,newIntrinsics] = undistortImage(imOrig,intrinsics,OutputView="full");
```

Find reference object in new image.

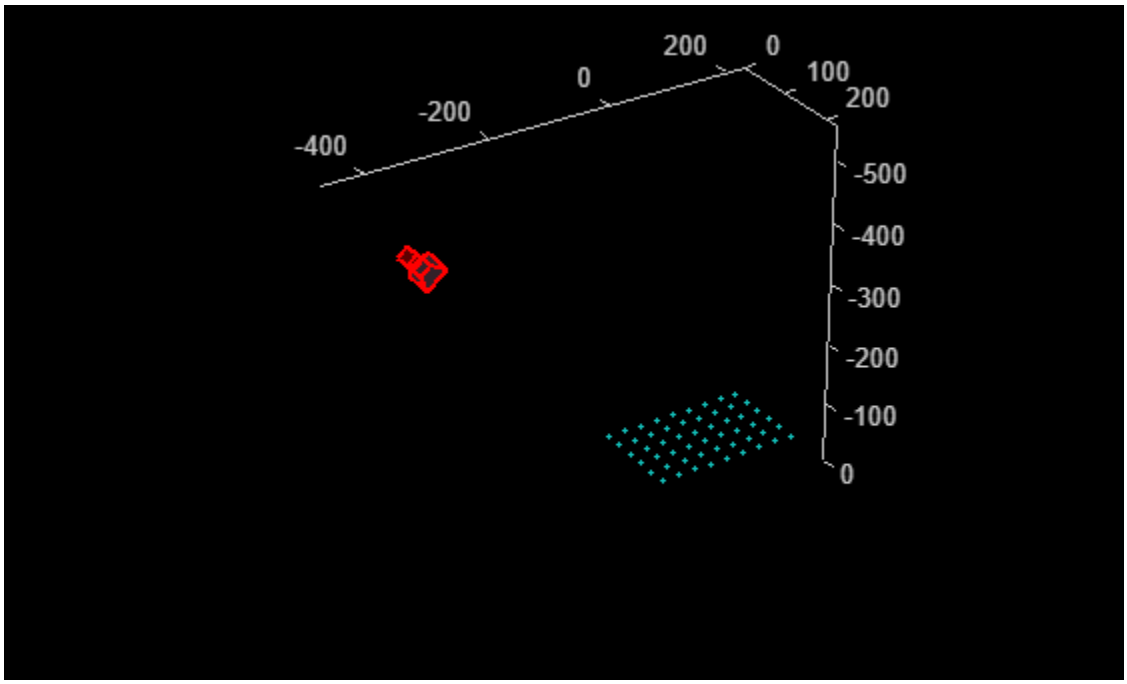
```
[imagePoints,patternDims] = detectCheckerboardPoints(im);
```

Compute new extrinsics.

```
[rotationMatrix, translationVector] = ...
    extrinsics(imagePoints,worldPoints,newIntrinsics);
```

Compute camera pose.

```
[orientation, location] = extrinsicsToCameraPose(rotationMatrix,translationVector);
figure
plotCamera(Location=location,Orientation=orientation,Size=20);
hold on
pcshow([worldPoints,zeros(size(worldPoints,1),1)], ...
  VerticalAxisDir="down",MarkerSize=40);
```



Input Arguments

`imagePoints` — Image coordinates of points

M-by-2 array

Image coordinates of points, specified as an *M*-by-2 array. The array contains *M* number of [*x*, *y*] coordinates. The `imagePoints` and `worldPoints` inputs must both be `double` or both be `single`.

Data Types: `single` | `double`

`worldPoints` — World coordinates corresponding to image coordinates

M-by-2 matrix

World coordinates corresponding to image coordinates, specified as an *M*-by-2 matrix. The `imagePoints` and `worldPoints` inputs must both be `double` or both be `single`. The function assumes that the points are coplanar with $z = 0$ and the number of points, *M*, must be at least 4.

Data Types: `single` | `double`

`cameraParams` — Camera parameters

`cameraParameters` object | `cameraIntrinsics` object | `fisheyeIntrinsics` object

Object for storing camera parameters, specified as a `cameraParameters`, `cameraIntrinsics`, or `fisheyeIntrinsics` object. These objects are returned by the `estimateCameraParameters` function, the `estimateFisheyeParameters` function, or the Camera Calibrator app. The object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

Output Arguments

`rotationMatrix` — 3-D rotation

3-by-3 matrix

3-D rotation, returned as a 3-by-3 matrix. The rotation matrix together with the translation vector allows you to transform points from the world coordinate to the camera coordinate system.



If you set the `imagePoints` and `worldPoints` inputs to class `double`, then the function returns the `rotationMatrix` and `translationVector` as `double`. Otherwise, they are `single`.

`translationVector` — 3-D translation

1-by-3 vector

3-D translation, returned as a 1-by-3 vector. The rotation matrix together with the translation vector allows you to transform points from the world coordinate to the camera coordinate system.



If you set the `imagePoints` and `worldPoints` inputs to class `double`, then the function returns the `rotationMatrix` and `translationVector` as `double`. Otherwise, they are `single`.

Algorithms

The `extrinsics` function uses two different algorithms to compute the extrinsics depending on whether `worldPoints` are specified as an *M*-by-2 matrix. Use an *M*-by-2 matrix for coplanar points where $z = 0$.

The `extrinsics` function computes the rotation matrix and translation vector for a single image in closed form. During calibration, the extrinsics are estimated numerically to minimize the reprojection errors for all calibration images. Therefore, using the `extrinsics` function on one of the calibration images returns rotation matrix and translation vector slightly different from the ones obtained during calibration.

Version History

Introduced in R2014a

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `extrinsics` function uses the postmultiply convention. Although there are no plans to remove `extrinsics` at this time, you can streamline your geometric transformation workflows by switching to the `estimateExtrinsics`

function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `extrinsics` to `estimateExtrinsics`.
- Specify the `cameraParams` argument as a `cameraIntrinsics` or `fisheyeIntrinsics` object. If you have a `cameraParameters` object, then you can get a `cameraIntrinsics` object by querying the `Intrinsics` property. If the `Intrinsics` property is empty according to the `isempty` function, then set the `ImageSize` property of the `cameraParameters` object to an arbitrary vector before querying the `Intrinsics` property. For example:

  ```
  load worldToImageCorrespondences.mat
  if(isempty(cameraParams.Intrinsics))
      cameraParams.ImageSize = [128 128];
  end
  intrinsics = cameraParams.Intrinsics;
  ```
- Replace the two output arguments `rotationMatrix` and `translationVector` with a single output argument, `camExtrinsics`. The `camExtrinsics` argument returns a `rigidtform3d` object. If you need to obtain the orientation matrix and location vector, then you can query the `R` and `Translation` properties of the object. Note that the value of `R` is the transpose of `rotationMatrix`.

The table shows examples of how to update your code.

Discouraged Usage	Recommended Replacement

This example calculates the location of a calibrated camera using the `exrinsics` function with the `cameraParams` argument specified as a `cameraParameters` object.
 	
`[rotMatrix,transVector] = extrinsics( ...`
`  imagePoints,worldPoints,cameraParams);`
	This example gets the camera intrinsics using the `Intrinsics` property of a `cameraParameters` object, then calculates the location of the calibrated camera using the `estimateExtrinsics` function.
	
	`intrinsics = cameraParams.Intrinsics;`
	`camExtrinsics = estimateExtrinsics( ...`
	`  imagePoints,worldPoints,intrinsics);`
	
	If you need to obtain the camera orientation and location, then you can query properties of `camExtrinsics`.
	
	`rotMatrix = camExtrinsics.R;`
	`transVector = camExtrinsics.Translation;`
	
	If you want the orientation in the postmultiply convention, take the transpose of `camExtrinsics.R`.
	
	`rotMatrix = camExtrinsics.R';`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.
- Use of a `fisheyeIntrinsics` object as the `cameraParams` input is not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Camera Calibrator | `estimateExtrinsics` | `plotCamera` | `estimateCameraParameters` | `cameraProjection` | `cameraParameters` | `cameraIntrinsics` | `estimateFisheyeParameters` | `fisheyeIntrinsics` | `img2world2d` | `world2img` | `extr2pose` | `pose2extr`

Topics
“Code Generation for Depth Estimation from Stereo Video”

generalizedDice

Generalized Sørensen-Dice similarity coefficient for image segmentation

Syntax

```
similarity = generalizedDice(X,target)
similarity = generalizedDice(X,target,'DataFormat',dataFormat)
```

Description

The generalized Dice similarity coefficient measures the overlap between two segmented images. Generalized Dice similarity is based on Sørensen-Dice similarity and controls the contribution that each class makes to the similarity by weighting classes by the inverse size of the expected region. When working with imbalanced data sets, class weighting helps to prevent the more prevalent classes from dominating the similarity score.

`similarity = generalizedDice(X,target)` calculates the generalized Sørensen-Dice similarity coefficient between test image `X` and target image `target`.

`similarity = generalizedDice(X,target,'DataFormat',dataFormat)` also specifies the dimension labels, `dataFormat`, of unformatted image data. You must use this syntax when the input are unformatted `dlarray` objects.

Examples

Calculate Generalized Dice Similarity

Load a pretrained network.

```
data = load("triangleSegmentationNetwork");
net = data.net;
```

Load the triangle image data set using `imageDatastore`.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
testImageDir = fullfile(dataDir,"testImages");
imds = imageDatastore(testImageDir);
```

Load ground truth labels for the triangle data set using `pixelLabelDatastore`.

```
labelDir = fullfile(dataDir,"testLabels");
classNames = ["triangle" "background"];
pixelLabelID = [255 0];
pxdsTruth = pixelLabelDatastore(labelDir,classNames,pixelLabelID);
```

Read a sample image and the corresponding ground truth labels.

```
I = readimage(imds,1);
gTruthLabels = readimage(pxdsTruth,1);
```

Run semantic segmentation on the image.

```
[predictions,scores] = semanticseg(I,net,Classes=classNames);
```

Encode the categorical predictions and targets using the `onehotencode` function.

```
featureDim = ndims(predictions) + 1;
encodedPredictions = onehotencode(predictions,featureDim);
encodedGroundTruthLabels = onehotencode(gTruthLabels,featureDim);
```

Ignore any undefined classes in the encoded data.

```
encodedPredictions(isnan(encodedPredictions)) = 0;
encodedGroundTruthLabels(isnan(encodedGroundTruthLabels)) = 0;
```

Compute generalized Dice similarity coefficient between the segmented image and the ground truth.

```
gDice = generalizedDice(encodedPredictions,encodedGroundTruthLabels)
```

```
gDice =
0.9346
```

Calculate Generalized Dice Loss of `dlarray` Input

Create input data as a formatted `dlarray` object containing 32 observations with unnormalized scores for ten output categories.

```
spatial = 10;
numCategories = 10;
batchSize = 32;
X = dlarray(rand(spatial,numCategories,batchSize),'SCB');
```

Convert unnormalized scores to probabilities of membership of each of the ten categories.

```
X = sigmoid(X);
```

Create target values for membership in the second and sixth category.

```
targets = zeros(spatial,numCategories,batchSize);
targets(:,2,:) = 1;
targets(:,6,:) = 1;
targets = dlarray(targets,'SCB');
```

Compute the generalized Dice similarity coefficient between probability vectors X and targets for multi-label classification.

```
Z = generalizedDice(X,targets);
whos Z
```

```
  Name      Size              Bytes  Class      Attributes

  Z         1x1x32              262  dlarray
```

Calculate the generalized Dice loss.

```
loss = 1 - mean(Z,'all')
```

```
loss =
  1(S) × 1(C) × 1(B) dlarray
```

```
1
```

Input Arguments

X — Test image

numeric array | `dlarray` object

Test image to be analyzed, specified as one of these values.

- A numeric array of any dimension. The last dimension must correspond to classes.
- An unformatted `dlarray` object. You must specify the data format using the `dataFormat` argument.
- A formatted `dlarray` object. The `dlarray` input must contain a channel dimension, `'C'` and can contain a batch dimension, `'B'`.

`dlarray` input requires Deep Learning Toolbox.

target — Target image

numeric array | `dlarray` object

Target image, specified as a numeric array or a `dlarray` object. The size and format of `target` must match the size and format of the test image, `X`. `dlarray` input requires Deep Learning Toolbox.

dataFormat — Dimension labels

string scalar | character vector

Dimension labels for unformatted `dlarray` image input, specified as a string scalar or character vector. Each character in `dataFormat` must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations

The format must include one channel label. The format cannot include more than one channel label or batch label. Do not specify the '`dataFormat`' argument when the input images are formatted `dlarray` objects.

Example: `'SSC'` indicates that the array has two spatial dimensions and one channel dimension

Example: `'SSCB'` indicates that the array has two spatial dimensions, one channel dimension, and one batch dimension

Output Arguments

similarity — Generalized Dice similarity coefficient

numeric scalar | `dlarray` object

Generalized Dice similarity coefficient, returned as a numeric scalar or a `dlarray` object with values in the range [0, 1]. A `similarity` of 1 means that the segmentations in the two images are a perfect match.

- If the input arrays are numeric images, then `similarity` is a numeric scalar.
- If the input arrays are `dlarray` objects, then `similarity` is a `dlarray` object of the same dimensionality as the input images. The spatial and channel dimensions of `similarity` are singleton dimensions. There is one generalized Dice measurement for each element along the batch dimension.

More About

Generalized Dice Similarity

Generalized Dice similarity is based on Sørensen-Dice similarity for measuring overlap between two segmented images.

The generalized Dice similarity function S used by `generalizedDice` for the similarity between one image Y and the corresponding ground truth T is given by:

$$S = \frac{2\sum_{k=1}^{K} w_k \sum_{m=1}^{M} Y_{km} T_{km}}{\sum_{k=1}^{K} w_k \sum_{m=1}^{M} Y_{km}^2 + T_{km}^2}$$

K is the number of classes, M is the number of elements along the first two dimensions of Y, and w_k is a class specific weighting factor that controls the contribution each class makes to the score. This weighting helps counter the influence of larger regions on the generalized Dice score. w_k is typically the inverse area of the expected region:

$$w_k = \frac{1}{\left(\sum_{m=1}^{M} T_{km}\right)^2}$$

There are several variations of generalized Dice scores [1], [2]. The `generalizedDice` function uses squared terms to ensure that the derivative is 0 when the two images match [3].

Version History

Introduced in R2021a

References

[1] Crum, William R., Oscar Camara, and Derek LG Hill. "Generalized overlap measures for evaluation and validation in medical image analysis." *IEEE Transactions on Medical Imaging*. 25.11, 2006, pp. 1451–1461.

[2] Sudre, Carole H., et al. "Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations." *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, Cham, 2017, pp. 240–248.

[3] Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". *Fourth International Conference on 3D Vision (3DV)*. Stanford, CA, 2016: pp. 565–571.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

`dice` | `dlarray` | `semanticseg` | `onehotencode`

generateCheckerboardPoints

Generate checkerboard corner locations

Syntax

```
worldPoints = generateCheckerboardPoints(boardSize,squareSize)
```

Description

`worldPoints = generateCheckerboardPoints(boardSize,squareSize)` returns an *M*-by-2 matrix containing *M* [*x*, *y*] corner coordinates for the squares on a checkerboard. The point [0,0] corresponds to the lower-right corner of the top-left square of the board.

Examples

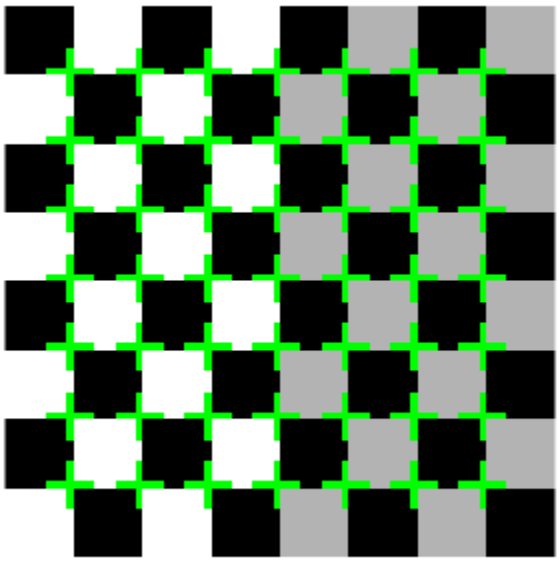
Generate and Plot Corners of an 8-by-8 Checkerboard

Generate the checkerboard, and obtain world coordinates.

```
I = checkerboard;
squareSize = 10;
worldPoints = generateCheckerboardPoints([8 8], squareSize);
```

Offset the points, placing the first point at the lower-right corner of the first square.

```
imshow(insertMarker(I, worldPoints + squareSize));
```

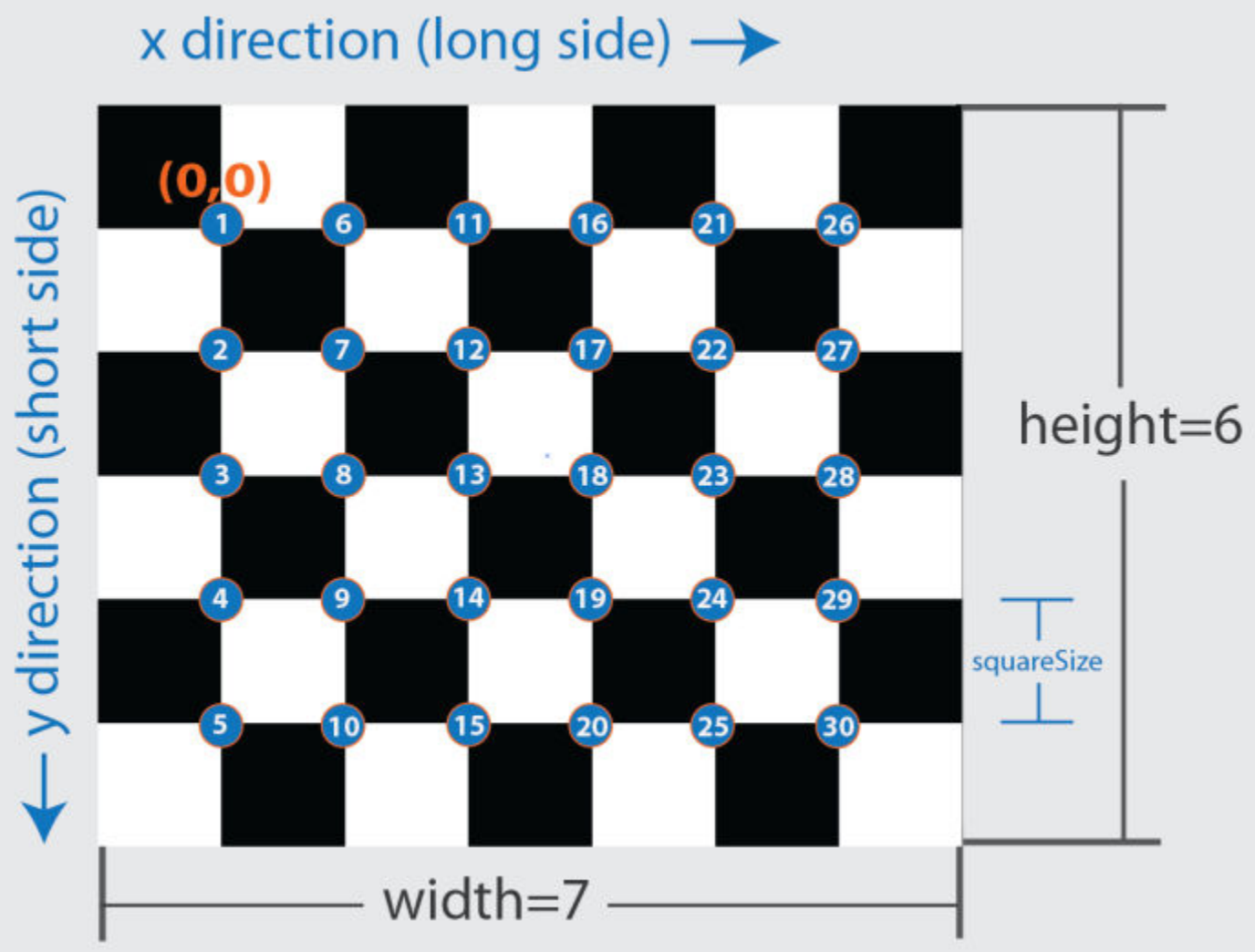


Input Arguments

`boardSize` — Generated checkerboard dimensions

2-element [*height*, *width*] vector

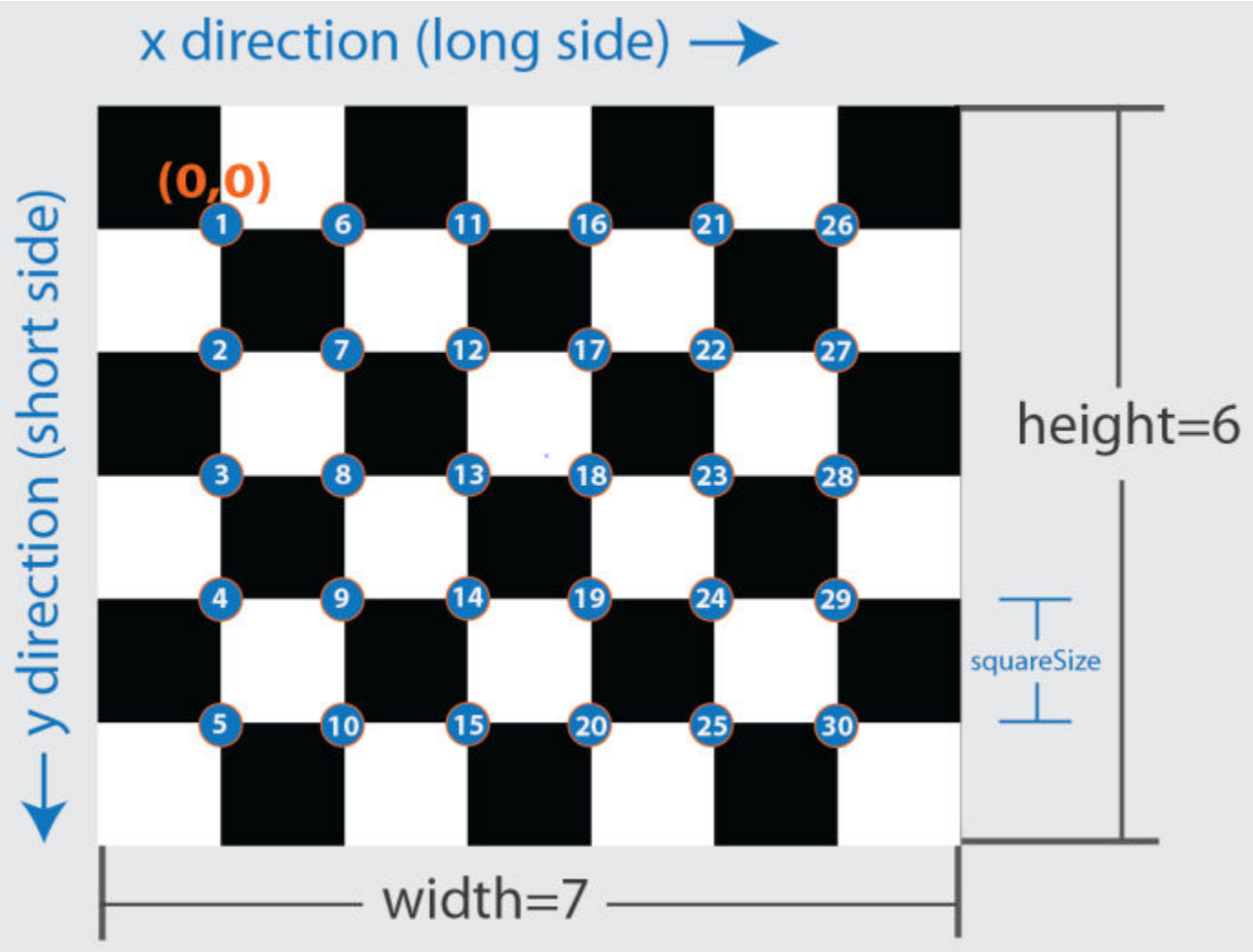
Generated checkerboard dimensions, specified as a 2-element [*height*, *width*] vector. You express the dimensions of the checkerboard in number of squares.



`squareSize` — Generated checkerboard square side length

scalar

Checkerboard square side length, specified as a scalar in world units. You express world units as a measurement, such as millimeters or inches.



Output Arguments

`worldPoints` — Generated checkerboard corner coordinates
M-by-2 matrix

Generated checkerboard corner coordinates, returned as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates. The coordinates represent the corners of the squares on the checkerboard. The point [0,0] corresponds to the lower-right corner of the top-left square of the board. The number of points, *M*, that the function returns are based on the number of squares on the checkerboard. This value is set with the `boardSize` parameter.
M = (`boardSize`(1)-1) * (`boardSize`(2)-1)

Version History

Introduced in R2013b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`estimateCameraParameters` | `detectCheckerboardPoints` | `cameraParameters` | `stereoParameters` | Camera Calibrator

Topics

“Measuring Planar Objects with a Calibrated Camera”
“Using the Single Camera Calibrator App”

imageMetrics

Evaluate per-image object detection performance metrics

Syntax

```
imageSummary = imageMetrics(metrics)
[imageSummary,imagePredictions,imageGroundTruth] = imageMetrics(metrics)
[___] = imageMetrics(metrics,imageIndices)
[___] = imageMetrics(___,Name=Value)
```

Description

`imageSummary = imageMetrics(metrics)` evaluates the object detection metrics, `metrics`, for each image in the data set.

`[imageSummary,imagePredictions,imageGroundTruth] = imageMetrics(metrics)` evaluates the object detection metrics, `metrics`, for each image in the data set, and returns the performance summary, predictions, and ground truth information for each image.

`[___] = imageMetrics(metrics,imageIndices)` evaluates the object detection metrics, `metrics`, for images in the data set specified using the image indices, `imageIndices`.

`[___] = imageMetrics(___,Name=Value)` specifies options that configure per-image metrics using name-value arguments, in addition to the input arguments from the previous syntaxes. For example, `ClassNames=["person" "car"]` specifies to evaluate per-image metrics for the person and car classes.

Examples

Compute Per-Image Metrics for Multiclass Object Detection

Load Detection Results and Ground Truth Annotations

Load precomputed detection results and ground truth labels. The ground truth data is a subset of the Indoor Object Detection Dataset created by Bishwo Adhikari, and is in the form of a `boxLabelDatastore` that returns the object bounding boxes and their corresponding class labels for every image [1]. The MAT files containing the precomputed results and annotations are attached to this example as supporting files. To train an object detector on a custom dataset and generate the detection results, see Multiclass Object Detection Using YOLO v2 Deep Learning.

```
load("yolov2IndoorObjectDetectorAnnotation100.mat");
load("yolov2IndoorObjectDetectorResults100.mat");
```

Evaluate and Summarize Object Detection Metrics

Evaluate the object detection results against the ground truth using the `evaluateObjectDetection` function. Compute the metrics summary across the dataset and classes using the `summarize` function.

```
evalMetrics = evaluateObjectDetection(results,blds,[0.5,0.75,0.9]);
[summaryDataset,summaryClass] = summarize(evalMetrics);
disp(summaryDataset)
```

NumObjects	mAPOverlapAvg	mAP0.5	mAP0.75	mAP0.9
171	0.40012	0.74165	0.44671	0.012002

```
disp(summaryClass)
```

	NumObjects	APOverlapAvg	AP0.5	AP0.75	AP0.9
exit	53	0.60733	0.9619	0.82106	0.039027
fireextinguisher	49	0.54426	0.93878	0.68903	0.0049712
chair	55	0.35992	0.88449	0.19463	0.00062696
clock	13	0.4891	0.92308	0.52885	0.015385
trashbin	1	0	0	0	0
screen	0	NaN	NaN	NaN	NaN
printer	0	NaN	NaN	NaN	NaN

The summary across classes shows that the "trashbin" class has only one ground truth object in this evaluation dataset, while the "screen" and "printer" classes do not have any ground truth objects present.

Evaluate Per-Image Metrics

Compute per-image metrics for the data set.

```
imageSummary = imageMetrics(evalMetrics);
```

Display the first ten images and their summarized per-image metrics.

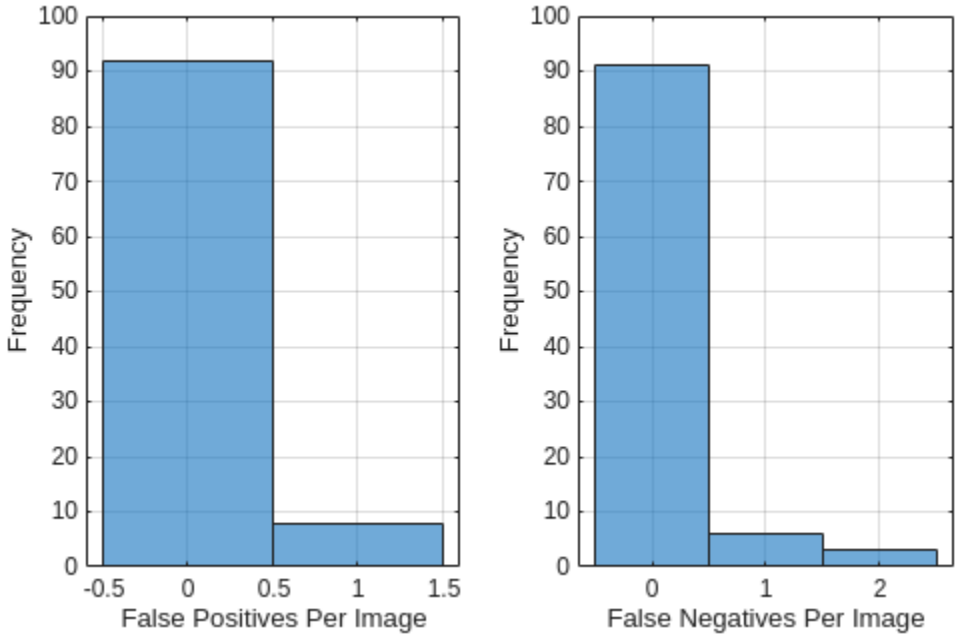
```
disp(imageSummary(1:10,:) )
```

ImageIndex	NumPredictedObjects	NumGroundTruthObjects	TP	FP	FN	Precision
1	2	2	2	0	0	1
2	2	2	2	0	0	1
3	1	1	1	0	0	1
4	0	2	0	0	2	NaN
5	3	2	2	1	0	0.66667
6	2	3	2	0	1	1
7	2	2	2	0	0	1
8	2	2	2	0	0	1
9	2	2	2	0	0	1
10	3	2	2	1	0	0.66667

Plot the histogram of false positive (FP) and false negative (FN) counts for each image.

```
figure;
subplot(1,2,1)
histogram(imageSummary.FP)
xlabel("False Positives Per Image")
ylabel("Frequency")
grid on
subplot(1,2,2)
```

```
histogram(imageSummary.FN)
xlabel("False Negatives Per Image")
ylabel("Frequency")
grid on
```



Most images have very few mistakes. The FP histogram shows that 8 images have one false positive detection. The FN histogram shows that 6 images have one false negative, and only 3 images have two false negatives.

[1] Adhikari, Bishwo; Peltomaki, Jukka; Huttunen, Heikki. (2019). Indoor Object Detection Dataset [Data set]. 7th European Workshop on Visual Information Processing 2018 (EUVIP), Tampere, Finland.

Analyze Object Detector Performance Across Size and Class

Load precomputed detection results and ground truth labels. The ground truth data is a subset of the Indoor Object Detection Dataset created by Bishwo Adhikari [1]. The MAT files containing the precomputed results and annotations are attached to this example as supporting files.

```
load("yolov2IndoorObjectDetectorAnnotation100.mat");
load("yolov2IndoorObjectDetectorResults100.mat");
```

Evaluate Object Detection Results

Evaluate the object detection results against the ground truth using the `evaluateObjectDetection` function. Specify three overlap thresholds for the evaluation: 0.5, 0.75, and 0.9.

```
evalMetrics = evaluateObjectDetection(results,blds,[0.5 0.75 0.9]);
```

Analyze Detector Performance Across Object Size and Class

Compute per-image metrics for the data set.

```
[imageSummary,imagePredictions,imageGroundTruths] = imageMetrics(evalMetrics);
```

To analyze object size- and class-based detector performance, create a visual representation of the number of true positives, false positives, false negatives for the entire data set. First, concatenate all prediction tables from all images into a single table `tblPreds`.

```
tblPreds = vertcat(imagePredictions{:});
```

Create a logical array indicating which predictions are matched to ground truth objects, or are true positives (TPs).

```
isMatchedPred = logical(tblPreds.IsMatched);
```

Extract true positives (TPs) by keeping only the `Label` (class name) and `Area` (object size) columns for rows where predictions are matched to ground truth.

```
tp = tblPreds(isMatchedPred,["Label" "Area"]);
```

Extract false positives (FPs) by keeping only the `Label` (class name) and `Area` (object size) columns for rows where predictions are not matched to ground truth.

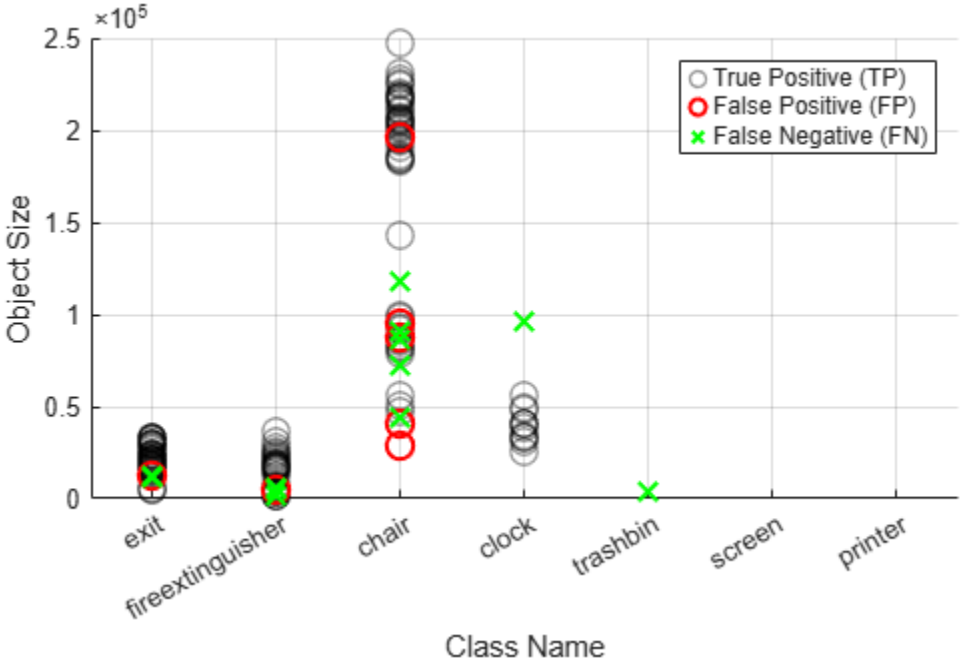
```
fp = tblPreds(~isMatchedPred,["Label" "Area"]);
```

Concatenate all ground truth tables into `tblTruths`, and create a logical array indicating which ground truths were matched by predictions. Extract false negatives (FNs) by keeping only `Label` and `Area` for ground truths not matched to any prediction.

```
tblTruths = vertcat(imageGroundTruths{:});
isMatchedGT = logical(tblTruths.IsMatched);
fn = tblTruths(~isMatchedGT,["Label" "Area"]);
```

Plot TPs as black circles using a swarm chart, with class name on the x-axis and object size on the y-axis. Plot false positives as red circles. Plot false negatives as green 'x' markers.

```
figure;
swarmchart(tp.Label,tp.Area,[],"k",XJitter="none",...
    LineWidth=1,SizeData=100,MarkerEdgeAlpha=0.4,DisplayName="True Positive (TP)")
hold on
swarmchart(fp.Label,fp.Area,[],"r",XJitter="none",...
    LineWidth=1.5,SizeData=100,DisplayName="False Positive (FP)")
swarmchart(fn.Label,fn.Area,[],"gx",XJitter="none",...
    LineWidth=1.5,SizeData=100,DisplayName="False Negative (FN)")
hold off
ylabel("Object Size")
xlabel("Class Name")
grid on
legend()
```



[1] Adhikari, Bishwo; Peltomaki, Jukka; Huttunen, Heikki. (2019). Indoor Object Detection Dataset [Data set]. 7th European Workshop on Visual Information Processing 2018 (EUVIP), Tampere, Finland.

Analyze Threshold-Based Object Detector Performance Metrics

Load precomputed detection results and ground truth labels. The ground truth data is a subset of the Indoor Object Detection Dataset created by Bishwo Adhikari [1]. The MAT files containing the precomputed results and annotations are attached to this example as supporting files.

```
load("yolov2IndoorObjectDetectorAnnotation100.mat");
load("yolov2IndoorObjectDetectorResults100.mat");
```

Evaluate Object Detection Results

Evaluate the object detection results against the ground truth using the `evaluateObjectDetection` function. Specify three overlap thresholds for the evaluation: 0.5, 0.75, and 0.9.

```
evalMetrics = evaluateObjectDetection(results,blds,[0.5 0.75 0.9]);
```

Compute Per-Image Metrics for Subset of Images

Compute the per-image metrics for a subset of images, and display the results.

```
imageResultsSubset = imageMetrics(evalMetrics,1:2:10);
disp(imageResultsSubset)
```

ImageIndex	NumPredictedObjects	NumGroundTruthObjects	TP	FP	FN	Precision

```
     1                2                2                2     0     0           1
     3                1                1                1     0     0           1
     5                3                2                2     1     0     0.66667
     7                2                2                2     0     0           1
     9                2                2                2     0     0           1
```

Compute Image Metrics at Specific Score Threshold

Specify a higher score threshold using the `ScoreThreshold` name-value argument, and compute the per-image metrics. At a higher score threshold, low confidence score predictions are dropped, leading to which will lead to fewer detected objects and more misses, or false negatives (FNs):

```
imageResultsThresh = imageMetrics(evalMetrics,1:2:10,ScoreThreshold=0.7);
disp(imageResultsThresh)
```

```
    ImageIndex    NumPredictedObjects    NumGroundTruthObjects    TP    FP    FN    Precision
    __________    ___________________    _____________________    __    __    __    _________

        1                 2                      2               2     0     0          1
        3                 1                      1               1     0     0          1
        5                 1                      2               1     0     1          1
        7                 1                      2               1     0     1          1
        9                 1                      2               1     0     1          1
```

Compute Image Metrics at Specific Overlap Threshold

Specify a higher overlap threshold using the `OverlapThreshold` name-value argument, and compute the per-image metrics. The `OverlapThreshold` value must be one of the overlap thresholds at which the detection results were evaluated using the `evaluateObjectDetection` function.

```
imageResultsThresh = imageMetrics(evalMetrics,1:2:10,ScoreThreshold=0.7,OverlapThreshold=0.75);
disp(imageResultsThresh)
```

```
    ImageIndex    NumPredictedObjects    NumGroundTruthObjects    TP    FP    FN    Precision
    __________    ___________________    _____________________    __    __    __    _________

        1                 2                      2               2     0     0          1
        3                 1                      1               1     0     0          1
        5                 1                      2               0     1     2          0
        7                 1                      2               1     0     1          1
        9                 1                      2               1     0     1          1
```

When you specify a higher overlap threshold, the detector misses more detections (more FNs).

[1] Adhikari, Bishwo; Peltomaki, Jukka; Huttunen, Heikki. (2019). Indoor Object Detection Dataset [Data set]. 7th European Workshop on Visual Information Processing 2018 (EUVIP), Tampere, Finland.

Analyze Object Detector Error Using Per-Image Metrics

Load precomputed detection results and ground truth labels. The ground truth data is a subset of the Indoor Object Detection Dataset created by Bishwo Adhikari [1]. The MAT files containing the precomputed results and annotations are attached to this example as supporting files.

```
load("yolov2IndoorObjectDetectorAnnotation100.mat");
load("yolov2IndoorObjectDetectorResults100.mat");
```

Evaluate Object Detection Results

Evaluate the object detection results against the ground truth using the `evaluateObjectDetection` function. Specify three overlap thresholds for the evaluation: 0.5, 0.75, and 0.9.

```
evalMetrics = evaluateObjectDetection(results,blds,[0.5 0.75 0.9]);
```

Identify Challenging Images Across All Classes

To identify images where the object detector performs poorly across all classes, analyze per-image detector performance using the false positive (FP) and false negative (FN) counts. First, compute the per-image metrics.

```
[summaryError,predsError,truthsError] = imageMetrics(evalMetrics,SortMetric="FP+FN",SortOrder="de
```

Select the 20 images with the highest number of incorrect detections, or the total number of FNs and FPs.

```
summaryError = summaryError(1:20,:);
predsError = predsError(1:20);
truthsError = truthsError(1:20);
```

Display the computed per-image metrics for the five images with the most incorrect detections.

```
disp(summaryError(1:5,:))
```

ImageIndex	NumPredictedObjects	NumGroundTruthObjects	TP	FP	FN	Precision
4	0	2	0	0	2	NaN
39	4	4	3	1	1	0.75
66	0	2	0	0	2	NaN
82	1	3	1	0	2	1
5	3	2	2	1	0	0.66667

Display the object predictions for the image with the index 39. The third "chair" class detection for this image is a FP, with its `IsMatched` column field set to `false`.

```
disp(predsError{2})
```

ImageIndex	Label	Score	OverlapScore	IsMatched	MatchedTruthLabel	MatchedTr
39	chair	0.87222	0.74144	true	chair	2
39	chair	0.86277	0.70789	true	chair	1
39	chair	0.66098	0	false	<undefined>	0
39	chair	0.62006	0.67055	true	chair	4

Identify Challenging Images for Specific Class

To identify images which are challenging for the object detector in detecting a specified class, analyze per-image detector performance using the FP and FN counts. To identify images where the detector struggled to identify objects of the "exit" class, specify the `ClassNames` name-value argument as `"exit"`.

```
[summaryErrorClass,predsErrorClass,truthsErrorClass] = imageMetrics(evalMetrics,ClassNames="exit"
    SortMetric="FP+FN",SortOrder="descend");
```

Select the 20 images with the highest number of incorrect detections, or the total number of FNs and FPs.

```
summaryErrorClass = summaryErrorClass(1:20,:);
predsErrorClass = predsErrorClass(1:20);
truthsErrorClass = truthsErrorClass(1:20,:);
```

Display the computed per-image metrics for the five images with the most incorrect detections.

```
disp(summaryErrorClass(1:5,:))
```

ImageIndex	NumPredictedObjects	NumGroundTruthObjects	TP	FP	FN	Precision
20	1	1	0	1	1	0
26	1	1	0	1	1	0
47	2	2	1	1	1	0.5
55	2	2	1	1	1	0.5
100	1	1	0	1	1	0

Display the object predictions for the image with the image index 47, which has one FP and one FN. The second "exit" class detection for this image is a FP, with its `IsMatched` column field set to `false`.

```
disp(predsErrorClass{3})
```

ImageIndex	Label	Score	OverlapScore	IsMatched	MatchedTruthLabel	MatchedTr
47	exit	0.90114	0.91124	true	exit	1
47	exit	0.68702	0.67807	false	<undefined>	0

Display the ground truth information for the image. The second ground truth row for this image indicates a FN, with its `IsMatched` column field set to `false`.

```
disp(truthsErrorClass{3})
```

ImageIndex	Label	IsMatched	Area	BoundingBox			
47	exit	true	19890	463	221	195	102
47	exit	false	27032	746	305	248	109

[1] Adhikari, Bishwo; Peltomaki, Jukka; Huttunen, Heikki. (2019). Indoor Object Detection Dataset [Data set]. 7th European Workshop on Visual Information Processing 2018 (EUVIP), Tampere, Finland.

Evaluate Per-Image Object Detection Using F-1 Metric

Load precomputed detection results and ground truth labels. The ground truth data is a subset of the Indoor Object Detection Dataset created by Bishwo Adhikari [1]. The MAT files containing the precomputed results and annotations are attached to this example as supporting files.

```
load("yolov2IndoorObjectDetectorAnnotation100.mat");
load("yolov2IndoorObjectDetectorResults100.mat");
```

Evaluate Object Detection Results

Evaluate the object detection results against the ground truth using the `evaluateObjectDetection` function. Specify three overlap thresholds for the evaluation: 0.5, 0.75, and 0.9.

```
evalMetrics = evaluateObjectDetection(results,blds,[0.5 0.75 0.9]);
```

Compute F1 Score Per Image

The F1 score is the harmonic mean of precision and recall, providing a balanced measure of classification performance. An F1 score of 1 indicates perfect precision and recall, while 0 indicates the worst performance.

Compute the image metrics over the data set.

```
[summaryError,predsError,truthsError] = imageMetrics(evalMetrics);
```

Display the first five rows of the per-image summary.

```
disp(summaryError(1:5,:))
```

ImageIndex	NumPredictedObjects	NumGroundTruthObjects	TP	FP	FN	Precision
1	2	2	2	0	0	1
2	2	2	2	0	0	1
3	1	1	1	0	0	1
4	0	2	0	0	2	NaN
5	3	2	2	1	0	0.66667

Note that precision is the ratio of the number of true positives and the total number of predicted positives. The value of precision on the 4th row is `NaN`, since there are no predictions on this particular image, leading to a division by zero in the ratio calculation. Recall is the ratio of the number of true positives and the number of ground truth objects in that image. The value of recall for this image is zero, since no objects are detected, while there are two ground truth objects actually present in the image.

Compute the F1 score over the data set using the precision and recall values from the `summaryError` table.

```
precision = summaryError.Precision;
recall = summaryError.Recall;
f1Score = 2 * (precision .* recall) ./ (precision + recall);
```

Organize the precision, recall and F1 score values into a table.

```
prfTable = table(precision, recall, f1Score, VariableNames={'Precision', 'Recall', 'F1 Score'});
```

Display the precision, recall, and F1 score values for the first five images.

```
disp(prfTable(1:5,:))
```

Precision	Recall	F1 Score

```
          1       1           1
          1       1           1
          1       1           1
        NaN       0         NaN
    0.66667       1         0.8
```

[1] Adhikari, Bishwo; Peltomaki, Jukka; Huttunen, Heikki. (2019). Indoor Object Detection Dataset [Data set]. 7th European Workshop on Visual Information Processing 2018 (EUVIP), Tampere, Finland.

Input Arguments

`metrics` — Object detection metrics

`objectDetectionMetrics` object

Object detection metrics, specified as an `objectDetectionMetrics` object.

`imageIndices` — Indices of images

N-element numeric vector

Indices of images for which to compute per-image object detection metrics, specified as a *N*-element numeric vector. Each element of the vector is a positive integer that represents an image index, and *N* is the number of images for which to compute metrics. By default, per-image metrics are returned for all images.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `imageMetrics(metrics,ClassNames=["person" "car"])` specifies to evaluate per-image metrics for the person and car classes.

`ClassNames` — Names of classes

`metrics.ClassNames` (default) | string scalar | character vector

Names of the classes for which to evaluate per-image metrics, specified as a string scalar or character vector. If multiple classes are specified, the function computes averaged metrics across the classes. By default, metrics will be computed for all classes stored in the ClassNames property of the `objectDetectionMetrics` object `metrics`.

`ScoreThreshold` — Confidence score threshold

`0.5` (default) | non-negative scalar

Confidence score threshold, specified as a non-negative scalar. The `imageMetrics` function returns per-image metrics for detections with a confidence score below this threshold. The value of `ScoreThreshold` must be in the predicted confidence score range specified by the `detectionResults` argument of the `evaluateObjectDetection` function. Set the score threshold above the confidence threshold at which the detector was run.

`OverlapThreshold` — Bounding box overlap threshold

`metrics.OverlapThreshold(1)` (default) | numeric scalar | numeric vector

Bounding box overlap threshold, specified as a numeric scalar or numeric vector. Each element of the numeric vector represents a box overlap threshold value over which the mean average precision is computed. By default, the overlap threshold is the first element of the OverlapThreshold property of the `objectDetectionMetrics` object `metrics`.

When the intersection over union (IoU) of the pixels in the ground truth bounding box and the predicted bounding box is equal to or greater than the overlap threshold, the detection is a match to the ground truth. The IoU is the number of pixels in the intersection of the bounding boxes divided by the number of pixels in the union of the bounding boxes.

`SortMetric` — Sorting metric
`"none"` (default) | `"FP"` | `"FN"` | `"FP+FN"` | `"Precision"` | `"Recall"`

Sorting metric that sorts the rows of the `imageMetrics` function output, specified as one of these options:

- `"none"` — Do not sort.
- `"FP"` — Sort by the number of false positive detections.
- `"FN"` — Sort by the number of false negative detections.
- `"FP+FN"` — Sort by the total number of false positive and false negative detections.
- `"Precision"` — Sort by the precision value.
- `"Recall"` — Sort by the recall value.

Tip Use the `SortMetric` argument to identify images with the most False Positives or False Negatives. This can help determine the types of images for which the detector exhibits the lowest performance.

`SortOrder` — Sorting order
`"descend"` (default) | `"ascend"`

Sorting order for the rows of the `imageMetrics` function output, specified as one of these options:

- `"descend"` — Sort in the descending order of the metric specified by the `SortMetric` name-value argument.
- `"ascend"` — Sort in the descending order of the metric specified by the `SortMetric` name-value argument.

Output Arguments

`imageSummary` — Per-image metrics summary
table

Metrics by area, returned as a table with *N* rows. *N* is the number of evaluated images. The columns are specified by these table variables.

- `ImageIndex` — Index of the image, based on image order in the `detectionResults` argument of the `evaluateObjectDetection` function, returned as a non-negative integer.
- `NumPredictedObjects` — The number of predicted objects in each image, returned as a positive integer.

- `NumGroundTruthObjects` — Number of ground truth objects in each image, returned as a positive integer.
- `TP` — Number of true positive detections, returned as a positive integer.
- `FP` — Number of false positives detections, returned as a positive integer.
- `FN` — Number of false negatives, returned as a positive integer.
- `Precision` — Precision, returned as a positive numeric scalar. Precision is the ratio of the number of true positives (*TP*) and the total number of predicted positives.
  Precision = *TP* / (*TP* + *FP*)

  FP is the number of false positives. Larger precision scores indicate that most detected objects match ground truth objects.
- `Recall` — Recall, returned as a positive numeric scalar. Recall is the ratio of the number of true positives (*TP*) and the number of ground truth objects – the sum of true positives (*TP*) and false negatives (*FN*).
  Recall = *TP* / (*TP* + *FN*)

  FN is the number of false negatives. Larger recall scores indicate that more of ground truth objects are detected.

`imagePredictions` — Per-image object predictions
N-by-1 cell array

Per-image object predictions, returned as an *N*-by-1 cell array. *N* is the number of evaluated images.

Each element of the cell array is a table with *P* rows, where *P* is the number of predictions each image, and these columns.

- `ImageIndex` — Index of the image, based on the image order in the `detectionResults` argument of the `evaluateObjectDetection` function, returned as a non-negative integer.
- `Label` — Predicted class label, returned as a string scalar or character vector.
- `Score` — Confidence score of the detector prediction, returned as a positive numeric scalar.
- `OverlapScore` — Bounding box overlap score between the ground truth and detected bounding boxes, returned as a positive numeric scalar.
- `IsMatched` — Logical that specifies if a predicted bounding box is a true positive or a false positive, returned as a numeric or logical `1` (`true`) or `0` (`false`), respectively.
- `MatchedTruthLabel` — If the predicted bounding box is a true positive (matched to a ground truth), then `MatchedTruthLabel` specifies the class of the ground truth box, returned as a categorical array. Otherwise, it is undefined.
- `MatchedTruthIndex` — If the predicted bounding box is a true positive (matched to a ground truth), then `MatchedTruthIndex` specifies the row index of the matched ground truth in the `imageGroundTruth` table, returned as a positive integer. Otherwise, it is *0*.
- `Area` — Area of the predicted bounding box, in pixels, returned as a positive integer.
- `BoundingBox` — Bounding box, returned as a 4-element or 5-element numeric vector. For an axis-aligned bounding box, the vector is of the form [*x y width height*]. For a rotated bounding box, the vector is of the form [x_{center} y_{center} *width height yaw*]. *x* and `y` specify the upper-left corner of the rectangle. *width* and *height* specify the width and height of the rectangle, which are its lengths along the *x*- and *y*-axes, respectively.

If you specify `"AOS"` as an additional metric to compute using the `AdditionalMetrics` name-value argument of the `evaluateObjectDetection` function, each table contains these additional columns.

- `Orientation` — The orientation angle of the predicted bounding box, returned as a non-negative numeric scalar. The angle specifies the clockwise rotation around the center of the bounding box. Units are in degrees.
- `OrientationDifference` — The difference between the ground truth and predicted bounding box orientations, returned as a non-negative numeric scalar. Units are in degrees.

`imageGroundTruth` — Per-image ground truth
table

Per-image ground truth, returned as an *N*-by-1 cell array. *N* is the number of evaluated images.

Each element of the cell array is a table with *M* rows and these columns. *M* is the number of ground truth objects for a given image.

- `ImageIndex` — Index of the image, based on the image order in the `detectionResults` argument of the `evaluateObjectDetection` function, returned as a non-negative integer.
- `Label` — Predicted class labels of the ground truth, returned as a string scalar or character vector.
- `IsMatched` — Logical that specifies if a prediction is matched to the ground truth or is a false negative, returned as a numeric or logical `1` (`true`) or `0` (`false`), respectively.
- `Area` — Area of the ground truth bounding box, in pixels, returned as a positive integer.
- `BoundingBox` — Bounding box, returned as a 4-element or 5-element numeric vector. For an axis-aligned bounding box, the vector is of the form [*x y width height*]. For a rotated bounding box, the vector is of the form [x_{center} y_{center} *width height yaw*]. *x* and `y` specify the upper-left corner of the rectangle. *width* and *height* specify the width and height of the rectangle, which are its lengths along the *x*- and *y*-axes, respectively.

If you specify `"AOS"` as an additional metric to compute using the `AdditionalMetrics` name-value argument of the `evaluateObjectDetection` function, each table contains this additional column.

- `Orientation` — The orientation angle of the ground truth bounding box, returned as a non-negative numeric scalar. The angle specifies the clockwise rotation around the center of the bounding box. Units are in degrees.

Version History

Introduced in R2026a

See Also

`objectDetectionMetrics` | `evaluateObjectDetection` | `summarize`

Topics

“Evaluate Object Detector Performance”
“Multiclass Object Detection Using YOLO v2 Deep Learning”

indexImages

Create image search index

Syntax

```
imageIndex = indexImages(imds)

imageIndex = indexImages(imds,bag)
imageIndex = indexImages( ___ ,Name,Value)
```

Description

`imageIndex = indexImages(imds)` creates an `invertedImageIndex` object, `imageIndex`, that contains a search index for `imds`. Use `imageIndex` with the `retrieveImages` function to search for images.

`imageIndex = indexImages(imds,bag)` returns a search index that uses a custom `bagOfFeatures` object, `bag`. Use this syntax with the `bag` you created when you want to modify the number of visual words or the feature type used to create the image search index for `imds`.

`imageIndex = indexImages( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

This object supports parallel computing using multiple MATLAB workers. Enable parallel computing from the “Computer Vision Toolbox Preferences” dialog box. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Then select **Computer Vision Toolbox** .

Examples

Search Image Set Using a Query Image

Create an image set.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets','cups');
imds = imageDatastore(setDir);
```

Index the image set.

```
imageIndex = indexImages(imds)
```

```
Creating an inverted image index using Bag-Of-Features.
-------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.
```

```
* Extracting features from 6 images...done. Extracted 1708 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 1366.
** Using the strongest 1366 features from each of the other image categories.

* Creating a 1366 word visual vocabulary.
* Number of levels: 1
* Branching factor: 1366
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 1366
* Number of clusters          : 1366
* Initializing cluster centers...100.00%.
* Clustering...completed 1/100 iterations (~0.03 seconds/iteration)...converged in 1 iterations.

* Finished creating Bag-Of-Features


Encoding images using Bag-Of-Features.
---------------------------------------

* Encoding 6 images...done.
Finished creating the image index.

imageIndex =
  invertedImageIndex with properties:

         ImageLocation: {6×1 cell}
            ImageWords: [6×1 vision.internal.visualWords]
         WordFrequency: [0.1667 0.1667 0.1667 0.3333 0.1667 0.1667 0.1667 0.5000 0.3333 0.1667 0
         BagOfFeatures: [1×1 bagOfFeatures]
               ImageID: [1 2 3 4 5 6]
        MatchThreshold: 0.0100
    WordFrequencyRange: [0.0100 0.9000]
```

Display the image set using the `montage` function.

```
thumbnailGallery = [];
for i = 1:length(imds.Files)
    I = readimage(imds,i);
    thumbnail = imresize(I,[300 300]);
    thumbnailGallery = cat(4,thumbnailGallery,thumbnail);
end

figure
montage(thumbnailGallery);
```



Select a query image.

```
queryImage = readimage(imds,2);
figure
imshow(queryImage)
```



Search the image set for similar image using query image. The best result is first.

```
indices = retrieveImages(queryImage,imageIndex)
```

```
indices = 5×1 uint32 column vector

   2
   1
   5
   4
   3
```

```
bestMatchIdx = indices(1);
```

Display the best match from the image set.

```
bestMatch = imageIndex.ImageLocation{bestMatchIdx}
```

```
bestMatch =
'B:\runnable\matlab\toolbox\vision\visiondata\imageSets\cups\blueCup.jpg'
```

```
figure
imshow(bestMatch)
```



Create Search Index Using Custom Bag of Features

Create an image set.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets','cups');
imds = imageDatastore(setDir);
```

Display image set.

```
thumbnailGallery = [];
for i = 1:length(imds.Files)
```

```
    I = read(imds);
    thumbnail = imresize(I, [300 300]);
    thumbnailGallery = cat(4, thumbnailGallery, thumbnail);
end

figure
montage(thumbnailGallery);
```



Train a bag of features using a custom feature extractor.

```
extractor = @exampleBagOfFeaturesExtractor;
bag = bagOfFeatures(imds,'CustomExtractor',extractor);
```

```
Creating Bag-Of-Features.
-------------------------

* Extracting features using a custom feature extraction function: exampleBagOfFeaturesExtractor.

* Extracting features from 6 images...done. Extracted 115200 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 92160
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 46/100 iterations (~1.09 seconds/iteration)...converged in 46 iterations

* Finished creating Bag-Of-Features
```

Use the trained bag of features to index the image set.

```
imageIndex = indexImages(imds,bag,'Verbose',false)
```

```
imageIndex =
  invertedImageIndex with properties:

         ImageLocation: {6×1 cell}
            ImageWords: [6×1 vision.internal.visualWords]
         WordFrequency: [1 1 1 1 1 1 1 0.8333 1 1 1 1 1 1 1 1 1 1 1 1 0.8333 1 1 1 1 1 1 0.8333
         BagOfFeatures: [1×1 bagOfFeatures]
               ImageID: [1 2 3 4 5 6]
        MatchThreshold: 0.0100
    WordFrequencyRange: [0.0100 0.9000]
```

```
queryImage = readimage(imds,4);

figure
imshow(queryImage)
```



Search for the image from image index using query image.

```
indices = retrieveImages(queryImage,imageIndex);
bestMatch = imageIndex.ImageLocation{indices(1)};
figure
imshow(bestMatch)
```



Input Arguments

`imds` — Images

`imageDatastore` object

Images, specified as an `imageDatastore` object. The object stores a collection of images.

`bag` — Bag of visual words

`bagOfFeatures` object

Bag of visual words, specified as a `bagOfFeatures` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Verbose',true` sets the `'Verbose'` property to `true`

`SaveFeatureLocations` — Save feature locations

`true` (default) | `false`

Save feature locations, specified as the comma-separated pair consisting of '`SaveFeatureLocations`' and a logical scalar. When set to `true`, the image feature locations are saved in the `imageIndex` output object. Use location data to verify spatial or geometric image search results. If you do not require feature locations, set this property to false to reduce memory consumption.

`Verbose` — Display progress information
`true` (default) | `false`

Display progress information, specified as the comma-separated pair consisting of '`Verbose`' and a logical scalar.

Output Arguments

`imageIndex` — Image search index
`invertedImageIndex` object

Image search index, returned as an `invertedImageIndex` object.

Algorithms

`imageIndex` uses the bag-of-features framework with the speeded-up robust features (SURF) detector and extractor to learn a vocabulary of 20,000 visual words. The visual words are then used to create an index that maps visual words to the images in `imds`. You can use the index to search for images within `imds` that are similar to a given query image.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

For code generation, the `indexImages` function does not support the `ImageDatastore` object as input. Instead, specify the input as a structure with fields `Images`, `Locations`, and `Labels`. The `Images` field must contain a cell array of grayscale or RGB color images. The `Locations` field must contain a cell array of the file paths to the locations of input images. The `Labels` field when non-empty, must contain image labels stored as categorical arrays.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`retrieveImages` | `evaluateImageRetrieval` | `imageDatastore` | `bagOfFeatures` | `invertedImageIndex`

Topics
“Image Retrieval Using Customized Bag of Features”
“Image Retrieval with Bag of Visual Words”

integralFilter

Filter using integral image

Syntax

```
J = integralFilter(intI,H)
```

Description

`J = integralFilter(intI,H)` filters an image given its integral image, `intI`, and a box filter, `H`.

Examples

Blur an Image Using an Average Filter

Read and display the input image.

```
I = imread('pout.tif');
imshow(I);
```



Compute the integral image.

```
intImage = integralImage(I);
```

Apply a 7-by-7 average filter.

```
avgH = integralKernel([1 1 7 7], 1/49);
J = integralFilter(intImage, avgH);
```

Cast the result back to the same class as the input image.

```
J = uint8(J);
figure
imshow(J);
```



Find Vertical and Horizontal Edges in Image

Construct Haar-like wavelet filters to find vertical and horizontal edges in an image.

Read the input image and compute the integral image.

```
I = imread('pout.tif');
intImage = integralImage(I);
```

Construct Haar-like wavelet filters. Use the dot notation to find the vertical filter from the horizontal filter.

```
horiH = integralKernel([1 1 4 3; 1 4 4 3],[-1, 1]);
vertH = horiH.'
```

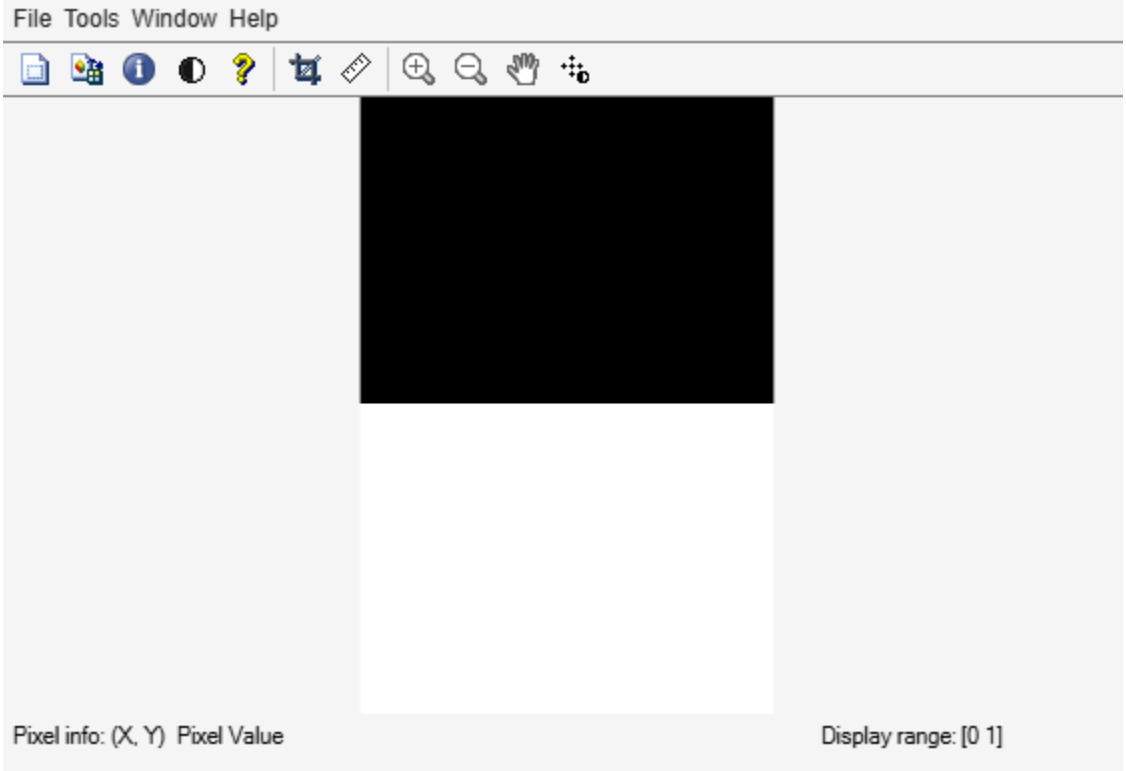
```
vertH =
  integralKernel with properties:

    BoundingBoxes: [2×4 double]
          Weights: [-1 1]
     Coefficients: [4×6 double]
           Center: [2 3]
```

```
            Size: [4 6]
     Orientation: 'upright'
```

Display the horizontal filter.

```
imtool(horiH.Coefficients, 'InitialMagnification','fit');
```

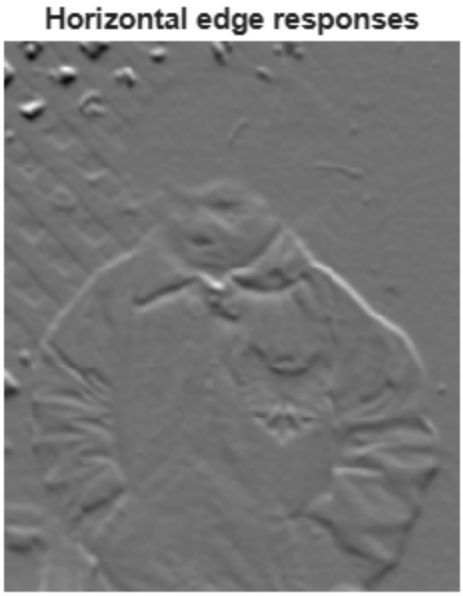


Compute the filter responses.

```
horiResponse = integralFilter(intImage,horiH);
vertResponse = integralFilter(intImage,vertH);
```

Display the results.

```
figure;
imshow(horiResponse,[]);
title('Horizontal edge responses');
```




```
figure;
imshow(vertResponse,[]);
title('Vertical edge responses');
```



Compute a Rotated Edge Response Using Integral Filter

Read the input image.

```
I = imread("pout.tif");
```

Compute 45 degree edge responses of the image.

```
intImage = integralImage(I,"rotated");
figure;
imshow(I);
title("Original Image");
```

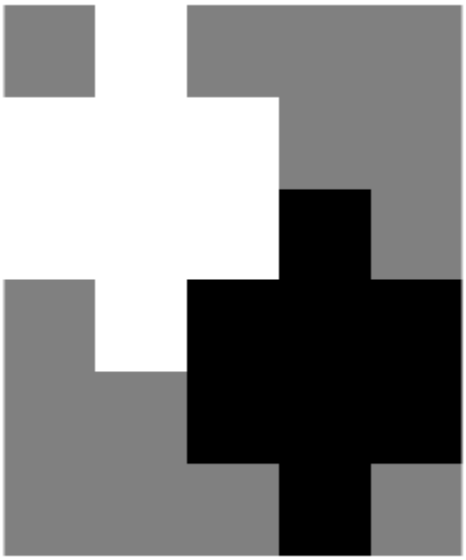


Construct 45 degree rotated Haar-like wavelet filters.

```
rotH = integralKernel([2 1 2 2;4 3 2 2],[1 -1],"rotated");
rotHTrans = rotH.';
```

Visualize the filter rotH.

```
figure;
imshow(rotH.Coefficients, []);
```

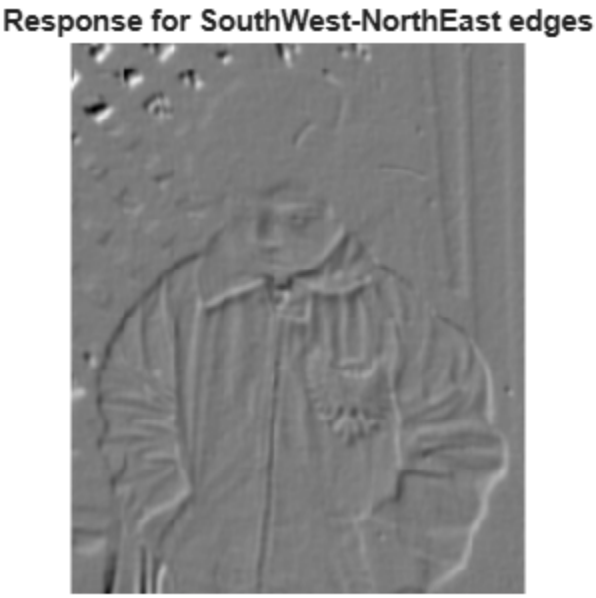


Compute filter responses.

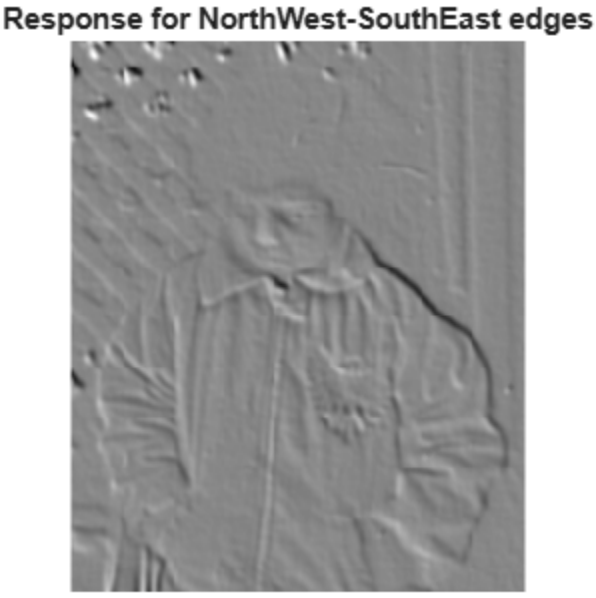
```
rotHResponse = integralFilter(intImage,rotH);
rotHTransResponse = integralFilter(intImage,rotHTrans);
```

Display results.

```
figure;
imshow(rotHResponse, []);
title("Response for SouthWest-NorthEast edges");
```




```
figure;
imshow(rotHTransResponse, []);
title("Response for NorthWest-SouthEast edges");
```



Input Arguments

`intI` — Integral image

numeric array

Integral image, specified as a numeric array. You can obtain the integral image, `intI`, using the `integralImage` function.

Data Types: `single` | `double`

`H` — Box filter

`integralKernel` object

Box filter, specified as an `integralKernel` object.

Output Arguments

`J` — Filtered image

numeric array

Filtered image, returned as a numeric array. The filtered image, `J`, returns only the parts of correlation that are computed without padding. This results in `size(J) = size(intI) – H.Size` for an upright filter, and `size(J) = size(intI) – H.Size – [0 1]` for a rotated filter.

Tips

- The filter size does not affect the speed of the filtering operation. Thus, the `integralFilter` function is ideally suited to use for fast analysis of images at different scales, as demonstrated by the Viola-Jones algorithm [1].
- The `integralFilter` function uses correlation for filtering. Therefore, the filter is not rotated before calculating the result.

Version History

Introduced in R2012a

References

[1] Viola, Paul and Michael J. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features." *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. Volume: 1, pp.511–518.

See Also

`cumsum` | `integralImage` | `integralKernel`

insertMarker

Insert markers in image or video

Syntax

```
RGB = insertMarker(I,position)
RGB = insertMarker(I,position,marker)
RGB = insertMarker(___,Name=Value)
```

Description

`RGB = insertMarker(I,position)` insert plus (+) markers into the truecolor or grayscale image `I`. The function draws markers by overwriting pixel values at the specified positions. The output image `RGB` is always a truecolor image.

`RGB = insertMarker(I,position,marker)` inserts markers of the specified marker type `marker`.

`RGB = insertMarker(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of input arguments from previous syntaxes. For example, `MarkerColor="red"` specifies to render the inserted position markers in red.

Examples

Draw Markers on an Image

Read an image into the workspace.

```
I = imread("peppers.png");
```

Insert a plus (+) marker onto the image at specified position.

```
RGB = insertMarker(I,[147 279]);
```

Specify the positions, colors, and sizes of four cross (x) markers and insert onto the image.

```
pos = [120 248; 195 246; 195 312; 120 312];
color = {"red","white","green","magenta"};
RGB = insertMarker(RGB,pos,"x",MarkerColor=color,Size=10);
```

Display the image.

```
imshow(RGB)
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor or an *M*-by-*N* grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`position` — Position of markers

M-by-2 matrix | *M*-element vector of points objects

Position of markers, specified as an *M*-by-2 matrix, where each row is an image coordinate pair of the form [*x* *y*], or an *M*-element vector of one of the points objects listed on the “Point Feature Types” page. *M* is the number of markers to insert into the image. The function defines the position of each marker as the [*x* *y*] pair of coordinates in the matrix, as the value of the `Location` property of the corresponding point feature object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`marker` — Type of marker

`"plus"` (default) | character vector | string scalar

Type of marker, specified as a character vector or string scalar. You can specify the marker type using either the full text name or the corresponding symbol.

Full Text Name	Symbol
`"circle"`	`"o"`
`"x-mark"`	`"x"`

Full Text Name	Symbol
"plus"	"+"
"star"	"*"
"square"	"s"

Data Types: char | string

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: insertMarker(I,position,Color="yellow") sets the marker color to yellow.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: insertMarker(I,position,"Color","yellow") sets the marker color to yellow.

Size — Size of marker

3 (default) | scalar

Size of the marker in pixels, specified as a scalar in the range [1,Inf).

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

MarkerColor — Marker color

"green" (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

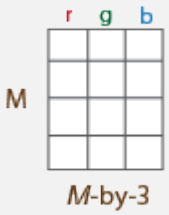
Marker color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of uint8 values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a uint8 data type, use the code uint8(255*intensity), where intensity is an RGB triplet value listed in the table.

You can specify a different color for each marker or one color for all markers. To specify one color for all markers, specify MarkerColor as a color name or an [*R G B*] vector.

Specification	Format	Example
Specify one color for all shapes (or markers)	Short color name or color name	"r" "red"
	RGB triplet	 [1 0 0]
Specify a color for each shape (or marker)	Vector of color names	["red","yellow","blue"]

Specification	Format	Example

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

This table lists the supported marker colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single` | `cell`

Output Arguments

RGB — Output image

M-by-*N*-by-3 truecolor image

Output image, returned as an *M*-by-*N*-by-3 truecolor image.

Version History

Introduced in R2013a

R2023b: Specify RGB color in the range [0,1]

Behavior changed in R2023b

Starting in R2023b, you can specify marker color with RGB values in the range [0,1].

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

- The input argument `marker` and the name-value argument `"MarkerColor"` must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`insertObjectAnnotation` | `cornerPoints` | `SURFPoints` | `MSERRegions` | `BRISKPoints` | `ORBPoints` | `insertShape` | `insertText` | “Point Feature Types”

Topics

“Insert Circle and Filled Shapes onto Image” on page 3-907
“Insert Text into Image” on page 3-919

insertObjectAnnotation

Annotate truecolor or grayscale image or video

Syntax

```
RGB = insertObjectAnnotation(I,shape,position,label)
___ = insertObjectAnnotation( ___ ,Name=Value)
```

Description

`RGB = insertObjectAnnotation(I,shape,position,label)` annotates a grayscale or truecolor image with the specified `shape` and `label` at the specified `position`. The function returns a truecolor image.

`___ = insertObjectAnnotation( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `FontSize=18` sets the annotation font size to `18`.

Examples

Annotate Image with Numbers and Strings

Read an image into the workspace.

```
I = imread("board.tif");
```

Create confidence value labels using floating point numbers.

```
label_str = cell(3,1);
conf_val = [85.212 98.76 78.342];
for ii = 1:3
    label_str{ii} = ['Confidence: ' num2str(conf_val(ii), '%0.2f') '%'];
end
```

Set the positions for the rectangular bounding boxes associated with the labels using the form [*x y width height*].

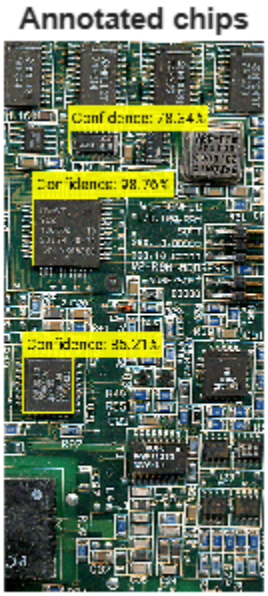
```
position = [23 373 60 66; 35 185 77 81; 77 107 59 26];
```

Insert the labels into the image.

```
RGB = insertObjectAnnotation(I,"rectangle",position,label_str,TextBoxOpacity=0.9,FontSize=18);
```

Display the annotated image.

```
figure
imshow(RGB)
title("Annotated chips")
```



Annotate Image with Integers

Read an image into the workspace.

```
I = imread("coins.png");
```

Set the positions on the image for the circular bounding boxes associated with the labels. The first two values of each row specify the center coordinates of a circle, (*x*,*y*) and the third value is the radius.

```
position = [96 146 31; 236 173 26];
```

Set the values for the labels to display as 5 and 10, in integers. These values represent US currency in the form of a nickle and dime, respectively.

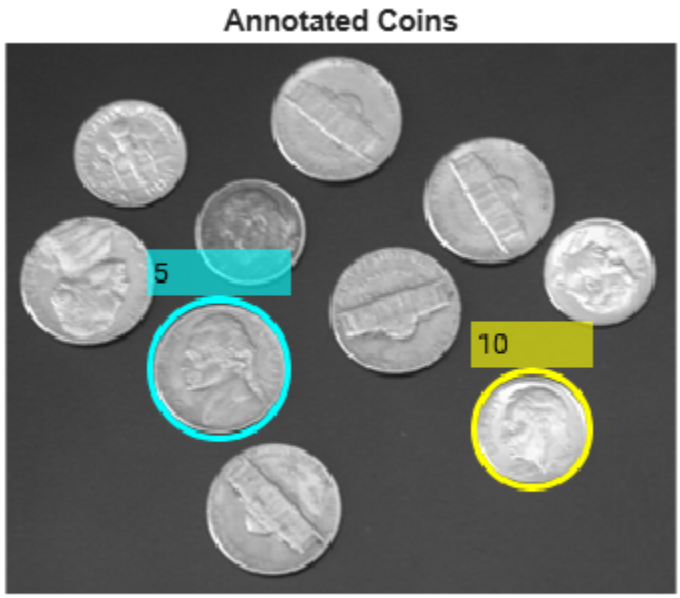
```
label = [5 10];
```

Insert the annotations into the image.

```
RGB = insertObjectAnnotation(I,"circle",position,label,LineWidth=3,AnnotationColor=["cyan","yello
```

Display the annotated image.

```
figure
imshow(RGB)
title("Annotated Coins")
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor or an *M*-by-*N* grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`shape` — Shape of annotation

`"rectangle"` | `"circle"` | `"projected-cuboid"` | `"ellipse"`

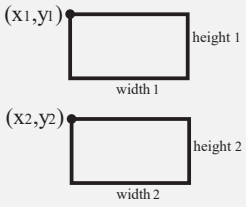
Shape of the annotation, specified as `"rectangle"`, `"circle"`, `"projected-cuboid"`, or `"ellipse"`. If you specify a `"rectangle"` annotation be specify the position of either an axis-aligned or rotated rectangle.

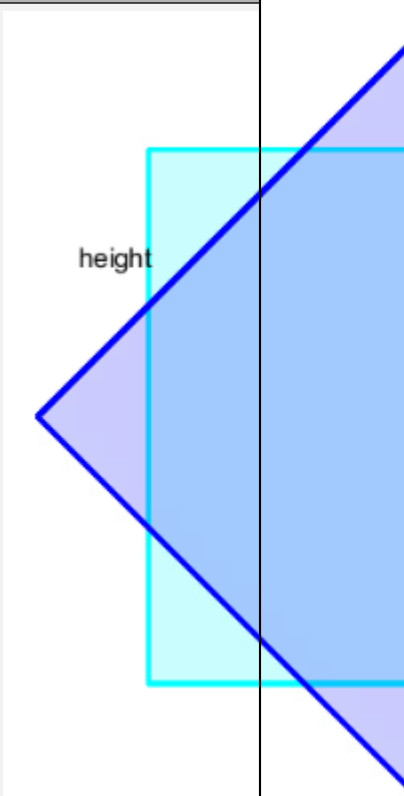
Data Types: `char` | `string`

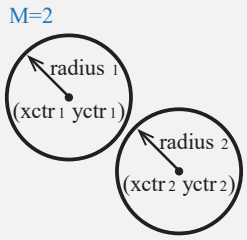
`position` — Position of shape

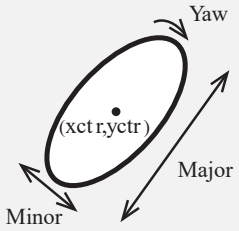
matrix | vector | cell array

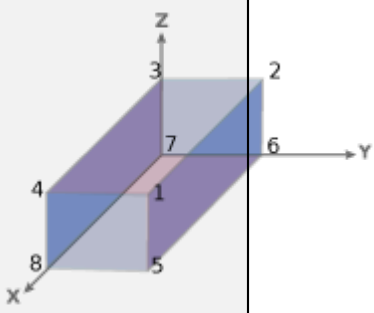
Position of the shape, specified according to the type of shape, as described in the table.

Shape	Position	Definition
`rectangle`	For one or more axis-aligned rectangles or filled rectangles, specify as an M-by-4 numeric matrix, where each row specifies a rectangle of the form $[x\ y\ width\ height]$. • M is the number of axis-aligned rectangles. • x and y specify the upper-left corner of the rectangle. • w specifies the width of the rectangle, which is its length along the x-axis. • h specifies the height of the rectangle, which is its length along the y-axis. $\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$	M=2 

Shape	Position	Definition
	For one or more rotated rectangles, specify in spatial coordinates as an *M*-by-5 numeric matrix, where each row specifies a rotated rectangle of the form [*xctr yctr w h yaw*]. • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the rectangle.	

Shape	Position	Definition
`circle`	For one or more circles, specify spatial coordinates as an *M*-by-3 numeric matrix, where each row specifies a circle of the form [*xctr yctr radius*]. • *M* is the number of circles. • *xctr* and *yctr* specify the center of the circle. • *radius* specifies the radius of the circle. $\begin{bmatrix} xctr_1 & yctr_1 & radius_1 \\ xctr_2 & yctr_2 & radius_2 \\ \vdots & \vdots & \vdots \\ xctr_M & yctr_M & radius_M \end{bmatrix}$	

Shape	Position	Definition
`ellipse`	For one or more ellipses, specify in spatial coordinates as an *M*-by-5 matrix, where each row specifies an ellipse of the form [*xctr yctr major minor yaw*] • *M* is the number of ellipses. • *xctr* and *yctr* specify the center of the ellipse. • *major* and *minor* are the lengths of the major and minor ellipse axes, respectively. • *yaw* is the angle of rotation in degrees. The angle of rotation is clockwise positive around the center of the ellipse.	

Shape	Position	Definition
`projected-cuboid`	For one or more projected cuboids, specify in spatial coordinates as an 8-by-2-by-*M* array or an *M*-by-8 matrix, where *M* is the number of projected cuboids. When specified as an 8-by-2-by-*M* array, each row must contain the [*x y*] location of a projected cuboid vertex. The vertices connect to form a cuboid with six faces. The order of the input vertices must match the order shown in the diagram. When specified as an *M*-by-8 matrix, each row specifies the dimensions of the front-facing and rear-facing sides of a projected cuboid in the form, [*x1 y1 w1 h1 x2 y2 w2 h2*], where [*x1 y1*] and [*x2 y2*] specify the upper-left coordinates of the front-facing and rear-facing sides, respectively, and [*w1 h1*] and [*w2 h2*] specify the corresponding widths and heights.	

Data Types: `single` | `double` | `cell` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`label` — Label to associate with shape

M-element numeric vector | string array | vector of categorical labels | cell array of ASCII character vectors

Label to associate with a shape, specified as an *M*-element numeric vector, string array, vector of categorical labels, or a cell array of ASCII character vectors. If you use character vectors, string scalars, or categorical labels, you must encode them as ASCII characters. If you specify a cell array, it must be equal in length to the number of specified shape positions. You can specify a single label for all shapes using a numeric scalar, string scalar, or scalar categorical label.

Example: `[5 10]` marks the first shape with the label `5`, and the second shape with the label `10`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `insertObjectAnnotation(I,"rectangle",position,label,FontSize=18);` sets the font size to use for inserting annotations to `18`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `insertObjectAnnotation(I,"rectangle",position,label,"FontSize","18");` sets the font size to use for inserting annotations to `18`.

`Font` — Font face of label text

`"Roboto-Regular"` (default) | character vector | string scalar

Font face of the label text, specified as a character vector or string scalar. The font face must be one of the available truetype fonts installed on your system. To get a list of available fonts on your system, use the `listTrueTypeFonts` function from the MATLAB command prompt.

Data Types: `char` | `string`

`FontSize` — Label text font size
`12` (default) | integer in range `[8 200]`

Label text font size, specified as an integer that corresponds to points in the range of `[8 200]`.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FontColor` — Font color
`"black"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

Font color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each font string or one color for all strings. To specify one color for all strings, specify `FontColor` as a color name or an [*R G B*] vector.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`AnnotationColor` — Annotation color
`"yellow"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

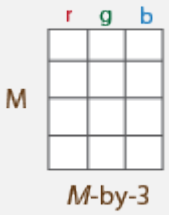
Annotation color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each marker or one color for all markers. To specify one color for all markers, specify `AnnotationColor` as a color name or an [*R G B*] vector.

Specification	Format	Example
Specify one color for all shapes (or markers)	Short color name or color name	`"r"` `"red"`
	RGB triplet	 `[1 0 0]`
Specify a color for each shape (or marker)	Vector of color names	`["red","yellow","blue"]`

Specification	Format	Example

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

Supported colors are listed in the table.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `cell` | `char` | `uint8` | `uint16` | `int16` | `double` | `single`

`TextBoxOpacity` — Opacity of text label box background
`0.6` (default) | scalar in the range of `[0 1]`

Opacity of the text label box background, specified as a scalar in the range of `[0, 1]`. A value of `0` renders the background of the label text box as fully transparent, while a value of `1` renders it as fully opaque.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`LineWidth` — Shape border line width
`1` (default) | positive integer

Shape border line width, specified as a positive integer, in pixels.

`ShowOrientation` — Original orientation indicator
`0` (`false`) (default) | `1` (`true`)

Original orientation indicator, specified as a numeric or logical `1` (`true`) or `0` (`false`). A value of `1` (`true`) displays the original orientation of the rotated rectangle by using an arrow annotation on the

bounding box. The original orientation is defined as a zero rotation angle. A value of `0` (`false`) does not show the arrow annotation.

This argument applies only when `Shape` describes a rotated rectangle or an ellipse.

Data Types: `logical` | `integer`

Output Arguments

`RGB` — Output image
M-by-*N*-by-3 truecolor image

Output image, returned as an *M*-by-*N*-by-3 truecolor image.

Version History

Introduced in R2012b

R2026a: Default value of `Font` changed to `Roboto-Regular`
Behavior changed in R2026a

Starting in R2026a, the default value of the `Font` name-value argument is `Roboto-Regular`. In previous releases, the default value was `LucidaSansRegular`.

R2024b: Support for ellipse shape

Added support for `"ellipse"` to visualize one or more ellipses on top of an image or on video data.

R2024b: Default value for orientation indicator changed from `true` to `false`
Behavior changed in R2024b

Starting in R2024b, the default value for the `ShowOrientation` name-value argument will be `false`.

R2023b: Specify RGB color in the range [0,1]
Behavior changed in R2023b

Starting in R2023b, you can specify marker color with RGB values in the range [0,1].

R2022b: Visualize projected cuboids

Added support to visualize projected cuboids.

R2023b: Rotated rectangle support for bounding box

You can specify a rotated rectangle bounding box using the form [*xctr yctr width height yaw*].

R2023b: Show orientation of rectangle of bounding box

You can now specify the `ShowOrientation` name-value argument to display the orientation and angle of a rotated rectangle bounding box.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Input image must be bounded. See “Specify Upper Bounds for Variable-Size Arrays in Generated Code” (MATLAB Coder).
- Code generation requires a C++ compiler. For a list of supported compilers, see Supported and Compatible Compilers

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`insertMarker` | `insertText` | `insertShape` | `insertObjectMask` | `cuboid2img`

Topics

“Choose Function to Visualize Detected Objects”

insertObjectMask

Insert masks in image or video stream

Syntax

```
RGB = insertObjectMask(I,BW)
RGB = insertObjectMask(I,maskstack)
RGB = insertObjectMask( ___ ,Name=Value)
```

Description

`RGB = insertObjectMask(I,BW)` inserts a mask `BW` into the specified image `I` and returns the result as a truecolor image `RGB`.

`RGB = insertObjectMask(I,maskstack)` inserts a set of masks `maskstack` into the specified image `I` and returns the result as a truecolor image `RGB`.

`RGB = insertObjectMask( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `(MaskColor="red")` sets the color of the mask to red.

Examples

Insert Masks with White Borders

Read an image into the workspace.

```
I = imread("visionteam1.jpg");
```

Load a stack of binary masks into the workspace.

```
load("visionteam1Maskstack.mat")
```

Insert the masks with white outlines into the image.

```
RGB = insertObjectMask(I,maskstack,LineColor="white",LineWidth=2);
```

Display the image with the masks inserted.

```
figure
imshow(RGB)
```



Insert Masks and Assign Unique Color to Each

Read an image into the workspace.

```
I = imread("visionteam1.jpg");
```

Load a stack of binary mask images.

```
load("visionteam1Maskstack.mat");
```

Insert the masks into the image and specify a unique color for each mask.

```
numMasks = size(maskstack,3);
RGB = insertObjectMask(I,maskstack,MaskColor=lines(numMasks));
```

Display the image with the inserted masks.

```
figure
imshow(RGB)
```



Input Arguments

`I` — Input image

truecolor image | grayscale image

Input image, specified as a truecolor (RGB) image, *m*-by-*n*-by-3 array, or a grayscale image, *m*-by-*n* array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`BW` — Input mask image

m-by-*n* logical matrix

Input mask image, specified as an *m*-by-*n* logical matrix.

Data Types: `logical`

`maskstack` — Stack of mask images

m-by-*n*-by-*P* logical array

Stack of mask images, specified as an *m*-by-*n*-by-*P* logical array, where *P* is the total number of masks in the stack.

`maskstack` has the same width and height as `I`.

Data Types: `logical`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `LineColor="white"` set the inserted line color to white.

`MaskColor` — Mask color

`"green"` (default) | character vector | cell array of character vectors | string scalar | string array | *M*-by-3 matrix

Mask color, specified as a character vector, cell array of character vectors, vector, or *M*-by-3 matrix of RGB values.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`, for example, `[0.4 0.6 0.7]`. When using a data type of `uint8`, which is defined within the range of `[0,255]`, conversions are necessary to ensure that the expected range is met. For example, to specify the previous vector, you must convert it by using `uint8(255*[0.4 0.6 0.7])`.

You can specify a different color for each mask or one color for all masks. To specify one color for all masks, set `MaskColor` to a color string or an [*R G B*] vector.

Specification	Format	Example

Specify one color for all shapes (or markers)	Short color name or color name	`"r"`
		
		`"red"`

	RGB triplet	 `[1 0 0]`
Specify a color for each shape (or marker)	Vector of color names	`["red","yellow","blue"]`

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

Supported colors are listed in the table.

Color Name	Short Name	RGB Triplet	Appearance
"red"	"r"	[1 0 0]	
"green"	"g"	[0 1 0]	
"blue"	"b"	[0 0 1]	
"cyan"	"c"	[0 1 1]	
"magenta"	"m"	[1 0 1]	
"yellow"	"y"	[1 1 0]	
"black"	"k"	[0 0 0]	
"white"	"w"	[1 1 1]	

Color values must be specified in the range of the data type support that you use to specify them. For example, if you specify an input of `double` datatype, then values must be in the range `[0,1]`. If you specify an input of `uint8` datatype, then values must be in the range `[0 255]`. For example, `[0,0,1]` and `uint8([0,0,255])` are two formats for the same color blue.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`Opacity` — Opacity of mask
`0.6` (default) | scalar in the range [0 1]

Opacity of mask, specified as a scalar value in the range [0 1]. The value `1` makes the mask completely opaque and the value `0` makes the mask completely transparent.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`LineColor` — Color of mask borders
`"auto"` (default) | RGB triplet | ColorSpec

Color of mask borders, specified as one of these values:

- `"auto"` — `LineColor` uses the same value or values as `MaskColor`.
- *P*-by-3 matrix of RGB triplets, where *P* is the total number of masks
- *P*-element vector of MATLAB ColorSpec names
- 1-by-3 RGB triplet or scalar MATLAB ColorSpec name, specifying the color to use for all masks.

  For a description of RGB color and how to specify it, see `MaskColor`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical` | `char` | `string`

`LineOpacity` — Opacity of mask borders
1 (default) | 0

Opacity of the mask borders, specified as a scalar value in the range of [0 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`LineWidth` — Width of mask borders
1 (default) | positive scalar

Width of mask borders, specified as a positive scalar in pixels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`RGB` — Output image
truecolor image

Output image, returned as a truecolor image of same datatype as the input image, with the same *m*-by-*n* dimensions as `I`.

Tips

- When masks overlap and the same pixel is in more than one mask, `maskstack(:,:,i)` takes precedence over `maskstack(:,:,j)`, where `i < j`.
- For better performance, set `LineOpacity` to `0` to disable drawing of the edges.

Version History

Introduced in R2020b

R2023b: Specify RGB color in the range [0,1]
Behavior changed in R2023b

Starting in R2023b, you can specify color with RGB values in the range [0,1].

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The value of the `LineWidth` parameter must be a compile-time constant.
- The values of the `MaskColor` and `LineColor` name-value arguments must be numeric. The MATLAB ColorSpec is not supported with code generation.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`insertObjectAnnotation` | `insertShape` | `labeloverlay` | `showShape`

Topics
“Choose Function to Visualize Detected Objects”

insertObjectInImage

Insert object from source image into image

Syntax

```
[newImage,boundingBox,newMask] = insertObjectInImage(destinationImage,
sourceImage,mask)
[newImage,boundingBox,newMask] = insertObjectInImage( ___ ,Name=Value)
```

Description

`[newImage,boundingBox,newMask] = insertObjectInImage(destinationImage, sourceImage,mask)` inserts an object from a source image, `sourceImage`, into a destination image, `destinationImage`. By default, the `insertObjectInImage` function inserts an object from the source image into a randomized location on the destination image. The function returns the result as an augmented image, `newImage`, a bounding box, `boundingBox`, and a corresponding object mask, `newMask`.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[newImage,boundingBox,newMask] = insertObjectInImage( ___ ,Name=Value)` specifies options using one or more name-value arguments, in addition to the input arguments from the previous syntax. For example, `BoundaryConstraintMode="inbounds-exact"` specifies to insert objects only when they are completely within the destination image.

Examples

Create Annotated Synthetic Image

Read the destination image into the workspace and display the image.

```
destImage = imread("parkroad.jpg");
imshow(destImage)
```



Read the source image containing the object, the small dog, into the workspace and display the image.

```
sourceImage = imread("DogTrio.jpg");
imshow(sourceImage)
```



Load the binary object mask into the workspace.

```
load("dogMask.mat");
```

Insert the object from the source image into a randomized location on the destination image. Generate the augmented synthetic image, the object mask, and the corresponding bounding box.

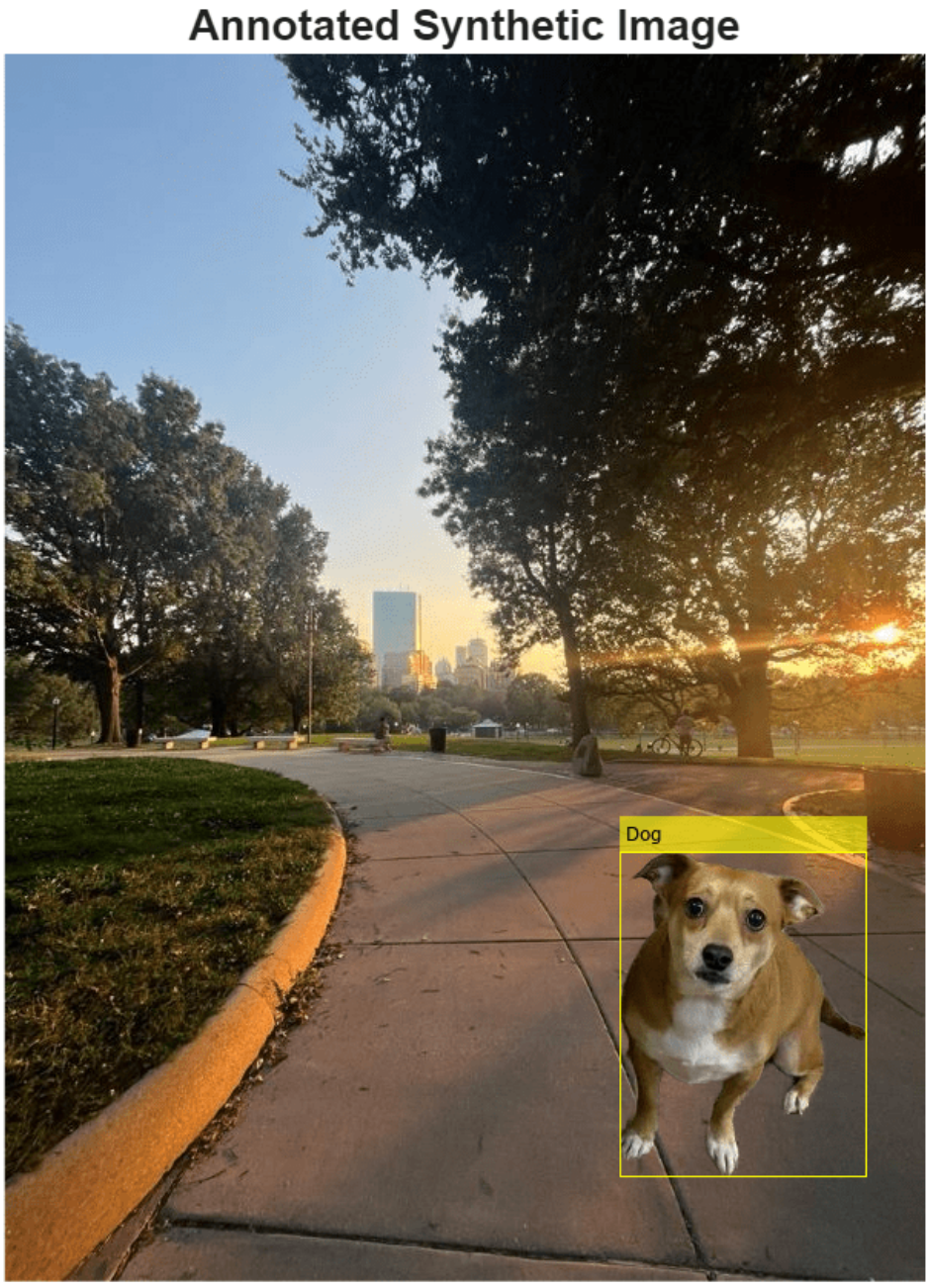
```
[newImage,boundingBox,newMask] = insertObjectInImage(destImage,sourceImage,dogMask);
```

Annotate the image at the position specified by the generated bounding boxes using the `insertObjectAnnotation` function.

```
annotedNewImage = insertObjectAnnotation(newImage,"rectangle",boundingBox,"Dog");
```

Display the annotated synthetic image.

```
figure
imshow(annotedNewImage)
title("Annotated Synthetic Image")
```


Annotated Synthetic Image
Dog

Input Arguments

`destinationImage` — Destination image

H-by-*W* matrix | *H*-by-*W*-by-3 numeric array

Destination image into which to insert an object, specified as one of these options:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
Binary	*H*-by-*W* logical matrix
RGB	*H*-by-*W*-by-3 numeric array

`sourceImage` — Source image

H-by-*W* matrix | *H*-by-*W*-by-3 numeric array

Source image containing an object to insert into the destination image, specified as one of these options:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
Binary	*H*-by-*W* logical matrix
RGB	*H*-by-*W*-by-3 numeric array

`mask` — Object mask

H-by-*W* logical matrix

Object mask of the source image, specified as an *H*-by-*W* logical matrix. *H* and *W* are the height and width of the source image, `sourceImage`, respectively.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `insertObjectInImage(destinationImage,sourceImage,mask,BoundaryConstraintMode="inbounds-exact")` specifies to insert objects only when they are completely within the destination image.

`RegionConstraintMask` — Region constraint mask

`true(size(destinationImage,[1 2]))` (default) | *M*-by-*N* logical array

Region constraint mask, specified as an *M*-by-*N* logical array. *M* and *N* are the height and width of the destination image `destinationImage`, respectively.

Specify the `RegionConstraintMask` value as `true` (or `1`) for only select pixel locations to insert the object from the source image into a specific location on the destination image. By default, the `insertObjectInImage` function inserts the object at a randomized location anywhere in the destination image.

`BoundaryConstraintMode` — Boundary constraint mode
`"inbounds-bbox"` (default) | `"inbounds-exact"` | `"center"`

Boundary constraint mode, specified as `"inbounds-bbox"`, `"inbounds-exact"`, or `"center"`. This name-value argument specifies the method used to constrain objects at the boundaries of images in the destination image `destinationImage`.

Value	Boundary Constraint Mode
`"inbounds-bbox"`	To insert an object, the entire bounding box containing the object must be completely within the destination image.
`"inbounds-exact"`	To insert an object, the object mask must be completely within the destination image.
`"center"`	To insert an object, only the object centroid must be inside the destination image, so objects can be partially visible and clipped at the image boundary.

`ObjectsInSceneMasks` — Stack of mask images containing inserted objects
`false(size(destinationImage,[1 2]))` (default) | *M*-by-*N*-by-*P* logical matrix

Stack of mask images containing inserted objects, specified as an *M*-by-*N*-by-*P* logical array. *P* is the total number of masks in the stack, and *M* and *N* are the height and width of the destination image `destinationImage`, respectively. The value of `ObjectsInSceneMasks` is `true` (or `1`) at the pixel locations which already contain an inserted object. By default, there are no existing inserted objects in the image.

`ObjectsInSceneMaxOverlap` — Maximum object overlap
`1` (default) | positive scalar in range [0, 1]

Maximum object overlap between an object in a source image `sourceImage` and a previously inserted object, specified as a positive scalar in range [0, 1]. Specify the `ObjectsInSceneMaxOverlap` value as `1` to insert new objects on top of existing objects. Decrease this value to decrease the overlap between the newly inserted objects and existing objects.

`MaxInsertionAttempts` — Maximum insertion attempts
`10` (default) | positive scalar

Maximum insertion attempts to satisfy the overlap value, specified as a positive scalar. The overlap value is specified by the `ObjectsInSceneMaxOverlap` name-value argument. If no insertion location can be found on the destination image after `MaxInsertionAttempts` iterations, the `insertObjectInImage` function will not insert the object from the source image into the destination image.

`BlendMethod` — Blend method
`"guidedfilter"` (default) | `"poisson"` | `"none"`

Blend method to use for inserting objects from the source image into the destination image, specified as one of these options:

BlendMethod Value	Blend Method
"guidedfilter"	Guided filter blending [1] – Blur the object mask of the source image at the mask-object boundary. Guided image filtering uses edge context from the source image to filter the mask prior to blending the object image with the destination image.
"poisson"	Poisson blending [2] – Blend the source object into the destination image by matching the color distribution of the source object to that of the boundary between the destination image and source object mask.
"none"	No blending – At insertion locations of the destination image, replace pixels of the destination image by the source image pixel values.

`GeometricAugmentation` — Geometric augmentation
`[ ]` (default) | function handle

Geometric augmentation, specified as a function handle that returns an `affinetform2d` object. The geometric augmentation warps a source image and the mask associated with the object in the image before it is inserted into the destination image. For example, specify the `GeometricAugmentation` value as `@() randomAffine2d("Scale",[0.9,1.1])` to apply a randomized scale transformation to an inserted object.

Output Arguments

`newImage` — Augmented image containing an inserted object
M-by-*N* matrix | *M*-by-*N*-by-*3* numeric array

Augmented image containing an inserted object, returned as one of these options:

Image Type	Data Format
Grayscale	*M*-by-*N* numeric matrix
Binary	*M*-by-*N* logical matrix
RGB	*M*-by-*N*-by-*3* numeric array

Tip Specify the `RegionConstraintMask` name-value argument to specify the region in the destination image in which to insert the object. Otherwise, the `insertObjectInImage` function inserts the object in the destination image at a randomized location.

`boundingBox` — Axis-aligned bounding box enclosing mask of inserted object region
4-element row vector

Axis-aligned bounding box enclosing mask of inserted object region, returned as a 4-element row vector. The row vector is of the form [*x y w h*], where:

- *x* and `y` specify the upper-left corner of the rectangle.
- *w* specifies the width of the rectangle, which is its length along the *x*-axis.
- *h* specifies the height of the rectangle, which is its length along the *y*-axis.

`newMask` — Mask containing inserted object region
M-by-*N* logical array

Mask containing inserted object region, returned as an *M*-by-*N* logical array. *M* and *N* are the height and width of the destination image, `destinationImage`, respectively. The value of the logical array returned by `newMask` is `true` (or `1`) at the pixel locations where object content is inserted into the destination image from the source image.

Tip Specify the `RegionConstraintMask` name-value argument to specify the region in the destination image in which to insert the object and the object mask. Otherwise, the `insertObjectInImage` function inserts the object and its corresponding mask in the destination image at a randomized location.

Version History

Introduced in R2025a

R2026a: Option to specify exact object mask for boundary constraint mode

Specify to insert an object only when the object mask is completely within the destination image by using the `BoundaryConstraintMode` name-value argument. By default, the `insertObjectInImage` function inserts an object when the bounding box containing the object is entirely within the destination image.

References

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. “Guided Image Filtering.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, no. 6 (June 2013): 1397–1409. doi:10.1109/TPAMI.2012.213.

[2] Pérez, Patrick, Michel Gangnet, and Andrew Blake. “Poisson Image Editing.” In *ACM SIGGRAPH 2003 Papers*, 313–18. San Diego California: ACM, 2003. doi:10.1145/1201775.882269.

See Also

`objectInsertionDatastore` | `insertObjectMask` | `imblend` | `affinetform2d` | `imguidedfilter`

Topics
“Get Started with Image Preprocessing and Augmentation for Deep Learning”

insertShape

Insert shapes in image or video

Syntax

```
RGB = insertShape(I,shape,position)
RGB = insertShape( ___ ,Name=Value)
```

Description

`RGB = insertShape(I,shape,position)` insert the specified `shape` into the truecolor or grayscale image `I`. The function draws the shapes by overwriting pixel values, and returns a truecolor image.

`RGB = insertShape( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments from the input arguments from the previous syntax. For example, `LineWidth=5` sets the line width value to `5` of the inserted shapes.

Examples

Insert Circle and Filled Shapes onto Image

Read an image into the workspace.

```
I = imread("peppers.png");
```

Place a circle on the image with a border line width of 5 pixels.

```
RGB = insertShape(I,"circle",[150 280 35],LineWidth=5);
```

Place a filled triangle and a filled hexagon on the image.

```
pos_triangle = [183 297 302 250 316 297];
pos_hexagon = [340 163 305 186 303 257 334 294 362 255 361 191];
RGB = insertShape(RGB,"filled-polygon",{pos_triangle pos_hexagon}, ...
    ShapeColor=["white","green"],Opacity=0.7);
```

Display the resulting image.

```
imshow(RGB)
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor or an *M*-by-*N* grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`shape` — Type of shape

`"rectangle"` | `"filled-rectangle"` | `"line"` | `"polygon"` | `"filled-polygon"` | `"circle"` | `"filled-circle"` | `"projected-cuboid"` | `"ellipse"` | `"filled-ellipse"`

Type of shape, specified as `"rectangle"`, `"filled-rectangle"`, `"line"`, `"polygon"`, `"filled-polygon"`, `"circle"`, `"filled-circle"`, `"projected-cuboid"`, `"ellipse"`, or `"filled-ellipse"`.

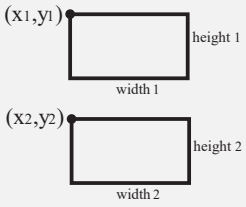
Data Types: `char` | `string`

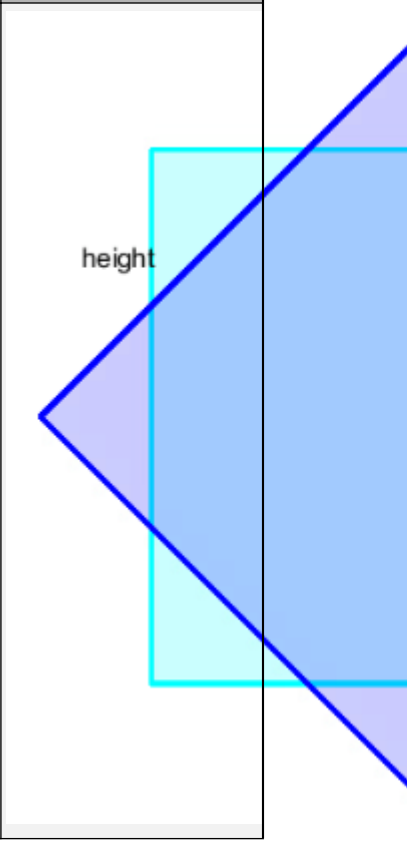
`position` — Position of shape

matrix | vector | cell array

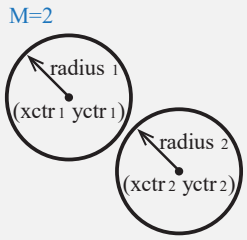
Position of shape, specified according to the type of shape, as described in the table.

Shape	Position	Definition

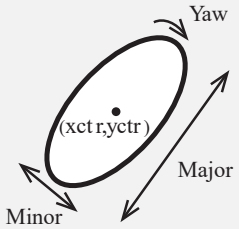
`rectangle`
 	
`filled-rectangle`
	For one or more axis-aligned rectangles or filled rectangles, specify as an M-by-4 numeric matrix, where each row specifies a rectangle of the form $[x\ y\ width\ height]$.
	
	• M is the number of axis-aligned rectangles.
	• x and y specify the upper-left corner of the rectangle.
	• w specifies the width of the rectangle, which is its length along the x-axis.
	• h specifies the height of the rectangle, which is its length along the y-axis.
	
	$\begin{bmatrix} x_1 & y_1 & width_1 & height_1 \\ x_2 & y_2 & width_2 & height_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & width_M & height_M \end{bmatrix}$
		M=2
		

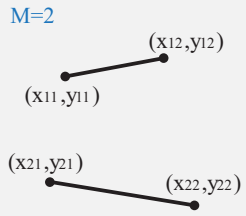
Shape	Position	Definition
	For one or more rotated rectangles, specify in spatial coordinates as an *M*-by-5 numeric matrix, where each row specifies a rotated rectangle of the form [*xctr yctr w h yaw*]. • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the rectangle.	

Shape	Position	Definition

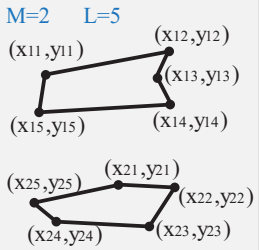
`circle`
 	
`filled-circle`
	For one or more circles, specify spatial coordinates as an *M*-by-3 numeric matrix, where each row specifies a circle of the form [*xctr yctr radius*].
	• *M* is the number of circles.
	• *xctr* and *yctr* specify the center of the circle.
	• *radius* specifies the radius of the circle.
	$\begin{bmatrix} xctr_1 & yctr_1 & radius_1 \\ xctr_2 & yctr_2 & radius_2 \\ \vdots & \vdots & \vdots \\ xctr_M & yctr_M & radius_M \end{bmatrix}$
		

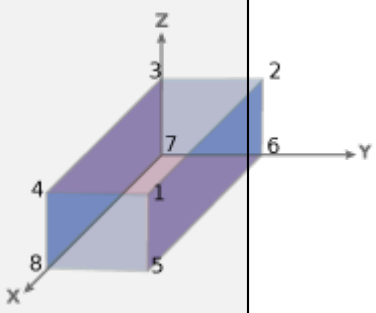
Shape	Position	Definition

`ellipse`
 	
`filled-ellipse`
	For one or more ellipses, specify in spatial coordinates as an *M*-by-5 matrix, where each row specifies an ellipse of the form [*xctr yctr major minor yaw*]
	• *M* is the number of ellipses.
	• *xctr* and *yctr* specify the center of the ellipse.
	• *major* and *minor* are the lengths of the major and minor ellipse axes, respectively.
	• *yaw* is the angle of rotation in degrees. The angle of rotation is clockwise positive around the center of the ellipse.
		

Shape	Position	Definition
`line`	For one or more lines, specify spatial coordinates using one of these formats to specify a single line with connected endpoints, multiple lines with the same number of endpoints, or multiple lines with different numbers of endpoints. • Single lines with connected endpoints — Specify as a *P*-by-2 matrix, where each row specifies an endpoint in the form [*x y*], and the endpoint in one row connects directly to the endpoint in the next. *P* is the number of endpoints in the line. • Multiple lines with the same number of endpoints — Specify as an *M*-by-2*P* matrix, where each row specifies the consecutive endpoints of a single line in the form [*x1 y1 x2 y2 ... xp yp*]. Each [*x y*] pair is the coordinates of a single endpoint. *M* is the number of specified lines, and *P* is the number of endpoints per line. • Multiple lines with different numbers of endpoints — Specify as an *M*-by-1 cell array, where each cell contains the endpoints for a single line as a *P*-by-2 matrix or a 1-by-2*P* vector]. *M* is the number of specified lines, and *P* is the number of endpoints for each line	

Shape	Position	Definition

`polygon`
 	
`filled-polygon`
	For one or more polygons, specify in spatial coordinates as an M-by-1 cell array, where each cell contains an L-by-2 matrix of [x y] vertex locations or a 1-by-2L vector of consecutive vertex locations of the form [x_1, y_1, x_2,y_2, … x_L,y_L].
	• M is the number of polygons.
	• Each [x y] pair describes a vertex location or endpoint.
	• L is the number of vertices in a polygon or endpoints in a line. Each polygon or line can have a different number of vertices or endpoints.
		

Shape	Position	Definition
`projected-cuboid`	For one or more projected cuboids, specify in spatial coordinates as an 8-by-2-by-*M* array or an *M*-by-8 matrix, where *M* is the number of projected cuboids. When specified as an 8-by-2-by-*M* array, each row must contain the [*x y*] location of a projected cuboid vertex. The vertices connect to form a cuboid with six faces. The order of the input vertices must match the order shown in the diagram. When specified as an *M*-by-8 matrix, each row specifies the dimensions of the front-facing and rear-facing sides of a projected cuboid in the form, [*x1 y1 w1 h1 x2 y2 w2 h2*], where [*x1 y1*] and [*x2 y2*] specify the upper-left coordinates of the front-facing and rear-facing sides, respectively, and [*w1 h1*] and [*w2 h2*] specify the corresponding widths and heights.	

Data Types: `single` | `double` | `cell` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `insertShape(I,"circle",position,Color="yellow")` sets the shape color to yellow.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `insertShape(I,"circle",position,"Color","yellow")` sets the shape color to yellow.

`LineWidth` — Shape border line width
`1` (default) | positive integer

Shape border line width, specified as a positive integer, in pixels. This argument applies to only the `"Rectangle"`, `"Line"`, `"Polygon"`, and `"Circle"` shapes.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ShapeColor` — Shape color
`"green"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

Shape color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity

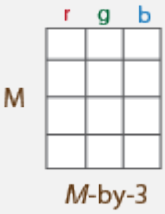
values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each shape or one color for all shapes. To specify one color for all markers, specify `ShapeColor` as a color name or an [*R G B*] vector.

Specification	Format	Example

Specify one color for all shapes (or markers)	Short color name or color name	`"r"`
		
		`"red"`

	RGB triplet	 `[1 0 0]`
Specify a color for each shape (or marker)	Vector of color names	`["red","yellow","blue"]`

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

This table lists the supported shape colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single` | `cell`

`Opacity` — Opacity of filled shape
`0.6` (default) | scalar in range [0,1]

Opacity of a filled shape, specified as a scalar in the range [0 1]. The `Opacity` argument applies to only the `"filled-rectangle"`, `"filled-polygon"`, and `"filled-circle"` shapes.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`SmoothEdges` — Smooth shape edges
`true` or `1` (default) | `false` or `0`

Smooth shape edges, specified as a logical `1` (`true`) or `0` (`false`). A `true` value enables an antialiasing filter to smooth shape edges. This value applies to only nonrectangular shapes. When you enable antialiasing requires additional time to draw the shapes.

Data Types: `logical`

Output Arguments

`RGB` — Output image
M-by-*N*-by-3 truecolor image

Output image, returned as an *M*-by-*N*-by-3 truecolor image.

Version History

Introduced in R2014a

R2024b: Support for ellipse shape

Added support for `"ellipse"` and `"filled-ellipse"` to visualize one or more ellipses on top of an image or on video data.

R2023b: Specify RGB color in the range [0,1]
Behavior changed in R2023b

Starting in R2023b, you can specify marker color with RGB values in the range [0,1].

R2022b: Visualize projected cuboids

Added support to visualize projected cuboids.

R2023b: Rotated rectangle support for bounding box

You can specify a rotated rectangle bounding box using the form [*xctr yctr width height yaw*].

R2023b: Show orientation of rectangle of bounding box

You can now specify the `ShowOrientation` name-value argument to display the orientation and angle of a rotated rectangle bounding box.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The input argument `shape` must be a compile-time constant.
- The `ShapeColor` and `SmoothEdges` name-value arguments must be compile-time constants.
- Code generation requires a C++ compiler. For a list of supported compilers, see Supported and Compatible Compilers

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`showShape` | `insertObjectAnnotation` | `insertObjectMask` | `insertMarker` | `insertText` | `cuboid2img`

Topics

“Draw Markers on an Image” on page 3-874
“Insert Text into Image” on page 3-919
“Choose Function to Visualize Detected Objects”

insertText

Insert text in image or video

Syntax

```
RGB = insertText(I,position,text)
RGB = insertText(I,position,numericValue)
RGB = insertText(___,Name=Value)
```

Description

`RGB = insertText(I,position,text)` insert text into the truecolor or grayscale image `I`. The function returns a truecolor image.

`RGB = insertText(I,position,numericValue)` inserts numeric values into the input image.

`RGB = insertText(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of input arguments from previous syntaxes. For example, `FontColor="red"` renders the inserted text or numeric values in red.

Examples

Insert Text into Image

Read the image into the workspace.

```
I = imread("board.tif");
```

Create text string that contain fractions.

```
text_str = cell(3,1);
conf_val = [85.212 98.76 78.342];
for ii = 1:3
   text_str{ii} = ['Confidence: ' num2str(conf_val(ii), '%0.2f') '%'];
end
```

Define the positions and colors of the text boxes.

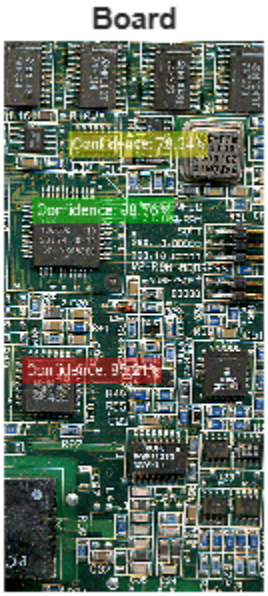
```
position = [23 373; 35 185; 77 107];
box_color = {"red","green","yellow"};
```

Insert the text strings with their respective box colors, and specify the font size, opacity, and text color of all strings.

```
RGB = insertText(I,position,text_str,FontSize=18,TextBoxColor=box_color, ...
    BoxOpacity=0.4,TextColor="white");
```

Display the image with annotations.

```
figure
imshow(RGB)
title("Board")
```



Insert Numeric Values into Image

Read the image into the workspace.

```
I = imread("peppers.png");
```

Define the numeric values and their *xy*-positions.

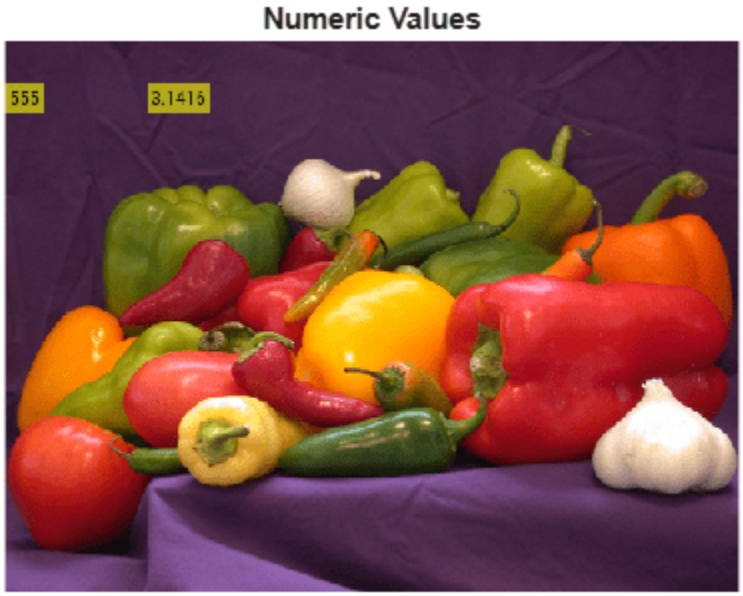
```
position =  [1 50; 100 50];
value = [555 pi];
```

Insert the numeric values using the bottom-left corner of each text box as its anchor point.

```
RGB = insertText(I,position,value,AnchorPoint="LeftBottom");
```

Display the image with the numeric values inserted.

```
figure
imshow(RGB)
title("Numeric Values")
```

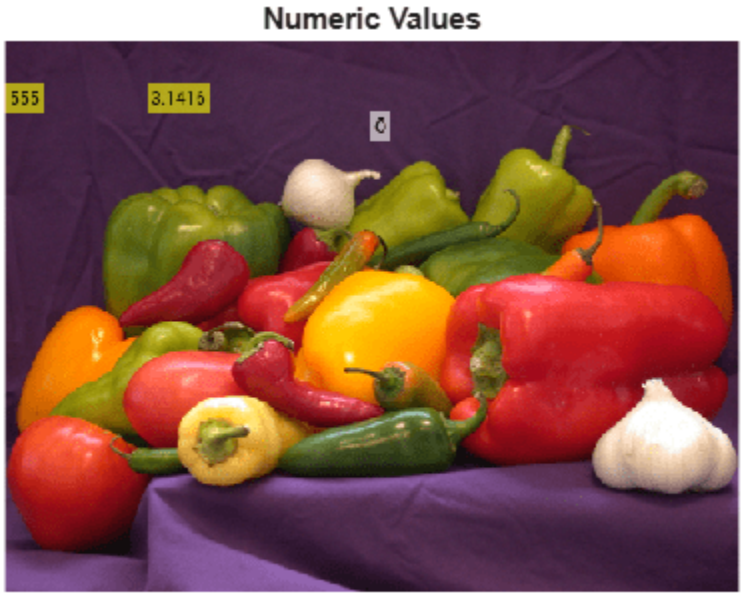


Insert the Unicode U+014C into the image.

```
OWithMacron = native2unicode([hex2dec("C5") hex2dec("8C")],"UTF-8");
RGB = insertText(RGB,[256 50],OWithMacron,font="Roboto-Regular",TextBoxColor="white");
```

Display the image with the Unicode character inserted.

```
figure
imshow(RGB)
title("Numeric Values");
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor or an *M*-by-*N* grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`text` — Unicode text character vector

character vector | string scalar | *M*-element cell array of character vectors | *M*-element string array

Unicode text, specified as a character vector, string scalar, *M*-element cell array of character vectors, or *M*-element string array. *M* is the number of specified text positions in `position`. The function overwrites pixels with the value of `text`. If you specify a single string or character vector, the function uses it for all positions in the `position` matrix. Most unicode fonts contain ASCII characters. You can display non-English and English characters, including English numeric values, with a single font.

Data Types: `char` | `string` | `cell`

`numericValue` — Numeric value text

numeric scalar | *M*-element numeric vector

Numeric value text, specified as a numeric scalar or an *M*-element numeric vector. *M* is the number of specified positions in `position`. If you specify a scalar value, the function uses that value for all specified positions. The function converts each numeric value to a character vector using the `sprintf` format '`%0.5g`'.

Data Types: `numeric`

`position` — Position of inserted text
M-by-2 matrix

Position of the inserted text, specified as *M*-by-2 matrix. *M* is the number of specified text positions, and each row specifies the *xy*-coordinates of the `AnchorPoint` of a text box.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `insertText(I,position,text,Font="Calibri")` sets the font face to `Calibri`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `insertText(I,position,text,"Font","Calibri")` sets the font face to `Calibri`.

`Font` — Font face of text
`"Roboto-Regular"` (default) | character vector | string scalar

Font face of the text, specified as a character vector or a string scalar. The font face must be one of the available truetype fonts installed on your system. To get a list of available fonts on your system, use the `listTrueTypeFonts` function from the MATLAB command prompt.

Data Types: `char` | `string`

`FontSize` — Font size
`12` (default) | positive integer in range [1, 200]

Font size, specified as a positive integer in the range [1, 200] in pixels.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FontColor` — Font color
`"green"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

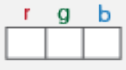
Font color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

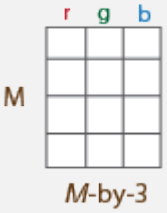
The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each font string or one color for all strings. To specify one color for all markers, specify `FontColor` as a color name or an [*R G B*] vector.

Specification	Format	Example

Specify one color for all shapes (or markers)	Short color name or color name	`"r"`
		
		`"red"`

	RGB triplet	 `[1 0 0]`
Specify a color for each shape (or marker)	Vector of color names	`["red","yellow","blue"]`

	Three-column matrix of RGB triplets	`[1 0 0`
		` 0 1 1`
		` 1 0 1`
		` 1 1 1]`
		

This table lists the supported marker colors.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`TextBoxColor` — Text box color

`"green"` (default) | short color name | color name | vector of color names | three-column matrix of RGB triplets

Text box color, specified as a short color name, color name, vector of color names, three-column matrix of RGB triplets.

The supported colors table lists RGB intensities in the range [0, 1], but you must specify RGB triplets in the range of your selected data type. For example, if specifying this argument as a matrix of `uint8` values, you must convert each intensity value to the range [0, 255]. To convert the listed intensity values to a `uint8` data type, use the code `uint8(255*intensity)`, where intensity is an RGB triplet value listed in the table.

You can specify a different color for each font string or one color for all strings. To specify one color for all strings, specify `FontColor` as a color name or an [*R G B*] vector.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`BoxOpacity` — Text box opacity
`0.6` (default) | scalar in the range of [`0` `1`]

Text box opacity of the text label box background, specified as a scalar in the range of `[0, 1]`. A value of `0` renders the background of the label text box as fully transparent, while a value of `1` renders it as fully opaque.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`AnchorPoint` — Text box reference point
`"LeftTop"` (default) | `"LeftCenter"` | `"LeftBottom"` | `"CenterTop"` | `"Center"` | `"CenterBottom"` | `"RightTop"` | `"RightCenter"` | `"RightBottom"`

Text box reference point, specified as one of these values:

- `"LeftTop"`
- `"LeftCenter"`
- `"LeftBottom"`
- `"CenterTop"`
- `"Center"`
- `"CenterBottom"`
- `"RightTop"`
- `"RightCenter"`
- `"RightBottom"`

The anchor point defines the relative location on each text box. You can position the text box specified by the corresponding row of `position`. For example, to place the center of the text box at the *xy*-coordinate specified by `position`, set `AnchorPoint` to `Center`.

Data Types: `char` | `string`

Output Arguments

`RGB` — Output image
M-by-*N*-by-3 truecolor image

Output image, returned as an *M*-by-*N*-by-3 truecolor image with the specified text inserted.

Limitations

- If you do not see a character in the output image, then the specified font does not contain the character. Select a different font. To get a list of available fonts on your system, at the MATLAB command prompt, enter `listTrueTypeFonts`.
- Increasing the font size also increases the preprocessing time and memory usage.
- The `insertText` function does not work for certain composite characters. For example, you cannot insert text when the rendering of one glyph corresponding to a character code influences the position, shape, or size of the adjacent glyph.

Version History

Introduced in R2013a

R2026a: Default value of `Font` changed to `Roboto-Regular`

Behavior changed in R2026a

Starting in R2026a, the default value of the `Font` name-value argument is `Roboto-Regular`. In previous releases, the default value was `LucidaSansRegular`.

R2023b: Specify RGB color in the range [0,1]

Behavior changed in R2023b

Starting in R2023b, you can specify marker color with RGB values in the range [0,1].

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `Font` and `FontSize` must be compile-time constants.
- Non-ASCII characters are not supported.
- Code generation requires a C++ compiler. For a list of supported compilers, see Supported and Compatible Compilers

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`insertObjectAnnotation` | `insertMarker` | `insertShape` | `listTrueTypeFonts`

Topics

“Draw Markers on an Image” on page 3-874
“Insert Circle and Filled Shapes onto Image” on page 3-907
“Choose Function to Visualize Detected Objects”

interp

Interpolate between transformations

Syntax

```
transformation0 = interp(transformation1,transformation2,points)
rotation0 = interp(rotation1,rotation2,points)
___  = interp( ___ ,transformation2,N)
```

Description

`transformation0 = interp(transformation1,transformation2,points)` interpolates at normalized positions `points` between transformations `transformation1` and `transformation2`.

The function interpolates rotations using a quaternion spherical linear interpolation, and linearly interpolates translations.

`rotation0 = interp(rotation1,rotation2,points)` interpolates at normalized rotations `points` between rotations `rotation1` and `rotation2`.

The function interpolates rotations using a quaternion spherical linear interpolation

`___  = interp( ___ ,transformation2,N)` interpolates `N` steps between the specified transformations or rotations.

Input Arguments

`transformation1` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects, where *N* is the total number of transformations. If you specify `transformation1` as an array, each element must be of the same type.

Either `transformation1` or `transformation2` must be a scalar transformation object of the same type. For example, if `transformation1` is an array of `se3` objects, `transformation2` must be a scalar `se3` object.

`transformation2` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects, where *N* is the total number of transformations. If you specify `transformation2` as an array, each element must be of the same type.

Either `transformation1` or `transformation2` must be a scalar transformation object of the same type. For example, if `transformation1` is an array of `se3` objects, `transformation2` must be a scalar `se3` object.

`rotation1` — First rotation
`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects, where *N* is the total number of rotations. If you specify `rotation1` as an array, each element must be of the same type.

Either `rotation1` or `rotation2` must be a scalar rotation object of the same type. For example, if `rotation1` is an array of `so3` objects, `rotation2` must be a scalar `so3` object.

`rotation2` — Last rotation
`so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects, where *N* is the total number of rotations. If you specify `rotation2` as an array, each element must be of the same type.

Either `rotation1` or `rotation2` must be a scalar rotation object of the same type. For example, if `rotation1` is an array of `se3` objects, `rotation2` must be a scalar `se3` object.

`points` — Normalized positions
N-element row vector of values in range `[0, 1]`

Normalized positions, specified as an *N*-element row vector of values in the range `[0, 1]`, where *N* is the total number of interpolated positions. Normalized positions `0` and `1` correspond to the first and last transformations or rotations, respectively.

Example: `interp(tf1,tf2,0.5)` interpolates a transformation halfway between `tf1` and `tf2`.

Example: `interp(r1,r2,0.5)` interpolates a rotation halfway between `r1` and `r2`.

`N` — Number of interpolated positions
positive integer

Number of interpolated positions, specified as a positive integer.

Example: `interp(tf1,tf2,5)` interpolates five transformations between transformations `tf1` and `tf2`.

Example: `interp(r1,r2,7)` interpolates seven rotations between rotations `r1` and `r2`.

Output Arguments

`transformation0` — Interpolated transformations
N-by-*M* matrix

Interpolated transformations, returned as an *N*-by-*M* matrix of the same transformation type as `transformation1` and `transformation2`, where *N* is the length of the longer argument between `transformation1` and `transformation2`, and *M* is the number of interpolated positions. Each row represents an interpolated transformation between `transformation1` and `transformation2`.

`rotation0` — Interpolated rotations
N-by-*M* matrix

Interpolated rotations, returned as an *N*-by-*M* matrix of the same rotation type as `rotation1` and `rotation2`, where *N* is the length of the longer argument between `rotation1` and `rotation2`, and

M is the number of interpolated positions. Each row represents an interpolated transformation between `rotation1` and `rotation2`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`dist` | `normalize` | `transform`

Objects

`se3` | `so3` | `quaternion`

isEpipoleInImage

Determine whether image contains epipole

Syntax

```
isIn = isEpipoleInImage(F,imageSize)
isIn = isEpipoleInImage(F',imageSize)
[isIn,epipole] = isEpipoleInImage( ___ )
```

Description

`isIn = isEpipoleInImage(F,imageSize)` determines whether the first stereo image associated with the fundamental matrix `F` contains an epipole. `imageSize` is the size of the first image, and is in the format returned by the function `size`.

`isIn = isEpipoleInImage(F',imageSize)` determines whether the second stereo image associated with the fundamental matrix `F'` contains an epipole.

`[isIn,epipole] = isEpipoleInImage( ___ )` also returns the epipole.

Examples

Determine Epipole Location in an Image

```
% Load stereo point pairs.
    load stereoPointPairs
    f = estimateFundamentalMatrix(matchedPoints1, matchedPoints2, 'NumTrials', 2000);
    imageSize = [200 300];

% Determine whether the image contains epipole and epipole location.
    [isIn,epipole] = isEpipoleInImage(f,imageSize)
```

```
isIn = logical
   1


epipole = 1×2

  256.5465  100.0140
```

Input Arguments

F — Fundamental matrix

3-by-3 matrix (default)

Fundamental matrix, specified as a 3-by-3 matrix computed from stereo images. `F` must be double or single. If P_1 represents a point in the first image I_1 that corresponds to P_2, a point in the second image I_2, then:

$[P_2,1] * F * [P_1,1]' = 0$

In computer vision, the fundamental matrix is a 3-by-3 matrix which relates corresponding points in stereo images. When two cameras view a 3-D scene from two distinct positions, there are a number of geometric relations between the 3-D points and their projections onto the 2-D images that lead to constraints between the image points. Two images of the same scene are related by epipolar geometry.

`imageSize` — Image size
row vector

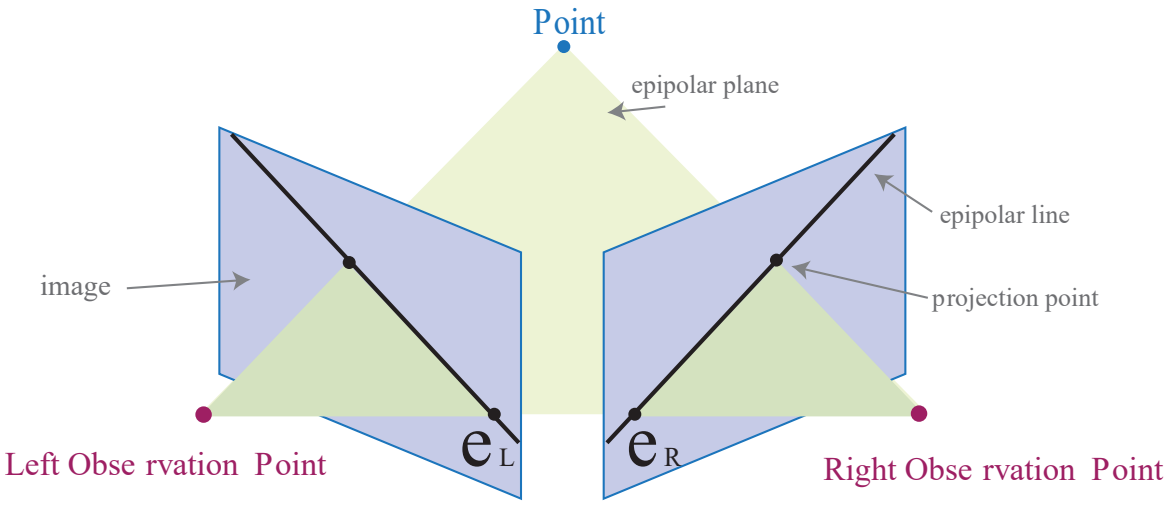
Image size, specified in the format returned by the `size` function.

Output Arguments

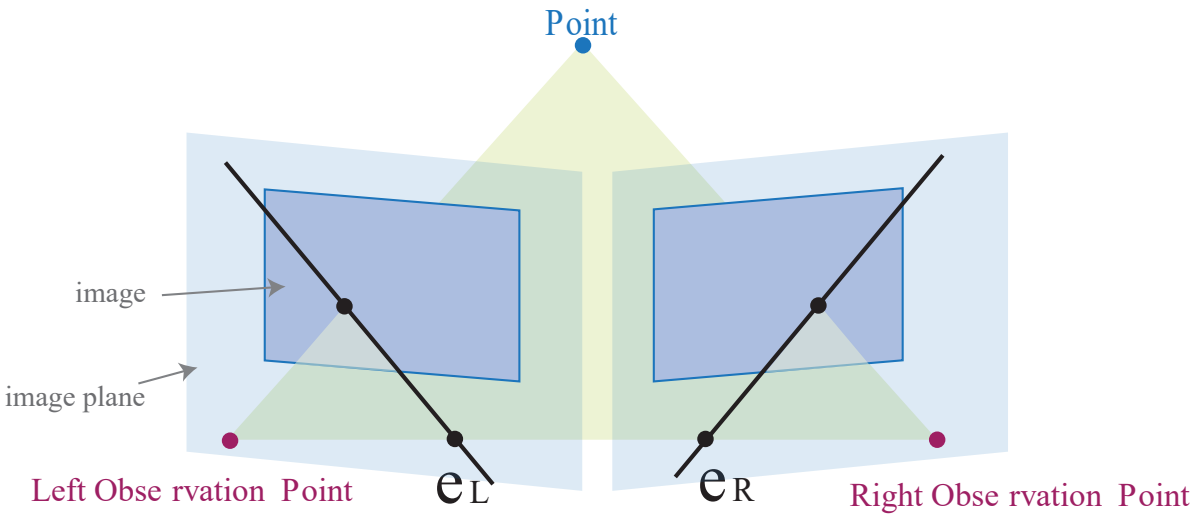
`isIn` — Valid epipole
logical

Valid epipole logical, specified as `true` when the image contains an epipole, and `false` when the image does not contain an epipole. When the image planes are at a great enough angle to each other, you can expect the epipole to be located in the image. When the image planes are at a more subtle angle to each other, you can expect the epipole to be located outside of the image, (but still in the image plane).

Epipole located in image.


Point
epipolar plane
epipolar line
image
projection point
e L
e R
Left Obse rvation Point
Right Obse rvation Point

Epipole located outside of image.


Point
image
image plane
Left Obse rvation Point
e L
e R
Right Obse rvation Point

`epipole` — Location of epipole
1-by-2 vector

Location of epipole, returned as a 1-by-2 vector.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`estimateFundamentalMatrix` | `epipolarLine` | `estimateUncalibratedRectification`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

isfilterseparable

Determine whether filter coefficients are separable

Syntax

```
isSeparable = isfilterseparable(H)
[isSeparable,hcol,hrow] = isfilterseparable(H)
```

Description

`isSeparable = isfilterseparable(H)` returns logical `1` (true) when the filter kernel `H` is separable, and `0` (false) otherwise.

`[isSeparable,hcol,hrow] = isfilterseparable(H)` also returns the vertical coefficients `hcol` and horizontal coefficients `hrow` when the filter kernel, `H`, is separable. Otherwise, `hcol` and `hrow` are empty.

Examples

Determine If Gaussian Filter Is Separable

Determine if the Gaussian filter created using the `fspecial` function is separable.

Create a Gaussian filter.

```
twoDimensionalFilter = fspecial('gauss');
```

Test the filter.

```
[isseparable,hcol,hrow] = isfilterseparable(twoDimensionalFilter)
```

```
isseparable = logical
   1


hcol = 3×1

   -0.1065
   -0.7870
   -0.1065


hrow = 1×3

   -0.1065   -0.7870   -0.1065
```

Input Arguments

`H` — Filter kernel

2-D numeric matrix | 2-D logical matrix

Filter kernel, specified as a non-sparse 2-D numeric or logical matrix.

Output Arguments

`isSeparable` — Filter is separable

`1` (true) | `0` (false)

Filter is separable, returned as logical `1` (true) when the filter is separable and `0` (false) when the filter is not separable.

Data Types: `logical`

`hcol` — Vertical coefficients

numeric vector | `[]`

Vertical coefficients when the filter kernel `H` is separable, returned as a numeric vector. When `H` is not separable, `hcol` is empty. If `H` is of data type `single`, then `hcol` is also of data type `single`. Otherwise, `hcol` is of data type `double`.

`hrow` — Horizontal coefficients

numeric vector | `[]`

Horizontal coefficients when the filter kernel `H` is separable, returned as a numeric vector. When `H` is not separable, `hrow` is empty. If `H` is of data type `single`, then `hrow` is also of data type `single`. Otherwise, `hrow` is of data type `double`.

More About

Separable two dimensional filters

Separable two-dimensional filters reflect the outer product of two vectors. Separable filters help reduce the number of calculations required.

A two-dimensional convolution calculation requires a number of multiplications equal to the *width* × *height* for each output pixel. The general case equation for a two-dimensional convolution is:

$$Y(m,n) = \sum_k \sum_l H(k,l)U(m-k,n-l)$$

If the filter H is separable then,

$$H(k,l) = H_{row}(k)H_{col}(l)$$

Shifting the filter instead of the image, the two-dimensional equation becomes:

$$Y(m,n) = \sum_k H_{row}(k) \sum_l H_{col}(l)\, U(m-k,n-l)$$

This calculation requires only (width + height) number of multiplications for each pixel.

Algorithms

The `isfilterseparable` function uses the singular value decomposition `svd` function to determine the rank of the matrix.

Version History

Introduced in R2006a

See Also

2-D FIR Filter | `svd` | `rank`

External Websites

MATLAB Central — Separable Convolution

lineToBorderPoints

Intersection points of lines in image and image border

Syntax

```
points = lineToBorderPoints(lines,imageSize)
```

Description

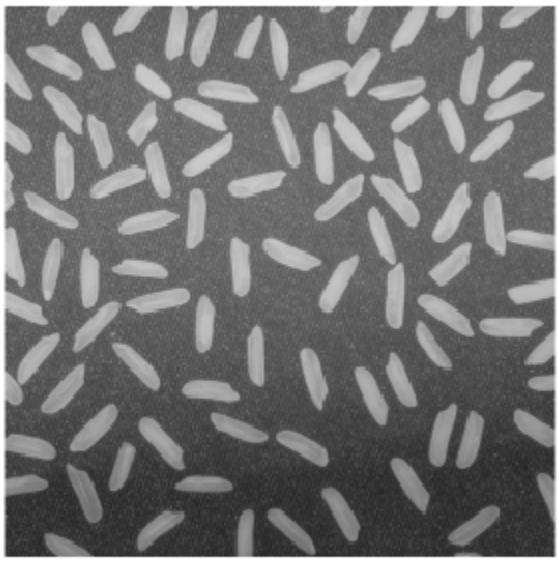
`points = lineToBorderPoints(lines,imageSize)` computes the intersection points between one or more lines in an image with the image border.

Examples

Find Intersection Points Between a Line and Image Border

Load and display an image.

```
I = imread('rice.png');
figure;
imshow(I);
hold on;
```



Define a line with the equation, 2 * x + y - 300 = 0.

```
aLine = [2,1,-300];
```

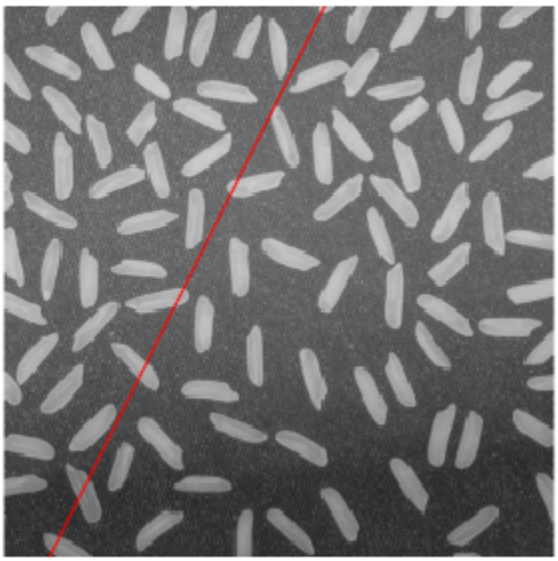
Compute the intersection points of the line and the image border.

```
points = lineToBorderPoints(aLine,size(I))
```

```
points = 1×4

  149.7500    0.5000   21.7500  256.5000
```

```
line(points([1,3]),points([2,4]),Color="r");
```



Input Arguments

`lines` — Line matrix

M-by-3 matrix (default)

Line matrix, specified as an *M*-by-3 matrix, where each row must be in the format, [*A*,*B*,*C*]. This matrix corresponds to the definition of the line:
$A * x + B * y + C = 0$.
M represents the number of lines.

`lines` must be `double` or `single`.

`imageSize` — Image size

integer (default) | row vector

Image size, specified as a row vector in the format returned by the `size` function.

Output Arguments

`points` — Intersection points

M-by-4 matrix

Intersection points, returned as an *M*-by-4 matrix. The function returns the matrix in the format of $[x_1, y_1, x_2, y_2]$. In this matrix, $[x_1\ y_1]$ and $[x_2\ y_2]$ are the two intersection points. When a line in the image and the image border do not intersect, the function returns `[-1,-1,-1,-1]`.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`size` | `line` | `epipolarLine`

matchFeatures

Find matching features

Syntax

```
indexPairs = matchFeatures(features1,features2)
[indexPairs,matchmetric] = matchFeatures(features1,features2)
[indexPairs,matchmetric] = matchFeatures(features1,features2,Name=Value)
```

Description

`indexPairs = matchFeatures(features1,features2)` returns indices of the matching features in the two input feature sets. The input feature must be either `binaryFeatures` objects or matrices.

`[indexPairs,matchmetric] = matchFeatures(features1,features2)` also returns the distance between the matching features, indexed by `indexPairs`.

`[indexPairs,matchmetric] = matchFeatures(features1,features2,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `matchFeatures(__,Method="Exhaustive")` sets the matching method to `Exhaustive`.

Examples

Find Corresponding Interest Points Between Pair of Images

Find corresponding interest points between a pair of images using local neighborhoods and the Harris algorithm.

Read the stereo images.

```
I1 = im2gray(imread("viprectification_deskLeft.png"));
I2 = im2gray(imread("viprectification_deskRight.png"));
```

Find the corners.

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
```

Extract the neighborhood features.

```
[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);
```

Match the features.

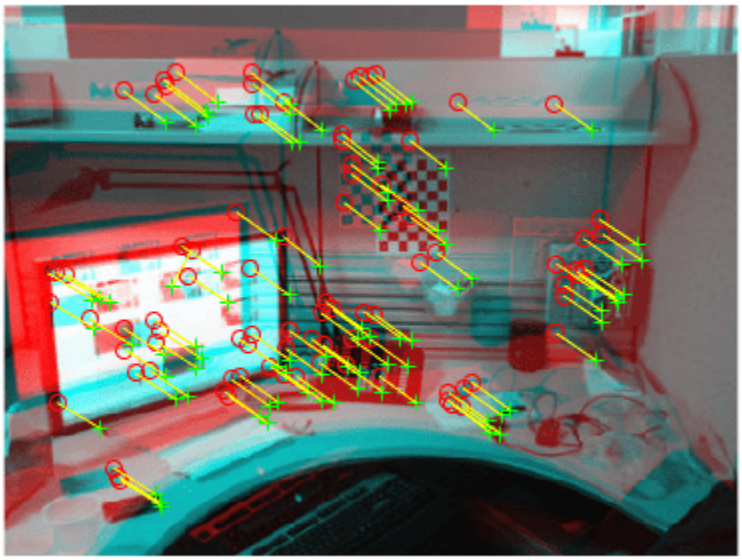
```
indexPairs = matchFeatures(features1,features2);
```

Retrieve the locations of the corresponding points for each image.

```
matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);
```

Visualize the corresponding points. You can see the effect of translation between the two images despite several erroneous matches.

```
figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
```



Find Corresponding Points Using SURF Features

Read the two images.

```
I1 = imread("cameraman.tif");
I2 = imresize(imrotate(I1,-20),1.2);
```

Find the SURF features.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract the features.

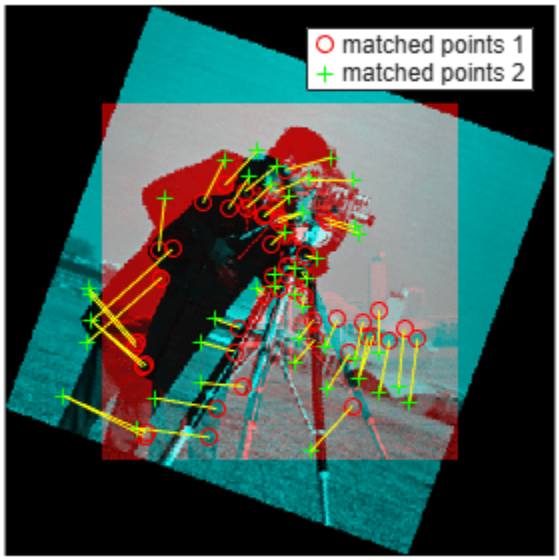
```
[f1,vpts1] = extractFeatures(I1,points1);
[f2,vpts2] = extractFeatures(I2,points2);
```

Retrieve the locations of matched points.

```
indexPairs = matchFeatures(f1,f2) ;
matchedPoints1 = vpts1(indexPairs(:,1));
matchedPoints2 = vpts2(indexPairs(:,2));
```

Display the matching points. The data still includes several outliers, but you can see the effects of rotation and scaling on the display of matched features.

```
figure; showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
legend("matched points 1","matched points 2");
```



Input Arguments

`features1` — Feature set 1

`binaryFeatures` object | *M1*-by-*N* matrix | point feature object

Features set 1, specified as a `binaryFeatures` object, an *M1*-by-*N* matrix, or as one of the point feature objects described in “Point Feature Types”. The matrix contains *M1* features, and *N* corresponds to the length of each feature vector. You can obtain the `binaryFeatures` object using the `extractFeatures` function with the fast retina keypoint (FREAK), Oriented FAST and Rotated BRIEF (ORB), or binary robust invariant scalable keypoints (BRISK) descriptor method.

`features2` — Feature set 2

M2-by-*N* matrix | `binaryFeatures` object | point feature object

Features set 2, specified as a `binaryFeatures` object, an *M2*-by-*N* matrix, or as one of the point feature objects described in “Point Feature Types”. The matrix contains *M2* features, and *N* corresponds to the length of each feature vector. You can obtain the `binaryFeatures` object using the `extractFeatures` function with the fast retina keypoint (FREAK), Oriented FAST and Rotated BRIEF (ORB), or binary robust invariant scalable keypoints (BRISK) descriptor method.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Method="Exhaustive"` sets the matching method to `Exhaustive`

`Method` — Matching method
`"Exhaustive"` (default) | `"Approximate"`

Matching method, specified as `"Exhaustive"` or `"Approximate"`. The method specifies how nearest neighbors between `features1` and `features2` are found. Two feature vectors match when the distance between them is less than the threshold set by the `MatchThreshold` parameter.

`"Exhaustive"`	Compute the pairwise distance between feature vectors in `features1` and `features2`.
`"Approximate"`	Use an efficient approximate nearest neighbor search. Use this method for large feature sets. [3]

`MatchThreshold` — Matching threshold
`10.0` or `1.0` (default) | percent value in the range (0, 100]

Matching threshold, specified as a scalar percent value in the range (0,100]. The default values are set to either `10.0` for binary feature vectors or to `1.0` for nonbinary feature vectors. You can use the match threshold for selecting the strongest matches. The threshold represents a percent of the distance from a perfect match.

Two feature vectors match when the distance between them is less than the threshold set by `MatchThreshold`. The function rejects a match when the distance between the features is greater than the value of `MatchThreshold`. Increase the value to return more matches.

Inputs that are `binaryFeatures` objects typically require a larger value for the match threshold. The `extractFeatures` function returns the `binaryFeatures` objects when extracting FREAK, ORB, or BRISK descriptors.

`MaxRatio` — Ratio threshold
`0.6` (default) | ratio in the range (0,1]

Ratio threshold, specified as a scalar ratio value in the range (0,1]. Use the max ratio for rejecting ambiguous matches. Increase this value to return more matches.

`Metric` — Feature matching metric
`"SSD"` (default) | `"SAD"`

Feature matching metric, specified as `"SAD"` or `"SSD"`.

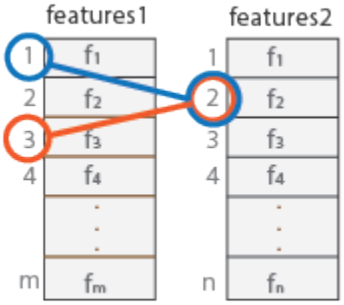
`"SAD"`	Sum of absolute differences
`"SSD"`	Sum of squared differences

This property applies when the input feature sets, `features1` and `features2`, are not `binaryFeatures` objects. When you specify the features as `binaryFeatures` objects, the function uses the Hamming distance to compute the similarity metric.

`Unique` — Unique matches
`false` (default) | `true`

Unique matches, specified as `false` or `true`. Set this value to `true` to return only unique matches between `features1` and `features2`.

When you set `Unique` to `false`, the function returns all matches between `features1` and `features2`. Multiple features in `features1` can match to one feature in `features2`.

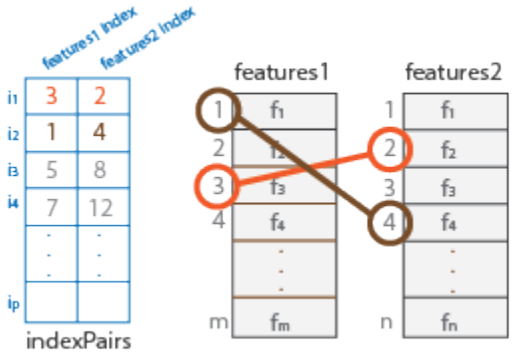


When you set `Unique` to `true`, the function performs a forward-backward match to select a unique match. After matching `features1` to `features2`, it matches `features2` to `features1` and keeps the best match.

Output Arguments

`indexPairs` — Indices to corresponding features
P-by-2 matrix

Indices of corresponding features between the two input feature sets, returned as a *P*-by-2 matrix of *P* number of indices. Each index pair corresponds to a matched feature between the `features1` and `features2` inputs. The first element indexes the feature in `features1`. The second element indexes the matching feature in `features2`.



`matchmetric` — Distance between matching features
p-by-1 vector

Distance between matching features, returned as a *p*-by-1 vector. The value of the distances are based on the metric selected. Each *i*th element in `matchmetric` corresponds to the *i*th row in the `indexPairs` output matrix. When `Metric` is set to either `SAD` or `SSD`, the feature vectors are normalized to unit vectors before computation.

Metric	Range	Perfect Match Value
`SAD`	`[0, 2*sqrt(size(features1, 2))]`.	`0`
`SSD`	`[0,4]`	`0`
`Hamming`	`[0, features1.NumBits]`	`0`

Version History

Introduced in R2011a

References

[1] Lowe, David G. "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision*. Volume 60, Number 2, pp. 91–110.

[2] Muja, M., and D. G. Lowe. "Fast Matching of Binary Features. "*Conference on Computer and Robot Vision*. CRV, 2012.

[3] Muja, M., and D. G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." *International Conference on Computer Vision Theory and Applications*.VISAPP, 2009.

[4] Rublee, E., V. Rabaud, K. Konolige and G. Bradski. "ORB: An efficient alternative to SIFT or SURF." In *Proceedings of the 2011 International Conference on Computer Vision*, 2564–2571. Barcelona, Spain, 2011.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Generates platform-dependent library for MATLAB host when using the `Approximate` method.
- Generates portable C code for non-host target only when using the `Exhaustive` method.
- Generates portable C code using a C++ compiler that links to OpenCV libraries when not using the `Exhaustive` method. See “Portable C Code Generation for Functions That Use OpenCV Library”.
- `'Method'` and `'Metric'` must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- CUDA code is generated only for the `exhaustive` matching method. If the `Approximate` method is selected, GPU Coder issues a warning and generates C/C++ code for this function.

See Also

`extractFeatures`

Topics

“Create Panorama”
“Point Feature Types”
“What Is Structure from Motion?”

matchFeaturesInRadius

Find matching features within specified radius

Syntax

```
indexPairs = matchFeaturesInRadius(features1,features2,points2,centerPoints,
radius)
[indexPairs,matchMetric] = matchFeaturesInRadius( ___ )
[indexPairs,matchMetric] = matchFeaturesInRadius( ___ ,Name,Value)
```

Description

`indexPairs = matchFeaturesInRadius(features1,features2,points2,centerPoints, radius)` returns the indices of the features most likely to correspond between the input feature sets within the specified radius or radii around each expected match location.

`[indexPairs,matchMetric] = matchFeaturesInRadius( ___ )` also returns the distance between the features in a matched pair in `indexPairs`.

`[indexPairs,matchMetric] = matchFeaturesInRadius( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition of the input arguments in previous syntaxes.

Examples

Match Features Between Two Images

Load a MAT file containing an image and camera data into the workspace.

```
data = load('matchInRadiusData.mat');
```

Convert the camera pose to extrinsics.

```
orientation = data.cameraPose2.Rotation;
location = data.cameraPose2.Translation;
[rotationMatrix,translationVector] = cameraPoseToExtrinsics(orientation,location);
```

Project the 3-D world points associated with feature set one onto the second image.

```
centerPoints = worldToImage(data.intrinsics,rotationMatrix,translationVector,data.worldPoints);
```

Match features between the two feature sets within spatial constraints.

```
indexPairs1 = matchFeaturesInRadius(data.features1,data.features2, ...
        data.points2,centerPoints,data.radius,'MatchThreshold',40, ...
        'MaxRatio',0.9);
```

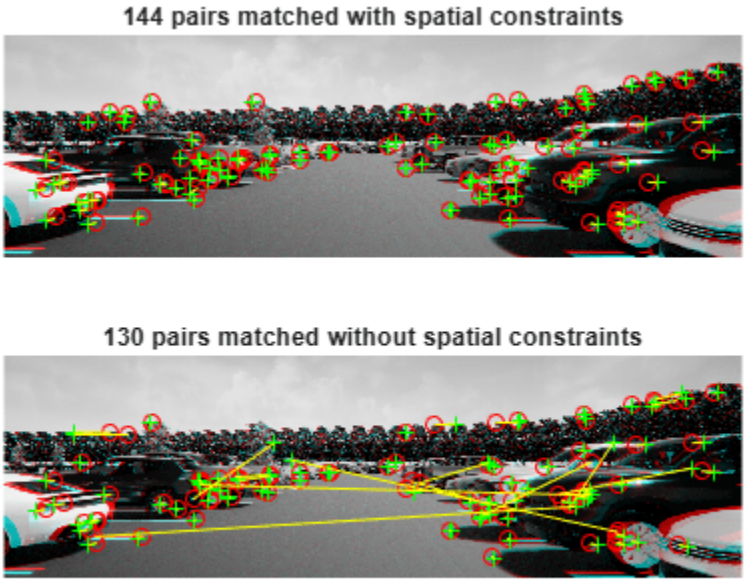
Match features between the two feature sets without using spatial constraints.

```
indexPairs2 = matchFeatures(data.features1,data.features2, ...
        'MatchThreshold',40,'MaxRatio',0.9);
```

Visualize and compare the results between the two ways of matching features.

```
figure
subplot(2,1,1)
showMatchedFeatures(data.I1,data.I2,data.points1( ...
    indexPairs1(:,1)),data.points2(indexPairs1(:,2)));
title(sprintf('%d pairs matched with spatial constraints',size(indexPairs1,1)));

subplot(2,1,2)
showMatchedFeatures(data.I1,data.I2,data.points1( ...
    indexPairs2(:,1)),data.points2(indexPairs2(:,2)));
title(sprintf('%d pairs matched without spatial constraints',size(indexPairs2,1)));
```



Input Arguments

`features1` — Feature set one

`binaryFeatures` object | *M*1-by-*N* matrix

Feature set one, specified as a `binaryFeatures` object or an *M*1-by-*N* matrix. The matrix contains *M*1 features, and *N* corresponds to the length of each feature vector.

You can obtain the `binaryFeatures` object using the `extractFeatures` function with the fast retina keypoint (FREAK), oriented fast and rotated brief (ORB), or binary robust invariant scalable keypoints (BRISK) descriptor method.

Data Types: `logical` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `binaryFeature object`

`features2` — Feature set two

*M*2-by-*N* matrix | `binaryFeatures` object

Feature set two, specified as a `binaryFeatures` object or an *M*2-by-*N* matrix. The matrix contains *M*2 features, and *N* corresponds to the length of each feature vector.

You can obtain the `binaryFeatures` object using the `extractFeatures` function with the fast retina keypoint (FREAK), oriented fast and rotated brief (ORB), or binary robust invariant scalable keypoints (BRISK) descriptor method.

Data Types: `logical` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `binaryFeature object`

`points2` — Feature points of feature set two
*M*2-by-2 matrix | *M*2-element feature point array

Feature points of feature set two, specified as either an *M*2-by-2 matrix in the format [*x y*] or an *M*2-element feature point array. For a list of point feature types, see “Point Feature Types”.

Data Types: `single` | `double` | `point feature type`

`centerPoints` — Expected matched locations
*M*1-by-2 matrix

Expected matched locations in the second image that correspond to the feature points from `features1`, specified as an *M*1-by-2 matrix of coordinates in the format [*x y*].

Data Types: `single` | `double`

`radius` — Search radius associated with center points
scalar | *M*1-element vector

Search radius associated with the center points, specified as a scalar or an *M*1-element vector. When you specify the `radius` as a scalar value, the function uses the same search radius for all center points.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Metric','SSD'` specifies the sum of squared differences metric for feature matching.

`MatchThreshold` — Matching threshold
`10.0` or `1.0` (default) | scalar value in the range (0, 100]

Matching threshold, specified as a scalar value in the range (0,100]. The default values are `10.0` for binary feature vectors or `1.0` for nonbinary feature vectors. You can use the matching threshold to select the strongest matches. The threshold represents a percent of the distance from a perfect match.

Two feature vectors match when the distance between them is less than the threshold set by '`MatchThreshold`'. The function rejects a match when the distance between the features is greater than the value of '`MatchThreshold`'. Increase the value to return more matches.

Inputs that are `binaryFeatures` objects typically require a larger value for the match threshold. The `extractFeatures` function returns a `binaryFeatures` object when extracting FREAK, ORB, or BRISK descriptors.

`MaxRatio` — Ratio threshold
`0.6` (default) | scalar value in the range (0, 1]

Ratio threshold, specified as a scalar value in the range (0, 1]. Use the ratio to reject ambiguous matches. Increase this value to return more matches.

`Metric` — Feature matching metric
`'SSD'` (default) | `'SAD'`

Feature matching metric, specified as either `'SAD'` or `'SSD'`.

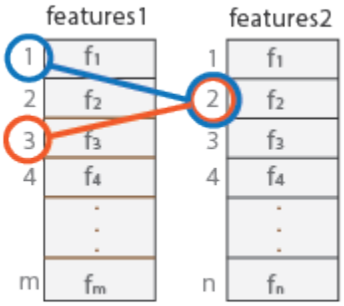
`'SAD'`	Sum of absolute differences
`'SSD'`	Sum of squared differences

This property applies when you specify the input feature sets, `features1` and `features2`, as matrices. When you specify the features as `binaryFeatures` objects, the function uses the Hamming distance to compute the similarity metric.

`Unique` — Unique matches
`false` or `0` (default) | `true` or `1`

Unique matches, specified as a logical `0` (`false`) or `1` (`true`). Set this value to `true` to return only unique matches between `features1` and `features2`.

When you set `Unique` to `false`, the function returns all matches between `features1` and `features2`. Multiple features in `features1` can match to one feature in `features2`.



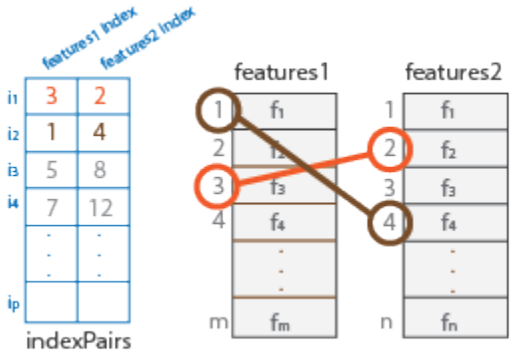
When you set `Unique` to `true`, the function performs a forward-backward match to select a unique match. After matching `features1` to `features2`, it matches `features2` to `features1` and keeps the best match.

Output Arguments

`indexPairs` — Indices of corresponding features
P-by-2 matrix

Indices of corresponding features between the two input feature sets, returned as a *P*-by-2 matrix. *P* is the number of matched pairs of features. Each index pair corresponds to a matched feature between

the `features1` and `features2` inputs. The first element indexes the feature in `features1`. The second element indexes the matching feature in `features2`.



`matchMetric` — Distance between matching features
P-by-1 vector

Distance between matching features, returned as a *P*-by-1 vector. The *i*th element in `matchMetric` corresponds to the *i*th row in the `indexPairs` output matrix. The values of the distances are based on the metric selected, but a perfect match is always `0`. When `Metric` is set to either `SAD` or `SSD`, the feature vectors are normalized to unit vectors before computation. The function returns `matchMetric` as a `double` data type when `features1` and `features2` are of type `double`. Otherwise, the returned vector is of type `single`.

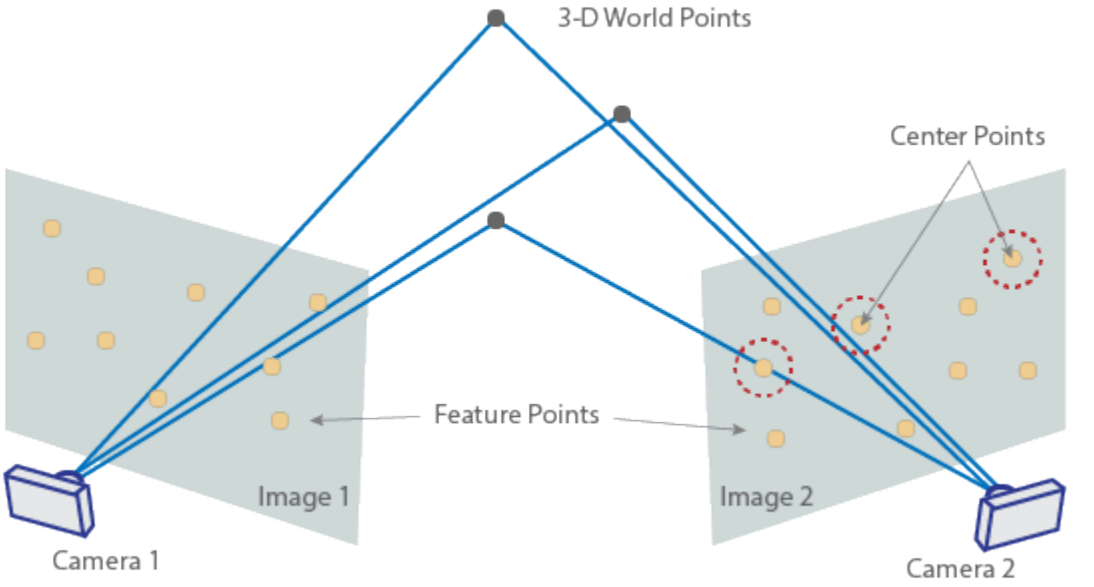
Metric	Range
SAD	`[0, 2*sqrt(size(features1,2))]`.
SSD	`[0, 4]`
Hamming	`[0, features1.NumBits]`

Note You cannot select the Hamming metric. The metric is selected automatically when the `features1` and `features2` inputs are `binaryFeatures`.

Data Types: `single` | `double`

Tips

- Use this function when the 3-D world points that correspond to feature set one `features1`, are known. `centerPoints` can be obtained by projecting a 3-D world point onto the second image. You can obtain the 3-D world points by triangulating matched image points from two stereo images.
- You can specify a circular area of points in feature set two to match with feature set one. Specify the origin as `centerPoints` with a radius specified by `radius`. Specify the points to match from feature set two as `points2`



Version History

Introduced in R2021a

References

[1] Fraundorfer, Friedrich, and Davide Scaramuzza. “Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications.” *IEEE Robotics & Automation Magazine* 19, no. 2 (June 2012): 78–90. https://doi.org/10.1109/MRA.2012.2182810.

[2] Lowe, David G. “Distinctive Image Features from Scale-Invariant Keypoints.” *International Journal of Computer Vision* 60, no. 2 (November 2004): 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[3] Muja, Marius, and David G. Lowe. “Fast Approximate Nearest Neighbors With Automatic Algorithm Configuration:” In *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications*, 331–40. Lisboa, Portugal: SciTePress - Science and Technology Publications, 2009. https://doi.org/10.5220/0001787803310340.

[4] Muja, Marius, and David G. Lowe. "Fast Matching of Binary Features." In *2012 Ninth Conference on Computer and Robot Vision*, 404–10. New York: Institute of Electrical and Electronics Engineers, 2012. https://doi.org/10.1109/CRV.2012.60.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `'Method'` and `'Metric'` must be compile-time constants.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions
`showMatchedFeatures` | `extractFeatures` | `detectHarrisFeatures` | `detectSURFFeatures` | `detectORBFeatures` | `detectFASTFeatures` | `detectBRISKFeatures` | `detectMinEigenFeatures` | `estimateFundamentalMatrix` | `estimateGeometricTransform` | `detectMSERFeatures` | `estimateWorldCameraPose` | `worldToImage`

Objects
`binaryFeatures`

Topics
“What Is Structure from Motion?”
“Monocular Visual Simultaneous Localization and Mapping”
“Implement Visual SLAM in MATLAB”

mrdivide, /

Transformation or rotation right division

Syntax

```
transformationC = transformationA/transformationB
rotationC = rotationA/rotationB
```

Description

`transformationC = transformationA/transformationB` right divides transformation `transformationA` by transformation `transformationB` and returns the quotient, transformation `transformationC`. `transformationC` is the same value as `transformationA*inv(transformationB)`.

You can use division to compose a sequence of transformations, so that `transformationC` represents a transformation where the inverse of `transformationB` is applied first, followed by `transformationA`.

`rotationC = rotationA/rotationB` right divides transformation `rotationA` by transformation `rotationB` and returns the quotient, transformation `rotationC`. `rotationC` is the same value as `rotationA*inv(rotationB)`.

Input Arguments

`transformationA` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationA` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`transformationB` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationB` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`rotationA` — First rotation
`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationA` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `so3` objects, `rotationB` must be a scalar `so3` object.

`rotationB` — Last rotation
`so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationB` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `se3` objects, `rotationB` must be a scalar `se3` object.

Output Arguments

`transformationC` — Transformation quotient
`se3` object | *N*-element array of transformation objects

Transformation quotient, returned as a scalar `se3` object or as an *N*-element array of the same transformation type as `transformationA` and `transformationB`. *N* is the length of the longer argument between `transformationA` and `transformationB` and each row represents the quotient between `transformationA` and `transformationB`.

`rotationC` — Rotation quotient
`so3` object | *N*-element array of rotation objects

Rotation quotient, returned as a scalar `so3` object or as an *N*-element array of the same rotation type as `rotationA` and `rotationB`. *N* is the length of the longer argument between `rotationA` and `rotationB` and each row represents the quotient between `rotationA` and `rotationB`.

See Also

Functions
`rdivide, ./` | `mtimes, *` | `times, .*`

Objects
`se3` | `so3`

mtimes, *

Transformation or rotation multiplication

Syntax

```
transformationC = transformationA*transformationB
rotationC = rotationA*rotationB
```

Description

`transformationC = transformationA*transformationB` performs transformation multiplication between transformation `transformationA` and transformation `transformationB` and returns the product, transformation `transformationC`.

You can use transformation multiplication to compose a sequence of transformations, so that `transformationC` represents a transformation where `transformationB` is applied first, followed by `transformationA`.

`rotationC = rotationA*rotationB` performs rotation multiplication between rotation `rotationA` and rotation `rotationB` and returns the product, rotation `rotationC`.

You can use rotation multiplication to compose a sequence of rotations, so that `rotationC` represents a rotation where `rotationB` is applied first, followed by `rotationA`.

Input Arguments

`transformationA` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationA` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`transformationB` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationB` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`rotationA` — First rotation
`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationA` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `so3` objects, `rotationB` must be a scalar `so3` object.

`rotationB` — Last rotation
`so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationB` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `se3` objects, `rotationB` must be a scalar `se3` object.

Output Arguments

`transformationC` — Transformation product
`se3` object | *N*-element array of transformation objects

Transformation product, returned as a scalar `se3` object or as an *N*-element array of the same transformation type as `transformationA` and `transformationB`. *N* is the length of the longer argument between `transformationA` and `transformationB` and each row represents the product between `transformationA` and `transformationB`.

`rotationC` — Rotation product
`so3` object | *N*-element array of rotation objects

Rotation product, returned as a scalar `so3` object or as an *N*-element array of the same rotation type as `rotationA` and `rotationB`. *N* is the length of the longer argument between `rotationA` and `rotationB` and each row represents the product between `rotationA` and `rotationB`.

See Also

Functions
`mrdivide, /` | `rdivide, ./` | `times, .*`

Objects
`se3` | `so3`

normalize

Normalize transformation or rotation matrix

Syntax

```
transformationN = normalize(transformation)
rotationN = normalize(rotation)
___ = normalize( ___ ,Method=normMethod)
```

Description

`transformationN = normalize(transformation)` normalizes the rotation of the transformation `transformation` and returns a transformation, `transformationN`, that is equivalent to `transformation`, but with normalized rotation.

`rotationN = normalize(rotation)` normalizes the rotation of the rotation `rotation` and returns a rotation, `rotationN`, that is equivalent to `rotation`, but with normalized rotation.

Note The transformation and rotation objects do not automatically normalize their rotations. You must use `normalize` each time you need to normalize a transformation or rotation. You may need to do this if:

- You specified an unnormalized input transformation or rotation at the creation of the transformation or rotation object.
- You performed many operations on the transformation or rotation objects such as `mtimes, *`, which may cause the transformation or rotation to become unnormalized due to data type precision.

`___ = normalize( ___ ,Method=normMethod)` specifies the normalization method `normMethod` that the `normalize` function uses to normalize the specified transformation or rotation.

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of transformation objects

Transformation, specified as a scalar `se3` object or an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformation` as an array, each element must be of the same type.

`rotation` — Rotation

`so3` object | *N*-element array of rotation objects

Rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotation` as an array, each element must be of the same type.

`normMethod` — Normalization method
`"quat"` (default) | `"cross"` | `"svd"`

Normalization method, specified as one of these options:

- `"quat"` — Convert the rotation submatrix into a normalized quaternion and then convert the normalized quaternion back to a transformation or rotation object. For more information, see the `normalize` of the `quaternion` object.
- `"cross"` — Normalize the third column of the rotation submatrix and then determine the other two columns through cross products.
- `"svd"` — Use singular value decomposition to find the closest orthonormal matrix by setting singular values to `1`. This solves the orthogonal Procrustes problem.

Data Types: `char` | `string`

Output Arguments

`transformationN` — Normalized transformation
`se3` object

Normalized transformation, returned as an `se3` object.

`rotationN` — Normalized rotation
`so3` object

Normalized rotation, returned as an `so3` object.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dist` | `interp` | `transform`

Objects
`se3` | `so3` | `quaternion`

ocrTrainingData

Create training data for OCR from ground truth

Syntax

```
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName)
```

Description

`[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName)` creates datastores for loading images, bounding boxes, and image text from ground truth.

`ocrTrainingData` creates training data that you can use to train and evaluate an optical character recognition (OCR) model from ground truth data. Use the `trainOCR` function to train an OCR model and the `evaluateOCR` function to evaluate the model.

Examples

Analyze OCR Ground Truth Data

This example shows how to analyze an OCR ground truth data to identify its character set and to understand the class distribution.

Load the ground truth data and then extract text labels.

```
ld = load("14SegmentGtruth.mat");
gTruth = ld.gTruth;
[~,~,txtds] = ocrTrainingData(gTruth,"Text","Word");
```

Read all ground truth text corresponding to each image and combine them.

```
allImagesText = txtds.readall;
allText = strjoin([allImagesText{:}], "");
```

Find the unique set of characters in the ground truth text.

```
[characterSet, ~, idx] = unique(char(allText));
```

Display the ground truth character set.

```
disp("Ground Character Set: " + string(characterSet))
```

```
Ground Character Set: +,-./3ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The ground truth data contains all the 26 alphabets of English language in capital case, the digit 3 and five special characters: +,-./.

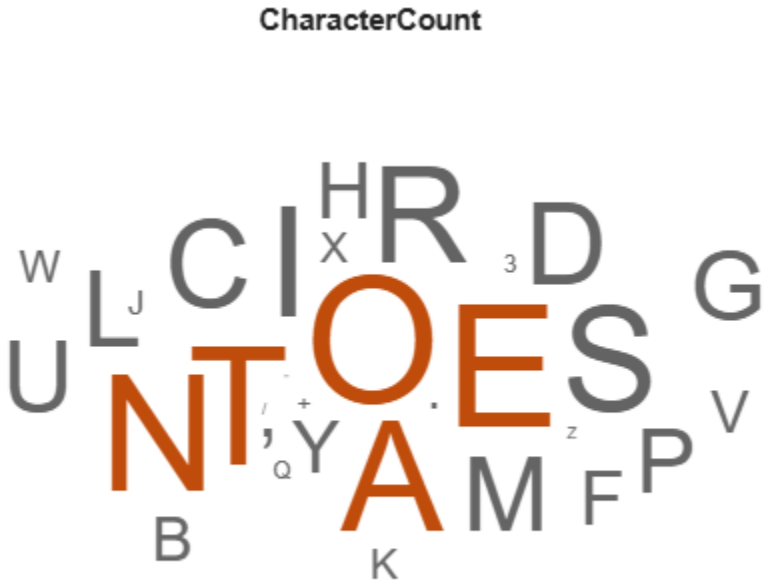
To understand the class distribution, count the character occurrences and tabulate the character count.

```
characterSet = cellstr(characterSet');
characterCount = accumarray(idx,1);
```

```
characterCountTbl = table(characterSet, characterCount, ...
    VariableNames=["Character", "CharacterCount"]);
characterCountTbl = sortrows(characterCountTbl, ...
    "CharacterCount", "descend");
```

Visualize the character count with a word cloud chart.

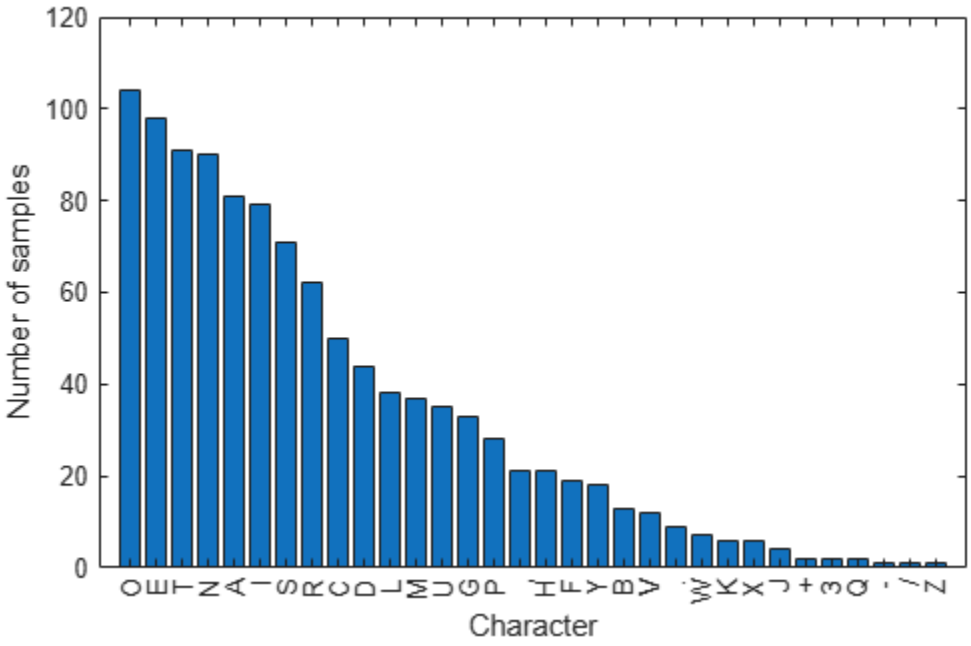
```
wordcloud(characterCountTbl, "Character", "CharacterCount")
```



The characters O, E, T, N and A have the highest character count and the characters -, +, /, . , 3 have the least character count.

Visualize the class distribution with a bar graph.

```
figure
numCharacters = numel(characterSet);
bar(1:numCharacters, characterCountTbl.CharacterCount)
xticks(1:numCharacters)
xticklabels(characterCountTbl.Character)
xlabel("Character")
ylabel("Number of samples")
```

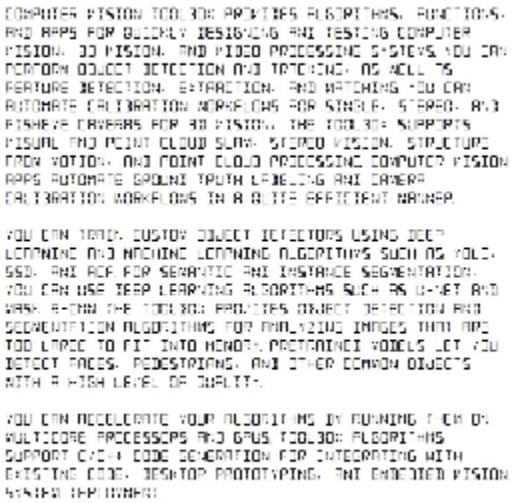


Prepare Data for OCR Training

This example shows preparing data to train an OCR model that can recognize fourteen-segment characters.

The training data contains word samples of fourteen-segment characters from a page of text. Read the training image and display it.

```
I = imread("CVT-DSEG14.jpg");
imshow(I)
```



This image was annotated with bounding boxes containing words and text labels were added to these bounding boxes as an attribute using the Image Labeler. To learn more about labeling images for OCR training, see “Train Custom OCR Model”. The labels were exported from the app as groundTruth object and saved in 14SegmentGtruth.mat file.

```
ld = load("14SegmentGtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Word";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
```

Combine the datastores.

```
cds = combine(imds,boxds,txtds);
```

The combined datastore can be used for training an OCR model using the `trainOCR` function.

Input Arguments

`gTruth` — Ground truth data

`groundTruth` object | *M*-by-1 array of `groundTruth` objects

Ground truth data, specified as a `groundTruth` object or an *M*-by-1 array of `groundTruth` objects exported from the Image Labeler app.

`labelName` — Name of rectangular ROI label

string scalar | character vector

Name of the rectangular ROI label used for labeling ground truth, specified as a string scalar or character vector. You must use the `Rectangle` label type for OCR ground truth labeling.

Use the Image Labeler app to label ground truth data. After loading your images into the app, select **Label** from the toolbar, then select `Rectangle`. A dialog box appears that provides the field for entering the label name.

`attributeName` — Attribute name
string scalar | character vector

Attribute name that corresponds to the label name, specified as a string scalar or character vector. The attribute identifies what the OCR detects in the specified ROI `labelName`. For example, `word`. To name an attribute in Image Labeler, after creating the ROI, select **Attribute** from the toolbar. A dialog box appears that provides the field for entering the attribute name.

Output Arguments

`imds` — Image datastore
`imageDatastore` object

Image datastore, returned as an `imageDatastore` object that contains images extracted from specified `groundTruth` object or objects.

`boxds` — Bounding box label datastore
`arrayDatastore` object

Bounding box label datastore associated with the ground truth images, returned as an `arrayDatastore` object.

`txtds` — Text label datastore
`arrayDatastore` object

Text label datastore that corresponds to the attribute name input, returned as an `arrayDatastore` object.

Version History

Introduced in R2023a

See Also

Apps
Image Labeler

Functions
`trainOCR` | `evaluateOCR` | `quantizeOCR` | `ocr`

Objects
`ocrText` | `ocrTrainingOptions`

Topics
“Getting Started with OCR”
“Automate Ground Truth Labeling for OCR”

“Train an OCR Model to Recognize Seven-Segment Digits”

evaluateOCR

Evaluate OCR results against ground truth

Syntax

```
metrics = evaluateOCR(resultTxt,groundTruthTxt)
metrics = evaluateOCR( ___ ,Name=Value)
```

Description

`metrics = evaluateOCR(resultTxt,groundTruthTxt)` evaluates the optical character recognition (OCR) results `resultTxt` against the ground truth `groundTruthTxt`. The function evaluates the quality of the OCR results by computing the character error rate and word error rate metrics across images and the entire data set.

`metrics = evaluateOCR( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Metrics="word-error-rate"` specifies to evaluate results using only the word error rate metric.

Examples

Evaluate Accuracy of OCR Model

This example shows how to evaluate the accuracy of an OCR model that can recognize seven-segment numerals on a dataset. The evaluation dataset contain images of energy meter displays that have seven-segment numerals in them.

Download and extract dataset.

```
datasetURL = "https://ssd.mathworks.com/supportfiles/vision/data/7SegmentImages.zip";
datasetZip = "7SegmentImages.zip";
if ~exist(datasetZip,"file")
    disp("Downloading evaluation dataset (" + datasetZip + " - 96 MB) ...");
    websave(datasetZip,datasetURL);
end

datasetFiles = unzip(datasetZip);
```

Load the evaluation ground truth.

```
ld = load("7SegmentGtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Digits";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
```

Combine the datastores.

```
cds = combine(imds,boxds,txtds);
```

Run OCR on the evaluation dataset.

```
results = ocr(cds, Model="seven-segment");
```

Evaluate the OCR results against the ground truth.

```
metrics = evaluateOCR(results,cds);
```

```
Evaluating ocr results
----------------------
* Selected metrics: character error rate, word error rate.
* Processed 119 images.
* Finalizing... Done.
* Data set metrics:

    CharacterErrorRate    WordErrorRate
    __________________    _____________

         0.082195             0.19958
```

Display accuracy of the OCR model.

```
modelAccuracy = 100*(1-metrics.DataSetMetrics.CharacterErrorRate);
disp("Accuracy of the OCR model= " + modelAccuracy + "%")
```

```
Accuracy of the OCR model= 91.7805%
```

Input Arguments

`resultTxt` — Prediction from OCR model

`ocrText` object | datastore

Prediction from OCR model, specified as a cell array of `ocrText` objects, or any datastore object that returns a cell array with at least two elements, such that the last two cells contain:

- Cell 1 — *N*-by-4 matrix, where *N* is the number of ROIs [in the original ground truth data.] text line bounding boxes.
- Cell 2 — *N*-element vector of strings, where *N* is the number of ROIs [in the original ground truth data.] of text predictions.

If specified as a cell array, `resultTxt` must contain one element for each image in the original ground truth data.

`groundTruthTxt` — Ground truth text

datastore object

Ground truth text, specified as a datastore object that returns a cell array with at least two elements, such that the last two cells contain:

- Cell 1 — *N*-by-4 matrix, where *N* is the number of ROIs [in the original ground truth data.] text line bounding boxes.
- Cell 2 — *N*-element vector of strings, where *N* is the number of ROIs [in the original ground truth data.] of text predictions.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `metrics = evaluateOCR(resultTxt,groundTruthTxt,Metrics="word-error-rate")` specifies to evaluate results using only the word error rate metric.

`Metrics` — Metrics to compute
`"all"` (default) | `"word-error-rate"` | `"character-error-rate"`

Metrics to compute, specified as `"all"`, `"word-error-rate"`, or `"character-error-rate"`.

`Verbose` — Progress information display
`true` or `1` (default) | `false` or `0`

Progress information display, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Output Arguments

`metrics` — OCR metrics
`ocrMetrics` object

OCR metrics, returned as an `ocrMetrics` object.

Tips

- Error rates are a percentage of characters (words) in the input `groundTruthTxt` that have been incorrectly predicted in `resultTxt`.
- To compute the number of incorrect predictions to use in the error rate calculation, the function uses the Levenshtein distance, which is defined as the minimum number of edits (such as insertions, deletions, or substitutions) required to change one word (or sentence) into another one.

  Error rate = $(S + D + I)/N$ where,

  - S — Number of substitutions
  - D — Number of deletions
  - I — Number of insertions
  - N — Maximum number of characters (words) between `groundTruthTxt` or `resultTxt`

Version History

Introduced in R2023a

See Also

Apps
Image Labeler

Functions
`trainOCR` | `ocrTrainingData` | `ocr` | `quantizeOCR`

Objects
`ocrText` | `ocrMetrics` | `ocrTrainingOptions`

Topics
“Getting Started with OCR”
“Automate Ground Truth Labeling for OCR”
“Train an OCR Model to Recognize Seven-Segment Digits”

quantizeOCR

Quantize OCR model

Syntax

```
outputFileName = quantizeOCR(inputFileName,outputModelName)
outputFileName = quantizeOCR( ___ ,OutputLocation=location)
```

Description

`outputFileName = quantizeOCR(inputFileName,outputModelName)` quantizes the specified optical character recognition (OCR) model `inputFileName` and stores it in the present working directory, specified by `outputModelName`.

`quantizeOCR` quantizes the weights of an OCR model to produce a new OCR model that is faster and use less memory, but at the cost of reduced accuracy in results.

`outputFileName = quantizeOCR( ___ ,OutputLocation=location)` specifies the location to store the returned quantized model, in addition to the input arguments from the previous syntax.. By default, `location` is set to the present working directory. Specify `location` as a character vector or string scalar of the absolute or relative path for the desired output location

Examples

Quantize OCR Model

This example shows how to quantize the seven-segment OCR model that ships with the Computer Vision Toolbox™.

Specify the fullpath to the OCR model.

```
model = fullfile(toolboxdir("vision"),"visionutilities",...
    "tessdata_best","seven_segment.traineddata");
```

Specify a filename to save the quantized OCR model.

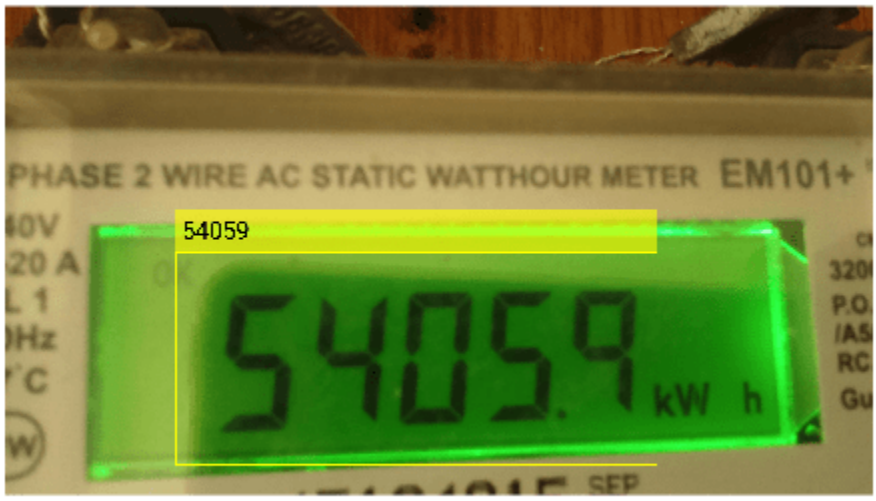
```
outputModelName = "quantizedModel";
```

Quantize the OCR model.

```
outputModel = quantizeOCR(model,outputModelName);
```

Use the quantized model to perform OCR on a test image and visualize the results.

```
I = imread("sevSegDisp.jpg");
roi = [506 725 1418 626];
ocrResults = ocr(I,roi,Model=outputModel,LayoutAnalysis="block");
Iocr = insertObjectAnnotation(I,"rectangle",...
            ocrResults.WordBoundingBoxes,ocrResults.Words,...
            LineWidth=5,FontSize=72);
imshow(Iocr)
```



Improve Runtime Performance of OCR Model with Quantization

This example shows how to improve runtime performance of an OCR model with quantization. This can be useful when deploying an OCR model in resource constrained systems.

Specify the fullpath to the OCR model.

```
originalModel = fullfile(toolboxdir("vision"),"visionutilities",...
    "tessdata_best","seven_segment.traineddata");
```

Specify a filename to save the quantized OCR model.

```
outputModelName = "quantizedModel";
```

Quantize the OCR model.

```
quantizedModel = quantizeOCR(originalModel,outputModelName);
```

Compare the runtime performance of the quantized model against the original model.

```
I = imread("sevSegDisp.jpg");
roi = [506 725 1418 626];
fOCR = @() ocr(I, roi, Model=originalModel, LayoutAnalysis="block");
tOCR = timeit(fOCR);

fQuantizedOCR = @() ocr(I, roi, Model=quantizedModel, LayoutAnalysis="block");
tQuantizedOCR = timeit(fQuantizedOCR);

perfRatio = tOCR/tQuantizedOCR;
disp("Quantized model is " + perfRatio + "x faster");
```

```
Quantized model is 1.7521x faster
```

Compare file size of the quantized model with that of the original model.

```
originalModelFile     = dir(originalModel);
originalModelFileSizeInMB = originalModelFile.bytes/1000000;

quantizedModelFile     = dir(quantizedModel);
quantizedModelFileSizeInMB = quantizedModelFile.bytes/1000000;

sizeRatio = originalModelFileSizeInMB/quantizedModelFileSizeInMB;
disp("Quantized model is " + sizeRatio + "x smaller");
```

```
Quantized model is 7.8483x smaller
```

While the quantized model is smaller and faster than the original model, these advantages of quantization comes at the expense of accuracy. To understand this trade-off, compare the accuracy of the two models by evaluating them against the YUVA EB dataset.

Download the dataset.

```
datasetURL = "https://ssd.mathworks.com/supportfiles/vision/data/7SegmentImages.zip";
datasetZip = "7SegmentImages.zip";
if ~exist(datasetZip,"file")
    disp("Downloading evaluation data set (" + datasetZip + " - 96 MB)...");
    websave(datasetZip,datasetURL);
end
```

```
Downloading evaluation data set (7SegmentImages.zip - 96 MB)...
```

```
datasetFiles = unzip(datasetZip);
```

Load ground truth to be used for evaluation.

```
ld = load("7SegmentGtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Digits";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
cds = combine(imds,boxds,txtds);
```

Run the two models on the dataset and evaluate recognition accuracy using `evaluateOCR`.

```
originalResult  = ocr(cds, Model=originalModel);
quantizedResult = ocr(cds, Model=quantizedModel);

metricsOriginal = evaluateOCR(originalResult, cds);
```

```
Evaluating ocr results
----------------------
* Selected metrics: character error rate, word error rate.
* Processed 119 images.
* Finalizing... Done.
* Data set metrics:
```

```
    CharacterErrorRate    WordErrorRate
    __________________    _____________

         0.082195            0.19958
```

```
metricsQuantized = evaluateOCR(quantizedResult, cds);
```

```
Evaluating ocr results
----------------------
* Selected metrics: character error rate, word error rate.
* Processed 119 images.
* Finalizing... Done.
* Data set metrics:

    CharacterErrorRate    WordErrorRate
    __________________    _____________

         0.13018             0.31933
```

Display the model accuracies.

```
originalModelAccuracy = 100*(1-metricsOriginal.DataSetMetrics.CharacterErrorRate);
quantizedModelAccuracy = 100*(1-metricsQuantized.DataSetMetrics.CharacterErrorRate);
disp("Original model accuracy = " +  originalModelAccuracy + "%")
```

```
Original model accuracy = 91.7805%
```

```
disp("Quantized model accuracy = " + quantizedModelAccuracy + "%")
```

```
Quantized model accuracy = 86.9816%
```

Tabulate the quantitative results.

```
trainedModelResults = [originalModelAccuracy; originalModelFileSizeInMB; tOCR];
quantizedModelResults = [quantizedModelAccuracy; quantizedModelFileSizeInMB; tQuantizedOCR];

table(trainedModelResults, quantizedModelResults, ...
    VariableNames=["OriginalModel", "QuantizedModel"], ...
    RowNames=["Accuracy (in %)", "File Size (in MB)", "Runtime (in seconds)"])
```

```
ans=3×2 table
    91.7805    86.9816
    11.2969     1.4394
     0.1057     0.0604
```

Input Arguments

`inputFileName` — Name of OCR model file

string scalar | character vector

Name of the OCR model file, specified as a string scalar or a character vector. If `inputFileName` does not include the full path, the function searches for the file in the current folder and along the MATLAB path. If you do not specify a file extension, the function appends `.traineddata`.

`outputModelName` — Output quantized model name

string scalar | character vector

Output quantized model name, specified as a string scalar or character vector.

Output Arguments

`outputFileName` — Name of quantized OCR model file
string scalar | character vector

Name of quantized OCR model file, specified as a string scalar or character vector.

Version History

Introduced in R2023a

See Also

Apps
Image Labeler

Functions
`trainOCR` | `ocrTrainingData` | `ocr` | `evaluateOCR`

Objects
`ocrText` | `ocrTrainingOptions`

Topics
“Getting Started with OCR”
“Automate Ground Truth Labeling for OCR”
“Train an OCR Model to Recognize Seven-Segment Digits”

quat

Convert transformation or rotation to numeric quaternion

Syntax

```
q = quat(transformation)
q = quat(rotation)
```

Description

`q = quat(transformation)` creates a quaternion `q` from the rotation of the transformation `transformation`.

`q = quat(rotation)` creates a quaternion `q` from the rotation `rotation`.

Examples

Convert SE(3) Transformation to Numeric Quaternion

Create SE(3) transformation with zero translation and a rotation defined by a numeric quaternion.

```
quat1 = [0.9659 0.2588 0 0]
```

```
quat1 = 1×4

    0.9659    0.2588         0         0
```

```
T = se3(quat1,"quat")
```

```
T = se3
    1.0000         0         0         0
         0    0.8660   -0.5000         0
         0    0.5000    0.8660         0
         0         0         0    1.0000
```

Convert the transformation back into a numeric quaternion.

```
quat2 = quat(T)
```

```
quat2 = 1×4

    0.9659    0.2588         0         0
```

Convert SO(3) Rotation to Numeric Quaternion

Create SO(3) rotation defined by a numeric quaternion. Use the `eul2quat` function to create the numeric quaternion.

```
quat1 = [0.9659 0.2588 0 0]
```

```
quat1 = 1×4

    0.9659    0.2588         0         0
```

```
R = so3(quat1,"quat")
```

```
R = so3
    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660
```

Convert the rotation into a numeric quaternion.

```
quat2 = quat(R)
```

```
quat2 = 1×4

    0.9659    0.2588         0         0
```

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of `se3` objects

Transformation, specified as an `se3` object or as an *N*-element array of `se3` objects. *N* is the total number of transformations.

`rotation` — Rotation

`so3` object | *N*-element array of `so3` objects

Rotation, specified as an `so3` object or as an *N*-element array of `so3` objects. *N* is the total number of rotations.

Output Arguments

`q` — Quaternion rotation angles

M-by-4 matrix

Quaternion rotation angles, returned as an *M*-by-4 matrix, where each row is of the form [*qw qx qy qz*]. *M* is the total number of transformations or rotations specified.

See Also

`se3` | `so3` | `quaternion`

trainOCR

Train OCR model to recognize text in image

Syntax

```
modelFileName = trainOCR(trainingData,modelName,baseModel,ocrOptions)
[modelFileName,info] = trainOCR( ___ )

[modelFileName,info] = trainOCR(trainingData,modelName,checkpoint,ocrOptions)
```

Description

Training

`modelFileName = trainOCR(trainingData,modelName,baseModel,ocrOptions)` trains a new OCR model by fine-tuning a pretrained base model using the hyperparameters specified in `ocrOptions`.

`[modelFileName,info] = trainOCR( ___ )` returns a structure that contains information on training progress, such as the training root mean squared error (RMSE) and learning rate for each iteration, using the input arguments from the previous syntax. For a list of the returned error rates, see the modelFileName on page 3-0 output argument.

Resume training

`[modelFileName,info] = trainOCR(trainingData,modelName,checkpoint,ocrOptions)` resumes training from an OCR training checkpoint. Use this syntax to improve the accuracy of your OCR model by using additional training data or to perform more training iterations.

Examples

Train OCR Model

This example shows how to train an OCR model that can recognize fourteen-segment characters. The training data contains word samples of fourteen-segment characters.

Unzip and extract training images.

```
datasetZip = 'dseg14.zip';
evalfiles = unzip(datasetZip);
```

The training images was annotated with bounding boxes containing words and text labels were added to these bounding boxes as an attribute using the Image Labeler. The labels were exported from the app as `groundTruth` object and saved in dseg14Gtruth.mat file.

```
ld = load("dseg14Gtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Word";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
```

Combine the datastores.

```
cds = combine(imds,boxds,txtds);
```

Split the data for training and validation with a training-to-validation ratio of 0.9

```
% Set the random number seed for reproducibility.
rng(0);

% Compute number of training and validation samples.
trainingToValidationRatio = 0.9;
numSamples = height(ld.gTruth.LabelData);
numTrainSamples = ceil(trainingToValidationRatio*numSamples);

% Divide the dataset into training and validation.
indices = randperm(numSamples);
trainIndices = indices(1:numTrainSamples);
validationIndices = indices(numTrainSamples+1:end);

cdsTrain = subset(cds, trainIndices);
cdsValidation = subset(cds, validationIndices);
```

Specify training options. Set the gradient decay factor for ADAM optimization to 0.9, and use an initial learning rate of 40e-4. Set the verbose frequency to 160 iterations and the maximum number of epochs for training to 5. Specify the checkpoint path to enable saving checkpoints and specify the validation data to enable validation.

```
outputDir = "OCRModel";
if ~exist(outputDir, "dir")
    mkdir(outputDir);
end

checkpointsDir = "Checkpoints";
if ~exist(checkpointsDir, "dir")
    mkdir(checkpointsDir);
end

ocrOptions = ocrTrainingOptions(GradientDecayFactor=0.9, ...
    InitialLearnRate=40e-4, MaxEpochs=5, VerboseFrequency=160, ...
    CheckpointPath=checkpointsDir, ValidationData=cdsValidation, ...
    OutputLocation=outputDir);
```

Train a new OCR model by fine-tuning the pretrained "english" model. The training will take about 3-4 minutes.

```
outputModelName = "fourteenSegment";
baseModel = "english";
outputModel = trainOCR(cdsTrain, outputModelName, baseModel, ocrOptions);
```

```
*************************************************************************
Starting OCR training

Model Name: fourteenSegment
Base Model: english
```

```
Preparing training data... 100.00 % completed.
Preparing validation data... 100.00 % completed.

Character Set: +,-./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ

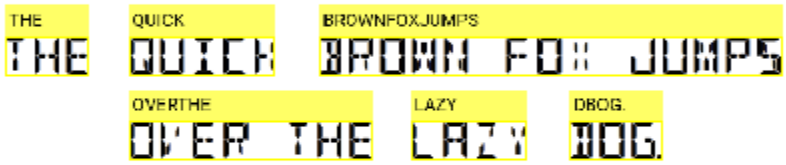
|=========================================================================================================
| Epoch | Iteration | Time Elapsed |           Training Statistics           |          Validatio
|       |           |  (hh:mm:ss)  |   RMSE   | Character Error | Word Error |   RMSE   | Charact
|=========================================================================================================
|   1   |     1     |   00:00:27   |   9.55   |     100.00      |   100.00   |   0.00   |       0
|   1   |    160    |   00:00:49   |   2.44   |      18.23      |   36.25    |   1.26   |       7
|   2   |    320    |   00:01:11   |   1.44   |      9.78       |   20.62    |   0.55   |       5
|   3   |    480    |   00:01:34   |   1.14   |      7.24       |   16.46    |   0.62   |       5
|   4   |    640    |   00:01:55   |   0.93   |      5.62       |   13.12    |   0.54   |       5
|   5   |    800    |   00:02:19   |   0.78   |      4.59       |   10.75    |   0.57   |       5
|   5   |    845    |   00:02:25   |   0.75   |      4.35       |   10.18    |   0.57   |       5
|=========================================================================================================

OCR training complete.
Exit condition: Reached maximum epochs.

Model file name: OCRModel\fourteenSegment.traineddata
************************************************************************
```

Use the trained model to perform OCR on a test image and visualize the results.

```
I = imread("DSEG14.png");
ocrResults = ocr(I,Model=outputModel);
Iocr = insertObjectAnnotation(I,"rectangle",...
            ocrResults.WordBoundingBoxes,ocrResults.Words,...
            LineWidth=2,FontSize=17);
imshow(Iocr)
```



Input Arguments

`trainingData` — Ground truth data

datastore

Ground truth data, specified as a `datastore` that returns a cell array or a table when input to the `read` function. The table must contain at least these three columns:

- 1st column — A cell vector of logical, grayscale, or RGB images.
- 2nd column — A cell vector in which each cell corresponds to an image and contains an *M*-by-4 matrix. *M* is the number of bounding boxes in the image, and each row of the matrix specifies a bounding box in the form [*x*,*y*,*width*,*height*]. cell vector that contains *M*-by-4 matrices with *M* bounding boxes of the form [*x*,*y*,*width*,*height*].
- 3rd column — A cell vector in which each cell corresponds to an image and contains *N* strings. *N* is the number of lines of text in the image, and each line must contain only text without newline characters.

`modelName` — New model name

string scalar | character vector

New model name, specified as a string scalar or character vector. If the folder already contains a file with the name specified by the `modelName` argument, then the `trainOCR` function overwrites it during training.

`baseModel` — Pretrained base model

string scalar | character vector

Pretrained base model, specified as a string scalar or character vector. You can specify any of these options:

- Language models shipped in the Computer Vision Toolbox, such as `"english"`, `"seven-segment"`, or `"japanese"`.
- One of the supported languages described in the “`Model” on page 3-0` argument of the `ocr` function. You cannot use quantized models, such as `"english-fast"`, `"seven-segment-fast"`, or `"japanese-fast"`, as base models.
- Full path to a custom trained model with a `.traineddata` extension.

`ocrOptions` — Hyperparameters for training

`ocrTrainingOptions` object

Hyperparameters for training, specified as an `ocrTrainingOptions` object.

`checkpoint` — OCR training checkpoint

string scalar | character vector | `CheckpointPath`

OCR training checkpoint, specified as a string scalar or character vector. You must specify a path to a file with a `.checkpoint.traineddata` extension, such as the path specified by the `CheckpointPath` property of an `ocrTrainingOptions` object. When you specify a value for the `CheckpointPath` argument of an `ocrTrainingOptions` object, the `trainOCR` function saves checkpoints at regular intervals during training. You can resume training from any one of these saved checkpoints.

Output Arguments

`modelFileName` — Model filename
string scalar

Model filename, returned as a string scalar.

`info` — Information on training progress
structure

Information on training progress, returned as a structure containing these fields:

- `BaseLearnRate` — Learning rate at each iteration.
- `TrainingRMSE` — Training RMSE at each iteration.
- `TrainingCharError` — Training character error rate at each iteration.
- `TrainingWordError` — Training word error rate at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `ValidationCharError` — Validation character error rate at each iteration.
- `ValidationWordError` — Validation word error rate at each iteration.
- `FinalValidationRMSE` — Final validation RMSE at the end of the training.
- `OutputModelIteration` — Iteration number of the returned model.

If you do not specify validation data , the structure contains empty `ValidationRMSE`, `ValidationCharError`, `ValidationWordError`, and `FinalValidationRMSE` fields.

Limitations

- Training OCR models with right-to-left scripts such as Arabic and Hebrew are not supported.

Algorithms

- The `trainOCR` function creates a temporary folder, "`<`*`modelName`*`>Training/`, where `<`*`modelName`*`>` is the value of the `modelName` argument, in the location specified by the `OutputLocation` property of the `ocrTrainingOptions` object. The folder contains training artifacts. If the folder does not already exist before you run the `trainOCR` function, the function deletes it at the end of training. If the folder already exists prior to training, the function does not delete the folder.
- Images read from `trainingData` must contain text of at least one-word length and up to a maximum of one-line length. The `trainOCR` function does not support images that contain multiple lines of text.
- The `trainOCR` function does not support on-the-fly data augmentation using a datastore transform. All the image data is read once from the training datastores at the start of training.

Version History

Introduced in R2023a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

Apps
Image Labeler

Functions
`ocrTrainingData` | `ocr` | `quantizeOCR` | `evaluateOCR`

Objects
`ocrText` | `ocrTrainingOptions`

Topics
“Getting Started with OCR”
“Train Custom OCR Models”
“Automate Ground Truth Labeling for OCR”
“Train an OCR Model to Recognize Seven-Segment Digits”

ocr

Recognize text using optical character recognition

Syntax

```
txt = ocr(I)
txt = ocr(I,roi)
txt = ocr(ds)
txt = ocr( ___ ,Name=Value)
```

Description

`txt = ocr(I)` returns an `ocrText` object that contains optical character recognition (OCR) information from the input image `I`. The object contains recognized characters, words, text lines, the locations of recognized words, and a metric indicating the confidence of each recognition result.

`txt = ocr(I,roi)` recognizes text in `I` within one or more rectangular regions.

`txt = ocr(ds)` returns a cell array of `ocrText` objects, each containing the recognition results for the ROIs specified in the datastore, `ds`, for the corresponding image. Use this syntax to perform OCR on a collection of images. By default, the `ocr` function assumes that each ROI contains only a single line of text. To process ROIs that may contain multiple lines of text, set the `LayoutAnalysis` name-value argument to `"block"`.

`txt = ocr( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `LayoutAnalysis="page"` treats the image as a page containing blocks of text.

Examples

Recognize Text Within an Image

Load an image with text into workspace.

```
businessCard = imread("businessCard.png");
ocrResults = ocr(businessCard)
```

```
ocrResults =
  ocrText with properties:

                     Text: '4 MathWorks:↵↵ ↵↵The MathWorks, Inc.↵↵-3 Apple Hill Drive↵Natick, MA
    CharacterBoundingBoxes: [107×4 double]
      CharacterConfidences: [107×1 single]
                     Words: {16×1 cell}
         WordBoundingBoxes: [16×4 double]
           WordConfidences: [16×1 single]
                 TextLines: {8×1 cell}
     TextLineBoundingBoxes: [8×4 double]
       TextLineConfidences: [8×1 single]
```

```
recognizedText = ocrResults.Text;
figure
imshow(businessCard)
text(600,150,recognizedText,BackgroundColor=[1 1 1]);
```



Recognize Text in Regions of Interest (ROIs)

Read an image into the workspace.

```
I = imread("handicapSign.jpg");
```

Define one or more rectangular regions of interest in which to recognize text within input image.

```
roi = [370 246 363 423];
```

Alternatively, you can use `drawrectangle` to select a region using a mouse.

For example,

```
figure;imshow(I)
```

```
roi = round(getPosition(drawrectangle))
```

Recognize text within the ROI.

```
ocrResults = ocr(I,roi);
```

Insert the recognized text into the original image. Display the image with the inserted recognized text.

```
Iocr = insertText(I,roi(1:2),ocrResults.Text,AnchorPoint="RightTop",FontSize=16);
figure
imshow(Iocr)
```



Recognize Digits from Seven-Segment Display

Read an image containing a seven-segment display into the workspace.

```
I = imread("sevSegDisp.jpg");
```

Specify the ROI that contains the seven-segment display.

```
roi = [506 725 1418 626];
```

To recognize the digits from the seven-segment display, specify the Model argument as "`seven-segment`".

```
ocrResults = ocr(I,roi,Model="seven-segment");
```

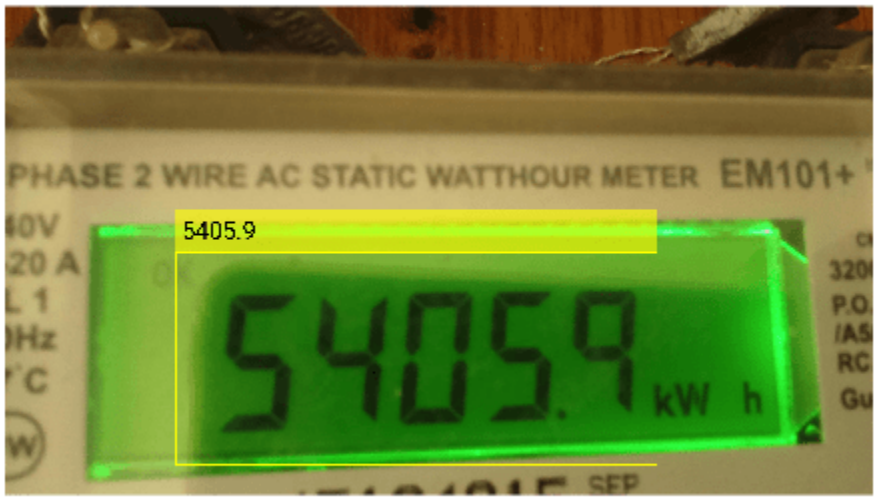
Display the recognized digits and detection confidence.

```
fprintf("Recognized seven-segment digits: ""%s""\nDetection confidence: %0.4f",cell2mat(ocrResul
```

```
Recognized seven-segment digits: "5405.9"
Detection confidence: 0.7948
```

Insert the recognized digits into the image.

```
Iocr = insertObjectAnnotation(I,"rectangle", ...
            ocrResults.WordBoundingBoxes,ocrResults.Words,LineWidth=5,FontSize=72);
figure
imshow(Iocr)
```



Display Bounding Boxes of Words and Recognition Confidences

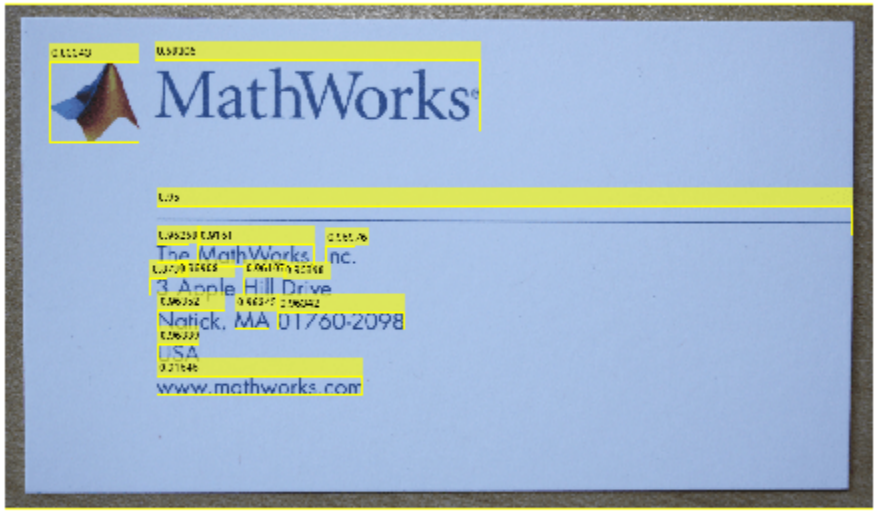
Read an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults = ocr(businessCard)
```

```
ocrResults =
  ocrText with properties:

                     Text: '4 MathWorks:↵↵ ↵↵The MathWorks, Inc.↵↵-3 Apple Hill Drive↵Natick, MA
    CharacterBoundingBoxes: [107×4 double]
      CharacterConfidences: [107×1 single]
                     Words: {16×1 cell}
         WordBoundingBoxes: [16×4 double]
           WordConfidences: [16×1 single]
                 TextLines: {8×1 cell}
     TextLineBoundingBoxes: [8×4 double]
       TextLineConfidences: [8×1 single]
```

```
Iocr = insertObjectAnnotation(businessCard,"rectangle", ...
                              ocrResults.WordBoundingBoxes, ...
                              ocrResults.WordConfidences);
figure
imshow(Iocr)
```



Find and Highlight Text in Image

Load an image containing text into the workspace.

```
businessCard = imread("businessCard.png");
ocrResults = ocr(businessCard);
bboxes = locateText(ocrResults,"Math",IgnoreCase=true);
Iocr = insertShape(businessCard,"FilledRectangle",bboxes);
figure
imshow(Iocr)
```



Evaluate Accuracy of OCR Model

This example shows how to evaluate the accuracy of an OCR model that can recognize seven-segment numerals on a dataset. The evaluation dataset contain images of energy meter displays that have seven-segment numerals in them.

Download and extract dataset.

```
datasetURL = "https://ssd.mathworks.com/supportfiles/vision/data/7SegmentImages.zip";
datasetZip = "7SegmentImages.zip";
if ~exist(datasetZip,"file")
    disp("Downloading evaluation dataset (" + datasetZip + " - 96 MB) ...");
    websave(datasetZip,datasetURL);
end

datasetFiles = unzip(datasetZip);
```

Load the evaluation ground truth.

```
ld = load("7SegmentGtruth.mat");
gTruth = ld.gTruth;
```

Create datastores that contain images, bounding boxes and text labels from the `groundTruth` object using the `ocrTrainingData` function with the label and attribute names used during labeling.

```
labelName = "Text";
attributeName = "Digits";
[imds,boxds,txtds] = ocrTrainingData(gTruth,labelName,attributeName);
```

Combine the datastores.

```
cds = combine(imds,boxds,txtds);
```

Run OCR on the evaluation dataset.

```
results = ocr(cds, Model="seven-segment");
```

Evaluate the OCR results against the ground truth.

```
metrics = evaluateOCR(results,cds);
```

```
Evaluating ocr results
----------------------
* Selected metrics: character error rate, word error rate.
* Processed 119 images.
* Finalizing... Done.
* Data set metrics:

    CharacterErrorRate    WordErrorRate
    __________________    _____________

         0.082195             0.19958
```

Display accuracy of the OCR model.

```
modelAccuracy = 100*(1-metrics.DataSetMetrics.CharacterErrorRate);
disp("Accuracy of the OCR model= " + modelAccuracy + "%")
```

```
Accuracy of the OCR model= 91.7805%
```

Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image | *M*-by-*N* binary image

Input image, specified as an *M*-by-*N*-by-3 truecolor image, *M*-by-*N* grayscale, or *M*-by-*N* binary image. The input image must consist of real, nonsparse values. The function converts truecolor or grayscale input images into a binary image using Otsu's thresholding technique, before performing character recognition. For best OCR results, specify an image in which the height of a lowercase *x*, or comparable character, is greater than 20 pixels. To improve results, remove any text rotations from the horizontal or vertical axes of greater than 10 degrees in either direction. The input image must be less than 2^16 (32,768) pixels in width or height.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`roi` — Rectangular regions of interest

M-by-4 element matrix

Rectangular regions of interest, specified as an *M*-by-4 matrix. Each row specifies a region of interest within the input image in the form [*x y width height*], where [*x y*] specifies the upper-left corner location, and [*width height*] specifies the size of the rectangular region of interest, in pixels. Each rectangle must be fully contained within the input image `I`. Before the recognition process, the function uses Otsu’s thresholding technique to convert truecolor and grayscale input regions of interest to binary regions.

To obtain best results when using `ocr` to recognize seven-segment digits, specify an `roi` that encloses only the part of the image that contains seven-segment digits.

ds — Evaluation data
datastore object

Evaluation data, specified as a datastore that returns a cell array or a table when input to the `read` function. The datastore must return a cell array or a table on the `read` functions with at least these two columns:

- 1st Column — A cell vector of logicals, grayscale images, or RGB images.
- 2nd Column — A cell vector that contains *numROIs*-by-4 matrices of bounding boxes of the form [*x y width height*], which specify text locations within each image.

When using a datastore input, the `LayoutAnalysis` name-value argument value must be either `"auto"` or `"none"`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `txt = ocr(I,LayoutAnalysis="page")` treats the text in the image as a page containing blocks of text.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

LayoutAnalysis — Type of layout analysis to perform for text segmentation
`"auto"` (default) | `"page"` | `"block"` | `"line"` | `"word"` | `"character"` | `"none"`

Type of layout analysis to perform for text segmentation, specified as one of these options:

LayoutAnalysis Value	Text Treatment
`"auto"`	If the `Model` argument value is `"seven-segment"`, treats the text in the image as though the `LayoutAnalysis` value is `"block"`. If you specify the `ds` input argument, treats the text in the image as though the `LayoutAnalysis` value is `"none"`. Otherwise, the function treats the text in the image as `"page"`.
`"page"`	Treats the text in the image as a page containing blocks of text.
`"block"`	Treats the text in the image as a single block of text.
`"line"`	Treats the text in the image as a single line of text.
`"word"`	Treats the text in the image as a single word of text.
`"character"`	Treats the text in the image as a single character.
`"none"`	Do not perform layout analysis. Best used for images with a single line of text.

You can use the `LayoutAnalysis` argument to determine the layout of the text within the input image. For example, you can set `LayoutAnalysis` to `"page"` to recognize text from a scanned

document that contains a specific format, such as a double column. This setting preserves the reading order in the returned text.

If your input image contains a few regions of text, or the text is located in a cluttered scene, the `ocr` function can return poor quality results. If you get poor OCR results, try a different layout that better matches the text in your image. If the text is located in a cluttered scene, try specifying an ROI around the text in your image in addition to trying a different layout.

`Model` — Model to use for recognition

`"english"` (default) | `"japanese"` | `"seven-segment"` | `<"ModelName">-fast"` | character vector | string scalar | cell array of character vectors | string array

Model to use for recognition, specified as one of these options;

- `"english"`, `"japanese"`, `"seven-segment"` — These specify the built-in models for detecting English text, Japanese text, or seven-segment digits, respectively.
- Character vector or string scalar — Use this option to specify a custom model or one of the additional language models included in the OCR Language Data Files support package.
- Cell array of character vectors or string array — Use this option to specify multiple models to use for detection simultaneously.

For faster performance using the built-in models (including any additional installed language models), you can append `-fast` to the language model string. For example, `"english-fast"`, `"japanese-fast"`, or `"seven-segment-fast"`.

You can also install additional models or add a custom model. For details, see “Install OCR Language Data Files”.

Specifying multiple languages enables simultaneous recognition of all the selected languages. However, selecting more than one language can reduce the accuracy of the `ocr` function and increase its processing time. To specify any of the additional languages that are contained in the “Install OCR Language Data Files” package, specify them the same way as the built-in languages. You do not need to specify the path. For example, to specify Finnish text recognition:

```
txt = ocr(img,Model="finnish");
```

List of Support Package OCR Languages

- `"afrikaans"`
- `"albanian"`
- `"ancientgreek"`
- `"arabic"`
- `"azerbaijani"`
- `"basque"`
- `"belarusian"`
- `"bengali"`
- `"bulgarian"`
- `"catalan"`
- `"cherokee"`

- "chinesesimplified"
- "chinesetraditional"
- "croatian"
- "czech"
- "danish"
- "dutch"
- "english"
- "esperanto"
- "esperantoalternative"
- "estonian"
- "finnish"
- "frankish"
- "french"
- "galician"
- "german"
- "greek"
- "hebrew"
- "hindi"
- "hungarian"
- "icelandic"
- "indonesian"
- "italian"
- "italianold"
- "japanese"
- "kannada"
- "korean"
- "latvian"
- "lithuanian"
- "macedonian"
- "malay"
- "malayalam"
- "maltese"
- "mathequation"
- "middleenglish"
- "middlefrench"
- "norwegian"
- "polish"
- "portuguese"
- "romanian"

- "russian"
- "serbianlatin"
- "slovakian"
- "slovenian"
- "spanish"
- "spanishold"
- "swahili"
- "swedish"
- "tagalog"
- "tamil"
- "telugu"
- "thai"
- "turkish"
- "ukrainian"

To use your own custom model for a `trainOCR` output, you can use this syntax:

```
ocrResults = ocr(I,Model=ocrModel);
```

where `ocrModel` contains the full path of the model.

For deployment targets generated by MATLAB Coder: The generated OCR executable and language model file folder must be colocated.

- For English: `C:/path/to/eng.traineddata`
- For Japanese: `C:/path/to/jpn.traineddata`
- For Seven-segment: `C:/path/to/seven_segment.traineddata`
- For custom data files: `C:/path/to/customlang.traineddata`
- `C:/path/ocr_app.exe`

You can copy the English, Japanese and seven-segment trained data files from this folder:

```
fullfile(matlabroot,"toolbox","vision","visionutilities","tessdata_best");
```

`CharacterSet` — Character subset

"" (default) | character vector | string scalar

Character subset, specified as a character vector or string scalar. By default, `CharacterSet` is set to the empty character vector `""`, which specifies the function to search for all characters in the language model specified by the `Model` name-value argument. You can set this value to a smaller set of known characters to constrain the classification process.

The `ocr` function selects the best match for detected text from the `CharacterSet`. Using deducible knowledge about the characters in the input image helps to improve text recognition accuracy. For example, if you set `CharacterSet` to all numeric digits, `"0123456789"`, the function attempts to match each character to only digits. In this case, the `ocr` function can incorrectly recognize a non-digit character as a digit.

If you specify the `Model` as `seven-segment`, the `ocr` function uses the `CharacterSet` value `"0123456789.:-"`.

Output Arguments

`txt` — Recognized text and metrics

`ocrText` object | array of `ocrText` objects | cell array of `ocrText` objects

Recognized text and metrics, returned as an `ocrText` object, array of `ocrText` objects, or a cell array of `ocrText` objects. Each object contains the recognized text and the location of the recognized text for an input image, as well as metrics indicating the confidence of the results. Confidence values are in the range [0,1], representing a percent probability. The shape of the `txt` output depends on whether you specify an entire single image or single image with one ROI, a single image with multiple ROIs, or a datastore to the `ocr` function.

Tips

Optical character recognition (OCR) language data files provide pretrained language models for the Tesseract OCR engine, enabling accurate and efficient text extraction in various languages. These files are designed for integration with Computer Vision Toolbox, allowing you to leverage advanced OCR capabilities across multiple languages. For step-by-step guidance on installing these language data files, enabling third-party language support, and using the pretrained models with the `ocr` function for multilingual text recognition, see “Install OCR Language Data Files”. For an overview of OCR workflows and basic usage, see “Getting Started with OCR”.

If your OCR results are not what you expect, try one or more of these options:

- Increase the image size by 2– 4 times.
- If the characters in the image are too close together or their edges are touching, use morphology to thin out the characters. Using morphology to thin out the characters helps create space between them.
- Use binarization to check for non-uniform lighting issues. Use the `graythresh` and `imbinarize` functions to binarize the image. If the characters are not visible in the results of the binarization, then the image has a potential non-uniform lighting issue. Try top-hat filtering, using the `imtophat` function, or other techniques that deal with removing non-uniform illumination.
- Use the `roi` argument to isolate the text. You can specify the `roi` manually or use text detection.
- If your image looks like a natural scene containing words, such as a street scene, rather than a scanned document, try setting the `LayoutAnalysis` argument to either `"Block"` or `"Word"`.
- Ensure that the image contains dark text on a light background. To achieve this, you can binarize the image and invert it before passing it to the `ocr` function.

Version History

Introduced in R2014a

R2024b: Generate optimized C/C++ code using Embedded Coder

Starting in R2024b, you can generate optimized C/C++ code for ARM processors using Embedded Coder.

R2023a: `Language` and `TextLayout` name-value arguments removed

The `Language` and `TextLayout` name-value arguments have been removed. Use the `Model` and `LayoutAnalysis`.

References

[1] Smith, Ray. *An Overview of the Tesseract OCR Engine*. In Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), 629–33. IEEE, 2007. https://doi.org/10.1109/ICDAR.2007.4376991."

[2] Smith, R., D. Antonova, and D. Lee. *Adapting the Tesseract Open Source OCR Engine for Multilingual OCR*. Proceedings of the International Workshop on Multilingual OCR, (2009).

[3] R. Smith. *Hybrid Page Layout Analysis via Tab-Stop Detection*. Proceedings of the 10th international conference on document analysis and recognition. 2009.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `LayoutAnalysis`, `Model`, and `CharacterSet` must be compile-time constants.
- Generated code for this function requires that the target platform has Tesseract and Leptonica installed. By default, generated code uses precompiled, platform-specific shared libraries. To generate portable code without relying on precompiled libraries, configure MATLAB Coder as follows:

  ```
  cfg = coder.config("lib");
  cfg.TargetLang = "C++";
  cfg.UsePrecompiledLibraries = "Avoid";
  ```

- You can also generate optimized C/C++ code using Embedded Coder, provided Tesseract and Leptonica are installed on the target platform.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Image Labeler

Functions

`trainOCR` | `evaluateOCR` | `quantizeOCR` | `ocrTrainingData` | `detectTextCRAFT` | `imbinarize`

Objects

`ocrText` | `visionSupportPackages` | `ocrTrainingOptions`

Topics

“Getting Started with OCR”

“Segment and Read Text in Image”
“Recognize Text Using Optical Character Recognition (OCR)”
“Train an OCR Model to Recognize Seven-Segment Digits”
“Recognize Seven-Segment Digits Using OCR”
“Automate Ground Truth Labeling for OCR”
“Install OCR Language Data Files”
“Install Computer Vision Toolbox Add-on Support Files”

detectTextCRAFT

Detect texts in images by using CRAFT deep learning model

Syntax

```
bboxes = detectTextCRAFT(I)
bboxes = detectTextCRAFT(I,roi)
bboxes = detectTextCRAFT( ___ ,Name=Value)
```

Description

`bboxes = detectTextCRAFT(I)` detects texts in images by using character region awareness for text detection (CRAFT) deep learning model. The `detectTextCRAFT` function uses a pretrained CRAFT deep learning model to detect texts in an image. The pretrained CRAFT model can detect 9 languages that include Chinese, Japanese, Korean, Italian, English, French, Arabic, German, and Bangla (Indian).

Note To use the pretrained CRAFT model, you must install the Computer Vision Toolbox Model for Text Detection. You can download and install the Computer Vision Toolbox Model for Text Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons. To run this function, you will require the Deep Learning Toolbox.

`bboxes = detectTextCRAFT(I,roi)` detects texts within a region-of-interest (ROI) in the image.

`bboxes = detectTextCRAFT( ___ ,Name=Value)` specifies additional options by using name-value pair arguments. You can use the name-value pair arguments to fine-tune the detection results.

Examples

Detect Texts in Images by Using CRAFT Model

Read an input image into the MATLAB workspace.

```
I = imread("handicapSign.jpg");
```

Compute the text detection results by using the `detectTextCRAFT` function. The region and the affinity thresholds are set to default values. The output is a set of bounding boxes that contain the detected text regions.

```
bboxes = detectTextCRAFT(I);
```

Draw the output bounding boxes on the image by using the `insertShape` function.

```
Iout = insertShape(I,"rectangle",bboxes,LineWidth=3);
```

Display the text detection results.

```
figure
imshow(Iout)
```



Detect Texts in ROI by Using CRAFT

Read an input image into the MATLAB workspace.

```
visiondatadir = fullfile(toolboxdir('vision'),'visiondata');
I = imread(fullfile(visiondatadir,'imageSets','books','pairOfBooks.jpg'));
```

Specify a region of interest (ROI) within the input image.

```
roi = [120,80,250,200];
```

Detect texts within the specified ROI by using the `detectTextCRAFT` function. The region and affinity thresholds are set to default values. The output is a set of bounding boxes that contain the detected text regions.

```
bboxes = detectTextCRAFT(I,roi);
```

Draw the ROI and the output bounding boxes on the input image. Display the text detection results.

```
I = insertObjectAnnotation(I,"rectangle",roi,"ROI",Color="green");
Iout = insertShape(I,"rectangle",bboxes,LineWidth=3);
figure
imshow(Iout)
```



Detect Characters by Modifying Affinity Threshold

This example shows how to detect each character in the text regions of an input image by using the CRAFT model. You can achieve this by modifying the affinity threshold. This example also demonstrates the effect of different affinity threshold values on the detection results.

Read an input image into the MATLAB workspace.

```
visiondatadir = fullfile(toolboxdir('vision'),'visiondata');
I = imread(fullfile(visiondatadir,'bookCovers','book27.jpg'));
```

Specify the affinity threshold values to consider for detecting the text regions in the image.

```
threshold = [1 0.1 0.01 0.001 0.0004];
```

Preallocate a 4-D array `Iout` to store the output image with detection results.

```
Iout = zeros(size(I,1),size(I,2),size(I,3),length(threshold));
```

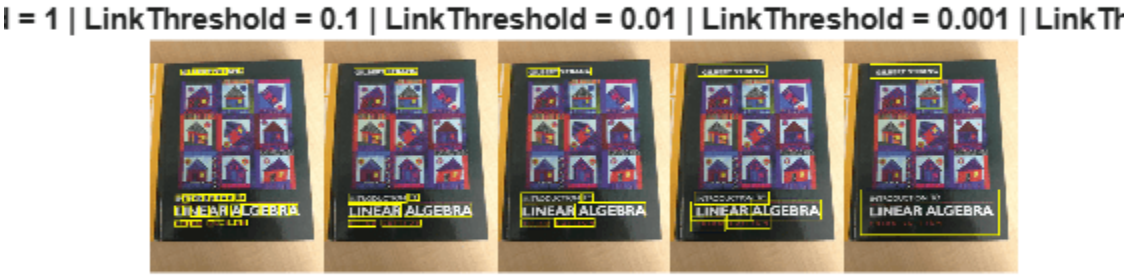
Compute the output for each affinity threshold value specified at the input. The output is a set of bounding boxes that contain the detected text regions. Draw the output bounding boxes on the image by using the `insertShape` function. The region threshold is set to the default value, 0.4.

```
for cnt = 1:length(threshold)
    bboxes = detectTextCRAFT(I,LinkThreshold=threshold(cnt));
    Iout(:,:,:,cnt) = insertShape(I,"rectangle",bboxes,LineWidth=3);
end
```

Display the text detection results obtained for different values of affinity threshold. You can notice that as the affinity threshold value decrease, the characters with less affinity scores are considered as

connected components and are grouped as a single instance. For good localization and detection results, the affinity threshold must be greater than zero.

```
figure
montage(uint8(Iout),Size=[1 5],BackgroundColor="white");
title(['LinkThreshold = ' num2str(threshold(1)) ' | LinkThreshold = ' num2str(threshold(2)) ' | L
    ' | LinkThreshold = ' num2str(threshold(4)) ' | LinkThreshold = ' num2str(threshold(5))]);
```



Input Arguments

`I` — Input image

2-D grayscale image | 2-D color image

Input image, specified as a 2-D grayscale image or 2-D color image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`roi` — Search rectangular region-of-interest

four-element vector

Search a rectangular region-of-interest in an image, specified as a four-element vector of the form [*x y width height*]. The vector specifies the upper left corner and size of a rectangular region in pixels. The region must be fully contained in the image.

When you specify this value, the `detectTextCRAFT` function detects texts that are present only within this ROI.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `bboxes = detectTextCRAFT(I,MaxSize=[10,10])` specifies the maximum size of the text region to detect in the input image

`CharacterThreshold` — Region threshold for characters
0.4 (default) | positive scalar

Region threshold for localizing each character in the image, specified as a positive scalar in the range [0, 1]. To increase the number of detections, lower the region threshold value. However, this will also result in false-positives. To reduce the number of false-positives, increase the region threshold value.

Data Types: `single` | `double`

`LinkThreshold` — Link threshold
0.4 (default) | positive scalar

Link threshold for grouping adjacent characters into a word, specified as a positive scalar in the range [0, 1]. You can increase the number of character level detections by increasing the link threshold. To detect each character in the image, set this value to 1. For good localization and detection results, the link threshold must be greater than zero.

Data Types: `single` | `double`

`MinSize` — Size of smallest detectable text region
[6,6] (default) | two-element vector

Size of smallest detectable text region in the image, specified as a two-element vector of form [*height width*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxSize` — Size of largest detectable text region
size of input image (default) | two-element vector

Size of largest detectable text region in the image, specified as a two-element vector of form [*height width*]. By default, this value is set to the height and width of the input image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"cpu"` | `"gpu"`

Hardware resource for processing images with the CRAFT model, specified as `"auto"`, `"gpu"`, or `"cpu"`.

ExecutionEnvironment	Description
`"auto"`	Use a GPU if available. Otherwise, use the CPU. The use of GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"gpu"`	Use the GPU. If a suitable GPU is not available, the function returns an error message.
`"cpu"`	Use the CPU.

Data Types: `char` | `string`

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as `"auto"`, `"mex"`, or `"none"`.

Acceleration	Description
`"auto"`	Automatically apply a number of optimizations suitable for the input network and hardware resource.
`"mex"`	Compile and execute a MEX function. This option is available when using a GPU only. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).
`"none"`	Disable all acceleration.

The default option is `"auto"`. If you use the `"auto"` option, MATLAB does not ever generate a MEX function.

Using the `"Acceleration"` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is only available when you are using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox). If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error.

Output Arguments

`bboxes` — Bounding boxes for detected text regions
M-by-4 matrix

Bounding boxes specifying the detected text regions, returned as an *M*-by-4 matrix. *M* is the number of detected text regions. Each row in the matrix is a vector of form [*x y width height*]. The vector specifies the upper left corner and size of the detected region in pixels.

Version History

Introduced in R2022a

R2024a: Generate CUDA code using GPU Coder

`detectTextCRAFT` now supports the generation of optimized CUDA code (requires GPU Coder).

R2024a: Generate C/C++ code using MATLAB Coder

`detectTextCRAFT` now supports the generation of C/C++ code (requires MATLAB Coder).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `roi` argument must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Code generation does not support variable-size data for the input argument `I`.
- Only the `CharacterThreshold`, `LinkThreshold`, `MinSize`, and `MaxSize` name-value arguments are supported.
- When you set `build_type` argument of `coder.config` object to `dll`, for generating code that does not use any third-party library, the `DynamicMemoryAllocationForFixedSizeArrays` property of the `coder.CodeConfig` object must be set to `true`.
- To avoid memory allocation error during library free C++ code generation on Windows platform, the `cfg.DynamicMemoryAllocationForFixedSizeArrays` property of the `coder.CodeConfig` object must be set to `true`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `roi` argument must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Code generation does not support variable-size data for the input argument `I`.
- Only the `CharacterThreshold`, `LinkThreshold`, `MinSize`, and `MaxSize` name-value arguments are supported.

See Also

`ocrText` | `ocr` | `insertShape` | `insertObjectAnnotation`

Topics

“Automatically Detect and Recognize Text Using Pretrained CRAFT Network and OCR”
“Recognize Text Using Optical Character Recognition (OCR)”
“Recognize Seven-Segment Digits Using OCR”
“Segment and Read Text in Image”

pcdenoise

Remove noise from 3-D point cloud

Syntax

```
ptCloudOut = pcdenoise(ptCloudIn)
[ptCloudOut,inlierIndices,outlierIndices] = pcdenoise(ptCloudIn)
[ptCloudOut, ___ ] = pcdenoise( ___ Name=Value)
```

Description

`ptCloudOut = pcdenoise(ptCloudIn)` returns a filtered point cloud that removes outliers.

`[ptCloudOut,inlierIndices,outlierIndices] = pcdenoise(ptCloudIn)` additionally returns the linear indices to the points that are identified as inliers and outliers.

`[ptCloudOut, ___ ] = pcdenoise( ___ Name=Value)` specifies a name-value argument in addition to any combination of arguments from previous syntaxes. For example, `Threshold=1.0` sets the outlier threshold to `1.0`.

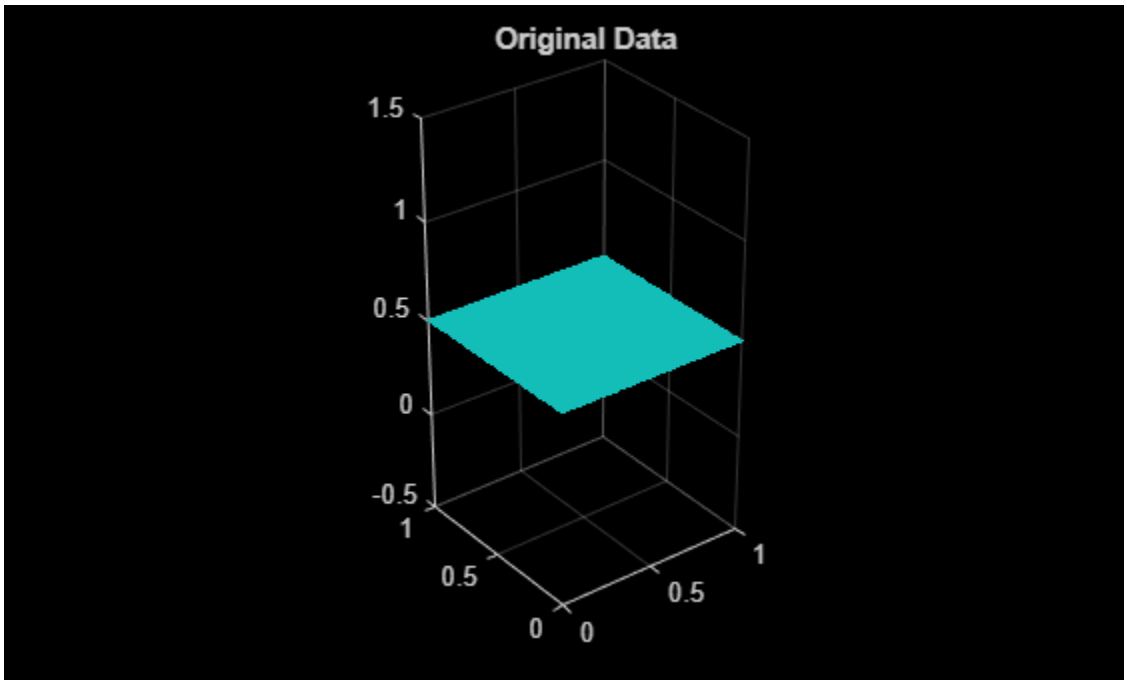
Examples

Remove Outliers from Noisy Point Cloud

Create a plane point cloud.

```
gv = 0:0.01:1;
[X,Y] = meshgrid(gv,gv);
ptCloud = pointCloud([X(:),Y(:),0.5*ones(numel(X),1)]);

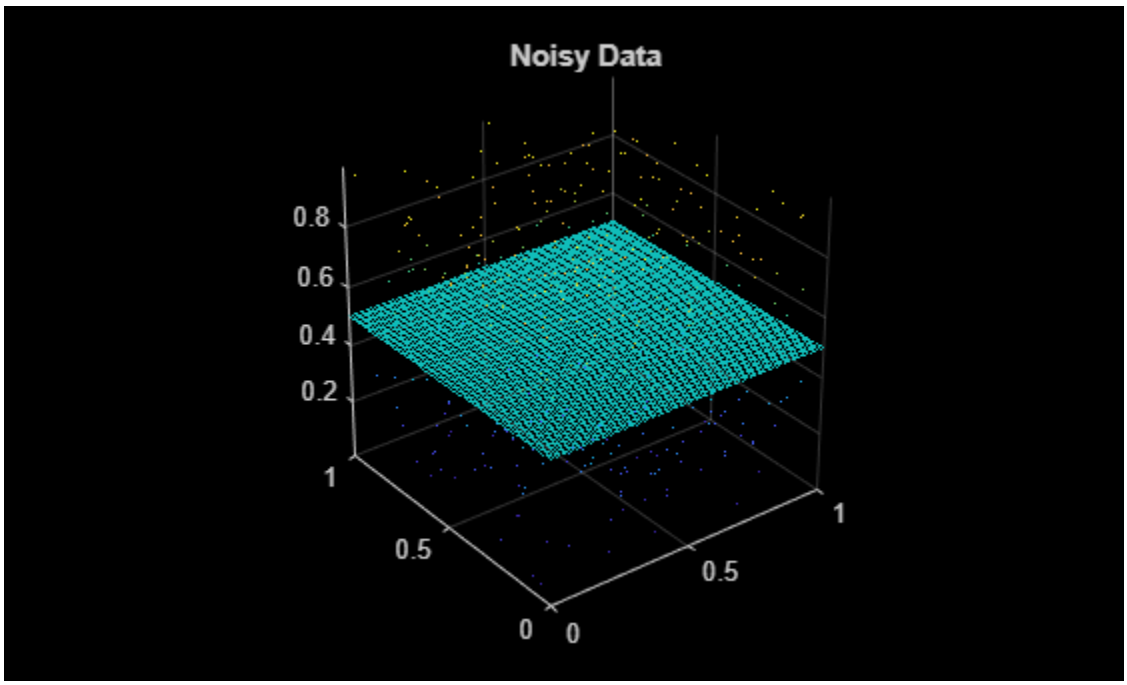
figure
pcshow(ptCloud);
title('Original Data');
```



Add uniformly distributed random noise.

```
noise = rand(500, 3);
ptCloudA = pointCloud([ptCloud.Location; noise]);

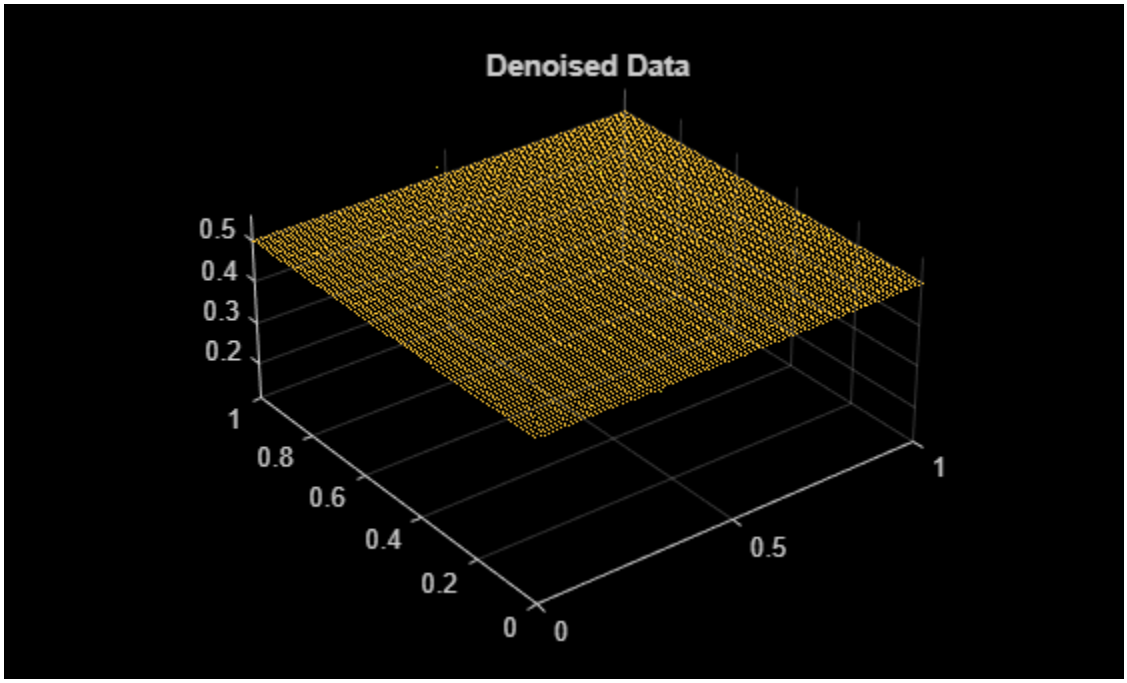
figure
pcshow(ptCloudA);
title('Noisy Data');
```



Remove outliers.

```
ptCloudB = pcdenoise(ptCloudA);

figure;
pcshow(ptCloudB);
title('Denoised Data');
```



Preserve Organized Structure of Point Cloud After Denoising

Load an organized point cloud from a saved MAT file.

```
ld = load('drivingLidarPoints.mat');
orgPtCloud = ld.ptCloud;
```

Remove noise from the point cloud.

```
orgPtCloudOut = pcdenoise(orgPtCloud,PreserveStructure=true);
```

Input Arguments

`ptCloudIn` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Threshold=1.0` sets the threshold to `1.0`.

`NumNeighbors` — Number of nearest neighbor points

4 (default) | positive integer

Number of nearest neighbor points, specified as the comma-separated pair consisting of '`NumNeighbors`' and a positive integer in pixels. The value is used to estimate the mean of the average distance to neighbors of all points. Decreasing this value makes the filter more sensitive to noise. Increasing this value increases the number of computations.

Data Types: `single` | `double`

`Threshold` — Outlier threshold

`1.0` (default) | positive scalar

Outlier threshold, specified as the comma-separated pair consisting of '`Threshold`' and a positive scalar. By default, the threshold is one standard deviation from the mean of the average distance to neighbors of all points. A point is considered to be an outlier if the average distance to its *k*-nearest neighbors is above the specified threshold.

Data Types: `single` | `double`

`PreserveStructure` — Preserve the organized structure of a point cloud

`false` (default) | `true`

Preserve the organized structure of a point cloud as an *M*-by-*N*-by-3 location matrix, specified as `true` or `false`. The table describes the point cloud structure according to the value of `PreserveStructure`.

`PreserveStructure`	The function returns
`true`	An organized, denoised, point cloud. The “Location” on page 2-0 property that describes the structure of the point cloud, contains an *M*-by-*N*-by-3 matrix. Points that are not selected in the denoised point cloud are filled with `NaN`, and the corresponding color is set to `[0 0 0]`. To return an organized point cloud, the input must be an organized point cloud.

PreserveStructure	The function returns
false	An unorganized, denoised, point cloud. The “Location” on page 2-0 property that describes the structure of the point cloud, contains an *M*-by-3 matrix.

Output Arguments

ptCloudOut — Filtered point cloud
pointCloud object

Filtered point cloud, returned as a pointCloud object.

inlierIndices — Linear index of inlier points
1-by-*N* vector

Linear index of inlier points, returned as a 1-by-*N* vector.

Data Types: uint32

outlierIndices — Linear index of outlier points
1-by-*N* vector

Linear index of outlier points, returned as a 1-by-*N* vector.

Data Types: uint32

Version History

Introduced in R2015a

References

[1] Rusu, R. B., Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. “Towards 3D Point Cloud Based Object Maps for Household Environments”. *Robotics and Autonomous Systems Journal*. 2008.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

- The PreserveStructure name-value argument is not supported for GPU code generation.

See Also

Functions
pcdownsample | pcplayer | pcshow | pcwrite | pcread | pcfitplane | pcmerge | pctransform | pcregistericp | pcregisterndt

Objects
`pointCloud` | `planeModel` | `affine3d`

Topics
“3-D Point Cloud Registration and Stitching”

pcmerge

Merge two 3-D point clouds

Syntax

```
ptCloudOut = pcmerge(ptCloudA,ptCloudB,gridStep)
ptCloudOut = pcmerge( ___ ,GridFilter=gridFilterMethod)
```

Description

`ptCloudOut = pcmerge(ptCloudA,ptCloudB,gridStep)` returns a merged point cloud using a box grid filter in the region of overlap. `gridStep` specifies the size of the 3-D box for the filter.

`ptCloudOut = pcmerge( ___ ,GridFilter=gridFilterMethod)` sets the method to downsample the region of overlap between the merged point clouds.

Examples

Merge Two Identical Point Clouds Using Box Grid Filter

Create a point cloud with X, Y, Z points in [0, 100].

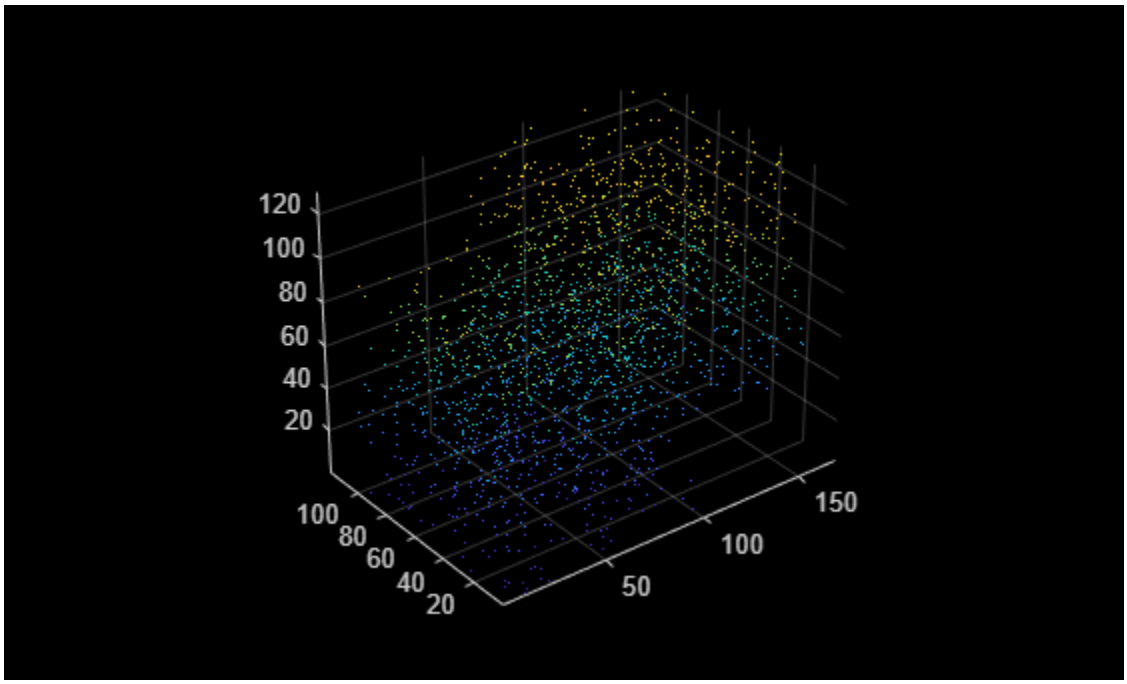
```
ptCloudA = pointCloud(100*rand(1000,3));
```

Create a partially overlapping point cloud.

```
ptCloudB = pointCloud([70 20 30] + 100*rand(1000,3));
```

Merge the two point clouds using a box filter.

```
ptCloudOut = pcmerge(ptCloudA, ptCloudB, 1);
pcshow(ptCloudOut);
```



Input Arguments

`ptCloudA` — Point cloud A
`pointCloud` object

Point cloud A, specified as a `pointCloud` object.

`ptCloudB` — Point cloud B
`pointCloud` object

Point cloud B, specified as a `pointCloud` object.

`gridStep` — Size of 3-D box for grid filter
positive scalar

Size of 3-D box for grid filter, specified as a positive scalar. Increasing the value of `gridStep` can reduce memory usage by merging more points, but it can also reduce the amount of detail captured. The grid filter returns a single point per 3-D box.

Data Types: `single` | `double`

`gridFilterMethod` — Method to downsample region of overlap
`"average"` | `"nearest"`

Method to downsample region of overlap between the merged point clouds, specified as one of:

- `"average"` (default) — Selects a point in the voxel by averaging the location, color, intensity, and normal of all points within the voxel.
- `"nearest"` — Selects the voxel's location, color, intensity, and normal based on the point nearest to the centroid of the voxel.

Output Arguments

`ptCloudOut` — Merged point cloud

`pointCloud` object

Merged point cloud, returned as a `pointCloud` object. The function computes the axis-aligned bounding box for the overlapped region between two point clouds. The bounding box is divided into grid boxes of the size specified by `gridStep`. Points within each grid box are downsampled using the specified grid filter type. Points outside the overlapped region are untouched.

Tips

- If the two point clouds do not have the same set of properties filled, such as `Color` or `Normal`, these properties will be cleared in the returned point cloud. For example, if `ptCloudA` has color but `ptCloudB` does not, then `ptCloudOut` will not contain color.
- The function filters out points with `NaN` or `Inf` values.

Version History

Introduced in R2015a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pointCloud` | `planeModel` | `affine3d` | `rigid3d`

Functions

`pccat` | `pcalign` | `pcplayer` | `pcshow` | `pcwrite` | `pcread` | `pctransform` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”

pcdownsample

Downsample 3-D point cloud

Syntax

```
ptCloudOut = pcdownsample(ptCloudIn,"random",percentage)
ptCloudOut = pcdownsample(ptCloudIn,"gridAverage",gridStep)
ptCloudOut = pcdownsample(ptCloudIn,"gridNearest",gridStep)
ptCloudOut = pcdownsample(ptCloudIn,"nonuniformGridSample",maxNumPoints)

[ptCloudOut,indices] = pcdownsample( ___ )
 ___  = pcdownsample( ___ ,Name=Value)
```

Description

`ptCloudOut = pcdownsample(ptCloudIn,"random",percentage)` returns a downsampled point cloud. Use this syntax for random sampling without replacement. The `percentage` input specifies the portion of the input to return to the output.

`ptCloudOut = pcdownsample(ptCloudIn,"gridAverage",gridStep)` returns a downsampled point cloud using a box grid filter. The `gridStep` input specifies the size of a 3-D box.

`ptCloudOut = pcdownsample(ptCloudIn,"gridNearest",gridStep)` returns a downsampled point cloud by selecting points closest to the centroid in each grid, using a box grid filter. The `gridStep` input specifies the size of a 3-D box.

`ptCloudOut = pcdownsample(ptCloudIn,"nonuniformGridSample",maxNumPoints)` returns a downsampled point cloud using nonuniform box grid filter. You must set the maximum number of points in the grid box, `maxNumPoints`, to at least `6`.

`[ptCloudOut,indices] = pcdownsample( ___ )` returns the linear indices for the points that are included in the downsampled point cloud. This syntax applies only when you use the `"random"`, `"gridNearest"`, or `"nonuniformGridSample"` downsampling methods.

` ___  = pcdownsample( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `pcdownsample(PtCloud,"random",percentage,PreserveStructure=true)` preserves the organized structure of a point cloud.

Examples

Downsample Point Cloud Using Box Grid Filter

Read a point cloud.

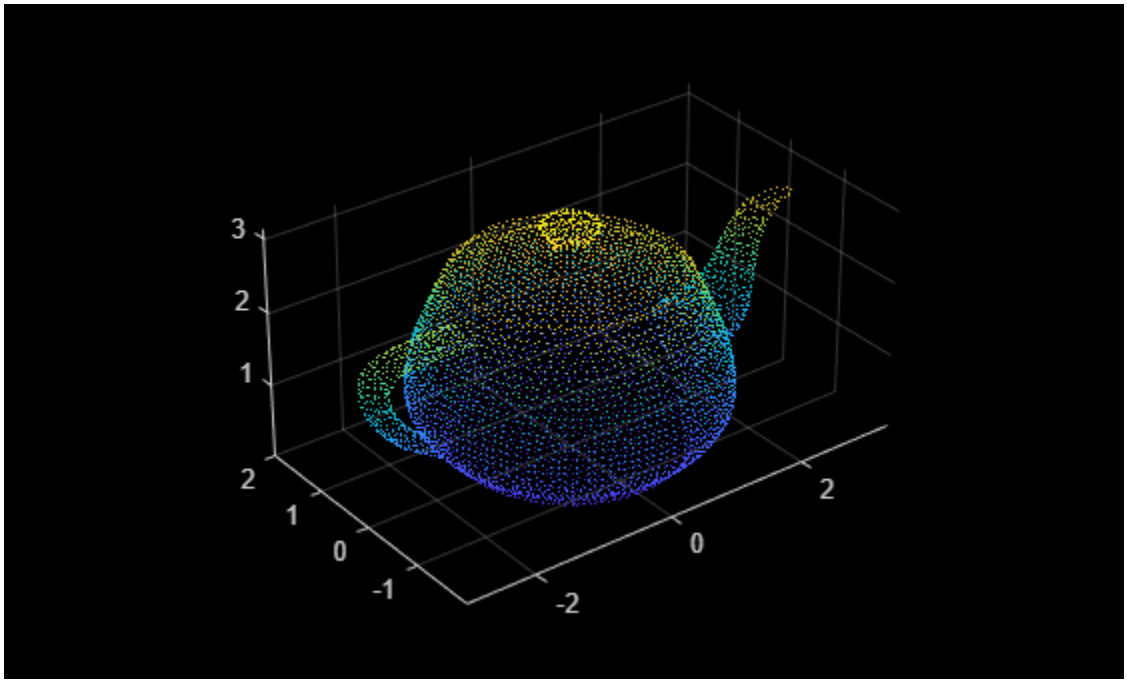
```
ptCloud = pcread("teapot.ply");
```

Set the 3-D resolution to be (0.1 x 0.1 x 0.1).

```
gridStep = 0.1;
ptCloudA = pcdownsample(ptCloud,'gridAverage',gridStep);
```

Visualize the downsampled data.

```
figure
pcshow(ptCloudA);
```

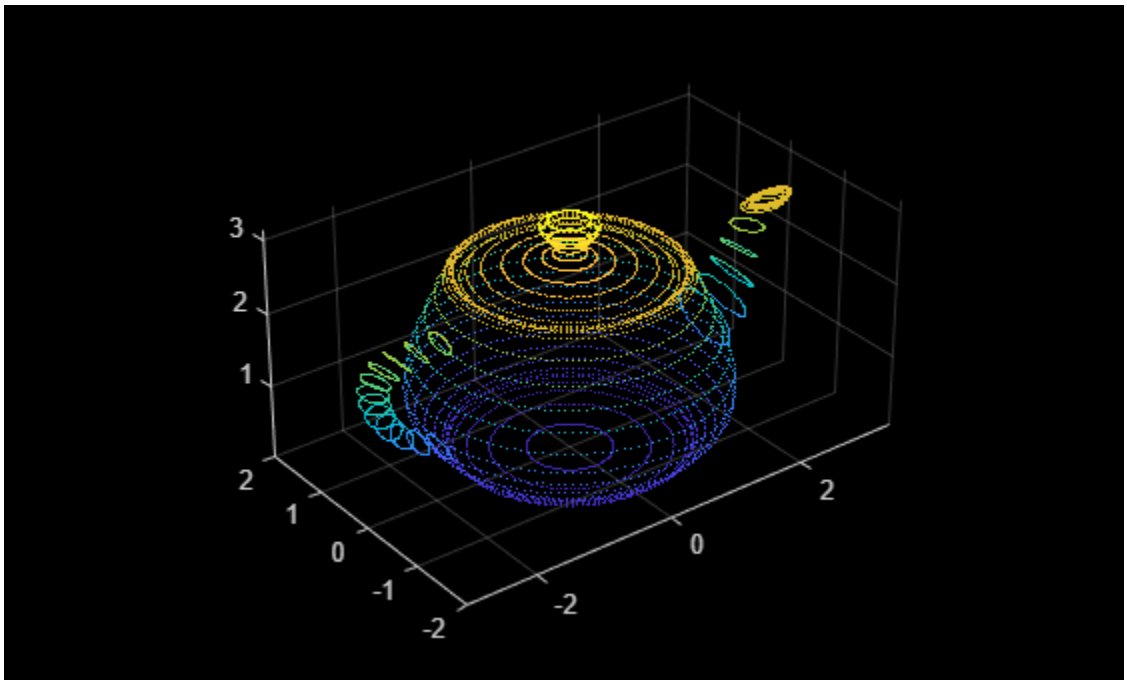


Compare the point cloud to data that is downsampled using a fixed step size.

```
stepSize = floor(ptCloud.Count/ptCloudA.Count);
indices = 1:stepSize:ptCloud.Count;
ptCloudB = select(ptCloud,indices);
```

Visualize the downsampled point cloud.

```
figure
pcshow(ptCloudB);
```



Remove Redundant Points from Point Cloud

Create a point cloud with all points sharing the same coordinates.

```
ptCloud = pointCloud(ones(100,3));
```

Set the 3-D resolution to a small value.

```
gridStep = 0.01;
```

The output now contains only one unique point.

```
ptCloudOut = pcdownsample(ptCloud,'gridAverage',gridStep)
```

```
ptCloudOut =
  pointCloud with properties:

     Location: [1 1 1]
        Count: 1
      XLimits: [1 1]
      YLimits: [1 1]
      ZLimits: [1 1]
        Color: [0×3 uint8]
       Normal: [0×3 double]
    Intensity: [0×1 double]
```

Preserve Organized Structure of Point Cloud After Downsampling

Load an organized point cloud from a saved MAT file.

```
ld = load('drivingLidarPoints.mat');
orgPtCloud = ld.ptCloud;
```

Downsample the point cloud. The output point cloud preserves the organized structure. Its location property is of size M-by-N-by-3.

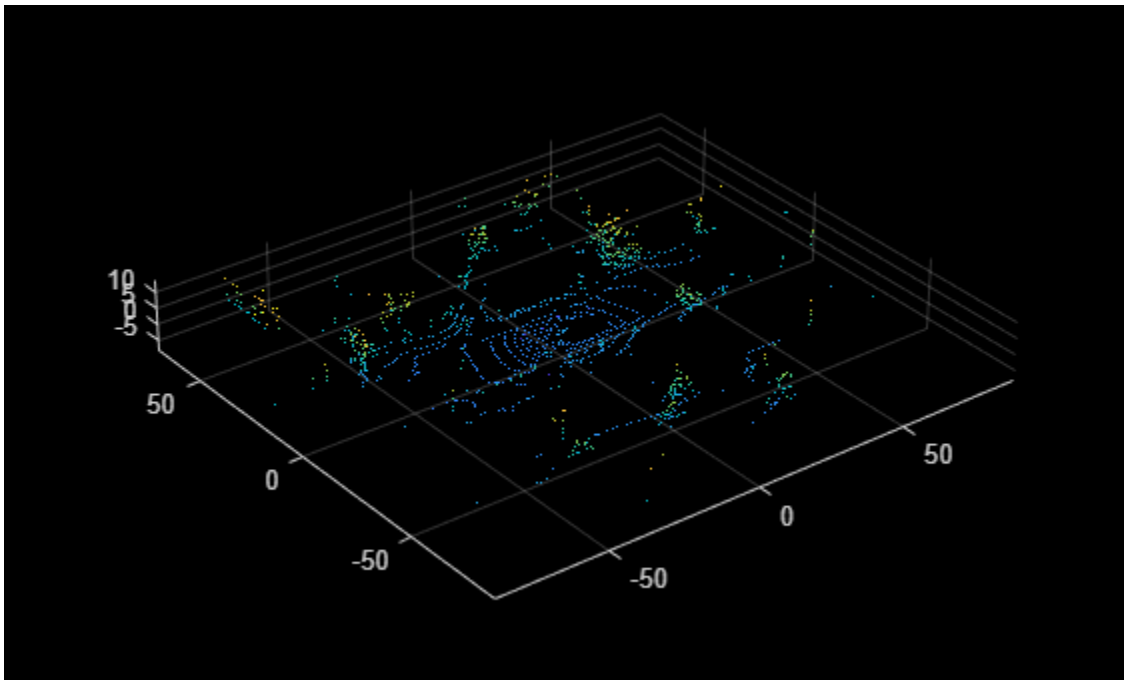
```
gridStep = 2;
orgPtCloudOut = pcdownsample(orgPtCloud,'gridNearest',gridStep,PreserveStructure=true)
```

```
orgPtCloudOut =
  pointCloud with properties:

     Location: [32×1083×3 double]
        Count: 34656
      XLimits: [-86.7523 90.5514]
      YLimits: [-85.8353 70.7870]
      ZLimits: [-8.2440 13.9009]
        Color: []
       Normal: []
    Intensity: [32×1083 double]
```

Visualize the downsampled point cloud.

```
figure
pcshow(orgPtCloudOut)
```



Input Arguments

`ptCloudIn` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`"random"` — Random downsample method

`"random"`

Random downsample method, specified as `"random"`. The `"random"` method is more efficient than the `"gridAverage"` downsample method, especially when it is applied before point cloud registration.

`percentage` — Percentage of input

nonnegative scalar

Percentage of input, specified as a nonnegative scalar in the range [0, 1]. The `percentage` input specifies the portion of the input for the function to return.

`"gridAverage"` — Grid average downsample method

`"gridAverage"`

Grid average downsample method, specified as `"gridAverage"`. Points within the same voxel are merged to a single point in the output voxel. Their color and normal properties are averaged accordingly. This method preserves the shape of the point cloud better than the `"random"` downsample method.

The function computes the axis-aligned bounding box for the entire point cloud. The bounding box is divided into a grid of voxels of size specified by `gridStep`. Points within each voxel are merged by averaging their locations, colors, and normals.

`"gridNearest"` — Nearest to centroid downsample method

`"gridNearest"`

Nearest to centroid downsample method, specified as `"gridNearest"`. Points are selected that are closest to the centroid in each voxel. Use the `gridNearest` downsampling technique to achieve a more precise depiction of color, intensity, and normal data in the downsampled point cloud.

The function computes the axis-aligned bounding box for the entire point cloud. The bounding box is divided into grid of voxels of size specified by `gridStep`.

`gridStep` — Size of 3-D box for grid filter

numeric value

Size of 3-D box for grid filter, specified as a numeric value. Increase the size of `gridStep` when there are not enough resources to construct a large fine-grained grid of voxels.

Data Types: `single` | `double`

`"nonuniformGridSample"` — Nonuniform grid sample method

`"nonuniformGridSample"`

Nonuniform grid sample method, specified as `"nonuniformGridSample"`. When you use the `"nonuniformGridSample"` algorithm, the normals are computed on the original data prior to downsampling. The downsampled output preserves more accurate normals.

`maxNumPoints` — Maximum number of points in grid box
integer

Maximum number of points in grid voxel, specified as an integer greater than `6`. The method randomly selects a single point from each voxel. If the normal was not provided in the input point cloud, this method automatically fills in the normal property in the `ptCloudOut` output.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `PreserveStructure=true` preserves the organized structure of a point cloud.

`PreserveStructure` — Preserve the organized structure of a point cloud
`false` (default) | `true`

Preserve the organized structure of a point cloud, specified as `true` or `false`. The table describes the point cloud structure according to the value of `PreserveStructure`.

`PreserveStructure`	The function returns
`true`	An organized, downsampled, point cloud. The "Location" on page 2-0 property that describes the structure of the point cloud, contains an *M*-by-*N*-by-3 matrix. Points that are not selected in the downsampled point cloud are filled with `NaN`, and the corresponding color is set to `[0 0 0]`. To return an organized point cloud, the input must be an organized point cloud, and the downsampling method must be `"random"`, `"nonuniformGridSample"`, or `"gridNearest"`.
`false`	An unorganized, downsampled, point cloud. The "Location" on page 2-0 property that describes the structure of the point cloud, contains an *M*-by-3 matrix.

Output Arguments

`ptCloudOut` — Downsampled point cloud
`pointCloud` object

Downsampled point cloud, returned as a `pointCloud` object.

`indices` — Indices of points in the downsampled point cloud
N-element vector

Indices of points in the downsampled point cloud, returned as a *N*-element vector.

Version History

Introduced in R2015a

R2024b: Specify `gridNearest` downsampling method

Use the `gridNearest` downsampling method, which selects the nearest point to the centroid of each grid.

R2025a: Generate C/C++ code for "`gridNearest`" method

The function now supports C/C++ code generation for the "`gridNearest`" method for downsampling point clouds.

References

[1] Pomerleau, F., F. Colas, R. Siegwart, and S. Magnenat. “Comparing ICP variants on real-world data sets.” *Autonomous Robots*. Vol. 34, Issue 3, April 2013, pp. 133–148.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- GPU code generation for nonuniform grid sample method is not optimized.
- For the random downsample method, GPU Coder uses the NVIDIA CUDA random library (cuRAND) to generate the random numbers. Because of architectural and implementation differences between the random number generators on the GPU and CPU, numerical verification does not always match.
- The `PreserveStructure` name-value argument is not supported for GPU code generation.
- GPU code generation does not support the "`gridNearest`" method for downsampling point clouds.

See Also

Functions
`pcdenoise` | `pcplayer` | `pcshow` | `pcwrite` | `pcread` | `pcfitplane` | `pcmerge` | `pctransform` | `pcregistericp` | `pcregisterndt`

Objects
`pointCloud` | `planeModel` | `affine3d`

Topics
“3-D Point Cloud Registration and Stitching”

pcread

Read 3-D point cloud from PLY or PCD file

Syntax

```
ptCloud = pcread(filename)
```

Description

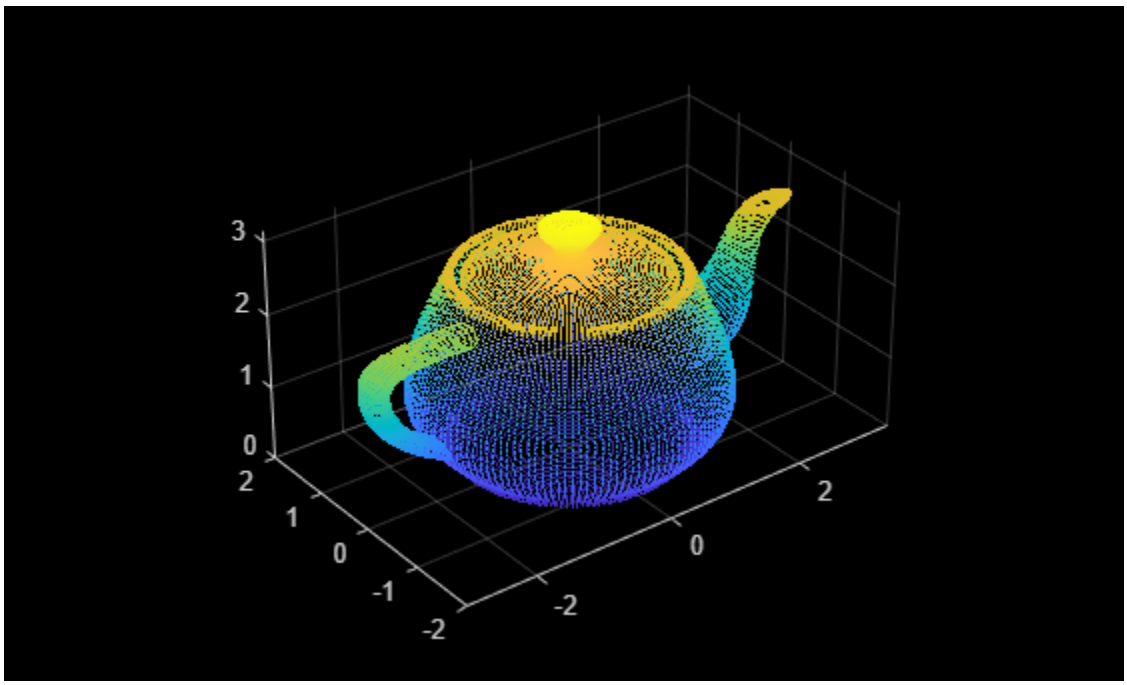
`ptCloud = pcread(filename)` reads a point cloud from the PLY or PCD file specified by the input `filename`. The function returns a `pointCloud` object, `ptCloud`.

Examples

Read Point Cloud from a PLY File

Load a PLY-format point cloud into the workspace.

```
ptCloud = pcread('teapot.ply');
pcshow(ptCloud);
```



Input Arguments

`filename` — File name
character vector | string scalar

File name, specified as a character vector or a string scalar. The input file type must be a PLY or a PCD format file.

The polygon (PLY) file format , also known as the Stanford triangle format stores three-dimensional data from 3-D scanners. It is a format for storing graphical objects that are described as a collection of polygons. A PLY file consists of a header, followed by a list of vertices and then, a list of polygons. The header specifies how many vertices and polygons are in the file. It also states what properties are associated with each vertex, such as (*x*,*y*,*z*) coordinates, normals, and color. The file format has two sub-formats: an ASCII representation and a binary version for compact storage and for rapid saving and loading. The header of both ASCII and binary files is ASCII text. Only the numeric data that follows the header is different between the two versions. See “The PLY Format” for details on the contents of a PLY file.

The point cloud data (PCD) file format also stores three-dimensional data. It was created by the authors of the widely used point cloud library (PCL) to accommodate additional point cloud data requirements. See The PCD (Point Cloud Data) file format.

Note This function only supports PCD file formats saved in version 0.7 (PCD_V7). It also only supports the header entries with the `COUNT` entry set to `1`. It does not support the `COUNT` entry set to a feature descriptor.

Output Arguments

`ptCloud` — Object for storing point cloud

`pointCloud` object

Object for storing point cloud, returned as a `pointCloud` object that contains the following PLY or PCD fields:

- `Location` property, stores the *x*, *y*, and *z* values.
- `Color` property, stores the red, green, and blue values.
- `Normal` property, stores the normal vectors for each point.
- `Intensity` property, stores the grayscale intensity for each point.

Version History

Introduced in R2015a

See Also

`pointCloud` | `pcplayer` | `pcshow` | `pcwrite` | `pcmerge` | `pcfitplane` | `planeModel` | `pctransform` | `pcdownsample` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”
“The PLY Format”

External Websites

The PCD (Point Cloud Data) file format

pcregistericp

Register two point clouds using ICP algorithm

Syntax

```
tform = pcregistericp(moving,fixed)
[tform,movingReg] = pcregistericp(moving,fixed)
[tform,movingReg,rmse] = pcregistericp(moving,fixed)
[___] = pcregistericp(moving,fixed,Name=Value)
```

Description

`tform = pcregistericp(moving,fixed)` returns a rigid transformation that registers a moving point cloud to a fixed point cloud.

The registration algorithm is based on the iterative closest point (ICP) algorithm. Best performance of this iterative process requires adjusting properties for your data. To improve the accuracy and efficiency of registration, consider downsampling point clouds using `pcdownsample` before using `pcregistericp`.

The registration algorithm requires point cloud normals when you select the `"pointToPlane"` or the `"planeToPlane"` (also known as Generalized-ICP or G-ICP) metric. If the `Normal` property of the input point cloud is empty, the function fills it.

`[tform,movingReg] = pcregistericp(moving,fixed)` additionally returns the transformed point cloud that aligns with the fixed point cloud.

`[tform,movingReg,rmse] = pcregistericp(moving,fixed)` additionally returns the root mean square error of the Euclidean distance between the aligned points, using any of the preceding syntaxes.

`[___] = pcregistericp(moving,fixed,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `pcregistericp(moving,fixed,Metric="planeToPlane")` sets the minimization metric to `"planeToPlane"`.

Examples

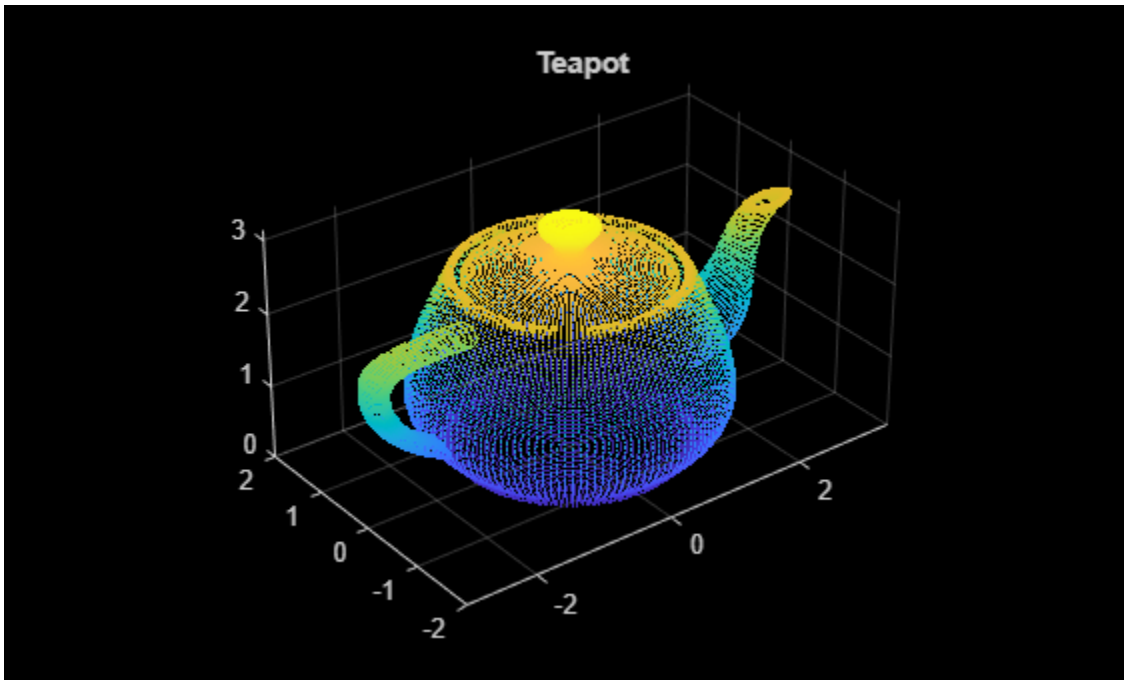
Align Two Point Clouds Using ICP Algorithm

Load point cloud data into the workspace.

```
ptCloud = pcread("teapot.ply");
```

Display the loaded point cloud.

```
pcshow(ptCloud)
title("Teapot")
```



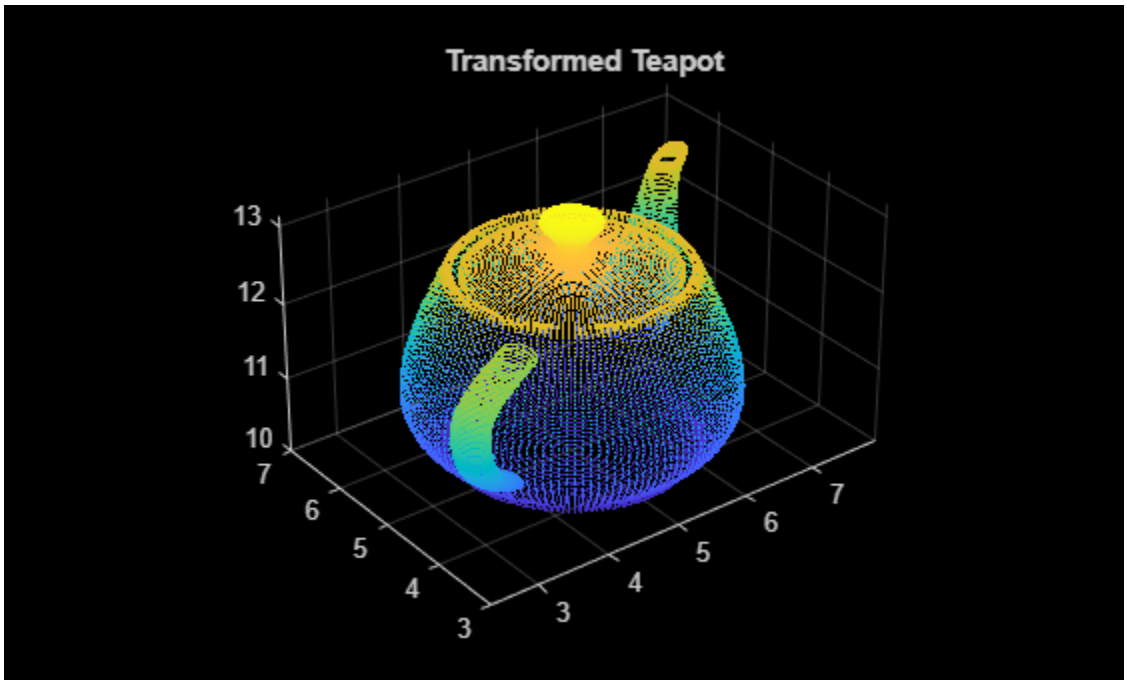
Create a transformation object with 30 degree rotation along the Z-axis and with a translation of [5 5 10].

```
rotationAngles = [0 0 30];
translation = [5 5 10];
tform1 = rigidtform3d(rotationAngles,translation);
```

Transform the point cloud and visualize the result.

```
ptCloudTformed = pctransform(ptCloud,tform1);

pcshow(ptCloudTformed)
title("Transformed Teapot")
```



Register the point clouds.

```
tform = pcregistericp(ptCloudTformed,ptCloud);
```

Compare the result with the true transformation.

```
disp(tform1.A);
```

```
    0.8660   -0.5000         0    5.0000
    0.5000    0.8660         0    5.0000
         0         0    1.0000   10.0000
         0         0         0    1.0000
```

```
tform2 = invert(tform);
disp(tform2.A);
```

```
    0.8660   -0.5000    0.0000    5.0000
    0.5000    0.8660    0.0000    5.0000
   -0.0000   -0.0000    1.0000   10.0000
         0         0         0    1.0000
```

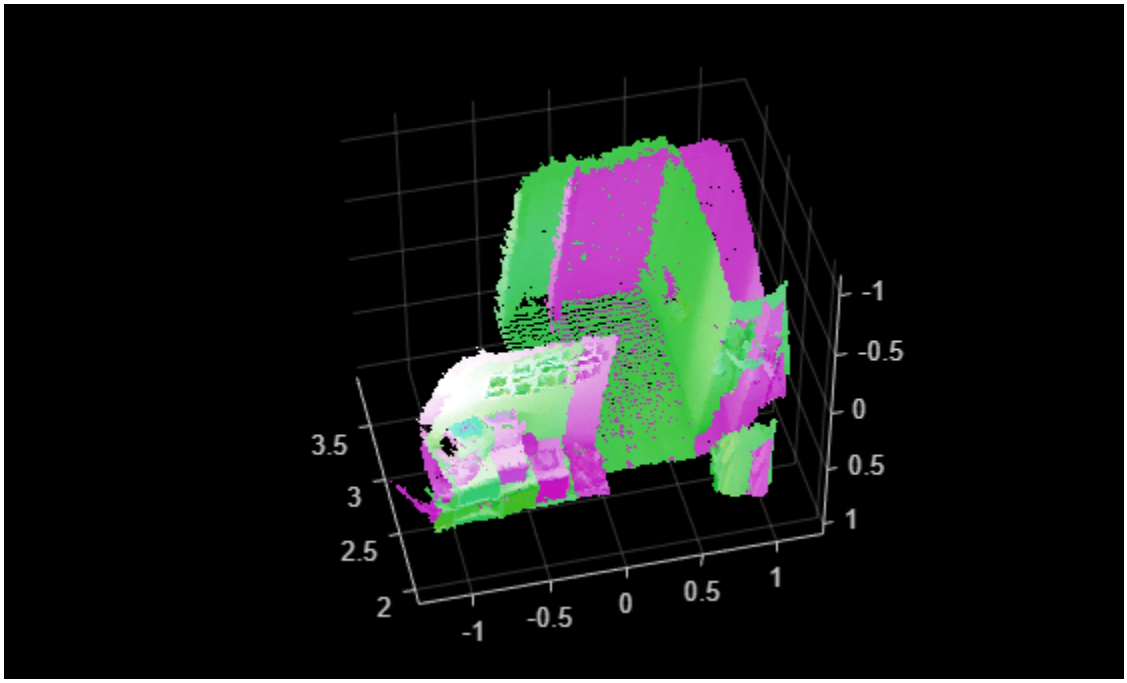
Align Two Point Clouds Using Plane-to-Plane ICP Registration

Load point cloud data into the workspace.

```
ld = load("livingRoom.mat");
moving = ld.livingRoomData{1};
fixed  = ld.livingRoomData{2};
```

Visualize the point cloud and set the direction to display the *y*-axis.

```
figure
pcshowpair(moving,fixed,VerticalAxis="Y",VerticalAxisDir="Down")
```



Downsample the point clouds using `nonuniformGridSample` method to improve the efficiency and accuracy of registration.

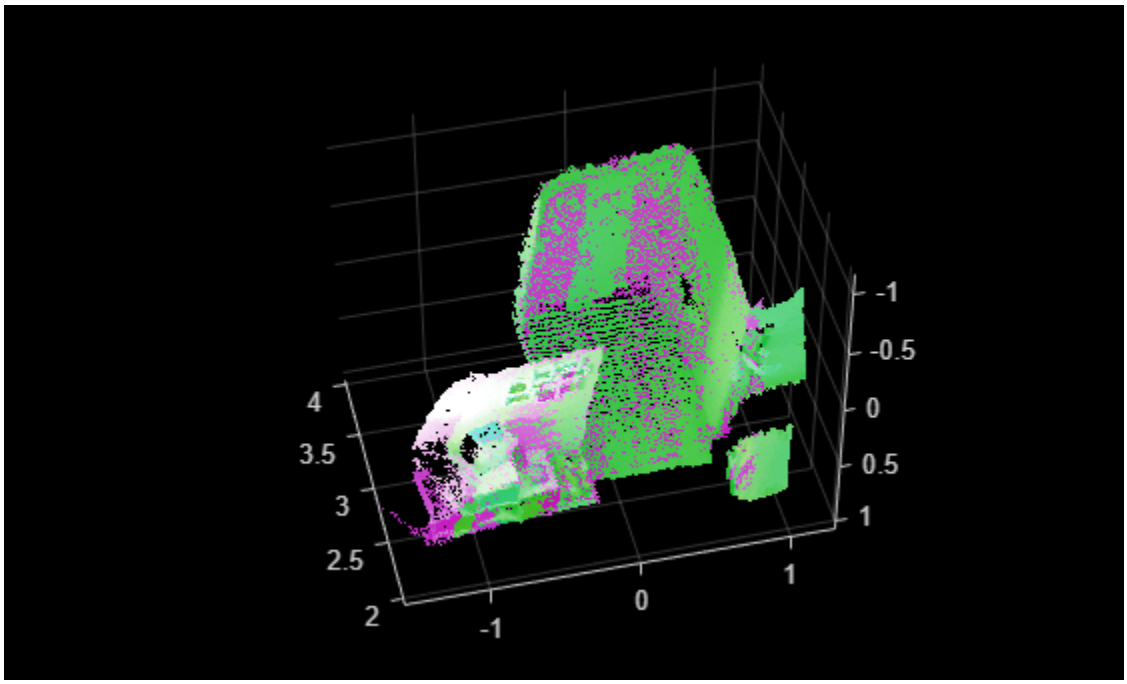
```
maxNumPoints = 12;
fixedDownsampled = pcdownsample(fixed,"nonuniformGridSample",maxNumPoints);
movingDownsampled = pcdownsample(moving,"nonuniformGridSample",maxNumPoints);
```

Align the point clouds using plane-to-plane (Generalized-ICP) registration.

```
[~,movingReg] = pcregistericp(movingDownsampled,fixedDownsampled,Metric="planeToPlane");
```

Visualize the point cloud alignment and set the direction to display the *y*-axis.

```
figure
pcshowpair(movingReg,fixed,VerticalAxis="Y",VerticalAxisDir="Down")
```



Align Two Colored Point Clouds

Load moving point cloud data having color information into the workspace.

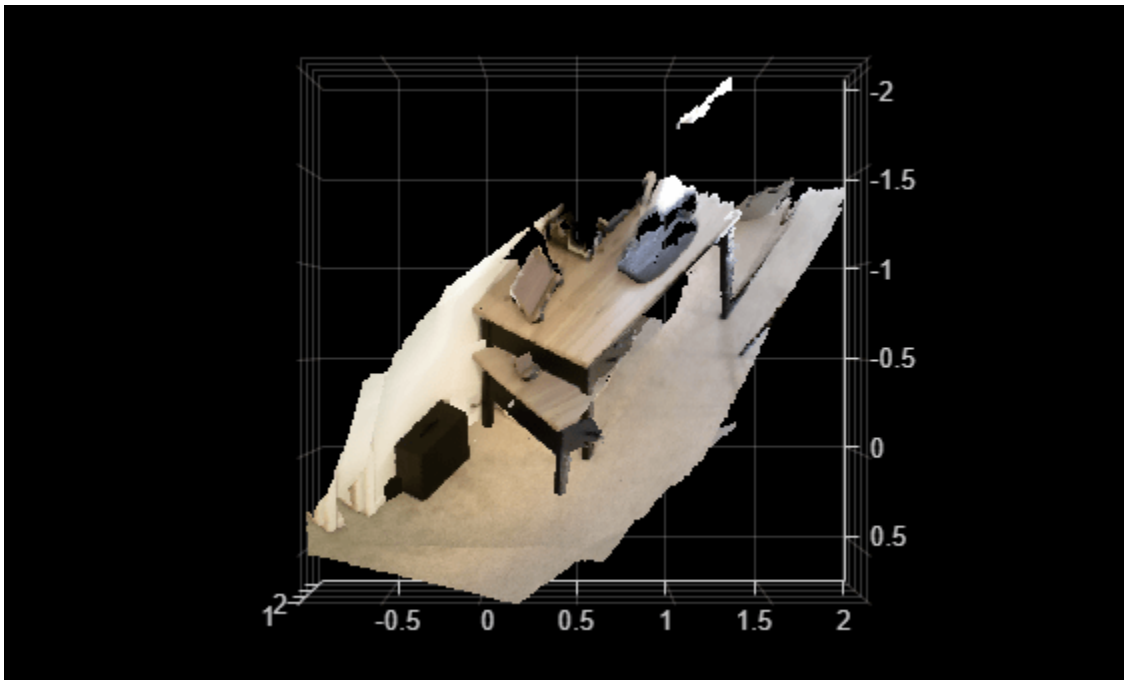
```
moving = pcread("apartmentMoving.pcd");
```

Load fixed point cloud data having color information into the workspace.

```
fixed  = pcread("apartmentFixed.pcd");
```

Visualize the colored point clouds and set the direction to display the *yx*-plane.

```
figure
pcshowpair(moving,fixed,"ViewPlane","YX","ColorSource","Color")
```

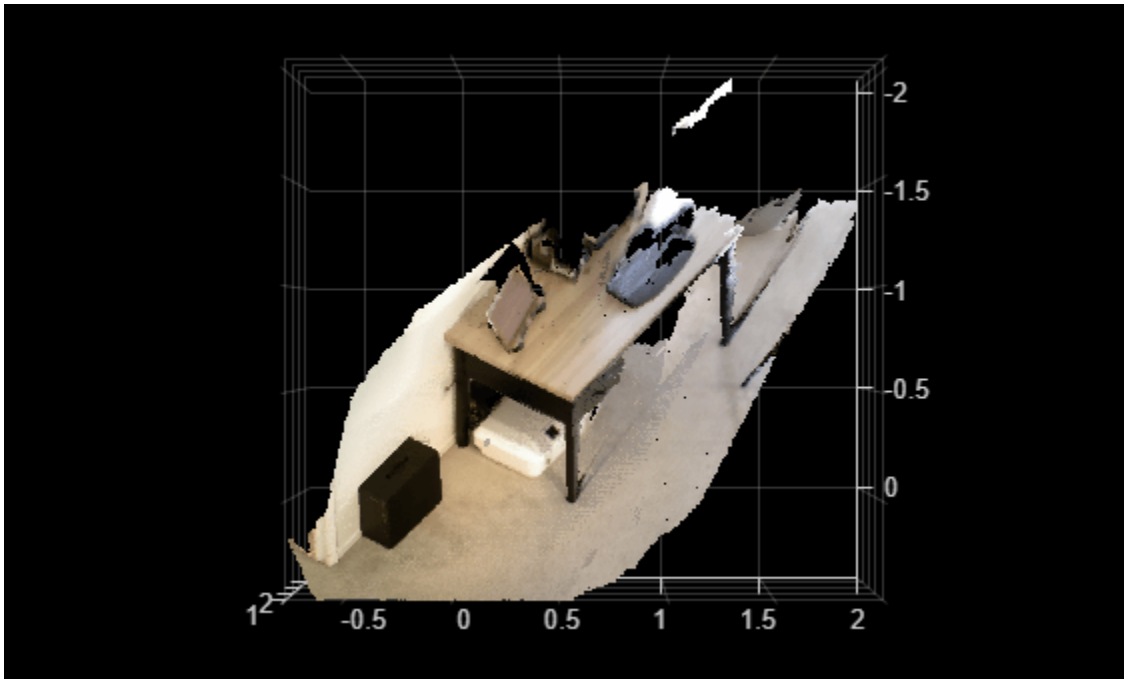


Align the colored point clouds using plane-to-plane (Generalized-ICP) registration and color information. Set the value to the `Metric` argument as `planeToPlaneWithColor`.

```
[~,movingReg] = pcregistericp(moving,fixed,Metric="planeToPlaneWithColor");
```

Visualize the point cloud alignment and set the direction to display the *yx*-plane.

```
figure
pcshowpair(movingReg,fixed,"ViewPlane","YX","ColorSource","Color")
```



Input Arguments

`moving` — Moving point cloud

`pointCloud` object

Moving point cloud, specified as a `pointCloud` object.

`fixed` — Fixed point cloud

`pointCloud` object

Fixed point cloud, specified as a `pointCloud` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Metric="planeToPlane"` sets the minimization metric for the ICP algorithm to `"planeToPlane"`.

`Metric` — Minimization metric

`"pointToPoint"` (default) | `"pointToPlane"` | `"planeToPlane"` | `"pointToPlaneWithColor"` | `"planeToPlaneWithColor"`

Minimization metric, specified as `"pointToPoint"`, `"pointToPlane"`, `"planeToPlane"`, `"planeToPlaneWithColor"`, or `"pointToPlaneWithColor"`. The iterative closest point (ICP) algorithm estimates the rigid transformation between the moving and fixed point clouds by minimizing the distance between the two point clouds according to the specified metric.

The normal based metrics `"pointToPlane"` and `"planeToPlane"` (also known as the Generalized ICP algorithm) can reduce the number of iterations required for registration, although they involve additional computation within each iteration. These metrics provide improved performance when registering planar surfaces. For point clouds that include color information, the `"pointToPlaneWithColor"` and `"planeToPlaneWithColor"` metrics provide further improvements in registration accuracy.

Downsample Method Selection

Downsample the point clouds using the `pcdownsample` function. If you select the `"pointToPlane"` or `"planeToPlane"` metric, the registration algorithm requires point cloud normals. The `"nonuniformGridSample"` metric computes normals on the original (pre-downsampled) data, which produces more accurate normals. As a result, using `"nonuniformGridSample"` can also improve accuracy for `"pointToPoint"` and `"pointToPlane"` ICP registration methods.

Use the `"random"`, `"gridAverage"`, or `"nonuniformGridSample"` downsampling methods with `pcdownsample`, as summarized in the table.

Metric	Moving Point Cloud Downsample Method	Fixed Point Cloud Downsample Method
"pointToPoint"	"random"	"random"
	"gridAverage"	"gridAverage"
"pointToPlane"	"gridAverage"	"gridAverage"
	"random"	"nonuniformGridSample"
"planeToPlane"	"gridAverage"	"gridAverage"
	"nonuniformGridSample"	"nonuniformGridSample"
"planeToPlaneWithColor"	"gridAverage"	"gridAverage"
	"nonuniformGridSample"	"nonuniformGridSample"

The "pointToPlaneWithColor" metric uses a multiscale registration approach where the point clouds are registered from coarser registration to finer registration. The result of the coarser registration is fed to the next finer registration step as the initial transformation. Therefore, the pcregistericp function does not use the normals in the input point clouds and thus the downsampling method does not affect its registration accuracy.

InlierRatio — Inlier percentage of matched pairs
1 (default) | scalar in the range (0, 1]

Inlier percentage of matched pairs, specified as a scalar between (0, 1]. This value specifies the percentage of matched pairs to use as inliers. The function considers a pair of matched points to be an inlier if the Euclidean distance between the two points is less than the specified percentage of the maximum Euclidean distance between any two points in a matched pair of the point clouds. By default, the function uses all matching pairs as inliers.

Specify only the InlierRatio or the InlierDistance name-value argument to exclude outliers. If you do not specify either argument, the function uses all matching pairs which is equivalent to setting the InlierRatio value to 1.

InlierDistance — Inlier distance threshold
scalar

Inlier distance threshold, specified as a scalar. This value sets the maximum distance between a point in the fixed point cloud and a matched point in the moving point cloud for the function to consider them as inliers. Decrease this value when you have a more accurate initial transformation, InitialTransform. Increase this value when there is more uncertainty in the initial transformation. Note that, increasing this value can decrease the speed of registration.

Specify only the InlierRatio or the InlierDistance name-value argument to exclude outliers. If you do not specify either argument, the function uses all matching pairs which is equivalent to setting the InlierRatio value to 1.

MaxIterations — Maximum number of iterations
30 (default) | positive integer

Maximum number of iterations, specified as a positive integer. This value specifies the maximum number of iterations over which the function attempts to make the two point clouds converge.

Tolerance — Tolerance between consecutive ICP iterations
[0.01 0.5] (default) | two-element vector

Tolerance between consecutive ICP iterations, specified as a two-element vector of the form [*Tdiff Rdiff*], where *Tdiff* and *Rdiff* represent the tolerances of absolute difference in translation and in rotation, respectively, estimated in consecutive ICP iterations. *Tdiff* measures the Euclidean distance between two translation vectors. *Rdiff* measures the angular difference in degrees. The algorithm stops when the average difference between estimated rigid transformations in the three most recent consecutive iterations is less than the specified tolerance values.

`InitialTransform` — Initial rigid transformation
`rigidtform3d` object

Initial rigid transformation, specified as a `rigidtform3d` object. This transformation must align the moving point cloud to the fixed point cloud. If not provided, the function uses a default translation that moves the centroid of the moving point cloud to the centroid of the fixed point cloud. The initial rigid transformation is useful when you have an external coarse estimation.

Note For the `"pointToPlaneWithColor"` metric, the default value of the `InitialTransform` name-value argument is a `rigidtform3d` object that performs an identity transformation.

`Verbose` — Display progress information
`false` or `0` (default) | `true` or `1`

Display progress information, specified as a logical `0` (`false`) or `1` (`true`). Specify `Verbose` as `true` to display progress information.

Output Arguments

`tform` — Rigid transformation
`rigidtform3d` object

Rigid transformation, returned as a `rigidtform3d` object. The rigid transformation registers a moving point cloud to a fixed point cloud. The `rigidtform3d` object describes the rigid 3-D transform. The iterative closest point (ICP) algorithm estimates the rigid transformation between the moving and fixed point clouds.

`movingReg` — Transformed point cloud
`pointCloud` object

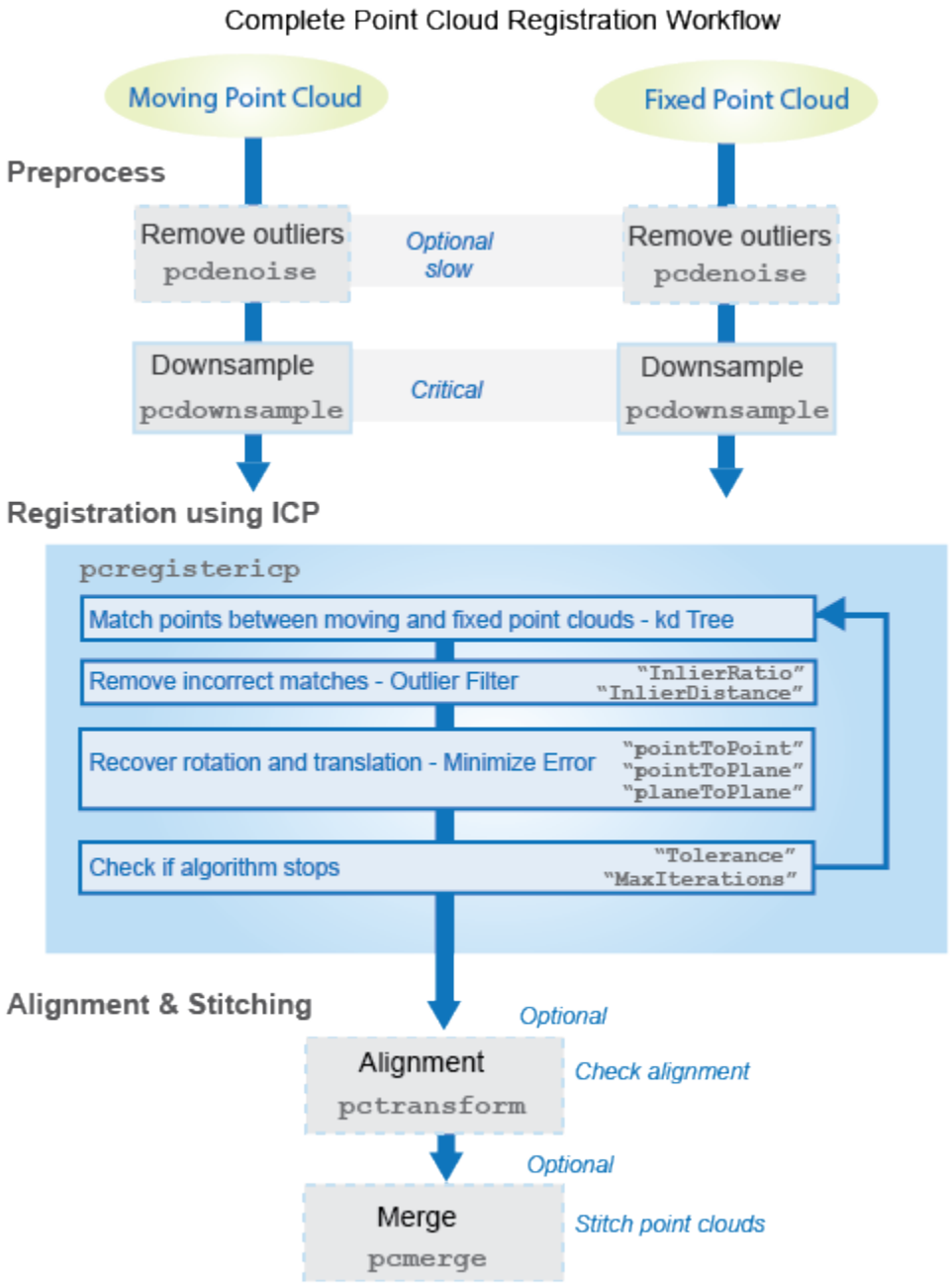
Transformed point cloud, returned as a `pointCloud` object. The transformed point cloud is aligned with the fixed point cloud.

`rmse` — Root mean square error
positive numeric scalar

Root mean square error, returned as a positive numeric scalar that represents the Euclidean distance between the aligned points. The aligned points is a result of the moving point cloud transformed by the `tform` output and the fixed point cloud. For each point in the fixed point cloud, the algorithm finds the closest point in the transformed point cloud, `movingReg`, then computes the Euclidean distance between the points, and then computes the `rmse`. A low `rmse` value indicates a more accurate registration. The `rmse` calculation is:

```
rmse = sqrt(sum(distance.^2,'all')/numel(distance));
```

Algorithms



Tips

- For ground vehicle point clouds, you can improve performance and accuracy by removing the ground using the `pcfitplane` or `segmentGroundFromLidarData` function before registration. For details on how to do this, see the `helperProcessPointCloud` helper function in the “Build a Map from Lidar Data” (Automated Driving Toolbox) example.

- To merge more than two point clouds, you can use the `pccat` function instead of the `pcmerge` function.

Version History

Introduced in R2018a

R2025a: `Extrapolate` name-value argument removed

Errors starting in R2025a

The `Extrapolate` name-value argument of the `pcregistericp` function has been removed. You can increase the `MaxIterations` argument or use the `pointToPlane` or `planeToPlane` values for the `Metric` name-value argument for similar accuracy.

R2024b: Generate optimized C/C++ code using Embedded Coder

Starting in R2024b, you can generate optimized C/C++ code for ARM processors using Embedded Coder.

R2024a: `InitialTransform` name-value argument default value changed for `"pointToPlaneWithColor"` metric

Behavior changed in R2024a

Starting in R2024a, for the `"pointToPlaneWithColor"` metric, the default value of the `InitialTransform` name-value argument is changed to a `rigidtform3d` object that performs an identity transformation. Prior to R2024a, the default value is a `rigidtform3d` object that contains the translation between the centroids of the moving and the fixed point clouds.

R2023b: Register point clouds with color information

Behavior changed in R2023b

Starting in R2023b, the `pcregistericp` function now supports the point cloud registration using color information. To include color information for registration, specify the `Metric` name-value argument as `"pointToPlaneWithColor"` or `"planeToPlaneWithColor"`. The `"pointToPlaneWithColor"` and `"planeToPlaneWithColor"` algorithm metrics do not support code generation.

R2023b: `Extrapolate` name-value argument will be removed in a future release

Not recommended starting in R2023b

The `Extrapolate` name-value argument of the `pcregistericp` function will be removed in a future release. You can increase the `MaxIterations` argument or use the `pointToPlane` or `planeToPlane` values for the `Metric` name-value argument for similar accuracy.

R2023b: `MaxIterations` name-value argument default value changed

Behavior changed in R2023b

The default value for the `MaxIterations` name-value argument was changed from `20` to `30` for `pointToPoint` and `pointToPlane` metrics.

R2023a: Returns RMSE of all points

Behavior changed in R2023a

Starting in R2023a, the `pcregistericp` function now returns the RMSE of all points. Previously, the function returned the RMSE of only the inliers. The function still outputs the RMSE of the inliers at each iteration when you set the `Verbose` argument to `true`.

R2023a: Default value modified for `InitialTransform` argument
Behavior changed in R2023a

Starting in R2023a, the `pcregistericp` function now uses a modified default value for the `InitialTransform` name-value argument. The default value is now a `rigidtform3d` object that contains a translation aligning the centroid of the moving point cloud to that of the fixed point cloud. Prior to R2023a, the default value is `rigidtform3d()`.

R2022b: Returns `rigidtform3d` object
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `pcregistericp` function now returns the output argument `tform` a `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `tform` as a `rigid3d` object, which uses the postmultiply convention.

You can now specify the `InitialTransform` name-value argument as a `rigidtform3d` object. Although you can still specify `InitialTransform` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2022b: Generalized-ICP registration algorithm
Behavior changed in R2022b

The `pcregistericp` function now supports the Generalized-ICP (G-ICP), or plane-to-plane, minimization metric. To use this registration algorithm, specify the `Metric` name-value argument as `"planeToPlane"`. The `"planeToPlane"` algorithm metric does not support code generation.

R2020a: Returns a `rigid3d` object
Behavior changed in R2020a

Starting in R2020a, the `pcregistericp` function returns a `rigid3d` object. Prior to R2020a, the function returned an `affine3d` object.

References

[1] Besl, P.J., and Neil D. McKay. “A Method for Registration of 3-D Shapes.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, no. 2 (February 1992): 239–256. https://doi.org/10.1109/34.121791.

[2] Chen, Yang, and Gérard Medioni. “Object Modelling by Registration of Multiple Range Images.” *Image and Vision Computing* 10, no. 3 (April 1992): 145–155. https://doi.org/10.1016/0262-8856(92)90066-C.

[3] Segal, A., Haehnel, D. and S. Thrun. "Generalized-ICP". *Robotics: Science and Systems V, Robotics: Science and Systems Foundation,*. (June 2009): 435-442. https://doi.org/10.15607/RSS.2009.V.021.

[4] Korn, Michael, Martin Holzkothen, and Josef Pauli. "Color supported generalized-ICP." *In 2014 International Conference on Computer Vision Theory and Applications (VISAPP),* 592–599. Lisbon, Portugal: IEEE, 2014. https://doi.org/10.5220/0004692805920599.

[5] Park, Jaesik, Qian-Yi Zhou, and Vladlen Koltun. "Colored point cloud registration revisited." *In Proceedings of the IEEE International Conference on Computer Vision (ICCV),* 143-152. Venice, Italy: IEEE, 2017. htttps://doi.org/10.1109/ICCV.2017.25.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The values for the `Extrapolate` and `Metric` arguments must be compile time constants.
- For C/C++ code generation on Windows platforms, only the Microsoft Visual C++ compiler is supported.

You can also generate optimized C/C++ code for ARM processors using Embedded Coder.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `"pointToPlaneWithColor"` and `"planeToPlaneWithColor"` algorithm metrics do not support GPU code generation.
- The point cloud output locations of the GPU-generated code differ from simulation by an order of `10e-3`, and the RMSE output differs by an order of `10e-4`, due to precision errors. However, the simulation and GPU code generation outputs have no notable visual differences.
- To ensure similar outputs from GPU code generation and simulation, the `Extrapolate` argument must be `false` or `0`.
- For enhanced GPU code generation performance, specify `"Location"`property of input point clouds of type `single`.
- For enhanced GPU code generation performance of `"planeToPlane"` metric, disable calls to cuBLAS and cuFFT CUDA libraries.

See Also

Functions

`pcregistercorr` | `pcregisterndt` | `pcregistercpd` | `pctransform` | `pcshowpair` | `pcshow` | `pcdownsample` | `pcfitplane` | `pcmerge` | `pcdenoise`

Objects

`pointCloud` | `rigidtform3d`

Topics

“3-D Point Cloud Registration and Stitching”
“Implement Point Cloud SLAM in MATLAB”

pcregisterndt

Register two point clouds using NDT algorithm

Syntax

```
tform = pcregisterndt(moving,fixed,gridStep)
[tform,movingReg] = pcregisterndt( ___ )
[tform,movingReg,rmse] = pcregisterndt( ___ )
[ ___ ] = pcregisterndt( ___ ,Name=Value)
```

Description

`tform = pcregisterndt(moving,fixed,gridStep)` returns the rigid transformation that registers the moving point cloud with the fixed point cloud. The fixed point cloud is voxelized into cubes of size `gridStep`.

The registration algorithm is based on the normal-distributions transform (NDT) algorithm [1] [2]. Best performance of this iterative process requires adjusting properties for your data. To improve accuracy and efficiency of registration, consider downsampling the moving point cloud by using `pcdownsample` before using `pcregisterndt`.

`[tform,movingReg] = pcregisterndt( ___ )` also returns the transformed point cloud that aligns with the fixed point cloud.

`[tform,movingReg,rmse] = pcregisterndt( ___ )` also returns the root mean square error of the Euclidean distance between the aligned point clouds.

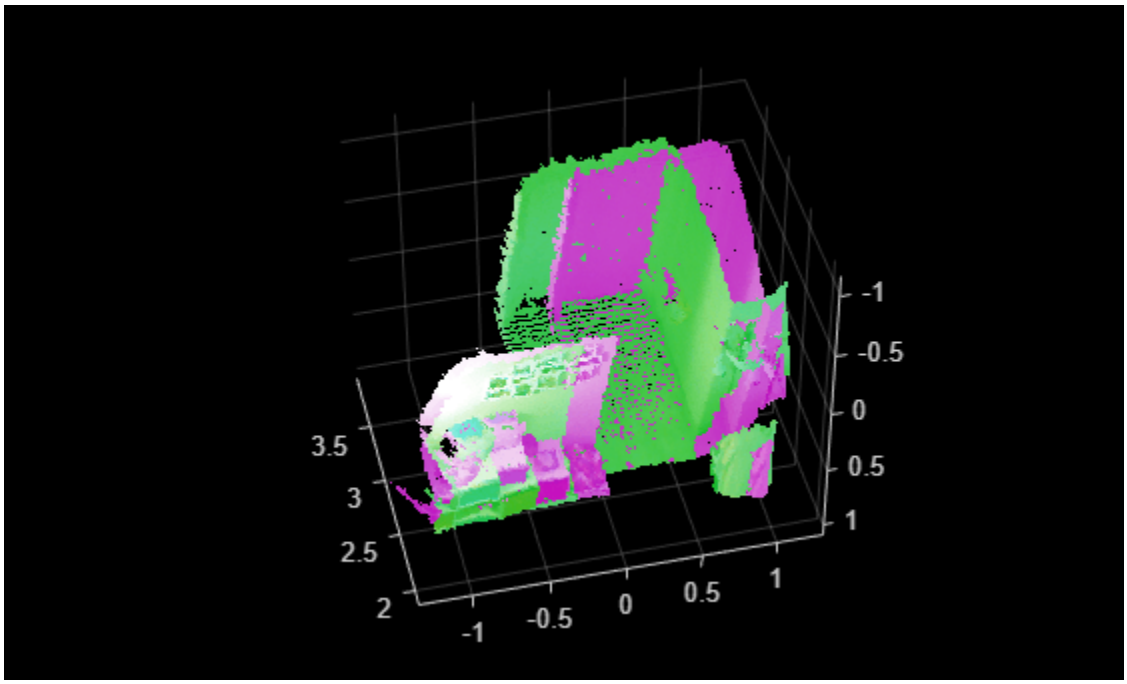
`[ ___ ] = pcregisterndt( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MaxIterations=20` stops the NDT algorithm after 20 iterations.

Examples

Align Two Point Clouds Using NDT Algorithm

Load point cloud data.

```
ld = load('livingRoom.mat');
moving = ld.livingRoomData{1};
fixed = ld.livingRoomData{2};
figure
pcshowpair(moving,fixed,VerticalAxis='Y',VerticalAxisDir='Down')
```



To improve the efficiency and accuracy of the NDT registration algorithm, downsample the moving point cloud.

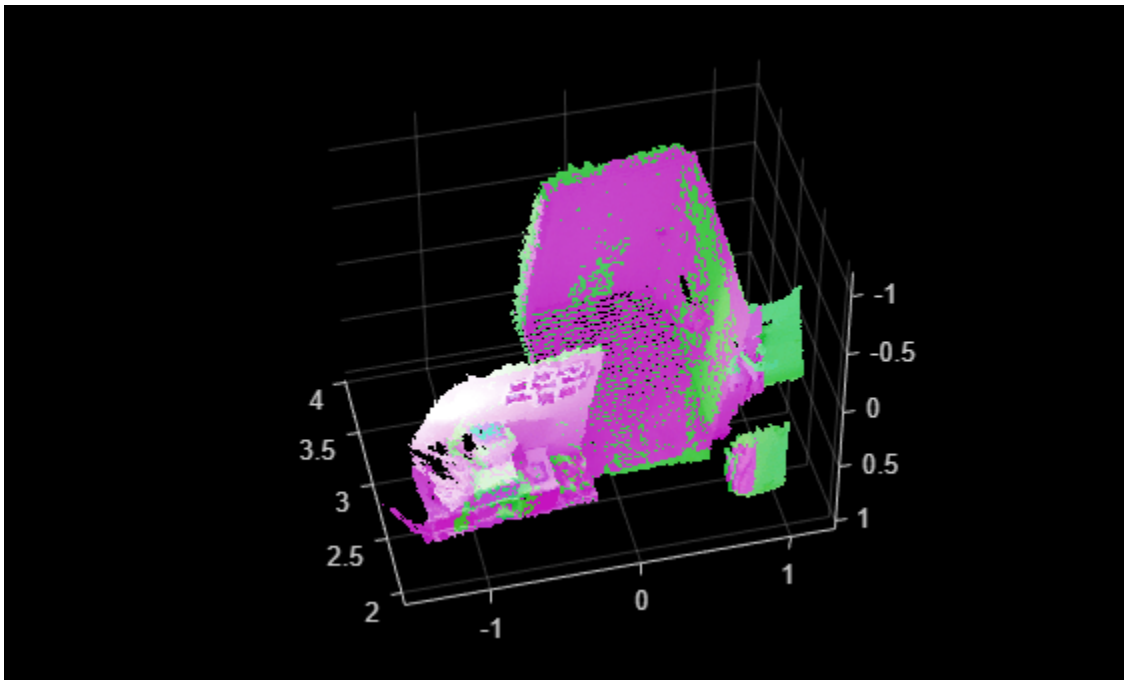
```
movingDownsampled = pcdownsample(moving,'gridAverage',0.1);
```

Voxelize the point cloud into cubes of sidelength 0.5. Apply the rigid registration using the NDT algorithm.

```
gridStep = 0.5;
tform = pcregisterndt(movingDownsampled,fixed,gridStep);
```

Visualize the alignment.

```
movingReg = pctransform(moving,tform);
figure
pcshowpair(movingReg,fixed,VerticalAxis='Y',VerticalAxisDir='Down')
```



Align Two Lidar Point Clouds

Construct a `velodyneFileReader` object.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Read and visualize two lidar point clouds.

```
frameNumber = 1;
skipFrame   = 5;
fixed = readFrame(veloReader,frameNumber);
moving = readFrame(veloReader,frameNumber + skipFrame);
```

Segment the ground plane and remove it from the fixed point cloud.

```
groundPtsIdxFixed = segmentGroundFromLidarData(fixed);
fixedSeg = select(fixed,~groundPtsIdxFixed,OutputSize='full');
```

Segment the ground plane and remove it from the moving point cloud.

```
groundPtsIdxMoving = segmentGroundFromLidarData(moving);
movingSeg = select(moving,~groundPtsIdxMoving,OutputSize='full');
```

Downsample the moving point cloud to improve accuracy and the efficiency of processing the points.

```
movingDownsampled = pcdownsample(movingSeg,gridAverage=0.2);
```

Register the moving point cloud with respect to the fixed point cloud.

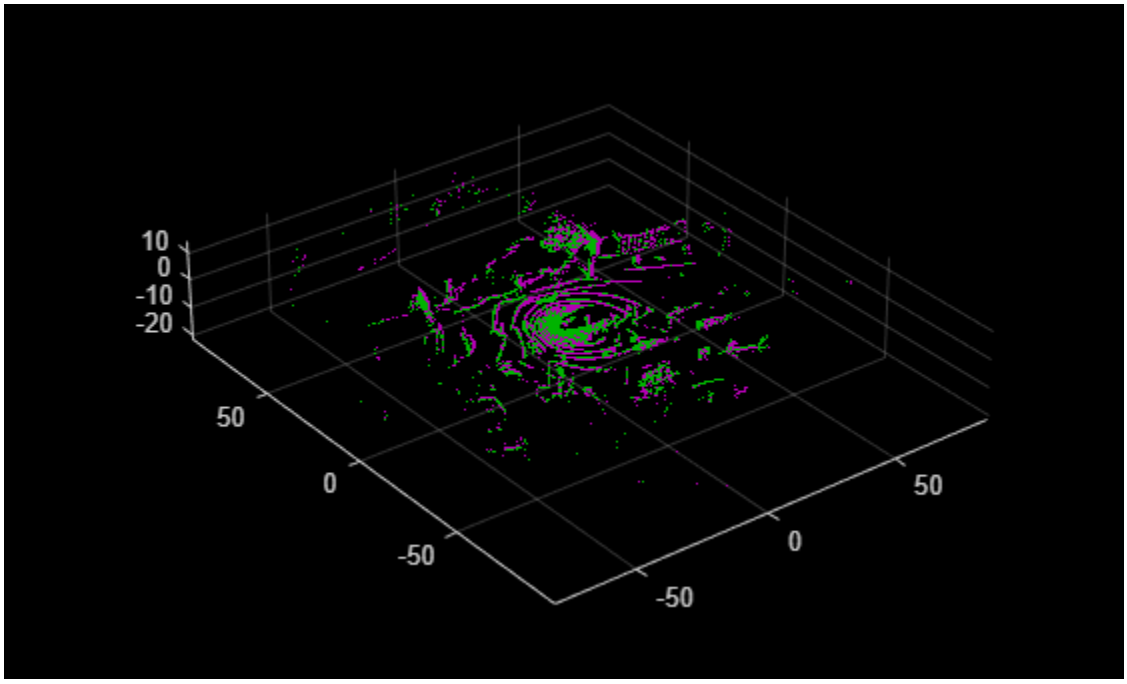
```
gridStep = 5;
tform = pcregisterndt(movingDownsampled,fixedSeg,gridStep);
```

Transform the moving point cloud using an estimated rigid transformation.

```
movingReg = pctransform(moving,tform);
```

Visualize the alignment of the point clouds.

```
figure
pcshowpair(movingReg,fixed)
```



Input Arguments

`moving` — Moving point cloud

`pointCloud` object

Moving point cloud, specified as a `pointCloud` object.

`fixed` — Fixed point cloud

`pointCloud` object

Fixed point cloud, specified as a `pointCloud` object.

`gridStep` — Size of voxels

positive scalar

Size of the 3-D cube that voxelizes the fixed point cloud, specified as a positive scalar.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxIterations=20` stops the NDT algorithm after 20 iterations.

`InitialTransform` — Initial rigid transformation
`rigidtform3d` object

Initial rigid transformation, specified as a `rigidtform3d` object. This transformation must align the moving point cloud to the fixed point cloud. If not provided, the function uses a default translation that moves the centroid of the moving point cloud to the centroid of the fixed point cloud.

`OutlierRatio` — Percentage of outliers
`0.55` (default) | scalar in the range [0, 1)

Expected percentage of outliers with respect to a normal distribution, specified as a scalar in the range [0, 1). The NDT algorithm assumes a point is generated by a mixture of a normal distribution for inliers and a uniform distribution for outliers. A larger value of `OutlierRatio` reduces the influence of outliers.

Data Types: `single` | `double`

`MaxIterations` — Maximum number of iterations
`30` (default) | positive integer

Maximum number of iterations before NDT stops, specified as a positive integer.

Data Types: `single` | `double`

`Tolerance` — Tolerance between consecutive NDT iterations
`[0.01 0.5]` (default) | 2-element vector

Tolerance between consecutive NDT iterations, specified as a 2-element vector with nonnegative values. The vector, [*Tdiff Rdiff*], represents the tolerance of absolute difference in translation and rotation estimated in consecutive NDT iterations. *Tdiff* measures the Euclidean distance between two translation vectors. *Rdiff* measures the angular difference in degrees. The algorithm stops when the difference between estimated rigid transformations in the most recent consecutive iterations falls below the specified tolerance value.

Data Types: `single` | `double`

`Verbose` — Display progress information
`false` (default) | `true`

Display progress information, specified as a numeric or logical `0` (`false`) or `1` (`true`). To display the progress information, set `Verbose` to `true`.

Output Arguments

`tform` — Rigid transformation
`rigidtform3d` object

Rigid transformation, returned as a `rigidtform3d` object. `tform` describes the rigid 3-D transformation that registers the moving point cloud, `moving`, to the fixed point cloud, `fixed`.

`movingReg` — Transformed point cloud
`pointCloud` object

Transformed point cloud, returned as a `pointCloud` object. The transformed point cloud is aligned with the fixed point cloud, `fixed`.

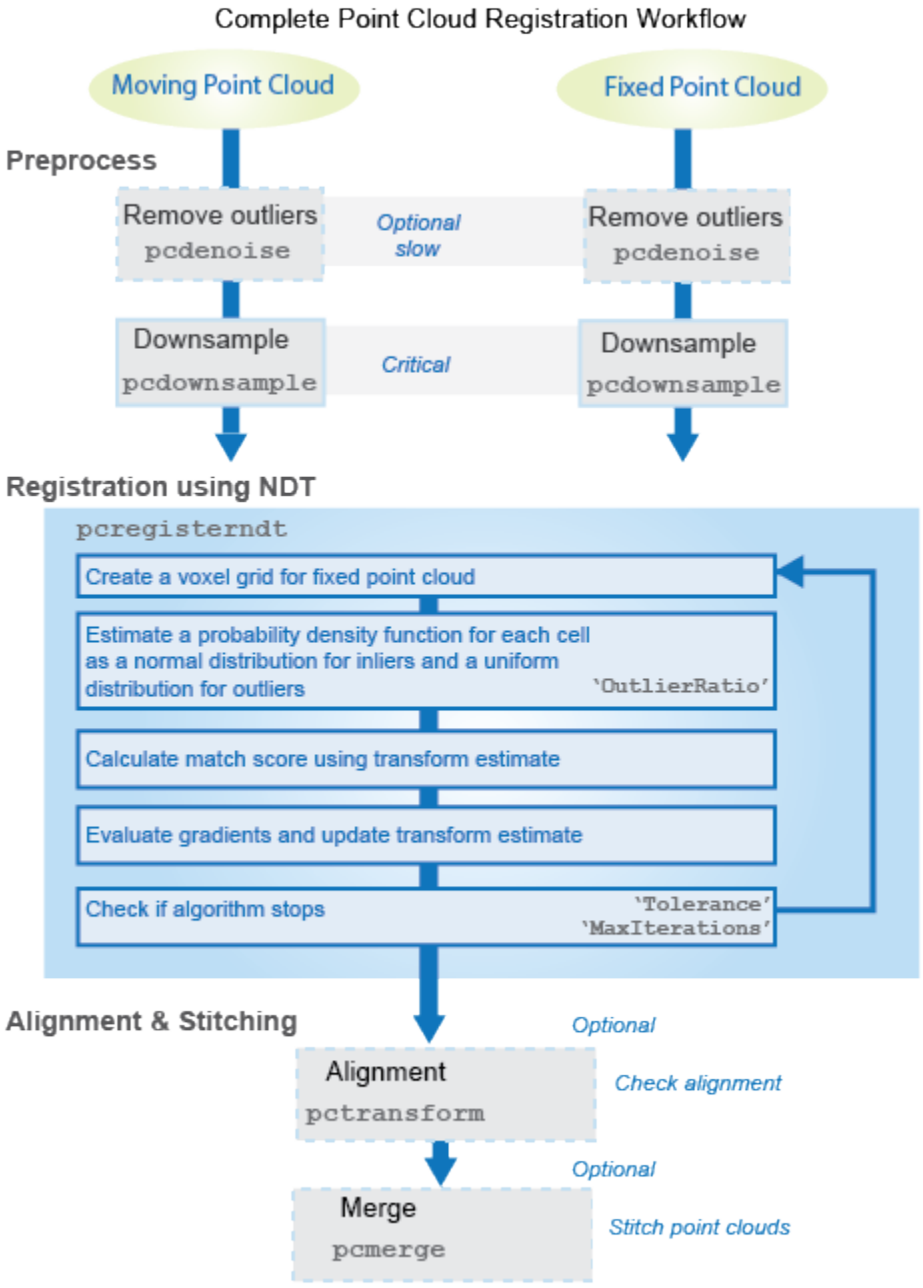
`rmse` — Root mean square error
positive number

Root mean square error, returned as a positive numeric scalar that represents the Euclidean distance between the aligned points. The aligned points is a result of the moving point cloud transformed by the `tform` output and the fixed point cloud. For each point in the fixed point cloud, the algorithm finds the closest point in the transformed point cloud, `movingReg`, then computes the Euclidean distance between the points, and then computes the `rmse`. A low `rmse` value indicates a more accurate registration. The `rmse` calculation is:

```
rmse = sqrt(sum(distance.^2,'all')/numel(distance));
```

Data Types: `single` | `double`

Algorithms



Tips

- For ground vehicle point clouds, you can improve performance and accuracy by removing the ground using `pcfitplane` or `segmentGroundFromLidarData` before registration. For details on how to do this, see the helper function, `helperProcessPointCloud` in the “Build a Map from Lidar Data” (Automated Driving Toolbox) example.

- To merge more than two point clouds, you can use `pccat` function instead of the `pcmerge` function.
- Prior to registration, the `pcregisterndt` function voxelizes the fixed point cloud using the specified `gridStep`. Decrease the grid step to capture distinct structures within the point cloud. However, decreasing this value can lead to an increase of the effect of the local noise from the sensor. Use the `pcdownsample` function to determine the effect of voxelization with this syntax:

  ```
  pcdownsample(fixed,"gridAverage",gridStep)
  ```

NDT Registration Overview

The NDT registration is based on normal distributions rather than other types of registration which match points or lines in order to find the transformation between two point clouds. The NDT approach uses statistical models with 3-D boxes, called voxels.

To register two point clouds, a moving point cloud and a fixed point cloud, using the NDT approach, the algorithm performs the following:

1. Computes the normal distributions for the fixed point cloud by dividing the area covered by the point cloud scan into 3-D boxes of constant size, referred to as "voxels". Each voxel contains a set of points. The algorithm computes the mean and covariance matrix for the points in each voxel.
2. Starting with an initial guess of the transformation, the algorithm aligns the moving point cloud to the fixed point cloud. It then finds the sum of the statistical likelihood of each aligned point to be located in the voxel that surrounds the point (in the moving point cloud), based on the normal distribution of the fixed point cloud.
3. To improve the registration, the algorithm maximizes the probability score of the moving point cloud on the normal distributions of the fixed point cloud. This is done by iteratively optimizing angular and translational estimations.
4. Repeats the alignment of the moving point cloud with the fixed point cloud using the new transformation from the previous step and then repeats the optimization.
5. The algorithm stops once the maximum iterations or the tolerance parameters, are met. The tolerance is a measure of how much the angular and translational estimations change from one iteration to another.

Version History

Introduced in R2018a

R2022b: Supports premultiply geometric transformation convention

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, the `pcregisterndt` function now returns the output argument `tform` as `rigidtform3d` object, which uses the premultiply convention. Before, the function returned `tform` as a `rigid3d` object, which uses the postmultiply convention.

You can now specify the `InitialTransform` name-value argument as a `rigidtform3d` object. Although you can still specify `InitialTransform` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2020a: Returns `rigid3d` object
Behavior changed in R2020a

Starting in R2020a, the `pcregisterndt` function returns a `rigid3d` object. Prior to R2020a, the function returned an `affine3d` object.

References

[1] Biber, P., and W. Straßer. “The Normal Distributions Transform: A New Approach to Laser Scan Matching.” *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV. Vol. 3, November 2003, pp. 2743–2748.

[2] Magnusson, M. “The Three-Dimensional Normal-Distributions Transform — an Efficient Representation for Registration, Surface Analysis, and Loop Detection.” Ph.D. Thesis. Örebro University, Örebro, Sweden, 2013.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Functions
`pcregistercorr` | `pcregistericp` | `pcregistercpd` | `pctransform` | `pcshow` | `pcshowpair` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcmerge`

Objects
`pointCloud` | `rigidtform3d` | `pcmapndt`

Topics
“3-D Point Cloud Registration and Stitching”
“Implement Point Cloud SLAM in MATLAB”

pcregrigid

(To be removed) Register two point clouds using ICP algorithm

Note The `pcregrigid` function will be removed in a future release. Use `pcregistericp` instead.

Syntax

```
tform = pcregrigid(moving,fixed)
[tform,movingReg] = pcregrigid(moving,fixed)
[tform,movingReg,rmse] = pcregrigid(moving,fixed)
[___] = pcregrigid(moving,fixed,Name,Value)
```

Description

`tform = pcregrigid(moving,fixed)` returns a rigid transformation that registers a moving point cloud to a fixed point cloud.

The registration algorithm is based on the "iterative closest point" (ICP) algorithm. Best performance of this iterative process requires adjusting properties for your data. Consider downsampling point clouds using `pcdownsample` before using `pcregrigid` to improve accuracy and efficiency of registration.

Point cloud normals are required by the registration algorithm when you select the `'pointToPlane'` metric. Therefore, if the input point cloud's `Normal` property is empty, the function fills it. When the function fills the `Normal` property, it uses 6 points to fit the local plane. Six points may not work under all circumstances. If registration with the `'pointToPlane'` metric fails, consider calling the `pcnormals` function which allows you to select the number of points to use.

`[tform,movingReg] = pcregrigid(moving,fixed)` additionally returns the transformed point cloud that aligns with the fixed point cloud.

`[tform,movingReg,rmse] = pcregrigid(moving,fixed)` additionally returns the root mean squared error of the Euclidean distance between the aligned point clouds, using any of the preceding syntaxes.

`[___] = pcregrigid(moving,fixed,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`moving` — Moving point cloud

`pointCloud` object

Moving point cloud, specified as a `pointCloud` object.

`fixed` — Fixed point cloud

`pointCloud` object

Fixed point cloud, specified as a `pointCloud` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Metric','pointToPoint'` sets the metric for the ICP algorithm to the `'pointToPoint'`.

`Metric` — Minimization metric
`'pointToPoint'` (default) | `'pointToPlane'`

Minimization metric, specified as the comma-separated pair consisting of '`Metric`' and `'pointToPoint'` or `'pointToPlane'`. The rigid transformation between the moving and fixed point clouds are estimated by the iterative closest point (ICP) algorithm. The ICP algorithm minimizes the distance between the two point clouds according to the given metric.

Setting '`Metric`' to `'pointToPlane'` can reduce the number of iterations to process. However, this metric requires extra algorithmic steps within each iteration. The `'pointToPlane'` metric improves the registration of planar surfaces.

Downsample Method Selection:
Downsample the point clouds using the `pcdownsample` function. Use either the `'random'` or `'gridAverage'` input for the `pcdownsample` function according to the `Metric` table below.

Metric	Point Cloud A Downsample Method	Point Cloud B Downsample Method
`'pointToPoint'`	`'random'`	`'random'`
	`'gridAverage'`	`'gridAverage'`
`'pointToPlane'`	`'gridAverage'`	`'gridAverage'`
	`'random'`	`'nonuniformGridSample'`

`Extrapolate` — Extrapolation
`false` (default) | `true`

Extrapolation, specified as the comma-separated pair consisting of '`Extrapolate`' and the boolean `true` or `false`. When you set this property to `true`, the function adds an extrapolation step that traces out a path in the registration state space, that is described in [2]. Setting this property to `true` can reduce the number of iterations to converge.

`InlierRatio` — Percentage of inliers
`1` (default) | scalar

Percentage of inliers, specified as the comma-separated pair consisting of '`InlierRatio`' and a scalar value. Use this value to set a percentage of matched pairs as inliers. A pair of matched points is considered an inlier if its Euclidean distance falls within the percentage set of matching distances. By default, all matching pairs are used.

`MaxIterations` — Maximum number of iterations
`20` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of 'MaxIterations' and a positive integer. This value specifies the maximum number of iterations before ICP stops.

`Tolerance` — Tolerance between consecutive ICP iterations
`[0.01, 0.009]` (default) | 2-element vector

Tolerance between consecutive ICP iterations, specified as the comma-separated pair consisting of 'Tolerance' and a 2-element vector. The 2-element vector, [*Tdiff*, *Rdiff*], represents the tolerance of absolute difference in translation and rotation estimated in consecutive ICP iterations. *Tdiff* measures the Euclidean distance between two translation vectors. *Rdiff* measures the angular difference in radians. The algorithm stops when the average difference between estimated rigid transformations in the three most recent consecutive iterations falls below the specified tolerance value.

`InitialTransform` — Initial rigid transformation
`affine3d()` object (default)

Initial rigid transformation, specified as the comma-separated pair consisting of 'InitialTransform' and an `affine3d` object. The initial rigid transformation is useful when you provide an external coarse estimation.

`Verbose` — Display progress information
`true` (default) | `false`

Display progress information, specified as the comma-separated pair consisting of 'Verbose' and a logical scalar. Set `Verbose` to `true` to display progress information.

Output Arguments

`tform` — Rigid transformation
`affine3d` object

Rigid transformation, returned as an `affine3d` object. The rigid transformation registers a moving point cloud to a fixed point cloud. The `affine3d` object describes the rigid 3-D transform. The iterative closest point (ICP) algorithm estimates the rigid transformation between the moving and fixed point clouds.

`movingReg` — Transformed point cloud
`pointCloud` object

Transformed point cloud, returned as a `pointCloud` object. The transformed point cloud is aligned with the fixed point cloud.

`rmse` — Root mean square error
positive numeric

Root mean square error, returned as the Euclidean distance between the aligned point clouds.

Version History

Introduced in R2015a

R2024a: Warns
Warns starting in R2024a

The `pcregrigid` function issues a warning that it will be removed in a future release. Use the `pcregistericp` object to perform ICP point cloud registration, instead.

R2023b: To be removed
Not recommended starting in R2023b

The `pcregrigid` function will be removed in a future release.

References

[1] Chen, Y. and G. Medioni. “Object Modelling by Registration of Multiple Range Images.” *Image Vision Computing*. Butterworth-Heinemann. Vol. 10, Issue 3, April 1992, pp. 145-155.

[2] Besl, Paul J., N. D. McKay. “A Method for Registration of 3-D Shapes.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Los Alamitos, CA: IEEE Computer Society. Vol. 14, Issue 2, 1992, pp. 239-256.

See Also

`pointCloud` | `pcregistericp` | `pcregisterndt`

Topics
“3-D Point Cloud Registration and Stitching”

pcsegdist

Segment point cloud into clusters based on Euclidean distance

Syntax

```
labels = pcsegdist(ptCloud,minDistance)
[labels,numClusters] = pcsegdist(ptCloud,minDistance)
[___] = pcsegdist(___,Name=Value)
```

Description

`labels = pcsegdist(ptCloud,minDistance)` segments a point cloud into clusters, with a minimum Euclidean distance of `minDistance` between points from different clusters. `pcsegdist` assigns an integer cluster label to each point in the point cloud, and returns the `labels` of all points.

`[labels,numClusters] = pcsegdist(ptCloud,minDistance)` also returns the number of clusters.

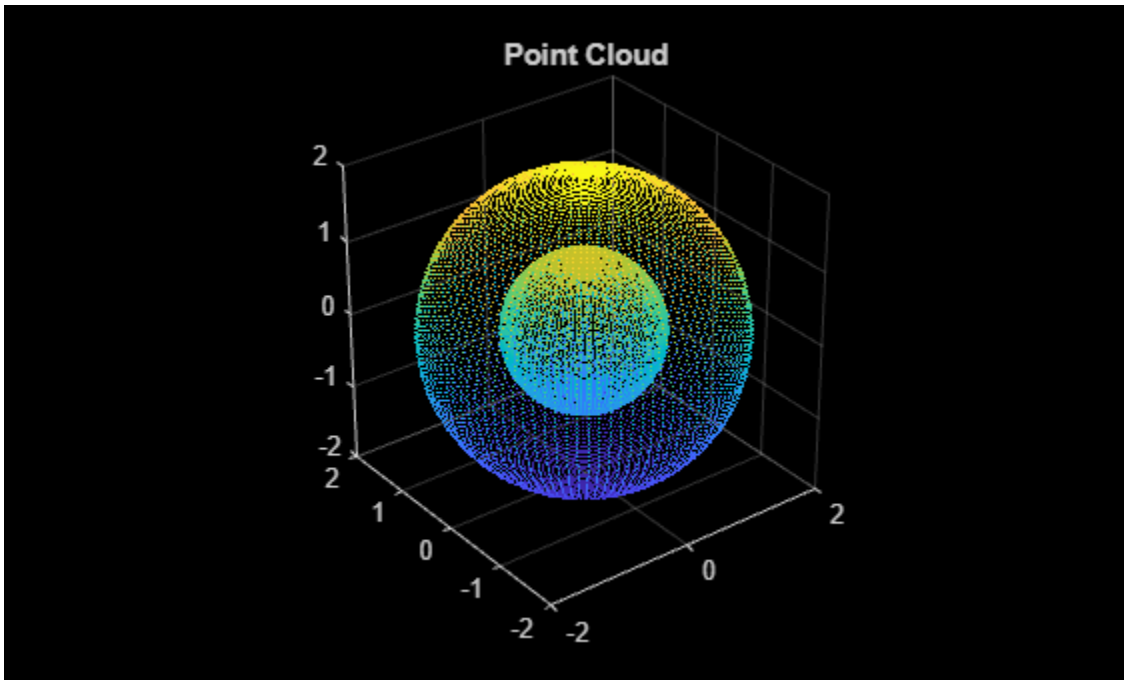
`[___] = pcsegdist(___,Name=Value)` sets properties using name-value arguments. For example, `labels = pcsegdist(ptCloud,minDistance,NumClusterPoints=[1,Inf])` sets the minimum and maximum number of points in each cluster to `[1,Inf]`.

Examples

Cluster Point Cloud Based on Euclidean Distance

Create two concentric spheres and combine them.

```
[X,Y,Z] = sphere(100);
loc1 = [X(:),Y(:),Z(:)];
loc2 = 2*loc1;
ptCloud = pointCloud([loc1;loc2]);
pcshow(ptCloud)
title('Point Cloud')
```



Set the minimum Euclidean distance between clusters.

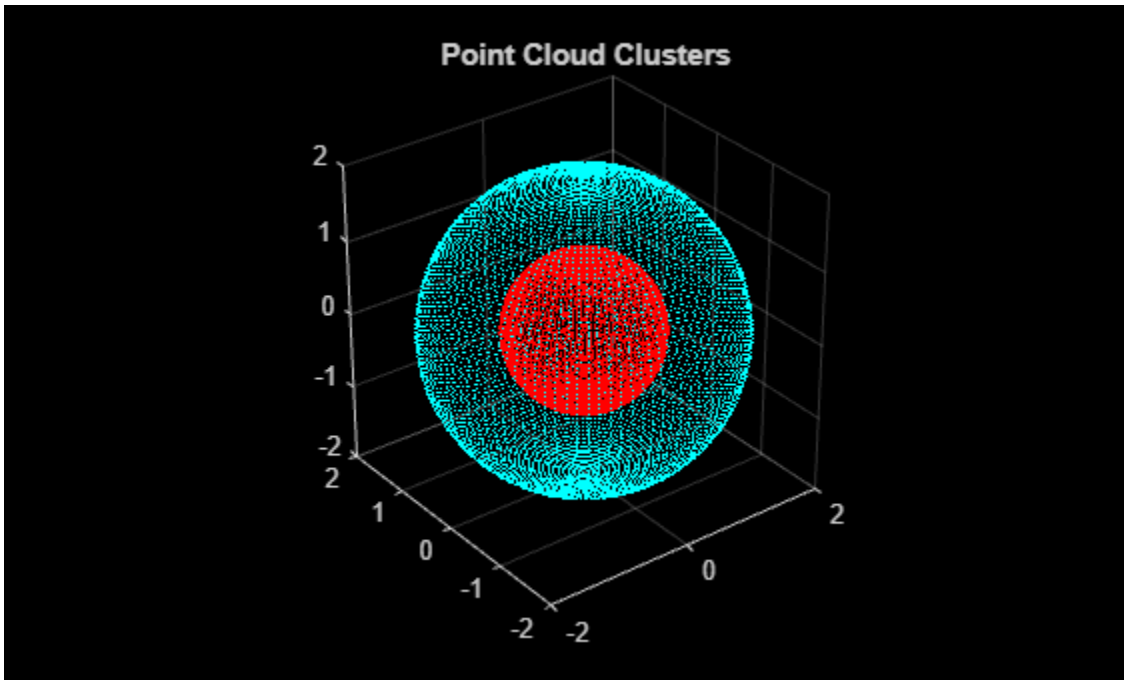
```
minDistance = 0.5;
```

Segment the point cloud.

```
[labels,numClusters] = pcsegdist(ptCloud,minDistance);
```

Plot the labeled results. The points are grouped into two clusters.

```
pcshow(ptCloud.Location,labels)
colormap(hsv(numClusters))
title('Point Cloud Clusters')
```



Cluster Lidar Point Cloud Based on Euclidean Distance

Load an organized lidar point cloud to the workspace.

```
ld = load('drivingLidarPoints.mat');
```

Detect the ground plane. Distance is measured in meters.

```
maxDistance = 0.9;
referenceVector = [0 0 1];
[~,inliers,outliers] = pcfitplane(ld.ptCloud,maxDistance,referenceVector);
```

Remove the ground plane points.

```
ptCloudWithoutGround = select(ld.ptCloud,outliers);
```

Cluster the point cloud with a minimum of 10 points per cluster using the exhaustive method.

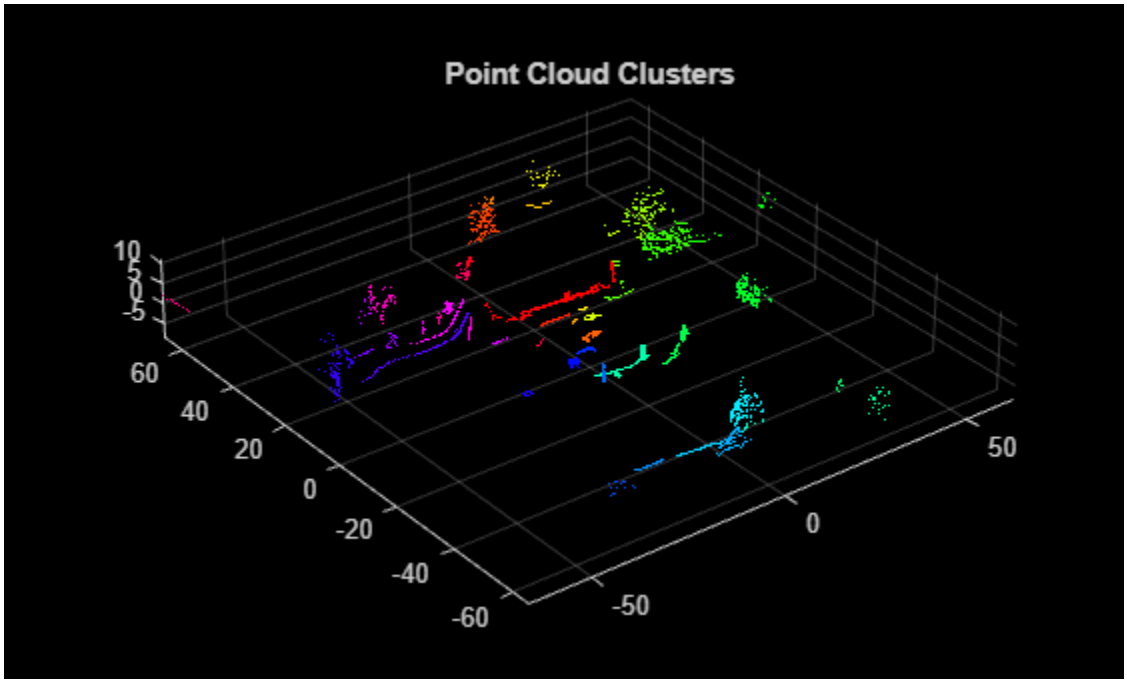
```
minDistance = 2;
minPoints = 10;
[labels,numClusters] = pcsegdist(ptCloudWithoutGround,minDistance,Method="exhaustive",...
    NumClusterPoints=minPoints);
```

Remove the points with a label value of `0`.

```
idxValidPoints = find(labels);
labelColorIndex = labels(idxValidPoints);
segmentedPtCloud = select(ptCloudWithoutGround,idxValidPoints);
```

Plot the labeled results.

```
figure
colormap(hsv(numClusters))
pcshow(segmentedPtCloud.Location,labelColorIndex)
title('Point Cloud Clusters')
```



Input Arguments

`ptCloud` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`minDistance` — Minimum Euclidean distance
positive scalar

Minimum Euclidean distance between points from two different clusters, specified as a positive scalar.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ParallelNeighborSearch=false` sets the `ParallelNeighborSearch` to `false`.

`NumClusterPoints` — Minimum and maximum number of points in each cluster
`[1,Inf]` (default) | 2-element vector | scalar

Minimum and maximum number of points in each cluster, specified as a scalar or a 2-element vector of the form [*minPoints*,*maxPoints*]. When you specify `NumClusterPoints` as a scalar, the maximum number of points in the cluster is unrestricted. The function sets `labels` to `0` when clusters are outside of the specified range.

`Method` — Method to segment the point cloud
`"approximate"` (default) | `"exhaustive"`

Method to segment the point cloud, specified as `"approximate"` or `"exhaustive"`. Set `Method` to `"exhaustive"` to ensure that no points outside of each cluster are less than `minDistance` away. This approach uses the density-based spatial clustering of applications with noise (DBSCAN) algorithm. Set `Method` to `"approximate"` for faster segmentation, but at the expense of accuracy.

`ParallelNeighborSearch` — Parallel neighbor search to segment point cloud data
`false` (default) | `true`

Parallel neighbor search to segment point cloud data, specified as `true` or `false`. Set this property to `true` when you expect there to be approximately 50 clusters or more with fewer than 100 points per cluster.

A parallel neighbor search can improve segmentation speed for some datasets. Improved speed depends on the dataset and the value of the `minDistance` input. This argument is not supported when you set the `Method` name-value argument to `"exhaustive"`.

Output Arguments

`labels` — Cluster labels
M-by-1 vector | *M*-by-*N* matrix

Cluster labels, returned as one of the following.

- If the point cloud, `ptCloud`, stores point locations as an unorganized *M*-by-3 matrix, then `labels` is an *M*-by-1 vector.
- If the point cloud, `ptCloud`, stores point locations as an organized *M*-by-*N*-by-3 matrix, then `labels` is an *M*-by-*N* matrix.

Each point in the point cloud has a cluster label, specified by the corresponding element in `labels`. The value of each label is an integer from `0` to the number of clusters of valid points, `numClusters`. The value `0` is reserved for invalid points, such as points with `Inf` or `NaN` coordinates.

Data Types: `uint32`

`numClusters` — Number of clusters
positive integer

Number of clusters, returned as a positive integer. The number of clusters excludes the label value `0`, which is reserved for invalid points. The function returns `numClusters` as a `single` data type when the value of the `Location` property of the `ptCloud` object is `single`. Otherwise, the function returns the value as a `double` data type.

Data Types: `single` | `double`

Version History

Introduced in R2018a

R2026a: Support for `NumClusterPoints` in GPU Code Generation

Starting in R2026a, `pcsegdist` function supports the `NumClusterPoints` name-value argument for GPU code generation.

R2024b: New `Method` name-value argument for segmenting point cloud

Configure the `Method` name-value argument to choose between `"exhaustive"` or `"approximate"` segmentation methods. The `ParallelNeighborhood` name-value argument does not support the `"exhaustive"` method.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

The `"exhaustive"` segmentation method, which can be set by using the `Method` name-value argument, does not support C/C++ code generation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The generated CUDA code segments the point cloud into clusters by using a combination of algorithms described in [1] and [2]. The output from the generated code can differ slightly with results from MATLAB simulation.
- The `"exhaustive"` segmentation method, which can be set by using the `Method` name-value argument, does not support GPU code generation.

References

[1] Andrade, Guilherme, Gabriel Ramos, Daniel Madeira, Rafael Sachetto, Renato Ferreira, and Leonardo Rocha. *"G-DBSCAN: A GPU Accelerated Algorithm for Density-Based Clustering." Procedia Computer Science* 18 (2013): 369–78. https://doi.org/10.1016/j.procs.2013.05.200.

[2] Kalentev, Oleksandr, Abha Rai, Stefan Kemnitz, and Ralf Schneider. *"Connected Component Labeling on a 2D Grid Using CUDA." Journal of Parallel and Distributed Computing* 71, no. 4 (April 2011): 615–20. https://doi.org/10.1016/j.jpdc.2010.10.012.

See Also

Objects

`pointCloud`

Functions
pcdenoise | pcfitplane | pcshow | segmentLidarData | pcbin

pcwrite

Write 3-D point cloud to PLY or PCD file

Syntax

```
pcwrite(ptCloud,filename)
pcwrite(ptCloud,filename,Encoding=encodingType)
```

Description

`pcwrite(ptCloud,filename)` writes the point cloud object, `ptCloud`, to the PLY or PCD file specified by the input `filename`.

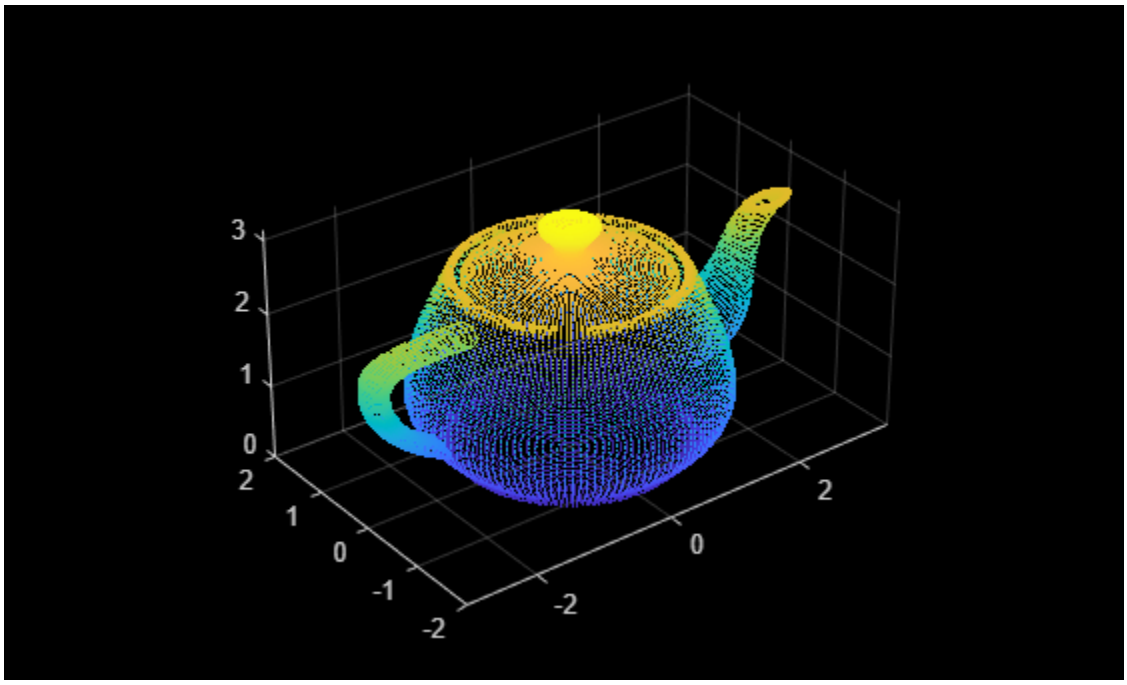
`pcwrite(ptCloud,filename,Encoding=encodingType)` writes a `pointCloud` object, `ptCloud`, to a PLY file that is in the specified format.

Examples

Write 3-D Point Cloud to PLY File

Load a point cloud into the workspace.

```
ptCloud = pcread("teapot.ply");
pcshow(ptCloud);
```

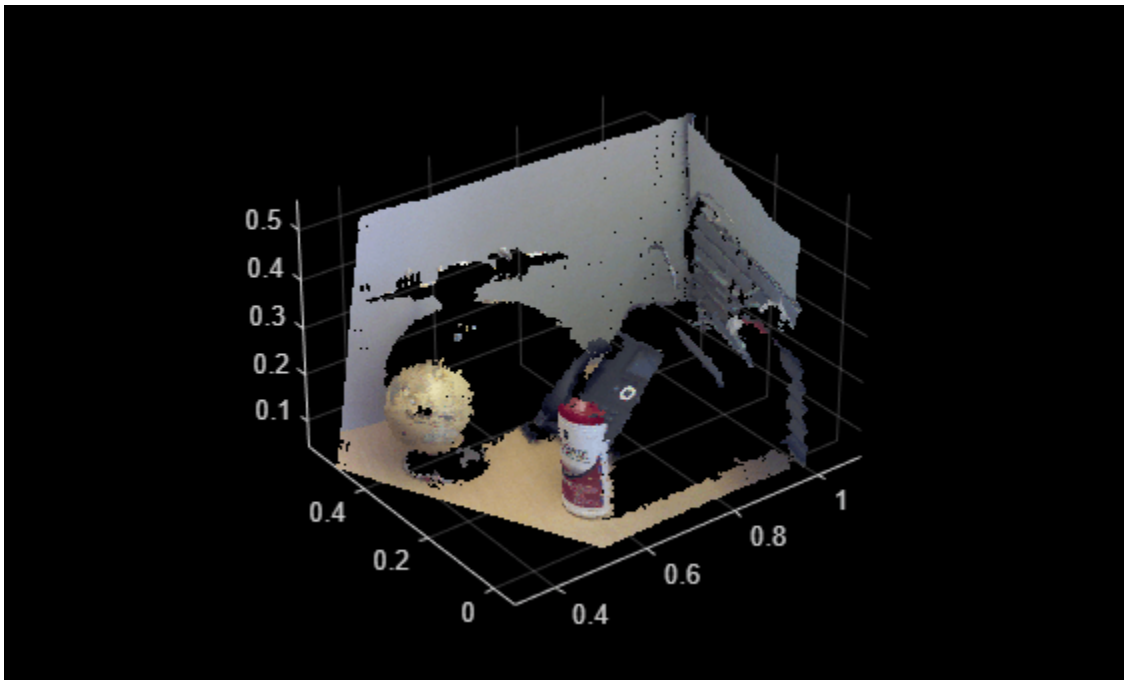



```
pcwrite(ptCloud,"teapotOut",Encoding="binary");
```

Write 3-D Organized Point Cloud to PCD File

Load a MAT file containing point cloud data into the workspace.

```
load("object3d.mat");
pcwrite(ptCloud,"object3d.pcd",Encoding="ascii");
pc = pcread("object3d.pcd");
pcshow(pc);
```



Input Arguments

`filename` — File name

character vector | string scalar

File name, specified as a character vector or string scalar. The input file type must be a PLY or PCD format file.

For a PLY-file, the `pcwrite` function converts an organized *M*-by-*N*-by-3 point cloud to an unorganized *M*-by-3 format. It converts the format because PLY files do not support organized point clouds. To preserve the organized format, you can save the point cloud as a PCD-file.

If you do not specify the file name with an extension, the function writes the file in a PLY-format. All `NaN` or `Inf` values will be skipped when writing to a PLY file.

`ptCloud` — Object for storing point cloud

`pointCloud` object

Object for storing point cloud, specified as a `pointCloud` object.

encodingType — Encoding type for output file
`"ascii"` | `"binary"` | `"compressed"`

Encoding type for the output file, specified as `"ascii"`, `"binary"`, or `"compressed"`. This table summarizes the valid and default encoding types for PLY and PCD file formats.

File Format	Valid Encodings
PLY	`"ascii"` `"binary"` (default)

PCD	`"ascii"`
	
	`"binary"`
	
	`"compressed"` (default)

To improve performance when writing large point clouds, use the `"compressed"` or `"binary"` encoding type.

Version History

Introduced in R2015a

R2026a: Write point clouds with additional Color property data types

You can now write a `pointCloud` object with a `Color` property of data type `uint8`, `uint16`, `single`, or `double` to a PLY or PCD file. In previous releases, the valid data types for the `Color` property depends on the file format you write to.

Valid Color Data Types

File Format	R2025b	R2026a
PLY	`uint8`, `uint16`	`uint8`, `uint16`, `single`, `double`
PCD	`uint8`	`uint8`, `uint16`, `single`, `double`

R2026a: Change in default encoding type
Behavior changed in R2026a

The default value of the `encodingType` argument is now `"binary"` for PLY files and `"compressed"` for PCD files. Before R2026a, the default value is `"ascii"` for both file formats.

In most cases, you do not need to make any changes to your code. However, if you specifically want to write point cloud data to a PLY or PCD file with ASCII encoding, you must specify the `encodingType` argument as `"ascii"`, as shown in this command:
`pcwrite(ptCloud,"sample.ply",Encoding="ascii")`.

R2026a: Improved performance

When you use the `pcwrite` function with the default encoding type, the function shows improved performance. When you use the `pcread` function to read a file generated by the `pcwrite` function, the `pcread` function also shows improved performance. The performance improvement increases as the number of points stored in the point cloud increases. For example, writing and then reading a point cloud using this code is about 17x and 11x faster, respectively, than in the previous release.

```
function [tWrite,tRead] = timingTest
    % Create a large point cloud
    rng("default");
    N = 1e6; % Number of points

    x = -50 + 100*rand(N,1);        % range [-50, 50]
    y = -50 + 100*rand(N,1);        % range [-50, 50]
    z = 10*rand(N,1);               % range [0, 10]

    ptCloud = pointCloud([x y z]);

    % Measure execution time of the operations
    tWrite = timeit(@()pcwrite(ptCloud,"temp.ply"));
    tRead = timeit(@()pcread("temp.ply"));
end
```

This table shows the approximate execution times required to write and read point cloud data to and from a PLY file.

Release	`pcwrite` Execution Time	`pcread` Execution Time
R2025b	2.19 s	0.56 s
R2026a	0.13 s	0.05 s

The code was timed on a Debian® 12, Intel® Xeon® CPU W-2133 @ 3.60 GHz test system by calling the function `timingTest`.

See Also

`pointCloud` | `pcread` | `pcplayer` | `pcshow` | `pcmerge` | `pcfitplane` | `planeModel` | `pctransform` | `pcdownsample` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”

pctransform

Transform 3-D point cloud

Syntax

```
ptCloudOut = pctransform(ptCloudIn,tform)
ptCloudOut = pctransform(ptCloudIn,D)
```

Description

`ptCloudOut = pctransform(ptCloudIn,tform)` applies the specified 3-D affine transform, `tform` to the point cloud, `ptCloudIn`. The transformation can be a rigid, similarity, or affine transformation.

`ptCloudOut = pctransform(ptCloudIn,D)` applies the displacement field `D` to the point cloud. Point cloud transformation using a displacement field define translations with respect to each point in the point cloud.

Examples

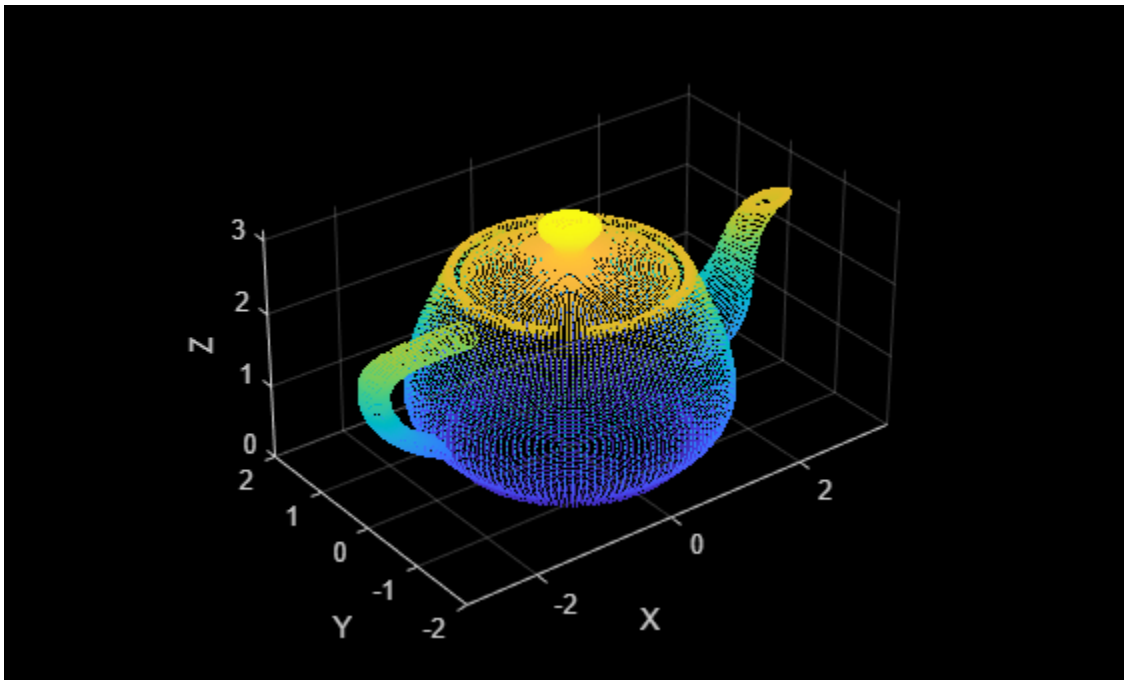
Rotate 3-D Point Cloud Using Rigid Transformation

Read a point cloud.

```
ptCloud = pcread('teapot.ply');
```

Plot the point cloud.

```
figure
pcshow(ptCloud)
xlabel('X')
ylabel('Y')
zlabel('Z')
```



Create a transformation object with a 45 degree rotation along the z-axis.

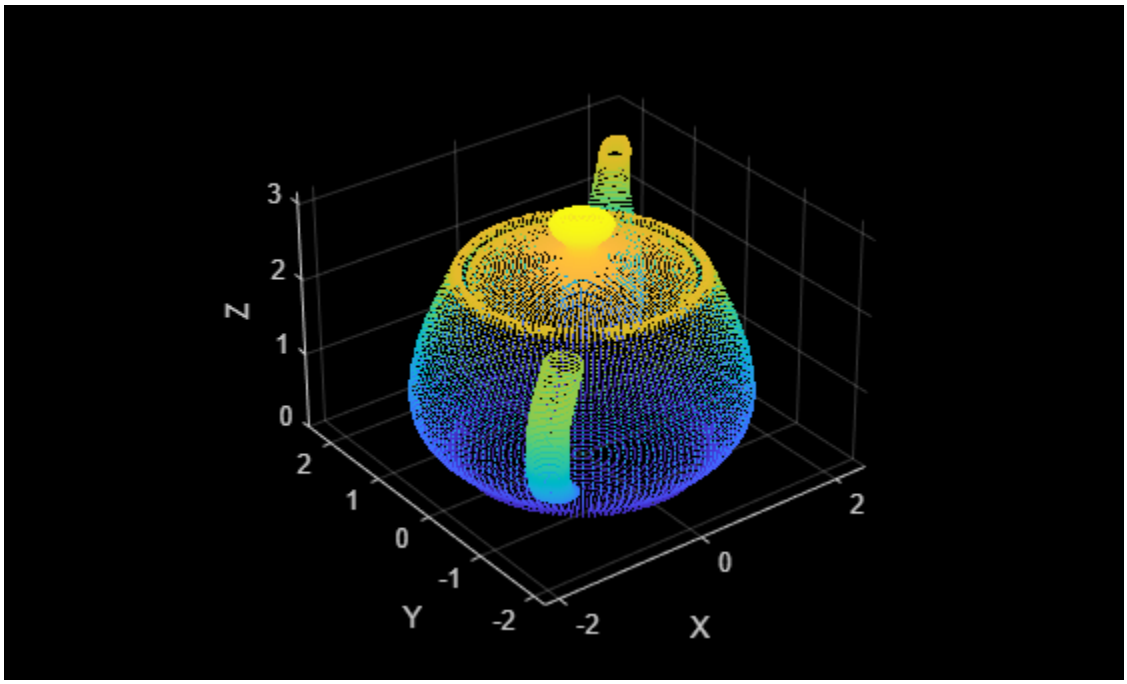
```
rotationAngles = [0 0 45];
translation = [0 0 0];
tform = rigidtform3d(rotationAngles,translation);
```

Transform the point cloud.

```
ptCloudOut = pctransform(ptCloud,tform);
```

Plot the transformed point cloud.

```
figure
pcshow(ptCloudOut)
xlabel('X')
ylabel('Y')
zlabel('Z')
```



Rigid and Nonrigid Transformations of 3-D Point Cloud

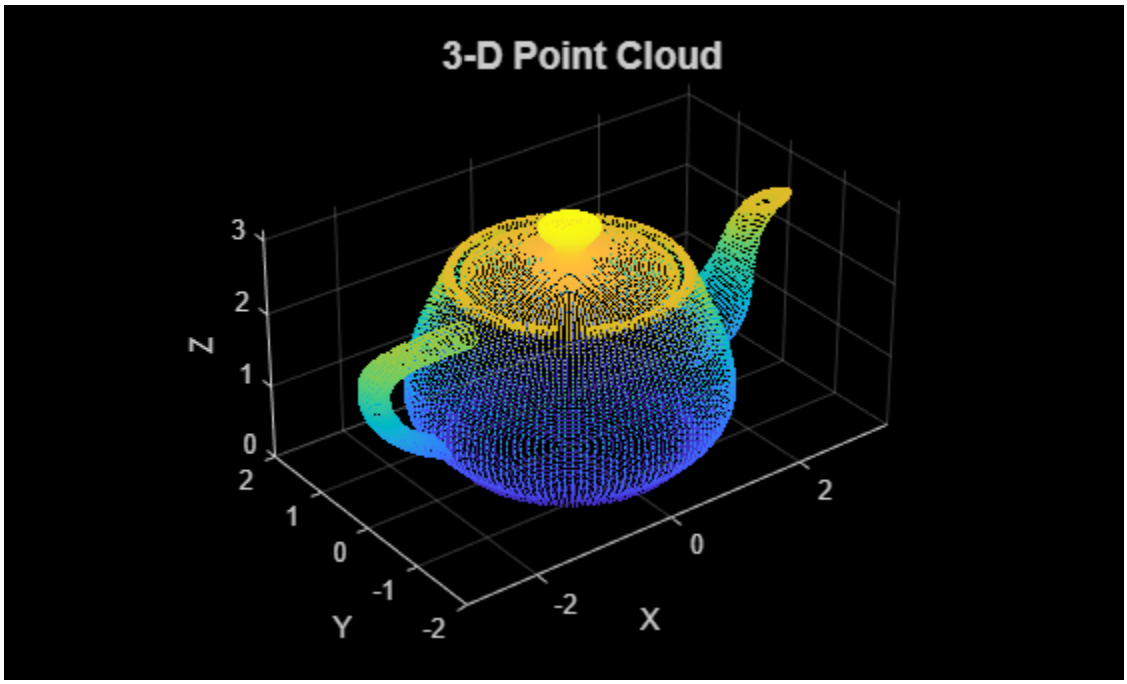
This example applies a rigid transformation (rotation) and a nonrigid transformation (shear) to a 3-D point cloud.

Read a point cloud into the workspace.

```
ptCloud = pcread('teapot.ply');
```

Plot the original 3-D point cloud.

```
figure1 = figure;
axes1 = axes(Parent=figure1);
pcshow(ptCloud,Parent=axes1,AxesVisibility='on');
xlabel('X');
ylabel('Y');
zlabel('Z');
title('3-D Point Cloud',FontSize=14)
```



Rotation of 3-D Point Cloud

Create a rigid transformation object that defines a 45 degree rotation along the *z*-axis.

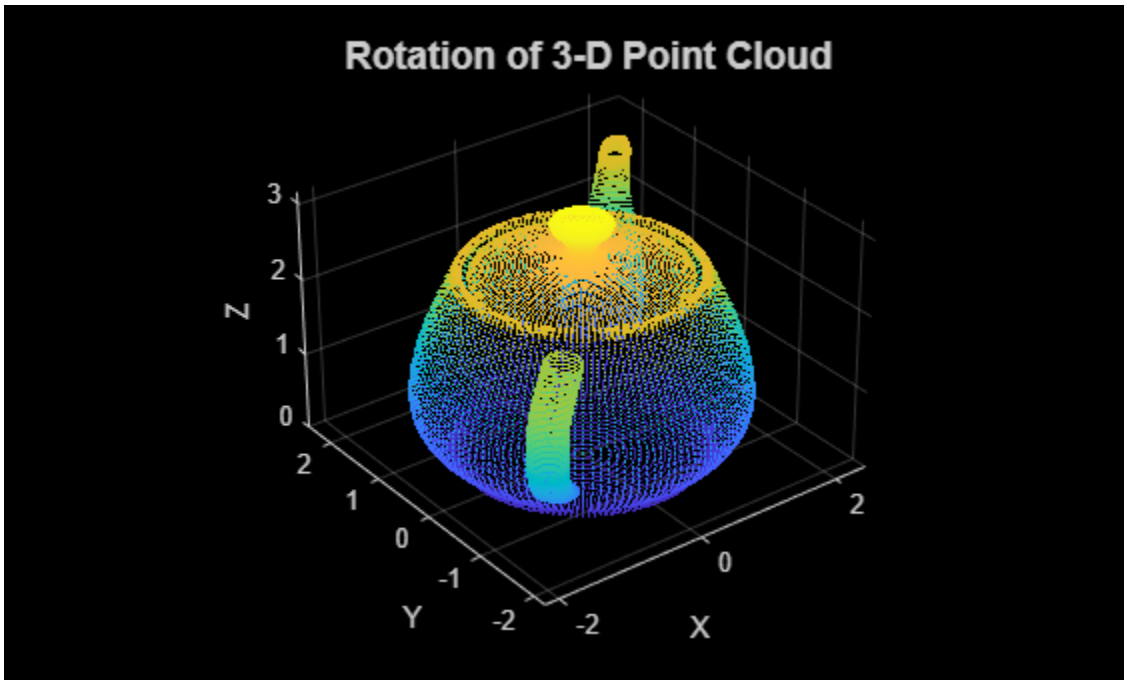
```
rotationAngles = [0 0 45];
translation = [0 0 0];
tform = rigidtform3d(rotationAngles,translation);
```

Transform the point cloud.

```
ptCloudOut1 = pctransform(ptCloud,tform);
```

Plot the rotated point cloud.

```
figure2 = figure;
axes2 = axes(Parent=figure2);
pcshow(ptCloudOut1,Parent=axes2,AxesVisibility='on');
xlabel('X');
ylabel('Y');
zlabel('Z');
title({'Rotation of 3-D Point Cloud'},FontSize=14)
```



Shearing of 3-D Point Cloud

Create an affine transformation object that defines shearing along the *x*-axis.

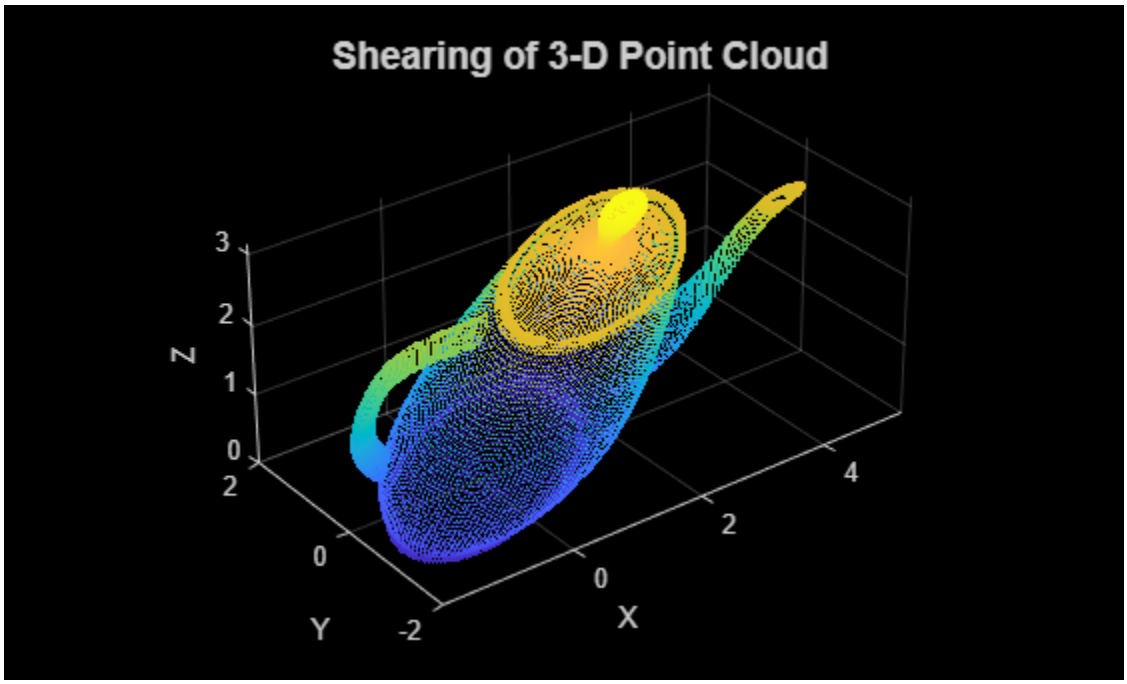
```
A = [1 0.75 0.75 0; ...
     0   1    0  0; ...
     0   0    1  0; ...
     0   0    0  1];
tform = affinetform3d(A);
```

Transform the point cloud.

```
ptCloudOut2 = pctransform(ptCloud,tform);
```

Plot the transformed point cloud.

```
figure3 = figure;
axes3 = axes(Parent=figure3);
pcshow(ptCloudOut2,Parent=axes3,AxesVisibility='on');
xlabel('X');
ylabel('Y');
zlabel('Z');
title({'Shearing of 3-D Point Cloud'},FontSize=14)
```



Point Cloud Transformation Using Displacement Field

Read a point cloud into the workspace.

```
ptCloud = pcread('teapot.ply');
```

Create a displacement field `D` of same size as the point cloud.

```
D = zeros(size(ptCloud.Location));
```

Set the displacement field value along *x*-axis for the first half of the points to 7.

```
pthalf = ptCloud.Count/2;
D(1:pthalf,1) = 7;
```

Extract the indices of points within a region-of-interest (ROI) using the `pointCloud` method `findNeighborsInRadius`. Set the displacement field value along the *x*-, *y*-, and *z*-axis for points within the ROI to 4, 4, and -2, respectively.

```
indices = findNeighborsInRadius(ptCloud,[0 0 3.1],1.5);
D(indices,1:2) = 4;
D(indices,3) = -2;
```

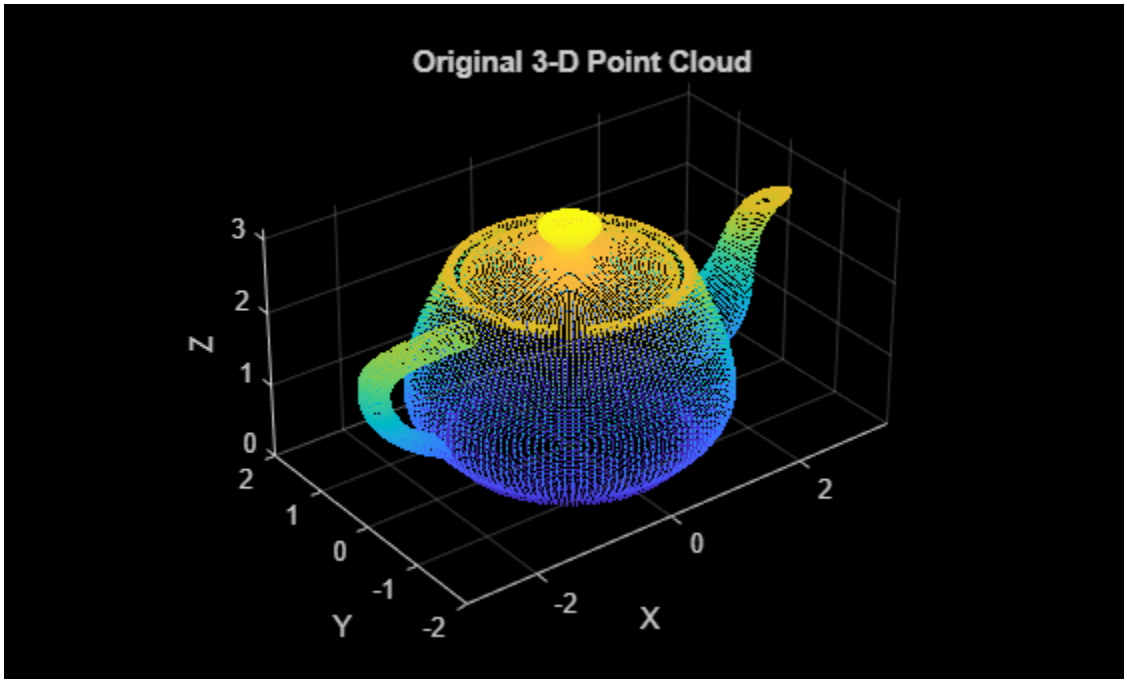
Transform the point cloud using the displacement field.

```
ptCloudOut = pctransform(ptCloud,D);
```

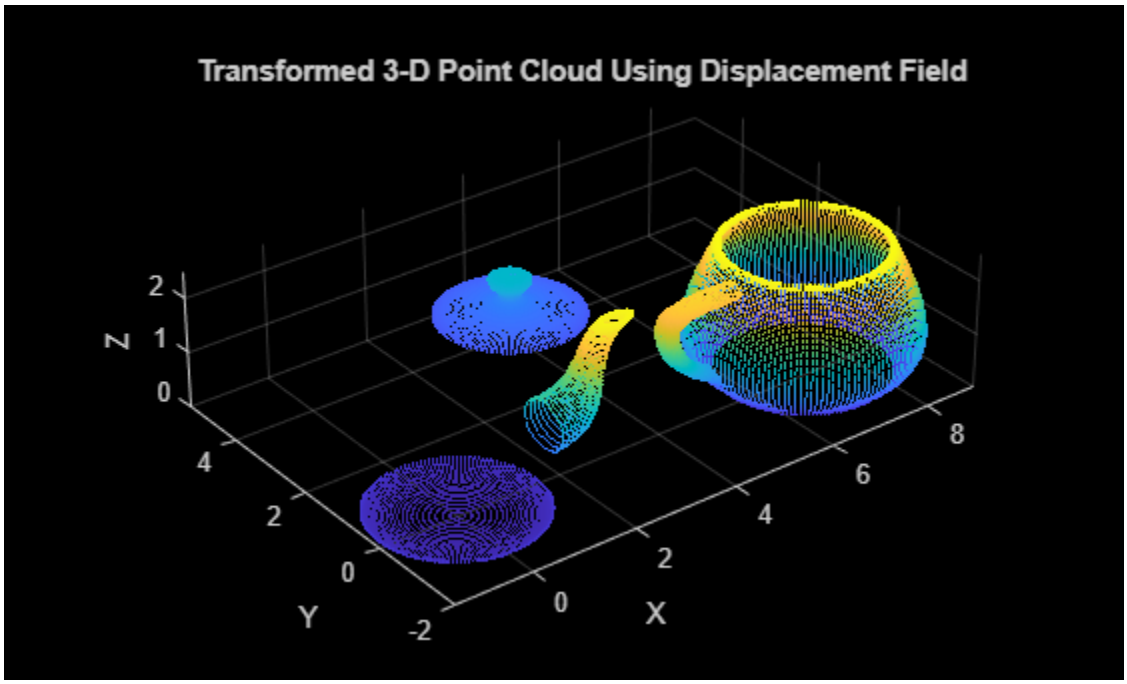
Display the original and transformed point cloud.

```
figure
pcshow(ptCloud)
xlabel('X');
```

```
ylabel('Y');
zlabel('Z');
title('Original 3-D Point Cloud')
```




```
figure
pcshow(ptCloudOut)
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Transformed 3-D Point Cloud Using Displacement Field')
```



Input Arguments

`ptCloudIn` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`tform` — 3-D geometric transformation
`rigidtform3d` object | `simtform3d` object | `affinetform3d` object

3-D geometric transformation, specified as a `rigidtform3d`, `simtform3d`, or `affinetform3d` object. See “Specify Transformation Matrix” for details on how to set up an affine 3-D `tform` input.

`D` — Displacement field
M-by-3 matrix | *M*-by-*N*-by-3 array

Displacement field, specified as either *M*-by-3 or an *M*-by-*N*-by-3 array. The displacement field is a set of displacement vectors that specify the magnitude and direction of translation for each point in the point cloud. The size of the displacement field must be the same as the size of the `Location` property of the `pointCloud` object.

Data Types: `single` | `double`

Output Arguments

`ptCloudOut` — Transformed point cloud
`pointCloud` object

Transformed point cloud, returned as a `pointCloud` object. The transformation applies to the coordinates of points and their normal vectors.

Version History

Introduced in R2015a

R2022b: Supports new geometric transformation objects

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify `tform` as a `rigidtform3d`, `simtform3d`, or `affinetform3d` object, which use the premultiply convention.

Although you can still specify `tform` as a `rigid3d` object or an `affine3d` object, these objects are not recommended because they use the postmultiply convention. You can streamline your geometric transformation workflows by switching to the new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Because of architectural and implementation differences between the GPU and CPU, the selection of neighboring points may differ during the computation of normal vectors. As a result, there may be numerical mismatches between MATLAB simulation and the output from code generation.

See Also

Objects
`pointCloud` | `planeModel` | `rigidtform3d` | `affinetform3d`

Functions
`pcplayer` | `pcshow` | `pcwrite` | `pcread` | `pcmerge` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp` | `pccat` | `pcalign`

Topics
“3-D Point Cloud Registration and Stitching”

pcnormals

Estimate normals for point cloud

Syntax

```
normals = pcnormals(ptCloud)
normals = pcnormals(ptCloud,k)
```

Description

`normals = pcnormals(ptCloud)` returns a matrix that stores a normal for each point in the input `ptCloud`. The function uses six neighboring points to fit a local plane to determine each normal vector.

`normals = pcnormals(ptCloud,k)` additionally specifies `k`, the number of points used for local plane fitting. The function uses this value rather than the six neighboring points as described in the first syntax.

Examples

Estimate Normals of Point Cloud

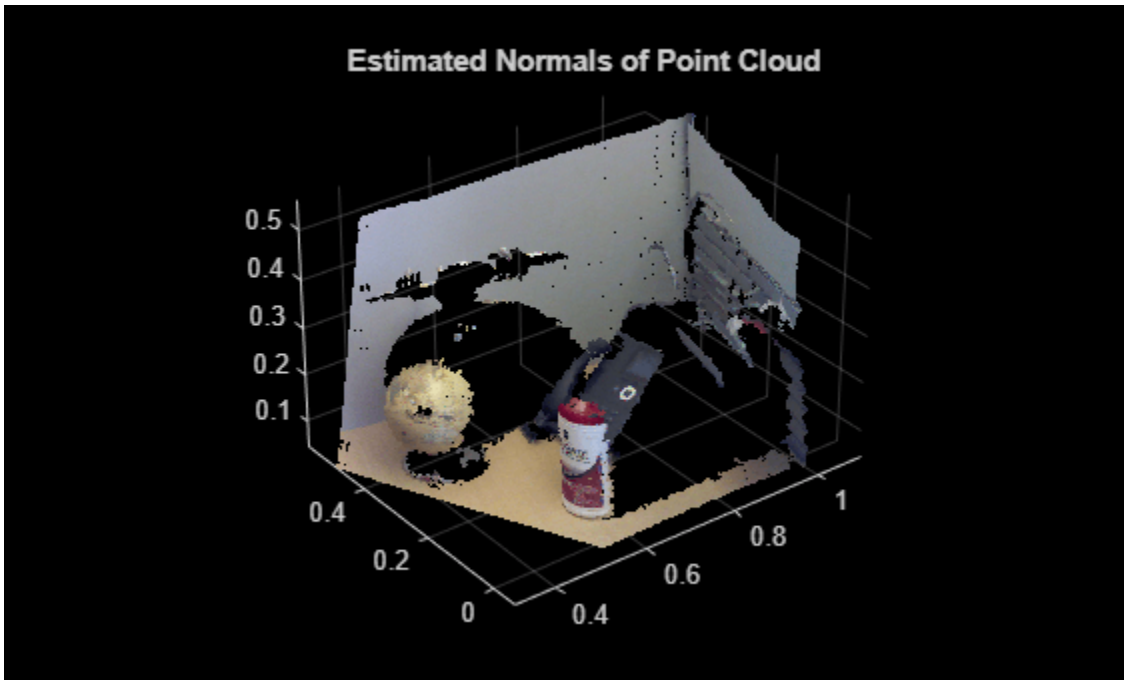
Load a point cloud.

```
load('object3d.mat');
```

Estimate the normal vectors.

```
normals = pcnormals(ptCloud);

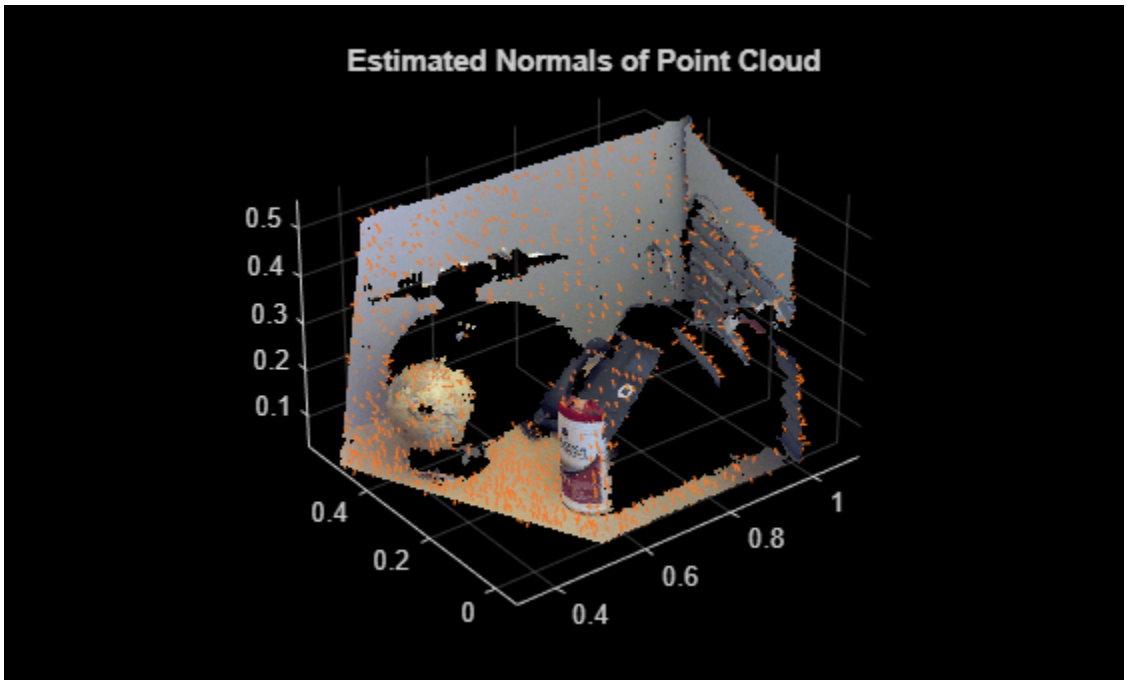
figure
pcshow(ptCloud)
title('Estimated Normals of Point Cloud')
hold on
```




```
x = ptCloud.Location(1:10:end,1:10:end,1);
y = ptCloud.Location(1:10:end,1:10:end,2);
z = ptCloud.Location(1:10:end,1:10:end,3);
u = normals(1:10:end,1:10:end,1);
v = normals(1:10:end,1:10:end,2);
w = normals(1:10:end,1:10:end,3);
```

Plot the normal vectors.

```
quiver3(x,y,z,u,v,w);
hold off
```

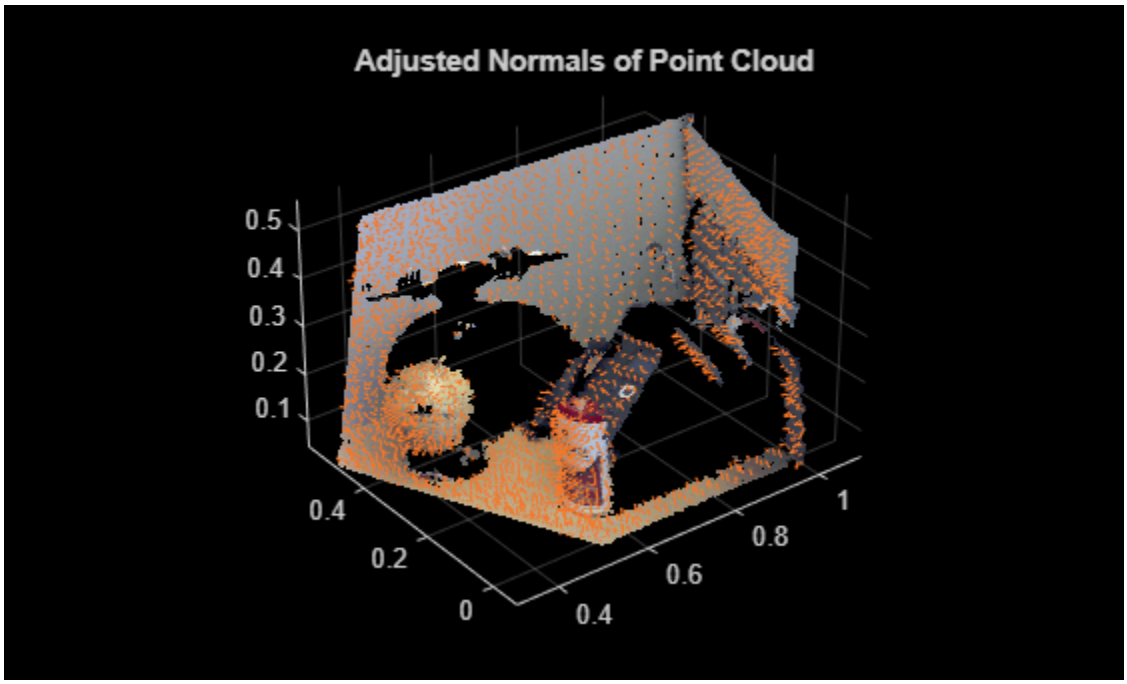


Flip the normals to point towards the sensor location. This step is necessary only for determining the inward or outward direction of the surface. The sensor center is set in *x* , *y* , *z* coordinates.

```
sensorCenter = [0,-0.3,0.3];
for k = 1 : numel(x)
   p1 = sensorCenter - [x(k),y(k),z(k)];
   p2 = [u(k),v(k),w(k)];
   % Flip the normal vector if it is not pointing towards the sensor.
   angle = atan2(norm(cross(p1,p2)),p1*p2');
   if angle > pi/2 || angle < -pi/2
       u(k) = -u(k);
       v(k) = -v(k);
       w(k) = -w(k);
   end
end
```

Plot the adjusted normals.

```
figure
pcshow(ptCloud)
title('Adjusted Normals of Point Cloud')
hold on
quiver3(x, y, z, u, v, w);
hold off
```



Input Arguments

`ptCloud` — Object for storing point cloud
`pointCloud` object

Object for storing point cloud, returned as a `pointCloud` object.

k — Number of points used for local plane fitting
integer greater than or equal to 3

Number of points used for local plane fitting, specified as an integer greater than or equal to 3. Increasing this value improves accuracy but slows down computation time. If you do not specify `k`, the function uses six neighboring points to fit a local plane to determine each normal vector.

Output Arguments

`normals` — Normals used to fit a local plane
M-by-3 matrix | *M*-by-*N*-by-3 matrix

Normals used to fit a local plane, returned as an *M*-by-3 or an *M*-by-*N*-by-3 matrix. The normal vectors are computed locally using the number of neighbors defined by the value of `k`. If `k` is not an input, it uses six neighboring points. The direction of each normal vector can be set based on how you acquired the points. The “Estimate Normals of Point Cloud” on page 3-1071 example, shows how to set the direction when the normal vectors are pointing towards the sensor.

Version History

Introduced in R2015b

References

[1] Hoppe, H., T. DeRose, T. Duchamp, J. Mcdonald, and W. Stuetzle. "Surface Reconstruction from Unorganized Points". *Computer Graphics (SIGGRAPH 1992 Proceedings)*. 1992, pp. 71–78.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- For GPU code generation, the number of neighbors must be a compile-time constant.
- Because of architectural and implementation differences between the GPU and CPU, the choice of neighboring points may differ during computation of normal vectors. As a result, there may be numerical mismatch between MATLAB simulation and the output from code generation.

See Also

`pointCloud` | `pcplayer` | `pcshow` | `pcwrite` | `pcmerge` | `pcfitplane` | `planeModel` | `pctransform` | `pcdownsample` | `pcdenoise` | `pcregistericp`

Topics

“3-D Point Cloud Registration and Stitching”
“The PLY Format”

pcfitcylinder

Fit cylinder to 3-D point cloud

Syntax

```
model = pcfitcylinder(ptCloudIn,maxDistance)
model = pcfitcylinder(ptCloudIn,maxDistance,referenceVector)
model = pcfitcylinder(ptCloudIn,maxDistance,referenceVector,
maxAngularDistance)

[model,inlierIndices,outlierIndices] = pcfitcylinder( ___ )
[model,inlierIndices,outlierIndices,meanError] = pcfitcylinder( ___ )
[ ___ ] = pcfitcylinder( ___ ,Name=Value)
```

Description

`model = pcfitcylinder(ptCloudIn,maxDistance)` fits a cylinder to a point cloud with a maximum allowable distance from an inlier point to the cylinder. This function uses the M-estimator SAmple Consensus (MSAC) algorithm to find the cylinder.

`model = pcfitcylinder(ptCloudIn,maxDistance,referenceVector)` fits a cylinder to the point cloud with additional orientation constraints specified by the 1-by-3 reference orientation input vector.

`model = pcfitcylinder(ptCloudIn,maxDistance,referenceVector, maxAngularDistance)` additionally specifies the maximum allowed absolute angular distance.

`[model,inlierIndices,outlierIndices] = pcfitcylinder( ___ )` additionally returns linear indices to the inlier and outlier points in the point cloud input.

`[model,inlierIndices,outlierIndices,meanError] = pcfitcylinder( ___ )` additionally returns the mean error of the distance of the inlier points to the model.

`[ ___ ] = pcfitcylinder( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MaxNumTrials=1000` sets the maximum number of random trials to 1000.

Examples

Extract Cylinder from Point Cloud

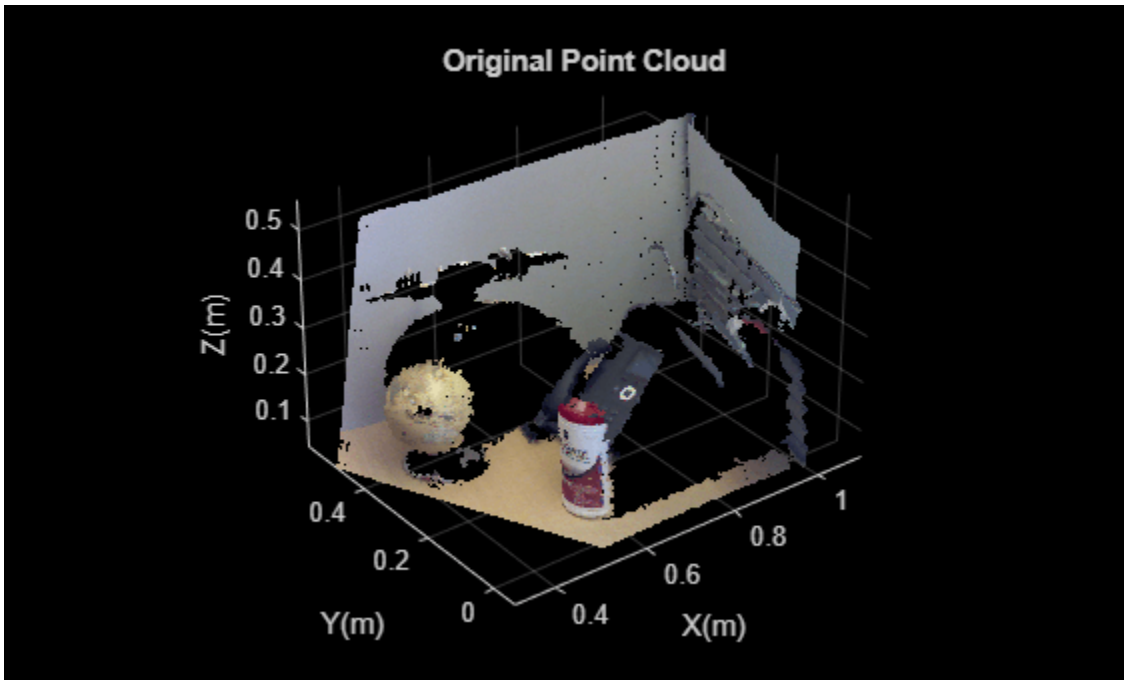
Load the point cloud.

```
load("object3d.mat");
```

Display the point cloud.

```
figure
pcshow(ptCloud)
```

```
xlabel("X(m)")
ylabel("Y(m)")
zlabel("Z(m)")
title("Original Point Cloud")
```



Set the maximum point-to-cylinder distance (5 mm) for cylinder fitting.

```
maxDistance = 0.005;
```

Set the region of interest to constrain the search.

```
roi = [0.4 0.6 -Inf 0.2 0.1 Inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set the orientation constraint.

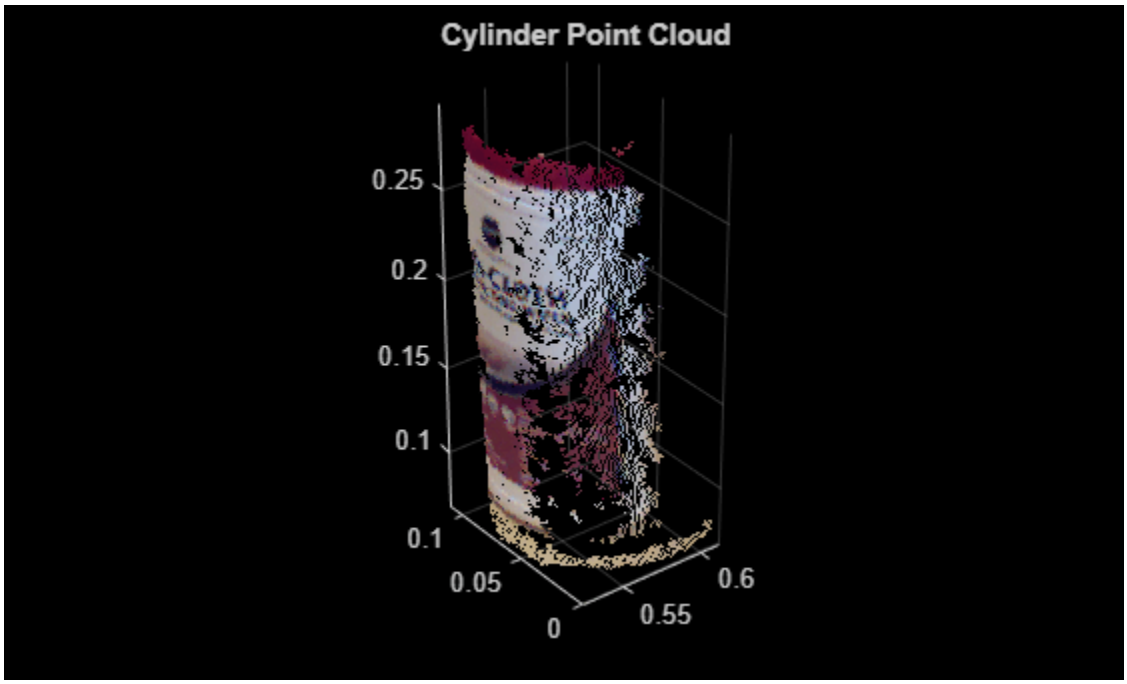
```
referenceVector = [0 0 1];
```

Detect the cylinder and extract it from the point cloud by specifying the inlier points.

```
[model,inlierIndices] = pcfitcylinder(ptCloud,maxDistance,...
        referenceVector,SampleIndices=sampleIndices);
pc = select(ptCloud,inlierIndices);
```

Plot the extracted cylinder.

```
figure
pcshow(pc)
title("Cylinder Point Cloud")
```



Detect Cylinder in Point Cloud

Load a MAT file containing a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud.

```
figure
pcshow(ptCloud)
title("Detect a Cylinder in a Point Cloud")
```

Set the maximum point-to-cylinder distance for cylinder fitting to 5mm.

```
maxDistance = 0.005;
```

Specify a region of interest (ROI) to constrain the fitting function.

```
roi = [0.4 0.6; -inf 0.2; 0.1 inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Set the orientation constraint for the fitting function

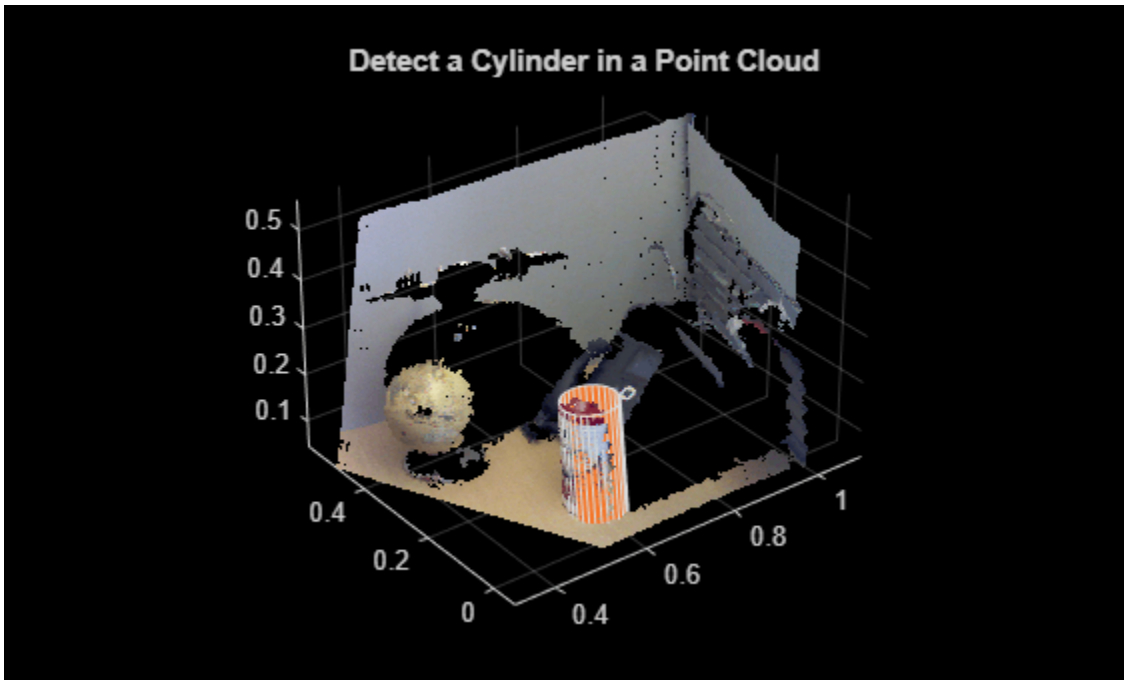
```
referenceVector = [0 0 1];
```

Detect the cylinder in the specified ROI of the point cloud and extract it.

```
model = pcfitcylinder(ptCloud,maxDistance,referenceVector, ...
        SampleIndices=sampleIndices);
```

Plot the model of the detected cylinder.

```
hold on
plot(model)
```



Input Arguments

`ptCloudIn` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object. If the `Normal` property of the input `ptCloud` is empty, the function populates it with values to meet the requirements of the fitting algorithm.

`maxDistance` — Maximum distance from an inlier point to the cylinder

scalar value

Maximum distance from an inlier point to the cylinder, specified as a scalar value. Specify the distance in units that are consistent with the units you are using for the point cloud.

Data Types: `single` | `double`

`referenceVector` — Reference orientation constraint

1-by-3 vector

Reference orientation constraint, specified as a 1-by-3 vector.

`maxAngularDistance` — Maximum absolute angular distance

5 (default) | scalar value

Maximum absolute angular distance, specified as a scalar value. The maximum angular distance is measured in degrees between the direction of the fitted cylinder and the reference orientation.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'SampleIndices',[]`.

`SampleIndices` — Linear indices of points to sample
`[]` (default) | column vector

Linear indices of points to sample in the input point cloud, specified as the comma-separated pair consisting of '`SampleIndices`' and a column vector. An empty vector means that all points are candidates to sample when fitting the cylinder during the RANSAC iteration. If you specify a subset of points, the function fits the model by sampling only those points in the subset. Providing a subset of points can significantly speed up the process by reducing the number of trials. You can generate the indices vector using the `findPointsInROI` method of the `pointCloud` object.

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | positive integer

Maximum number of random trials for finding inliers, specified as the comma-separated pair consisting of '`MaxNumTrials`' and a positive integer. To improve robustness of the output, increase this value. However, doing so adds additional computations.

`Confidence` — Percentage for finding maximum number of inliers
`99` (default) | numeric scalar in the range (0 100)

Percentage for finding maximum number of inliers, specified as the comma-separated pair consisting of '`Confidence`' and a numeric scalar, in the range (0 100). To improve the robustness of the output, increase this value. However, doing so adds additional computations.

Output Arguments

`model` — Geometric model of cylinder
`cylinderModel` object.

Geometric model of cylinder, returned as a `cylinderModel` object.

The coefficients for the output model are set to zero when:

- The input point cloud does not contain enough valid points.
- The algorithm cannot find enough inlier points.

`inlierIndices` — Linear indices of inlier points
column vector

Linear indices of the inlier points in the input point cloud, returned as a column vector.

`outlierIndices` — Linear indices of outlier points
column vector

Linear indices of the outlier points in the input point cloud returned as a column vector.

`meanError` — Mean square error
scalar value

Mean error of the distance of inlier points to the model, returned as a scalar value.

Algorithms

The function returns a geometric model that describes the cylinder. This function uses the M-estimator SAmple Consensus (MSAC) algorithm to find the cylinder. The MSAC algorithm is a variant of the RANdom SAmple Consensus (RANSAC) algorithm.

The fitting algorithm for the `pcfitcylinder` function requires point cloud normals. Therefore, if the `Normal` property for the input point cloud is empty, the function fills it. When the function fills the `Normal` property, it uses six points to fit the local cylinder. If six points do not work and the fitting fails, consider calling the `pcnormals` function which enables you to select the number of points to use.

Version History

Introduced in R2015b

References

[1] Torr, P. H. S., and A. Zisserman. “MLESAC: A New Robust Estimator with Application to Estimating Image Geometry.” *Computer Vision and Image Understanding*. Volume 78, Issue 1, April 2000, pp. 138-156.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

The optimized GPU implementation of `pcfitcylinder` uses `randi` to generate random numbers instead of `randperm` used in the MATLAB implementation. As a result, there might be numerical differences between MATLAB simulation and code generation.

See Also

Objects

`cylinderModel` | `planeModel` | `sphereModel` | `affine3d` | `pointCloud`

Functions

`pcfitplane` | `pcfitsphere` | `findPointsInROI` | `pcplayer` | `pcshow` | `pcwrite` | `pcread` | `pcmerge` | `pctransform` | `pcregistericp` | `pcdenoise`

Topics
“3-D Point Cloud Registration and Stitching”

pcfitplane

Fit plane to 3-D point cloud

Syntax

```
model = pcfitplane(ptCloudIn,maxDistance)
model = pcfitplane(ptCloudIn,maxDistance,referenceVector)
model = pcfitplane(ptCloudIn,maxDistance,referenceVector,maxAngularDistance)

[model,inlierIndices,outlierIndices] = pcfitplane( ___ )
[model,inlierIndices,outlierIndices,meanError] = pcfitplane( ___ )
[ ___ ] = pcfitplane( ___ ,Name=Value)
```

Description

`model = pcfitplane(ptCloudIn,maxDistance)` fits a plane to a point cloud that has a maximum allowable distance from an inlier point to the plane. The function returns a geometrical model that describes the plane.

This function uses the M-estimator SAmple Consensus (MSAC) algorithm to find the plane. The MSAC algorithm is a variant of the RANdom SAmple Consensus (RANSAC) algorithm.

`model = pcfitplane(ptCloudIn,maxDistance,referenceVector)` fits a plane to a point cloud that has additional orientation constraints specified by the 1-by-3 `referenceVector` input.

`model = pcfitplane(ptCloudIn,maxDistance,referenceVector,maxAngularDistance)` fits a plane to a point cloud that has a specified maximum angular distance.

`[model,inlierIndices,outlierIndices] = pcfitplane( ___ )` additionally returns the linear indices to the inlier and outlier points in the point cloud input.

`[model,inlierIndices,outlierIndices,meanError] = pcfitplane( ___ )` additionally returns the mean error of the distance of inlier points to the model, using any of the preceding syntaxes.

`[ ___ ] = pcfitplane( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `pcfitplane(ptCloud,maxDistance,referenceVector,maxAngularDistance,Confidence=95)` sets the confidence percentage for finding maximum number of inlier to `95`.

Examples

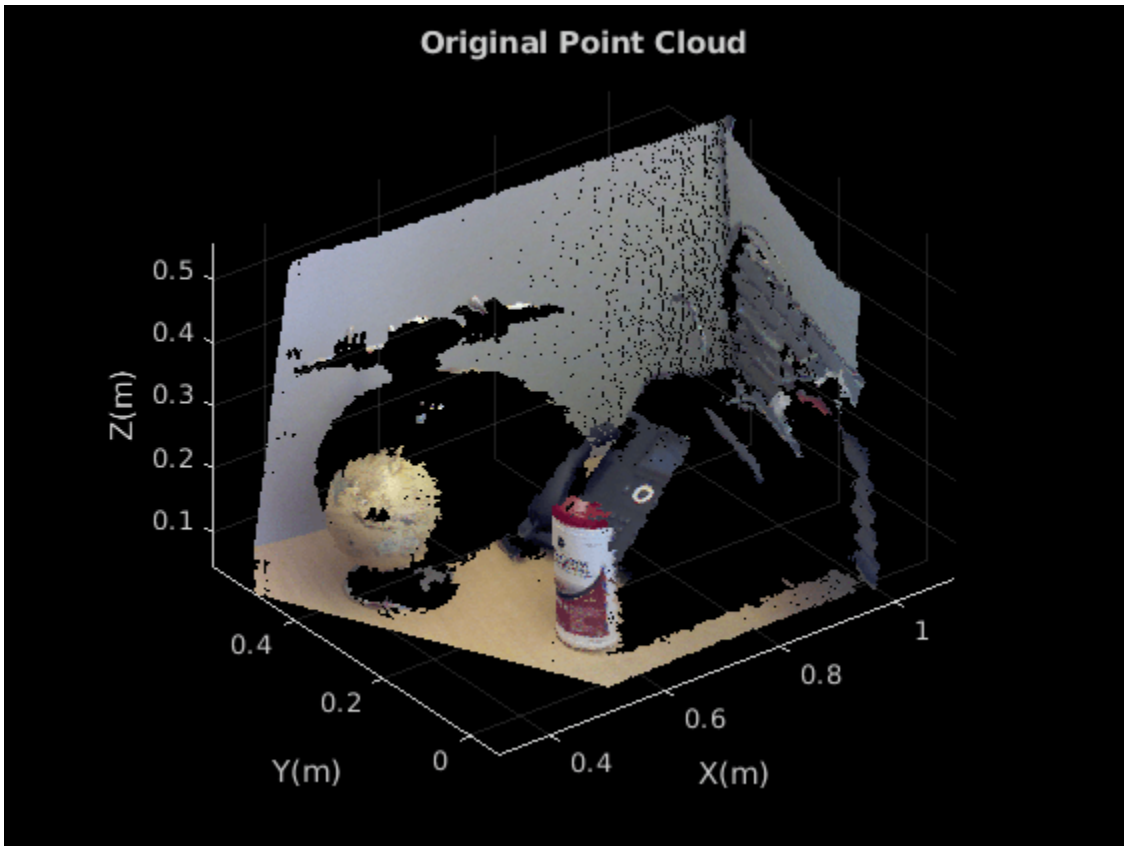
Detect Multiple Planes from Point Cloud

Load the point cloud.

```
load("object3d.mat")
```

Display the point cloud and label the figure.

```
figure
pcshow(ptCloud)
xlabel("X(m)")
ylabel("Y(m)")
zlabel("Z(m)")
title("Original Point Cloud")
```



Set the maximum point-to-plane distance (2cm) for plane fitting.

```
maxDistance = 0.02;
```

Set the normal vector of the plane.

```
referenceVector = [0,0,1];
```

Set the maximum angular distance to 5 degrees.

```
maxAngularDistance = 5;
```

Detect the first plane, the table, and extract it from the point cloud.

```
[model1,inlierIndices,outlierIndices] = pcfitplane(ptCloud,...
            maxDistance,referenceVector,maxAngularDistance);
plane1 = select(ptCloud,inlierIndices);
remainPtCloud = select(ptCloud,outlierIndices);
```

Set the region of interest to constrain the search for the second plane, left wall.

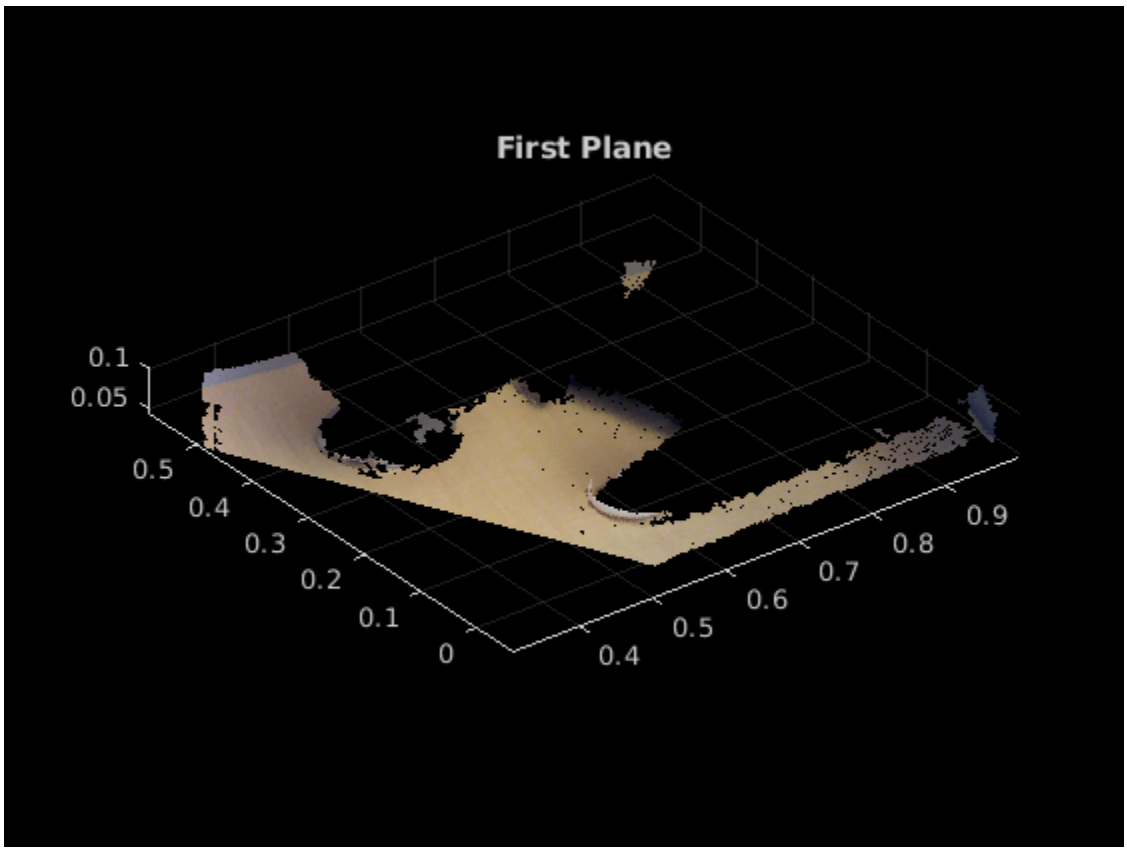
```
roi = [-inf,inf;0.4,inf;-inf,inf];
sampleIndices = findPointsInROI(remainPtCloud,roi);
```

Detect the left wall and extract it from the remaining point cloud.

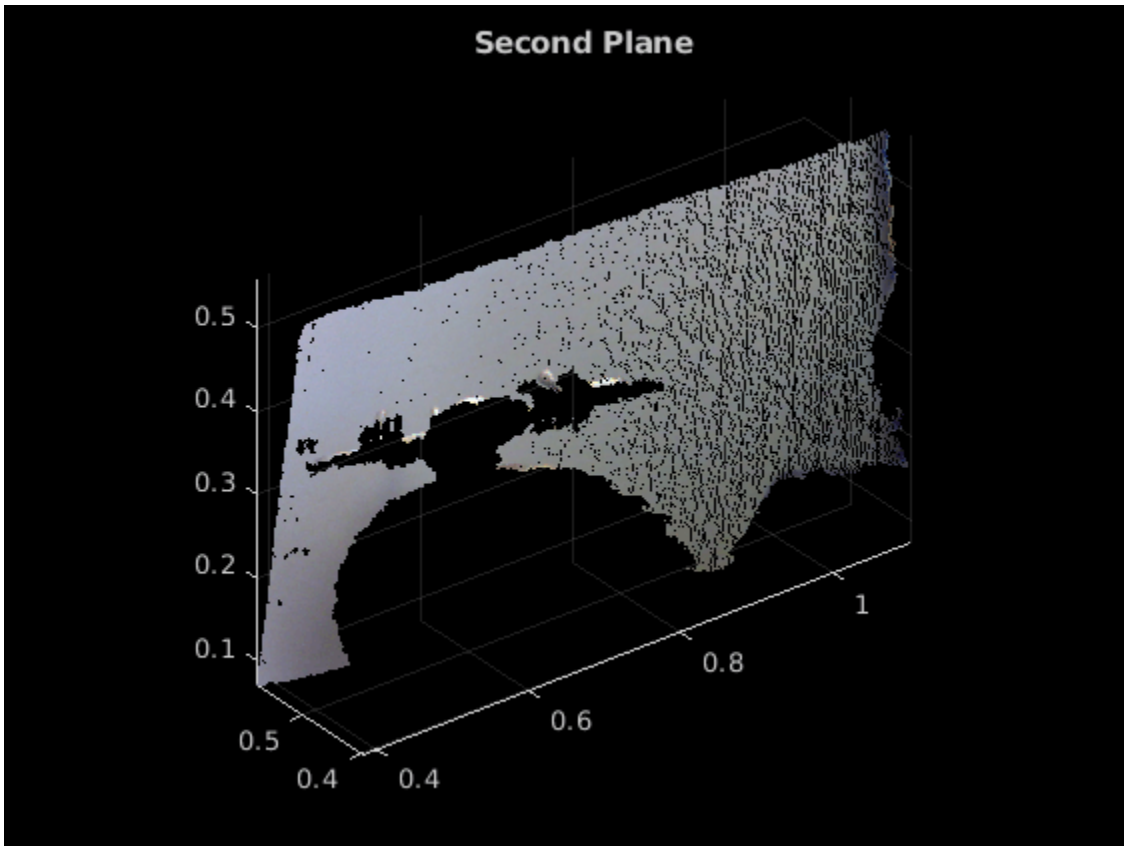
```
[model2,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,...
            maxDistance,SampleIndices=sampleIndices);
plane2 = select(remainPtCloud,inlierIndices);
remainPtCloud = select(remainPtCloud,outlierIndices);
```

Plot the two planes and the remaining points.

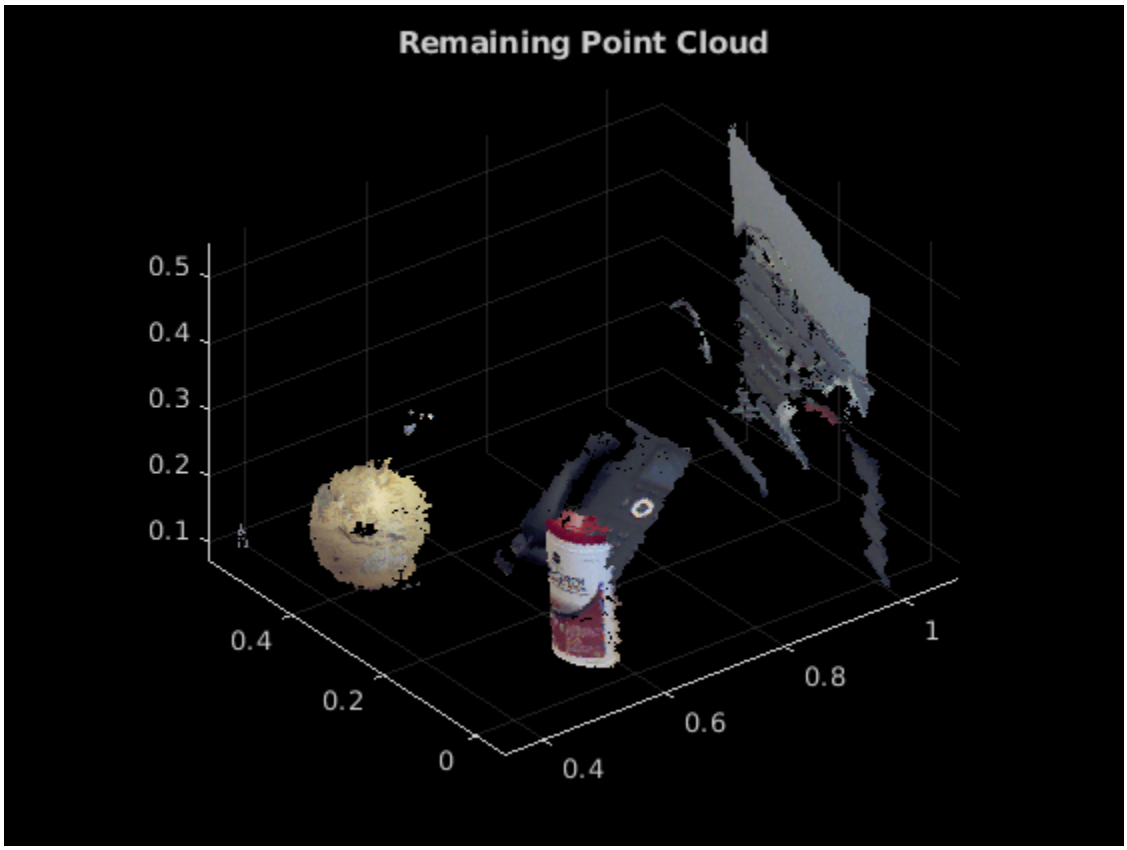
```
figure
pcshow(plane1)
title("First Plane")
```




```
figure
pcshow(plane2)
title("Second Plane")
```




```
figure
pcshow(remainPtCloud)
title("Remaining Point Cloud")
```



Input Arguments

`ptCloudIn` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`maxDistance` — Maximum distance from an inlier point to the plane
scalar value

Maximum distance from an inlier point to the plane, specified as a scalar value. Specify the distance in units that are consistent with the units you are using for the point cloud.

Data Types: `single` | `double`

`referenceVector` — Reference orientation constraint
1-by-3 vector

Reference orientation constraint, specified as a 1-by-3 vector. You must specify this argument for the function to apply an orientation constraint to fit a plane to the input point cloud. If you do not specify the reference vector, the function fits the model using the plane equation, $ax + by + cz + d = 0$.

Data Types: `single` | `double`

`maxAngularDistance` — Maximum absolute angular distance
5 degrees (default) | scalar value

Maximum absolute angular distance between the normal vector of the fitted plane and the reference orientation, specified as a scalar value in degrees.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `pcfitplane(ptCloud,maxDistance,referenceVector,maxAngularDistance,Confidence=95)` sets the confidence percentage for finding maximum number of inlier to 95.

`SampleIndices` — Linear indices of points to be sampled
`[]` (default) | column vector

Linear indices of points to sample in the input point cloud, specified as a column vector. An empty vector means that all points are candidates to sample in the RANSAC iteration to fit the plane. When you specify a subset, only points in the subset are sampled to fit a model.

Providing a subset of points can significantly speed up the process and reduce the number of trials. You can generate the indices vector using the `findPointsInROI` object function of the `pointCloud` object.

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | positive integer

Maximum number of random trials for finding inliers, specified as a positive integer. Increasing this value makes the output more robust but adds additional computations.

`Confidence` — Confidence percentage for finding maximum number of inliers
`99` (default) | numeric scalar in the range (0, 100)

Confidence percentage for finding maximum number of inliers, specified as a numeric scalar in the range (0 100). Increasing this value makes the output more robust but adds additional computations.

Output Arguments

`model` — Geometric model of plane
`planeModel` object

Geometric model of plane, returned as a `planeModel` object.

When the input point cloud does not contain enough valid points, or when the function cannot find enough inlier points, the coefficients for the output model are set to zero.

`inlierIndices` — Linear indices of inlier points
column vector

Linear indices of inlier points within the input point cloud, returned as a column vector.

`outlierIndices` — Linear indices of outlier points
column vector

Linear indices of outlier points within the input point cloud, returned as a column vector.

`meanError` — Mean square error
scalar value

Mean error of the distance of inlier points to the model, returned as a scalar value.

Version History

Introduced in R2015b

References

[1] Torr, P. H. S., and A. Zisserman. “MLESAC: A New Robust Estimator with Application to Estimating Image Geometry.” *Computer Vision and Image Understanding*. 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- For optimized CUDA code generation, the confidence percentage for finding maximum number of inliers is always set to 99.

See Also

Objects

`pointCloud` | `affinetform3d` | `planeModel`

Functions

`pcshow` | `pcwrite` | `pcread` | `pcfitsphere` | `pcfitcylinder` | `findPointsInROI` | `pcmerge` | `pctransform` | `pcregistericp` | `pcdenoise`

Topics

“3-D Point Cloud Registration and Stitching”

pcfitsphere

Fit sphere to 3-D point cloud

Syntax

```
model = pcfitsphere(ptCloudIn,maxDistance)
[model,inlierIndices,outlierIndices] = pcfitsphere( ___ )
[model,inlierIndices,outlierIndices,meanError] = pcfitsphere( ___ )
[ ___ ] = pcfitsphere( ___ ,Name=Value)
```

Description

`model = pcfitsphere(ptCloudIn,maxDistance)` fits a sphere to a point cloud that has a maximum allowable distance from an inlier point to the sphere. The function returns a geometrical model that describes the sphere.

This function uses the M-estimator SAmple Consensus (MSAC) algorithm to find the sphere. The MSAC algorithm is a variant of the RANdom SAmple Consensus (RANSAC) algorithm.

`[model,inlierIndices,outlierIndices] = pcfitsphere( ___ )` additionally returns linear indices to the inlier and outlier points in the point cloud input.

`[model,inlierIndices,outlierIndices,meanError] = pcfitsphere( ___ )` additionally returns the mean error of the distance of inlier points to the model, using any of the preceding syntaxes.

`[ ___ ] = pcfitsphere( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `MaxNumTrials=1000` sets the maximum number of random trials to 1000.

Examples

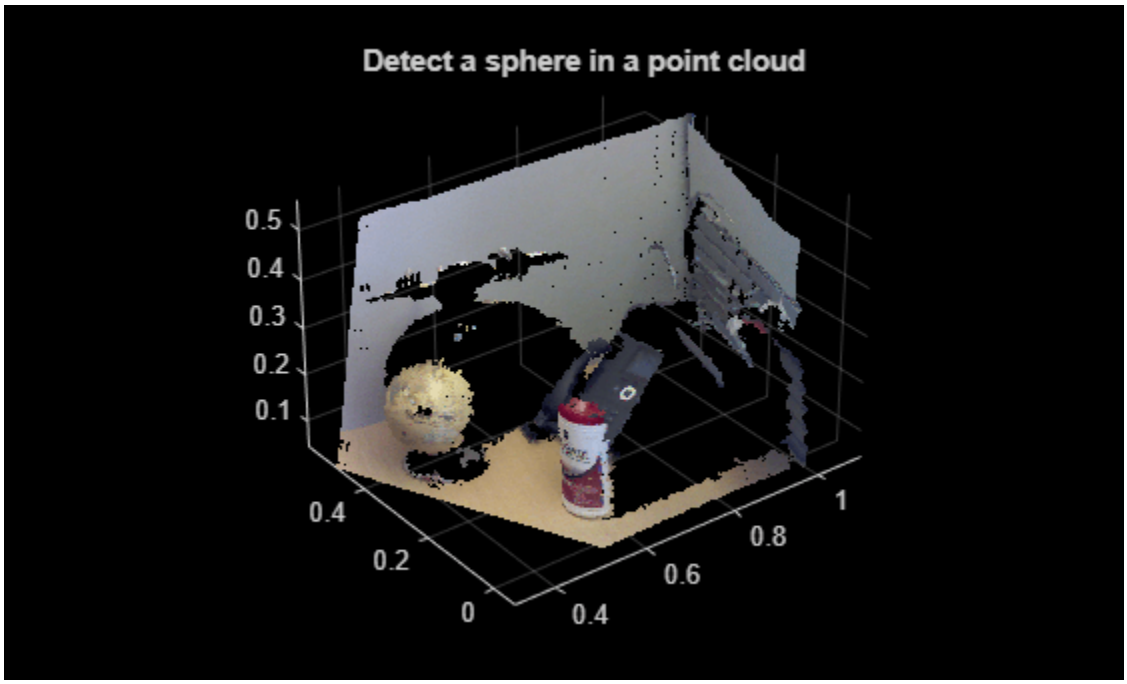
Detect Sphere in Point Cloud

Load a point cloud into the workspace.

```
load("object3d.mat");
```

Display the point cloud and label the figure.

```
figure
pcshow(ptCloud)
title("Detect a sphere in a point cloud")
```



Set the maximum point-to-sphere distance for sphere fitting to 1cm.

```
maxDistance = 0.01;
```

Set the region of interest to constrain the search.

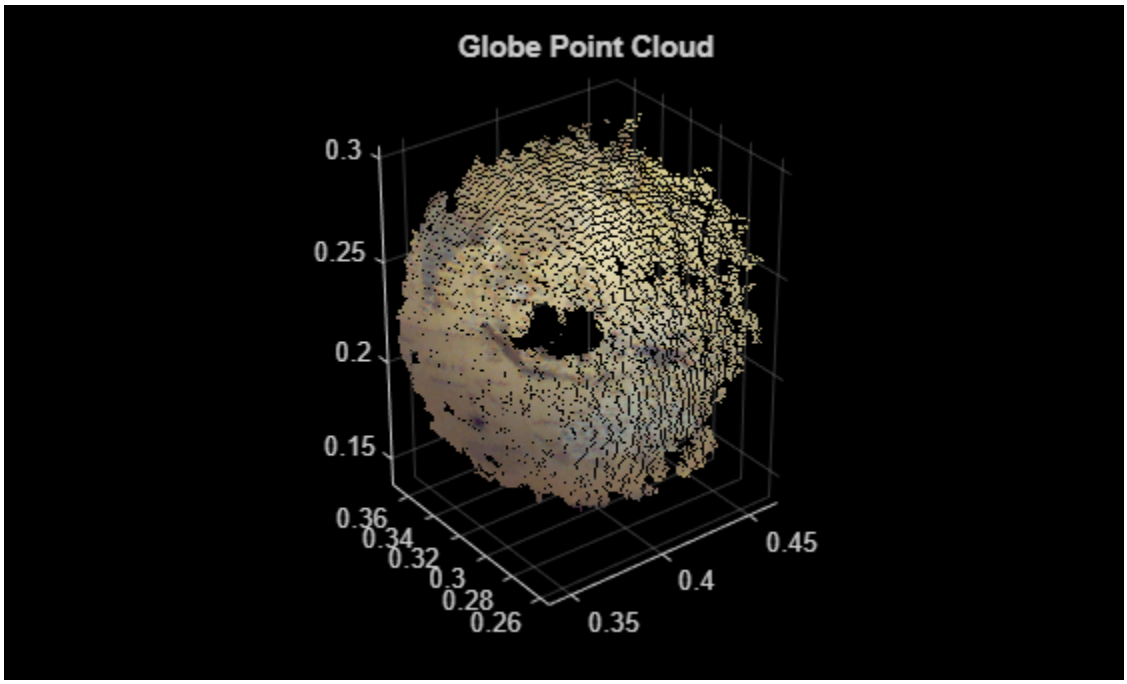
```
roi = [-inf,0.5;0.2,0.4;0.1,inf];
sampleIndices = findPointsInROI(ptCloud,roi);
```

Detect the globe in the point cloud and extract it.

```
[model,inlierIndices] = pcfitsphere(ptCloud,maxDistance,SampleIndices=sampleIndices);
globe = select(ptCloud,inlierIndices);
```

Plot the extracted globe.

```
figure
pcshow(globe)
title("Globe Point Cloud")
```



Input Arguments

`ptCloudIn` — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object.

`maxDistance` — Maximum distance from an inlier point to the sphere
scalar value

Maximum distance from an inlier point to the sphere, specified as a scalar value. Specify the distance in units that are consistent with the units you are using for the point cloud.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxNumTrials=1000` sets the maximum number of random trials to 1000.

`SampleIndices` — Linear indices of points to be sampled
`[]` (default) | column vector

Linear indices of points to sample in the input point cloud, specified as a column vector. An empty vector means that all points are candidates to sample in the RANSAC iteration to fit the sphere. When you specify a subset, only points in the subset are sampled to fit a model. Providing a subset of points

can significantly speed up the process and reduce the number of trials. You can generate the indices vector using the `findPointsInROI` object function of the `pointCloud` object.

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | positive integer

Maximum number of random trials for finding inliers, specified as a positive integer. Increasing this value makes the output more robust but adds additional computations.

`Confidence` — Confidence percentage for finding maximum number of inliers
`99` (default) | numeric scalar in the range (0 100)

Confidence percentage for finding maximum number of inliers, specified as a numeric scalar representing percentage, in the range (0 100). Increasing this value makes the output more robust but adds additional computations.

Output Arguments

`model` — Geometric model of sphere
`sphereModel` object

Geometric model of sphere, returned as a `sphereModel` object.

When the input point cloud does not contain enough valid points, or when the function cannot find enough inlier points, the coefficients for the output model are set to zero.

`inlierIndices` — Linear indices of inlier points
column vector

Linear indices of inlier points within the input point cloud, returned as a column vector.

`outlierIndices` — Linear indices of outlier points
column vector

Linear indices of outlier points within the input point cloud, returned as a column vector.

`meanError` — Mean square error
scalar value

Mean error of the distance of inlier points to the model, returned as a scalar value.

Version History

Introduced in R2015b

References

[1] Torr, P. H. S. and A. Zisserman. “MLESAC: A New Robust Estimator with Application to Estimating Image Geometry.” *Computer Vision and Image Understanding*. 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects

`pointCloud` | `planeModel` | `sphereModel` | `affine3d`

Functions

`pcfitplane` | `pcfitcylinder` | `findPointsInROI` | `pcshow` | `pcwrite` | `pcread` | `pcmerge` | `pctransform` | `pcregistericp` | `pcdenoise`

Topics

“3-D Point Cloud Registration and Stitching”

pixelLabelTrainingData

Create training data for semantic segmentation from ground truth

Syntax

```
[imds,pxds] = pixelLabelTrainingData(gTruth)
[imds,pxds] = pixelLabelTrainingData(gTruth,Name,Value)
```

Description

`[imds,pxds] = pixelLabelTrainingData(gTruth)` creates image datastore `imds` and pixel label datastore `pxds` from the specified ground truth. You can combine the returned datastores using the `combine` function and use the `trainnet` function to train deep learning segmentation networks. You can also use these datastores with the `evaluateSemanticSegmentation` function to evaluate the result from deep learning or classical segmentation methods.

This function supports parallel computing using multiple MATLAB workers. Enable parallel computing using the “Computer Vision Toolbox Preferences” dialog box.

`[imds,pxds] = pixelLabelTrainingData(gTruth,Name,Value)` returns image and pixel label datastores with additional options specified by one or more name-value arguments.

- If the `groundTruth` objects in `gTruth` were created using a video file, a custom data source, or an `imageDatastore` with different custom read functions, then you can specify any combination of name-value pair arguments.
- If the `groundTruth` objects were created from an image collection or image sequence data source, then you can specify only the `SamplingFactor` name-value pair argument

Examples

Prepare Data for Evaluating Semantic Segmentation Algorithm

Load a `groundTruth` object named `gTruth`. The ground truth contains pixel labels for triangles and background, annotated on a video with 100 frames.

```
visiondataPath = fullfile(matlabroot, 'toolbox', 'vision', 'visiondata');
addpath(fullfile(visiondataPath, 'triangleImages'));
addpath(fullfile(visiondataPath, 'triangleImages', 'testLabels'));
loadedData = load(fullfile(visiondataPath, 'triangleImages', 'triangleGroundTruth.mat'));
gTruth = loadedData.gTruth;
```

Create a folder in the current directory.

```
foldername = fullfile(tempdir,"videoFrames");
mkdir(foldername)
```

Create an `imageDatastore` and a `pixelLabelDatastore` from the video file and corresponding pixel labels. Write every fifth image to disk.

```
[imds,pxdsTruth] = pixelLabelTrainingData(gTruth,...
    'SamplingFactor',5,'WriteLocation',foldername);

Write images extracted for training to folder:
    /tmp/videoFrames

Writing 20 images extracted from triangleVideo.avi...Completed.
```

Confirm that the temporary folder contains every fifth image.

```
imds.Files

ans = 20×1 cell array
    "'/tmp/videoFrames/triangleVideo01.png'"
    "'/tmp/videoFrames/triangleVideo06.png'"
    "'/tmp/videoFrames/triangleVideo11.png'"
    "'/tmp/videoFrames/triangleVideo16.png'"
    "'/tmp/videoFrames/triangleVideo21.png'"
    "'/tmp/videoFrames/triangleVideo26.png'"
    "'/tmp/videoFrames/triangleVideo31.png'"
    "'/tmp/videoFrames/triangleVideo36.png'"
    "'/tmp/videoFrames/triangleVideo41.png'"
    "'/tmp/videoFrames/triangleVideo46.png'"
```

Remove the video and images from the path.

```
rmpath(fullfile(visiondataPath, 'triangleImages'));
rmpath(fullfile(visiondataPath, 'triangleImages', 'testLabels'));
```

Input Arguments

`gTruth` — Ground truth data

scalar `groundTruth` object | array of `groundTruth` objects

Ground truth data, specified as a scalar `groundTruth` object or an array of `groundTruth` objects. When `gTruth` is an array of `groundTruth` objects, the `LabelDefinitions` property of each object must contain the same pixel label names.

If you use custom data sources in `gTruth` with parallel computing enabled, then the reader function is expected to work with a pool of MATLAB workers to read images from the data source in parallel.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"SamplingFactor",5`

`SamplingFactor` — Factor for subsampling images

1 (default) | integer | vector of integers

Factor for subsampling images in the ground truth data source, specified an integer or a vector of integers. For a sampling factor of *N*, the returned training data includes every *N*th image in the ground truth data source. The function ignores ground truth images with empty label data.

Use sampled data to reduce repeated data, such as a sequence of images with the same scene and labels. It can also help in reducing training time.

Value	Sampling Factor
Integer	Manually set the sampling factor to apply to all data.
Vector of integers	When you input a vector of ground truth objects, the function uses the sampling factor specified by the corresponding vector element.

`ImageFormat` — Image file format
PNG (default) | string scalar | character vector

Image file format, specified as a string scalar or character vector. File formats must be supported by `imwrite`. This argument applies only for `groundTruth` objects created using a video file or a custom data source.

`WriteLocation` — Name of folder
`pwd` (current working folder) (default) | string scalar | character vector

Folder name to write extracted images to, specified as a string scalar or character vector. The specified folder must exist and have write permissions.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` with different custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`ImageFormat` — Image file format
PNG (default) | string scalar | character vector

Image file format, specified as a string scalar or character vector. File formats must be supported by `imwrite`.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` with different custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`NamePrefix` — Prefix for output image file names
string scalar | character vector

Prefix for output image file names, specified as a string scalar or character vector. The image files are named as:

```
<name_prefix><source_number>_<image_number>.<image_format>
```

The default value uses the name of the data source that the images were extracted from, `strcat(sourceName,"_")` for video and custom data source, or `"datastore"` for image datastore.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` with custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`Verbose` — Display training progress
`true` (default) | `false`

Display training progress on the MATLAB command line, specified as `true` or `false`. This argument applies only for `groundTruth` objects created using a video file or a custom data source.

Output Arguments

`imds` — Collection of images
`ImageDatastore` object

Collection of images extracted from the ground truth, `gTruth`, returned as an `ImageDatastore` object. Each image in `imds` has annotations with at least one class of pixel labels. `imds` ignores images that with no annotations.

`pxds` — Collection of pixel-labeled data
`PixelLabelDatastore` object

Collection of pixel-labeled data extracted from the ground truth, `gTruth`, returned as a `PixelLabelDatastore` object. The object contains a categorical matrix of pixel labels for each image contained in the image datastore, `imds`. Labels that do not correspond to pixel labels are ignored.

Version History

Introduced in R2018a

See Also

Apps

Image Labeler | Video Labeler

Functions

`objectDetectorTrainingData` | `semanticseg` | `evaluateSemanticSegmentation` | `trainnet`

Objects

`groundTruth` | `ImageDatastore` | `PixelLabelDatastore`

Topics

“Training Data for Object Detection and Semantic Segmentation”

plotCamera

Plot camera in 3-D coordinates

Syntax

```
cam = plotCamera
cam = plotCamera(camPoses)
cam = plotCamera(cameraTable)
cam = plotCamera(Name=Value)
```

Description

`cam = plotCamera` plots a default camera in 3-D coordinates in the current axes. The function returns `cam`, a `Camera` object that contains the properties of the plotted camera.

`cam = plotCamera(camPoses)` plots an array of camera visualization objects `camPoses`, in the current axes.

`cam = plotCamera(cameraTable)` plots one or more cameras specified by `cameraTable`.

`cam = plotCamera(Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Opacity=0.4` sets the opacity of the plotted camera to `0.4`.

The name-value pair arguments set the associated properties of the plotted camera visualization object.

Examples

Plot Animated Camera Using Absolute Pose

Create a `rigidtform3d` object.

```
initialRotation = [1  0  0;
                   0  0  1;
                   0 -1  0];
initialTranslation = [10 0 20];
pose = rigidtform3d(initialRotation,initialTranslation);
```

Plot a camera with an opacity of zero and an absolute pose based on the created `rigidtform3d` object.

```
cam = plotCamera(AbsolutePose=pose,Opacity=0)
```

```
cam =
  Camera with properties:

           Parent: [1×1 Axes]
             Size: 1
     AbsolutePose: [1×1 rigidtform3d]
```

```
          Visible: 1
        AxesVisible: 0
      ButtonDownFcn: ''
              Color: [1 0 0]
            Opacity: 0
              Label: ''
```

Set viewing properties for the current axes.

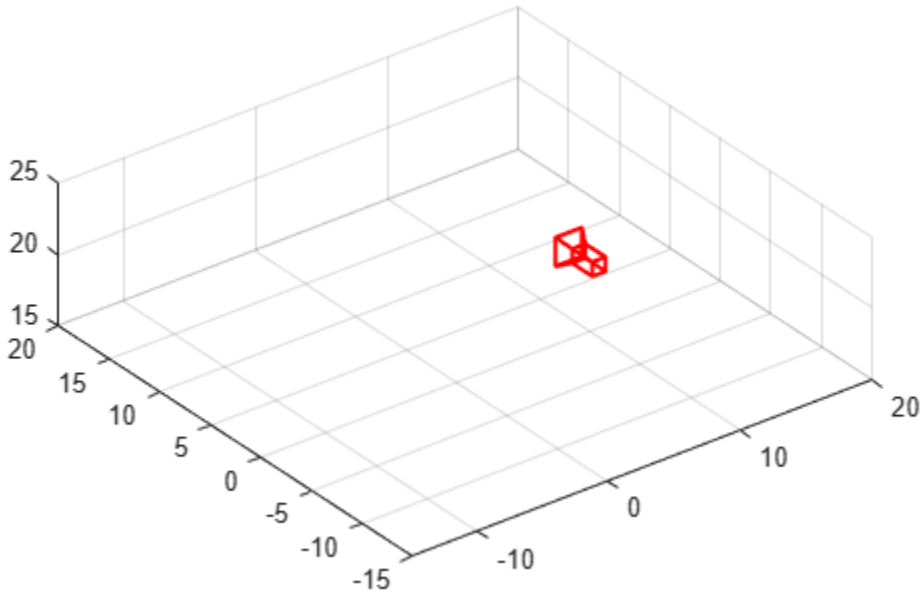
```
grid on
axis equal
axis manual
```

Expand the viewable limits of each axis. These changes enable the entire animation to be visible in the next step.

```
xlim([-15 20]);
ylim([-15 20]);
zlim([15 25]);
```

Rotate the camera around the *y*-axis.

```
for theta = 0:10:1800
    updateAngles = [0 -theta 0];
    updateTranslation = [10*cosd(theta) 10*sind(theta) 20];
    updateTform = rigidtform3d(updateAngles, updateTranslation);
    cam.AbsolutePose = rigidtform3d(initialRotation * updateTform.R, updateTform.Translation);
    drawnow();
end
```



Visualize Camera Extrinsics

Create a set of calibration images.

```
imds = imageDatastore(fullfile(toolboxdir("vision"),"visiondata","calibration","slr"));
```

Detect the checkerboard corners in the images.

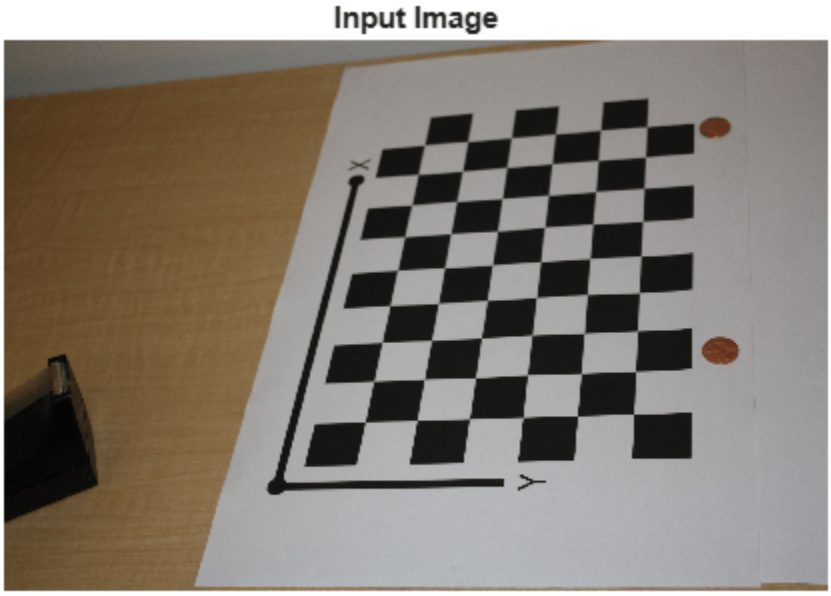
```
[imagePoints,patternDims] = detectCheckerboardPoints(imds.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0). Set the square size to 29 mm.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Load an image at its new location.

```
imOrig = imread(fullfile(toolboxdir("vision"),"visiondata","calibration","slr","image9.jpg"));
figure;
imshow(imOrig);
title("Input Image");
```



Calibrate the camera.

```
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=size(imOrig,1:2));
```

Undistort the image.

```
im = undistortImage(imOrig,cameraParams.Intrinsics);
```

Find the reference object in the new image.

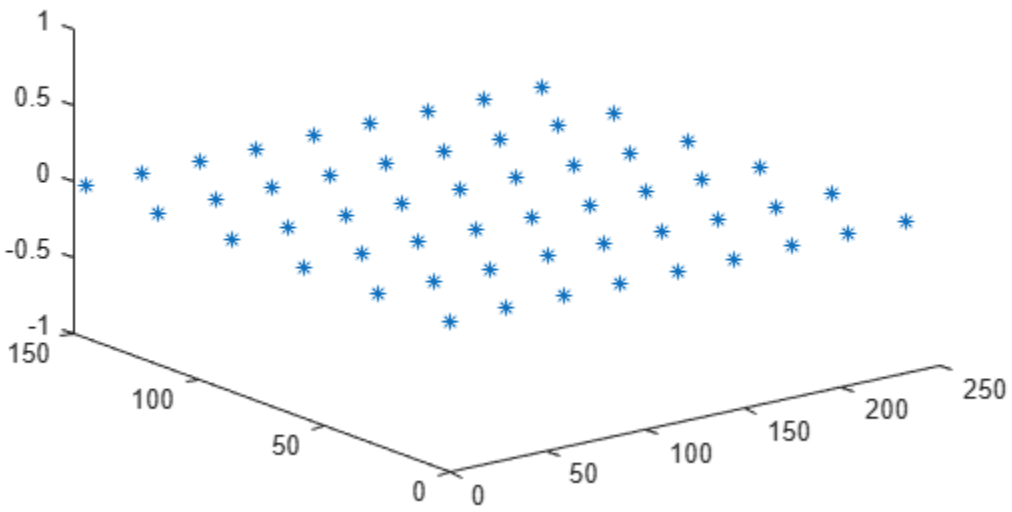
```
[imagePoints,patternDims] = detectCheckerboardPoints(im);
```

Compute the new extrinsics.

```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,cameraParams.Intrinsics);
```

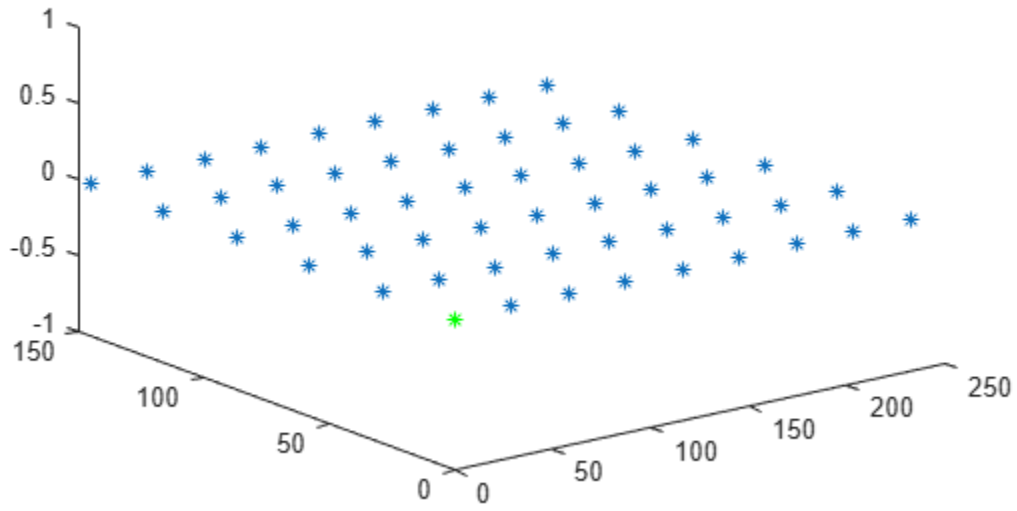
Plot the world points.

```
figure;
plot3(worldPoints(:,1),worldPoints(:,2),zeros(size(worldPoints, 1),1),"*");
hold on
```



Mark the origin.

```
plot3(0,0,0,"g*");
```

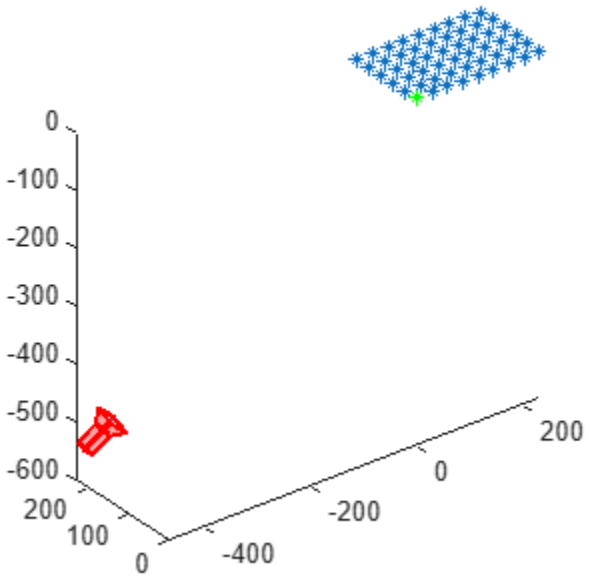


Compute the camera location and orientation.

```
absPose = extr2pose(camExtrinsics);
```

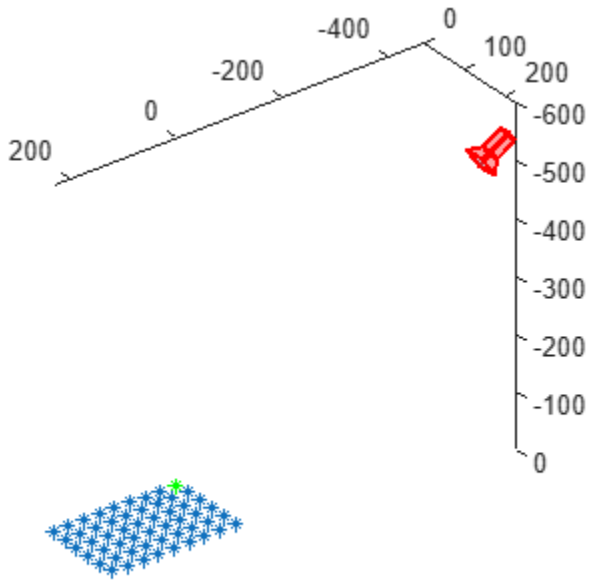
Plot the camera.

```
cam = plotCamera(AbsolutePose=absPose,Size=20);
```



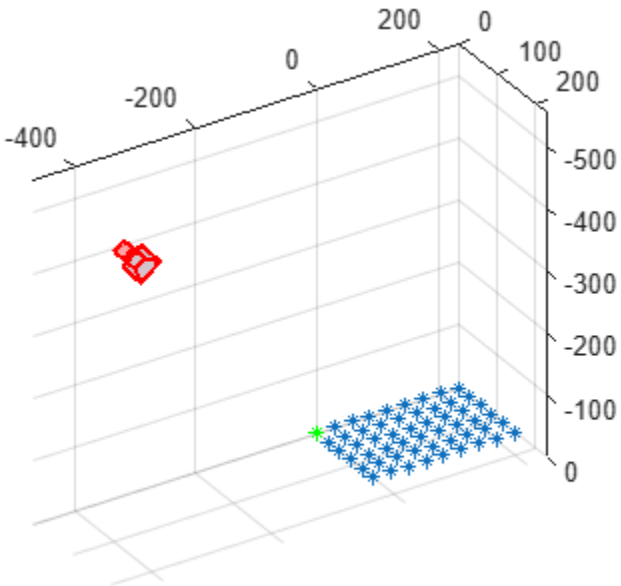
Make the z -axis point down.

```
set(gca,CameraUpVector=[0 0 -1]);
```



Set the view parameters.

```
camorbit(gca,-110,60,data=[0 0 1]);
axis equal
grid on
```

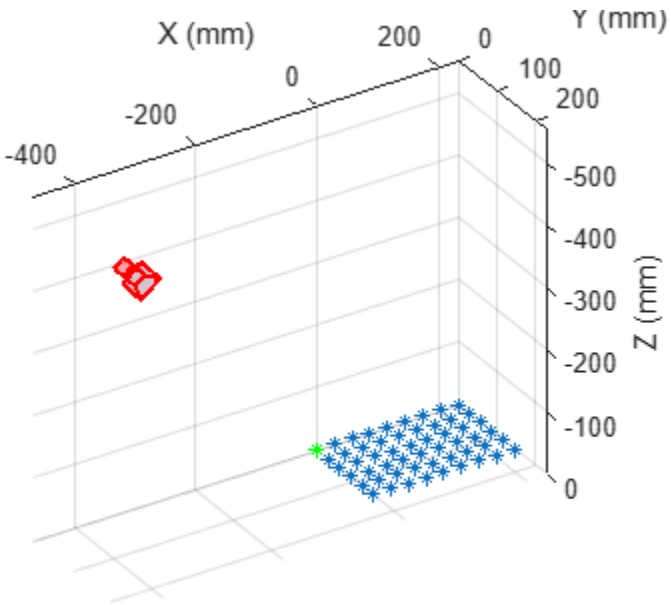


Turn on 3-D rotation.

```
cameratoolbar('SetMode','orbit');
```

Label the axes.

```
xlabel("X (mm)");
ylabel("Y (mm)");
zlabel("Z (mm)");
```



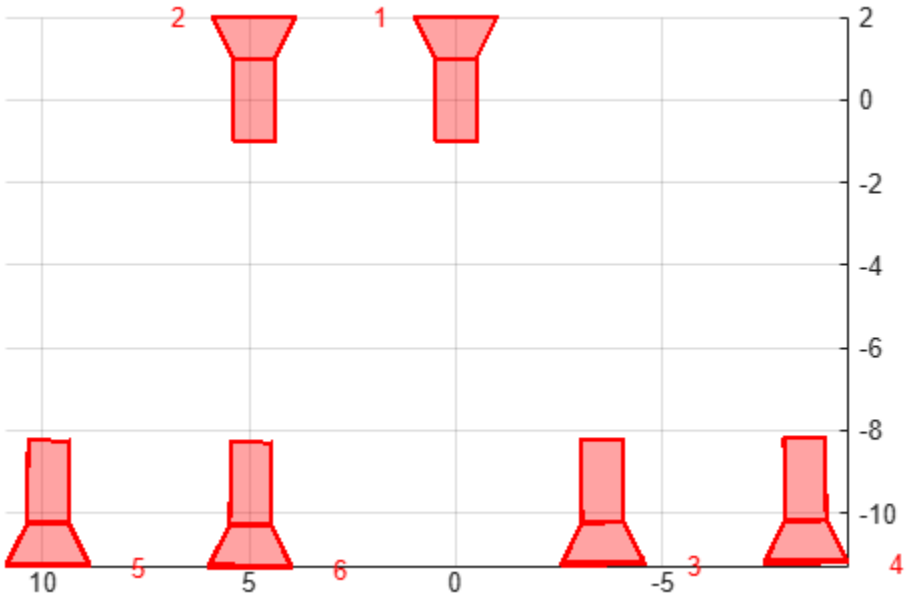
Plot Poses of Multiple Cameras

Load the poses of multiple cameras as a vector of `rigidtform3d` objects.

```
load("multiCameraPoses.mat")
```

Plot the cameras and show the results in a top-down view.

```
plotCamera(poses);
axis equal
view([0 1 0])
grid on
```



Input Arguments

`cameraTable` — Properties of cameras for visualization

table

Properties of cameras for visualization, specified as a table. Each row represents a single camera. Each column title must match the name-part of a name-value pair argument. The *n*th-row values set the properties for the *n*th element of `cam`. You cannot specify values for `Parent`. If the table contains a `ViewId` column, then the view IDs are used to set the `Label` values of the cameras.

`camPoses` — Camera visualization

scalar | array of `rigidtform3d` objects

Camera visualization, specified as a scalar or an array of `rigidtform3d` objects. These objects represent the camera poses in world coordinates.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Size=0.3` specifies the camera-base width as `0.3`.

`AbsolutePose` — Camera absolute pose

`rigidtform3d` object

Camera absolute pose in the world coordinate system, specified as a `rigidtform3d` object.

`Size` — Camera-base width
`1` (default) | positive real number

Camera-base width, specified as a positive real number.

`Label` — Camera label
`''` (default) | cell array of character vectors | array of strings

Camera label, specified as an array of strings or a cell array of character vectors. When camera poses are the initial input, the size of `Label` must match the size of `camPoses`.

`Color` — Camera color
`[1 0 0]` (red) (default) | RGB triplet

Camera color, specified as an RGB triplet. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0, 1].

`Opacity` — Camera opacity
`0.2` (default) | scalar in the range [0, 1]

Camera opacity, specified as a scalar in the range [0, 1].

`Visible` — Camera visibility
`true` or `1` (default) | `false` or `0`

Camera visibility, specified as a numeric or logical `1` (`true`) or `0` (`false`).

`AxesVisible` — Camera axes visibility
`false` or `0` (default) | `true` or `1`

Camera axes visibility, specified as a numeric or logical `1` (`true`) or `0` (`false`).

`ButtonDownFcn` — Callback function
`''` (default) | function handle

Callback function, specified as a function handle that executes when you click the camera.

`Parent` — Output axes
`axes` handle

Output axes, specified as an axes handle. By default, `plotCamera` uses the current axes handle. To return the current axes, use the `gca` function.

Output Arguments

`cam` — Camera visualization object
`Camera` object | row vector of `Camera` objects

Camera visualization object, returned as one of these options.

- `Camera` object — The function returns this option when plotting a single camera. Name-value arguments or `cameraTable` input elements set the corresponding `Camera` object properties.

- Row vector of `Camera` objects — The function returns this option when plotting multiple cameras. The *n*th -row values of the `cameraTable` input set the properties for the *n*th `Camera` object in this vector.

Camera objects are created using the `vision.graphics.Camera` class, which contains the following properties:

- “Parent” on page 3-0
- “Size” on page 3-0
- “AbsolutePose” on page 3-0
- “Visible” on page 3-0
- “AxesVisible” on page 3-0
- “ButtonDownFcn” on page 3-0
- “Color” on page 3-0
- “Opacity” on page 3-0
- “Label” on page 3-0

Version History

Introduced in R2015a

R2026a: Support for multiple camera poses

Added a new syntax to specify camera poses in world coordinates as a vector of `rigidtform3d` objects and enhanced the `Label` name-value argument to support multiple camera poses.

R2022b: Specify `AbsolutePose` as a `rigidtform3d` object

Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `AbsolutePose` name-value argument as a `rigidtform3d` object, which uses the premultiply convention.

Although you can still specify `AbsolutePose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to the new premultiply geometric transformation objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

R2020a: The `Location` and `Orientation` properties belonging to the `plotCamera` function are removed

The `plotCamera` function now accepts a `rigid3d` object to specify a camera pose. Starting in R2020a, you must use the `rigid3d` object instead of the `Location` and `Orientation` properties to specify the pose.

See Also

Functions
`estimateExtrinsics` | `showExtrinsics`

Objects
`rigidtform3d`

Topics
“Structure from Motion from Two Views”
“3-D Coordinate Systems”
“Using the Single Camera Calibrator App”
“Using the Stereo Camera Calibrator App”

pose2extr

Convert camera pose to extrinsics

Syntax

```
camExtrinsics = pose2extr(cameraPose)
```

Description

`camExtrinsics = pose2extr(cameraPose)` returns the camera extrinsics, or the transformation from world coordinates to camera coordinates, for a camera with pose `cameraPose` in world coordinates.

Examples

Convert World Coordinates to Camera Coordinates

Create a camera pose from a 3-D orientation matrix and location vector.

```
orientation = eye(3);
location = [0 0 10];
cameraPose = rigidtform3d(orientation,location);
```

Compute the transformation from world coordinates to camera coordinates.

```
tform = pose2extr(cameraPose)
```

```
tform =
  rigidtform3d with properties:

    Dimensionality: 3
       Translation: [0 0 -10]
                 R: [3×3 double]

                 A: [1     0     0     0
                     0     1     0     0
                     0     0     1   -10
                     0     0     0     1]
```

Input Arguments

cameraPose — Orientation and location of camera

`rigidtform3d` object | `se3`

Orientation and location of the camera in world coordinates, specified as a `rigidtform3d` or `se3` object.

Output Arguments

`camExtrinsics` — Camera extrinsics
`rigidtform3d` object | `se3` object

Camera extrinsics, returned as a `rigidtform3d` or `se3` object. The extrinsics are a transformation from world coordinates to camera coordinates that enables you to transform points from the world coordinate system to the camera coordinate system. `camExtrinsics` is the inverse of `cameraPose` as calculated using the `invert` function.

Version History

Introduced in R2022b

R2022b: Recommended over `cameraPoseToExtrinsics`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `cameraPoseToExtrinsics` function uses the postmultiply convention. Although there are no plans to remove `cameraPoseToExtrinsics` at this time, you can streamline your geometric transformation workflows by switching to the `pose2extr` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Objects
`se3` | `rigidtform3d`

Functions
`extr2pose` | `estimateExtrinsics` | `estrelpose` | `estworldpose`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

ransac

Fit model to noisy data

Syntax

```
[model,inlierIdx] = ransac(data,fitFcn,distFcn,sampleSize,maxDistance)
[___] = ransac(___,Name,Value)
```

Description

`[model,inlierIdx] = ransac(data,fitFcn,distFcn,sampleSize,maxDistance)` fits a model to noisy data using the M-estimator sample consensus (MSAC) algorithm, a version of the random sample consensus (RANSAC) algorithm.

Specify your function for fitting a model, `fitFcn`, and your function for calculating distances from the model to your data, `distFcn`. The `ransac` function takes random samples from your `data` using `sampleSize` and uses the fit function to maximize the number of inliers within `maxDistance`.

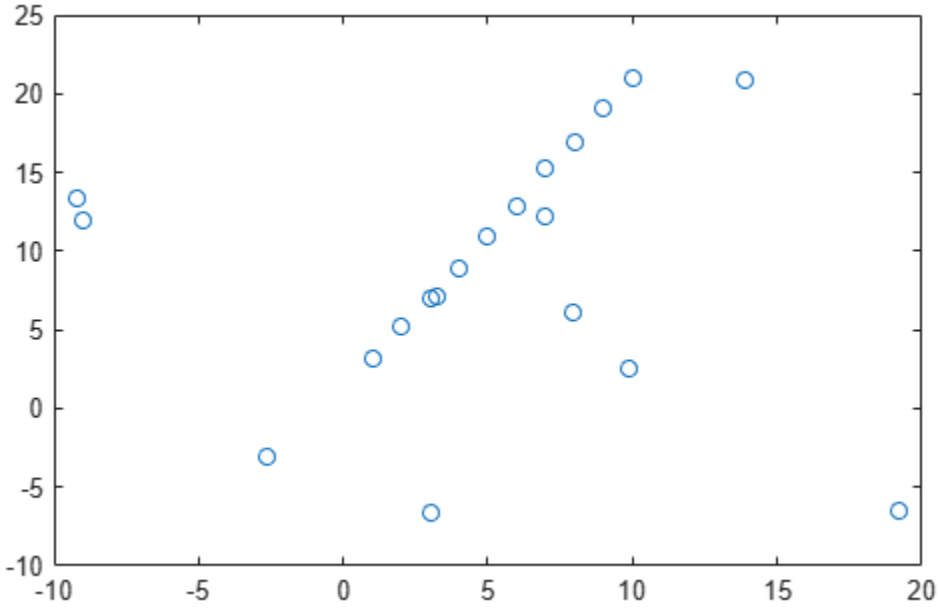
`[___] = ransac(___,Name,Value)` additionally specifies one or more `Name,Value` pair arguments.

Examples

Fit Line to 2-D Points Using Least Squares and RANSAC Algorithms

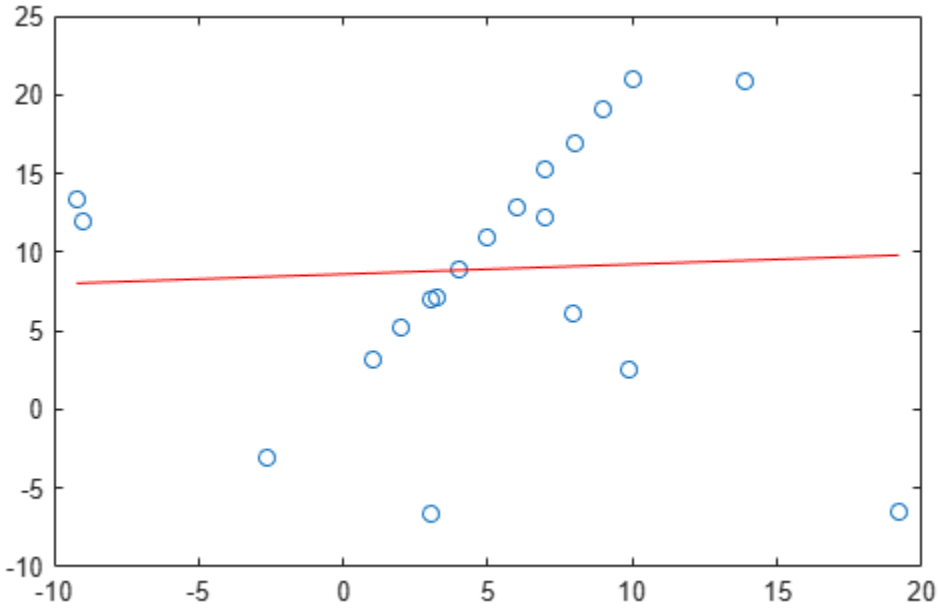
Load and plot a set of noisy 2-D points.

```
load pointsForLineFitting.mat
plot(points(:,1),points(:,2),'o');
hold on
```



Fit a line using linear least squares. Due to outliers, the line is not a good fit.

```
modelLeastSquares = polyfit(points(:,1),points(:,2),1);
x = [min(points(:,1)) max(points(:,1))];
y = modelLeastSquares(1)*x + modelLeastSquares(2);
plot(x,y,'r-')
```



Fit a line to the points using the MSAC algorithm. Define the sample size, the maximum distance for inliers, the fit function, and the distance evaluation function. Call `ransac` to run the MSAC algorithm.

```
sampleSize = 2; % number of points to sample per trial
maxDistance = 2; % max allowable distance for inliers

% Define the fit function using polyfit.
fitLineFcn = @(points)polyfit(points(:,1),points(:,2),1);

% Define the distance function to classify each point as an inlier or outlier
% based on the maxDistance threshold.
distLineFcn = @(model,points)(points(:, 2) - polyval(model, points(:,1))).^2;

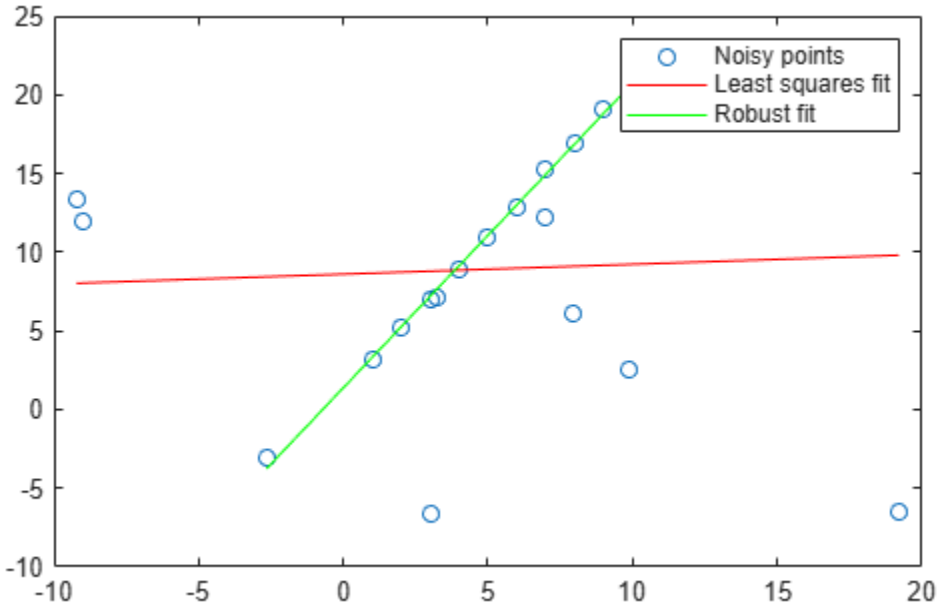
[modelRANSAC,inlierIdx] = ransac(points,fitLineFcn,distLineFcn, ...
  sampleSize,maxDistance);
```

Refit a line to the inliers using `polyfit`.

```
modelInliers = polyfit(points(inlierIdx,1),points(inlierIdx,2),1);
```

Display the final fit line. This line is robust to the outliers that `ransac` identified and ignored.

```
inlierPts = points(inlierIdx,:);
x = [min(inlierPts(:,1)) max(inlierPts(:,1))];
y = modelInliers(1)*x + modelInliers(2);
plot(x, y, 'g-')
legend('Noisy points','Least squares fit','Robust fit');
hold off
```



Input Arguments

data — Data to be modeled

m-by-*n* matrix

Data to be modeled, specified as an *m*-by-*n* matrix. Each row corresponds to a data point in the set to be modeled. For example, to model a set of 2-D points, specify the point data as an *m*-by-*2* matrix.

Data Types: `single` | `double`

`fitFcn` — Function to fit a subset of `data`
function handle

Function to fit a subset of `data`, specified as a function handle. The function must be of the form:

```
model = fitFcn(data)
```

If it is possible to fit multiple models to the data, then `fitFcn` returns the model parameters as a cell array.

`distFcn` — Function to compute distances from model
function handle

Function to compute distances from the model to the data, specified as a function handle. The function must be of the form:

```
distances = distFcn(model,data)
```

If `model` is an *n*-element array, then distances must be an *m*-by-*n* matrix. Otherwise, `distances` must be an *m*-by-1 vector.

`sampleSize` — Minimum sample size
positive scalar integer

Minimum sample size from `data` that is required by `fitFcn`, specified as a positive scalar integer.

`maxDistance` — Maximum distance for inlier points
positive scalar

Maximum distance from the fit curve to an inlier point, specified as a positive scalar. Any points further away than this distance are considered outliers. The distance is defined by `distFcn`. The RANSAC algorithm creates a fit from a small sample of points, but tries to maximize the number of inlier points. Lowering the maximum distance improves the fit by putting a tighter tolerance on inlier points.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MaxNumTrials',2000`

`ValidateModelFcn` — Function to validate model
function handle

Function to validate model, specified as the comma-separated pair consisting of '`ValidateModelFcn`' and a function handle. The function returns `true` if the model is accepted

based on criteria defined in the function. Use this function to reject specific fits. The function must be of the form:

```
isValid = validateModelFcn(model,varargin)
```

If no function is specified, all models are assumed to be valid.

`MaxSamplingAttempts` — Maximum number of sampling attempts
100 (default) | integer

Maximum number of attempts to find a sample that yields a valid model, specified as the comma-separated pair consisting of '`MaxSamplingAttempts`' and an integer.

`MaxNumTrials` — Maximum number of random trials
`1000` (default) | integer

Maximum number of random trials, specified as the comma-separated pair consisting of '`MaxNumTrials`' and an integer. A single trial uses a minimum number of random points from `data` to fit a model. Then, the trial checks the number of inliers within the `maxDistance` from the model. After all trials, the model with the highest number of inliers is selected. Increasing the number of trials improves the robustness of the output at the expense of additional computation.

`Confidence` — Confidence of final solution
`99` (default) | scalar from 0 to 100

Confidence that the final solution finds the maximum number of inliers for the model fit, specified as the comma-separated pair consisting of '`Confidence`' and a scalar from 0 to 100. Increasing this value improves the robustness of the output at the expense of additional computation.

Output Arguments

`model` — Best fit model
parameters defined in `fitFcn`

Best fit model, returned as the parameters defined in the `fitFcn` input. This model maximizes the number of inliers from all the sample attempts.

`inlierIdx` — Inlier points
logical vector

Inlier points, returned as a logical vector. The vector is the same length as `data`, and each element indicates if that point is an inlier for the model fit based on `maxDistance`.

Version History

Introduced in R2017a

References

[1] Torr, P. H. S., and A. Zisserman. "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry." *Computer Vision and Image Understanding*. Vol. 18, Issue 1, April 2000, pp. 138–156.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`fitPolynomialRANSAC` | `polyfit` | `polyval`

rdivide, ./

Element-wise transformation or rotation right division

Syntax

```
transformationC = transformationA./transformationB
rotationC = rotationA./rotationB
```

Description

`transformationC = transformationA./transformationB` divides transformations element-by-element by dividing each element of transformation `transformationA` with the corresponding element of transformation `transformationB` and returns the quotient, transformation `transformationC`.

`rotationC = rotationA./rotationB` divides rotations element-by-element by dividing each element of rotation `rotationA` with the corresponding element of rotation `rotationB` and returns the quotient, rotation `rotationC`.

Input Arguments

`transformationA` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationA` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`transformationB` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationB` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`rotationA` — First rotation

`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationA` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `so3` objects, `rotationB` must be a scalar `so3` object.

`rotationB` — Last rotation
`so2` object | `so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationB` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `se3` objects, `rotationB` must be a scalar `se3` object.

Output Arguments

`transformationC` — Transformation quotient
`se3` object | *N*-element array of transformation objects

Transformation quotient, returned as a scalar `se3` object or as an *N*-element array of the same transformation type as `transformationA` and `transformationB`. *N* is the length of the longer argument between `transformationA` and `transformationB` and each row represents the quotient between `transformationA` and `transformationB`.

`rotationC` — Rotation quotient
`so3` object | *N*-element array of rotation objects

Rotation quotient, returned as a scalar `so3` object or as an *N*-element array of the same rotation type as `rotationA` and `rotationB`. *N* is the length of the longer argument between `rotationA` and `rotationB` and each row represents the quotient between `rotationA` and `rotationB`.

See Also

Functions
`mrdivide, /` | `mtimes, *` | `times, .*`

Objects
`se3` | `so3`

reconstructScene

Reconstruct 3-D scene from disparity map

Syntax

```
xyzPoints = reconstructScene(disparityMap,reprojectionMatrix)
```

Description

`xyzPoints = reconstructScene(disparityMap,reprojectionMatrix)` returns an array of 3-D world point coordinates that reconstruct a scene from a disparity map. The 3-D world coordinates are relative to the optical center of camera 1 in a stereo system. You can use the `rectifyStereoImages` function to obtain the `reprojectionMatrix` from a pair of stereo images.

Examples

Reconstruct 3-D Scene from Stereo Image Pair Using Semi-Global Matching

Reconstruct a 3-D scene from a pair of stereo images by using the disparity map computed through semi-global matching.

Load the stereo parameters.

```
load('webcamsSceneReconstruction.mat');
```

Read in the stereo pair of images.

```
I1 = imread('sceneReconstructionLeft.jpg');
I2 = imread('sceneReconstructionRight.jpg');
```

Rectify the images.

```
[J1, J2, reprojectionMatrix] = rectifyStereoImages(I1,I2,stereoParams);
```

Display the images after rectification.

```
figure
imshow(cat(3,J1(:,:,1),J2(:,:,2:3)));
```



Compute the disparity.

```
disparityMap = disparitySGM(im2gray(J1),im2gray(J2));
figure
imshow(disparityMap,[0,64]);
```



Reconstruct the 3-D world coordinates of points corresponding to each pixel from the disparity map.

```
xyzPoints = reconstructScene(disparityMap,reprojectionMatrix);
```

Segment out a person located between 3.2 and 3.7 meters away from the camera.

```
Z = xyzPoints(:,:,3);
mask = repmat(Z > 3200 & Z < 3700,[1,1,3]);
J1(~mask) = 0;
imshow(J1);
```



Input Arguments

`disparityMap` — Disparity image

2-*D* array

Disparity image, specified as a 2-D array of disparity values for pixels in image 1 of a stereo pair. You can use either `disparityBM` or `disparitySGM` functions to generate the disparity image.

The disparity image can contain invalid values marked as `NaN`. These values correspond to pixels in image 1 that did not match with image 2. The function sets the world coordinates corresponding to invalid disparity value to `NaN`.

Pixels with zero disparity correspond to world points that are too far away to measure, given the resolution of the camera. The function sets the world coordinates corresponding to zero disparity to `Inf`.

When you specify the `disparityMap` input as a `double`, the function returns the coordinates as `double`. Otherwise, the function returns the coordinates as `single`.

Data Types: `single` | `double`

`reprojectionMatrix` — Reprojection matrix

4-by-4 matrix

Reprojection matrix, specified as a 4-by-4 matrix. You can use the `rectifyStereoImages` function to obtain the `reprojectionMatrix` from a pair of stereo images.

The reprojection matrix is represented as:

$$\begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/b & 0 \end{bmatrix}$$

where f and $[cx,cy]$ are the focal length and principal point of the rectified camera 1, respectively. b is the baseline of the virtual rectified stereo camera.

Data Types: `single` | `double`

Output Arguments

`xyzPoints` — Coordinates of world points

M-by-*N*-by-3 array

Coordinates of world points, returned as an *M*-by-*N*-by-3 array. The 3-D world coordinates are relative to the optical center of camera 1 in the stereo system represented by `stereoParams`.

The output array contains the $[x, y, z]$ coordinates of world points that correspond to the pixels in the `disparityMap` input. `xyzPoints`(:, :, 1) contains the x world coordinates of points corresponding to the pixels in the disparity map. `xyzPoints`(:, :, 2) contains the y world coordinates, and `xyzPoints`(:, :, 3) contains the z world coordinates. The 3-D world coordinates are relative to the optical center of camera 1 in the stereo system.

When you specify the `disparityMap` input as `double`, the function returns the `xyzPoints` output as `double`. Otherwise, the function returns it as `single`.

Data Types: `single` | `double`

Version History

Introduced in R2014a

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*, Sebastopol, CA: O'Reilly, 2008.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Functions

`disparitySGM` | `disparityBM` | `estimateCameraParameters` | `size` | `rectifyStereoImages` | `lineToBorderPoints`

Objects

`cameraParameters` | `stereoParameters`

Topics

“Structure from Motion from Two Views”
“Code Generation for Depth Estimation from Stereo Video”
“Coordinate Systems”

rectifyStereoImages

Rectify pair of stereo images

Syntax

```
[J1,J2,reprojectionMatrix] = rectifyStereoImages(I1,I2,stereoParams)
[___,camMatrix1,camMatrix2,R1,R2] = rectifyStereoImages(I1,I2,stereoParams)
[J1,J2] = rectifyStereoImages(I1,I2,tform1,tform2)

[J1,J2] = rectifyStereoImages(___,interp)
[J1,J2] = rectifyStereoImages(___,Name=Value)
```

Description

`[J1,J2,reprojectionMatrix] = rectifyStereoImages(I1,I2,stereoParams)` undistorts and rectifies versions of `I1` and `I2` input images using the stereo parameters of a stereo camera system stored in the `stereoParams` object. Use the `reconstructScene` function with the `reprojectionMatrix` to reproject a 2-D point in a disparity map to a 3-D point in the rectified camera coordinate system of camera 1.

Stereo image rectification projects images onto a common image plane in such a way that the corresponding points have the same row coordinates. This image projection makes the image appear as though the two cameras are parallel. Use the `disparityBM` or `disparitySGM` functions to compute a disparity map from the rectified images for 3-D scene reconstruction.

`[___,camMatrix1,camMatrix2,R1,R2] = rectifyStereoImages(I1,I2,stereoParams)` returns the 3-by-4 camera projection matrices `camMatrix1` and `camMatrix2` for the rectified cameras, and the corresponding rectification rotation matrices, `R1` and `R2`.

`[J1,J2] = rectifyStereoImages(I1,I2,tform1,tform2)` returns rectified versions of `I1` and `I2` input images by applying projective transformations `tform1` and `tform2`.

`[J1,J2] = rectifyStereoImages(___,interp)` additionally specifies the interpolation method to use for rectified images. You can specify the method as `"nearest"`, `"linear"`, or `"cubic"`.

`[J1,J2] = rectifyStereoImages(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `OutputView="valid"` sets the `OutputView` argument to `"valid"`.

Examples

Rectify Stereo Images

Specify images containing a checkerboard for calibration.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = detectCheckerboardPoints(...
    leftImages.Files,rightImages.Files);
```

Specify world coordinates of checkerboard keypoints.

```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Read in the images.

```
I1 = readimage(leftImages,1);
I2 = readimage(rightImages,1);
imageSize = [size(I1,1),size(I1,2)];
```

Calibrate the stereo camera system.

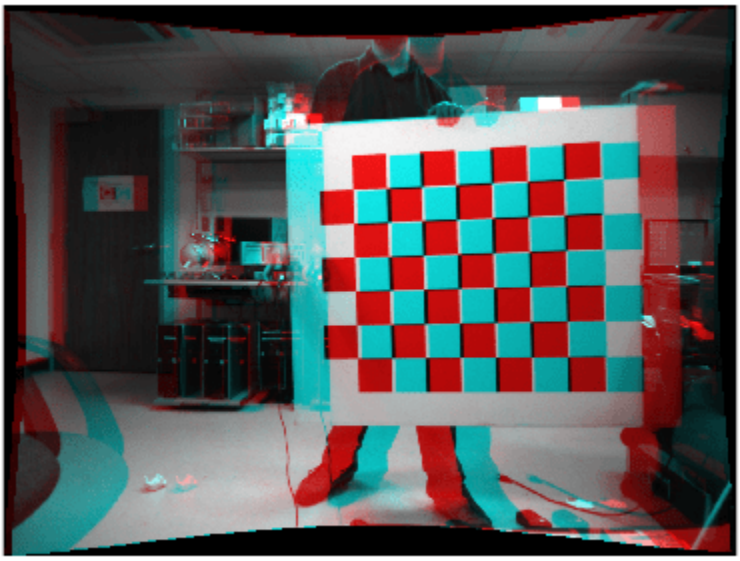
```
stereoParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Rectify the images using "full" output view.

```
[J1_full,J2_full] = rectifyStereoImages(I1,I2,stereoParams,OutputView="full");
```

Display the result for "full" output view.

```
figure;
imshow(stereoAnaglyph(J1_full,J2_full));
```

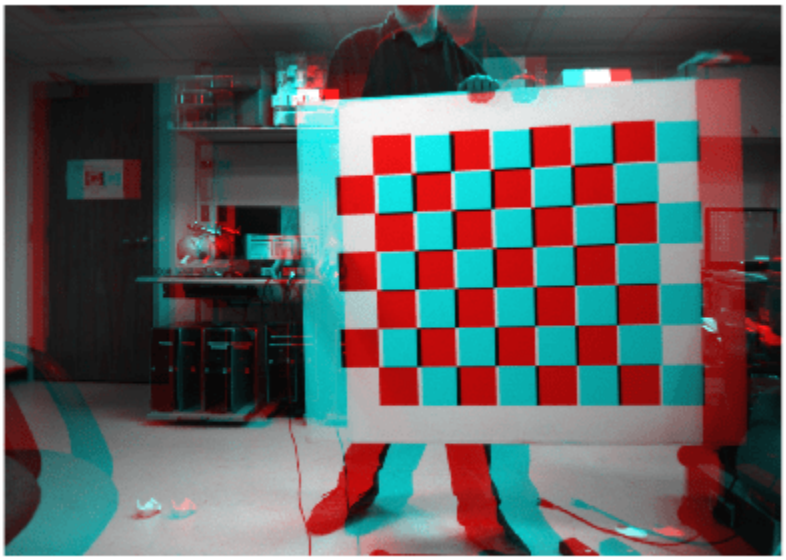


Rectify the images using "valid" output view. This is most suitable for computing disparity.

```
[J1_valid,J2_valid] = rectifyStereoImages(I1,I2,stereoParams,OutputView="valid");
```

Display the result for "valid" output view.

```
figure;
imshow(stereoAnaglyph(J1_valid,J2_valid));
```



Input Arguments

`I1` — Input image 1

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image corresponding to camera 1, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-D grayscale array. Input images `I1` and `I2` must also be real, finite, and nonsparse. The input images must be the same class.

Data Types: `uint8` | `uint16` | `int16` | `single` | `double` | `logical`

`I2` — Input image 2

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D truecolor image

Input image corresponding to camera 2, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-D grayscale array. Input images `I1` and `I2` must be real, finite, and nonsparse. The input images must also be the same class.

Data Types: `uint8` | `uint16` | `int16` | `single` | `double` | `logical`

`stereoParams` — Stereo camera system parameters

`stereoParameters` object

Stereo camera system parameters, specified as a `stereoParameters` object.

Data Types: `uint8` | `uint16` | `int16` | `single` | `double`

`tform1` — Projective transformation

`projtform2d` object

Projective transformations for image 1, specified as a `projtform2d` object. You can get a `projtform2d` object using the `estimateStereoRectification` function.

Data Types: `single` | `double`

`tform2` — Projective transformation
`projtform2d` object

Projective transformations for image 2, specified as a `projtform2d` object. You can get a `projtform2d` object using the `estimateStereoRectification` function.

Data Types: `single` | `double`

`interp` — Interpolation method
`"linear"` (default) | `"nearest"` | `"cubic"`

Interpolation method, specified as either `"linear"`, `"nearest"`, or `"cubic"`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `OutputView="valid"` sets the `OutputView` argument to `"valid"`.

`OutputView` — Size of rectified images
`"valid"` (default) | `"full"`

Size of rectified images, specified as `"full"` or `"valid"`. When you set this parameter to `"full"`, the rectified images include all pixels from the original images. When you set this value to `"valid"`, the output images are cropped to the size of the largest common rectangle containing valid pixels.

When there is no overlap between rectified images, set `OutputView` to `"full"`.

`FillValues` — Output pixel fill values
numeric scalar | three-element numeric vector

Output pixel fill values, specified as numeric scalar or three-element numeric vector. When the corresponding inverse-transformed location in the input image is completely outside the input image boundaries, use the fill values for output pixels. If `I1` and `I2` are 2-D grayscale images, then you must set `FillValues` to a numeric scalar. If `I1` and `I2` are truecolor images, then you can set `FillValues` to a scalar or a three-element vector of RGB values.

Output Arguments

`J1` — Undistorted and rectified image 1
M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Undistorted and rectified version of `I1`, returned as an *M*-by-*N*-by-3 truecolor image or as an *M*-by-*N* 2-D grayscale image.

Stereo image rectification projects images onto a common image plane in such a way that the corresponding points have the same row coordinates. This image projection makes the image appear

as though the two cameras are parallel. Use the `disparityBM` or `disparitySGM` functions to compute a disparity map from the rectified images for 3-D scene reconstruction.

`J2` — Undistorted and rectified image 2

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Undistorted and rectified version of `I2`, returned as an *M*-by-*N*-by-3 truecolor image or as an *M*-by-*N* 2-D grayscale image.

Stereo image rectification projects images onto a common image plane in such a way that the corresponding points have the same row coordinates. This image projection makes the image appear as though the two cameras are parallel. Use the `disparityBM` or `disparitySGM` functions to compute a disparity map from the rectified images for 3-D scene reconstruction.

`reprojectionMatrix` — Reprojection matrix

4-by-4 matrix

Reprojection matrix, returned as a 4-by-4 matrix of the form:

$$\begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/b & 0 \end{bmatrix},$$

where f and $[cx,cy]$ are the focal length and principal point of rectified camera 1, respectively. b is the baseline of the virtual rectified stereo camera.

Use the `reconstructScene` function with the `reprojectionMatrix` to reproject a 2-D point in a disparity map to a 3-D point in the rectified camera coordinate system of camera 1.

`camMatrix1` — Camera one rectified projection matrix

3-by-4 matrix

Rectified camera one projection matrix, returned as 3-by-4 matrix. Use `camMatrix1` and `camMatrix2` to project 3-D world points in camera one's coordinate system into the image plane of `J1` and `J2`, respectively.

Use `camMatrix1` to project 3-D world points in the rectified camera one coordinate system into the image plane of `J1`.

Data Types: `single` | `double`

`camMatrix2` — Camera two rectified projection matrix

3-by-4 matrix

Camera two rectified projection matrix, returned as 3-by-4 matrix. Use `camMatrix1` and `camMatrix2` to project 3-D world points in camera one's coordinate system into the image plane of `J1` and `J2`, respectively.

Use `camMatrix2` to project 3-D world points in the rectified camera two coordinate system into the image plane of `J2`.

Data Types: `single` | `double`

`R1` — Camera one rotation matrix

3-by-3 matrix

Camera one rotation matrix related to camera one rectified projection, returned as a 3-by-3 matrix. The `R1` rotation matrix relate 3-D points from the unrectified camera one coordinate system to points in the rectified camera one coordinate system.

Data Types: `single` | `double`

R2 — Camera two rotation matrix
3-by-3 matrix

Camera two rotation matrix related to camera one rectified projection, returned as a 3-by-3 matrix. The `R2` rotation matrix relate 3-D points from the unrectified camera two coordinate system to points in the rectified camera two coordinate system.

Data Types: `single` | `double`

Tips

- The Computer Vision Toolbox rectification algorithm requires that the epipole for each image lie outside of the image. If the epipole lies within the image, you can first transform the images into polar coordinates as described in the rectification method proposed by Marc Pollefeys, Reinhard Koch, and Luc Van Gool [2] on page 3-782.

Version History

Introduced in R2014a

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `tform1` and `tform2` as `projtform2d` objects, which use the premultiply convention.

Although you can still specify `tform1` and `tform2` as `projective2d` objects or 3-by-3 matrices returned by the `estimateUncalibratedRectification` function, these values are not recommended because they use the postmultiply convention. You can streamline your geometric transformation workflows by switching to the `projtform2d` objects. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV : Computer Vision with the OpenCV Library*. Sebastopol, CA: O'Reilly, 2008.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `'interp'` and `'OutputView'` must be compile-time constants.

- To generate code, use the `toStruct` function to pass `stereoParameters` to the `rectifyStereoImages` function.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`reconstructScene` | `disparityBM` | `disparitySGM` | `estimateCameraParameters` | `estimateStereoRectification`

Objects
`stereoParameters`

Topics
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Stereo Visual Simultaneous Localization and Mapping”
“Uncalibrated Stereo Image Rectification”
“Code Generation for Depth Estimation from Stereo Video”
“Coordinate Systems”

retrieveImages

Search image set for similar image

Syntax

```
imageIDs = retrieveImages(queryImage,imageIndex)
[imageIDs,scores] = retrieveImages(queryImage,imageIndex)
[imageIDs,scores,imageWords] = retrieveImages(queryImage,imageIndex)
[imageIDs, ___ ] = retrieveImages(queryImage,imageIndex,Name,Value)
```

Description

`imageIDs = retrieveImages(queryImage,imageIndex)` returns the image identifiers `imageIDs` that correspond to images within `imageIndex` that are visually similar to the query image. The `imageIDs` are returned in ranked order, from the most to least similar match.

`[imageIDs,scores] = retrieveImages(queryImage,imageIndex)` optionally returns the similarity scores used to rank the image retrieval results. The `scores` output contains the corresponding scores from 0 to 1.

`[imageIDs,scores,imageWords] = retrieveImages(queryImage,imageIndex)` optionally returns the visual words in `queryImage` that are used to search for similar images.

`[imageIDs, ___ ] = retrieveImages(queryImage,imageIndex,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments, using any of the preceding syntaxes.

Examples

Search Image Set Using Query Image

Create an image set of book covers.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata','bookCovers');
bookCovers = imageDatastore(dataDir);
```

Display the data set.

```
thumbnailGallery = [];
for i = 1:length(bookCovers.Files)
    I = readimage(bookCovers,i);
    thumbnail = imresize(I,[300 300]);
    thumbnailGallery = cat(4,thumbnailGallery,thumbnail);
end

figure
montage(thumbnailGallery);
```



Index the image set. This step may take a few minutes.

```
imageIndex = indexImages(bookCovers);
```

```
Creating an inverted image index using Bag-Of-Features.
-------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 58 images...done. Extracted 29216 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 23373.
** Using the strongest 23373 features from each of the other image categories.

* Creating a 20000 word visual vocabulary.
* Number of levels: 1
* Branching factor: 20000
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 23373
* Number of clusters          : 20000
* Initializing cluster centers...100.00%.
* Clustering...completed 5/100 iterations (~1.23 seconds/iteration)...converged in 5 iterations.
```

```
* Finished creating Bag-Of-Features


Encoding images using Bag-Of-Features.
--------------------------------------

* Encoding 58 images...done.
Finished creating the image index.
```

Select and display the query image.

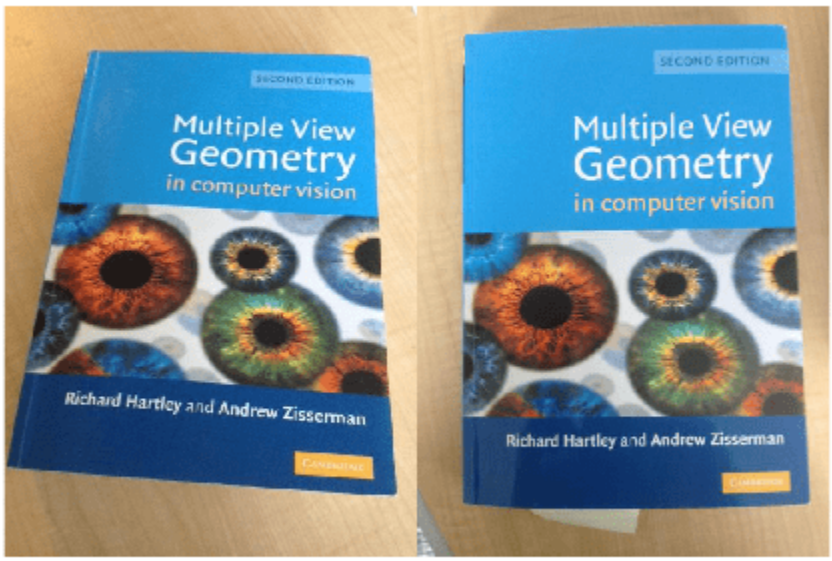
```
queryDir = fullfile(dataDir,'queries',filesep);
queryImage = imread([queryDir 'query3.jpg']);

imageIDs = retrieveImages(queryImage,imageIndex);
```

Show the query image and its best match, side-by-side.

```
bestMatch = imageIDs(1);
bestImage = imread(imageIndex.ImageLocation{bestMatch});

figure
imshowpair(queryImage,bestImage,'montage')
```



Search Image Set for Specific Object Using ROIs

Search an image set for an object using a region of interest (ROI) for the query image.

Define a set of images to search.

```
imageFiles = ...
  {'elephant.jpg', 'cameraman.tif', ...
```

```
  'peppers.png',  'saturn.png',...
  'pears.png',    'stapleRemover.jpg', ...
  'football.jpg', 'mandi.tif',...
  'kids.tif',     'liftingbody.png', ...
  'office_5.jpg', 'gantrycrane.png',...
  'moon.tif',     'circuit.tif', ...
  'tape.png',     'coins.png'};

imds = imageDatastore(imageFiles);
```

Create a search index.

```
 imageIndex = indexImages(imds);

Creating an inverted image index using Bag-Of-Features.
---------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 16 images...done. Extracted 3680 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 2944.
** Using the strongest 2944 features from each of the other image categories.

* Creating a 2944 word visual vocabulary.
* Number of levels: 1
* Branching factor: 2944
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 2944
* Number of clusters          : 2944
* Initializing cluster centers...100.00%.
* Clustering...completed 1/100 iterations (~0.13 seconds/iteration)...converged in 1 iterations.

* Finished creating Bag-Of-Features


Encoding images using Bag-Of-Features.
---------------------------------------

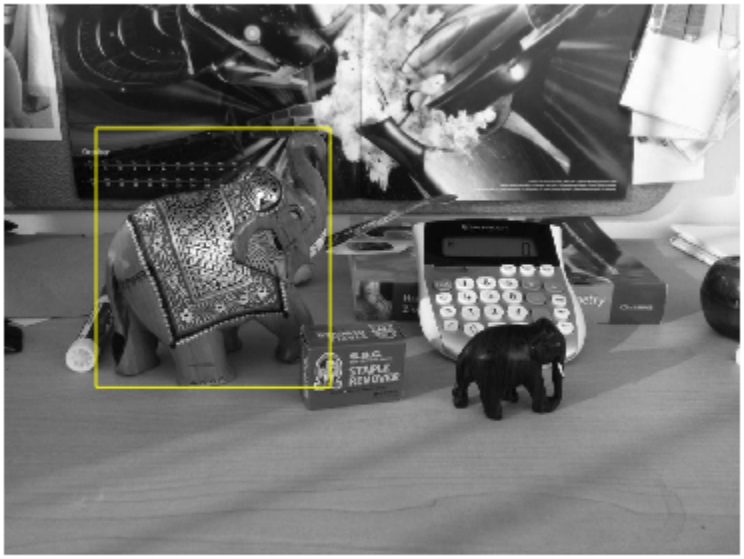
* Encoding 16 images...done.
Finished creating the image index.
```

Specify a query image and an ROI. The ROI outlines the object, an elephant, for the search.

```
queryImage = imread('clutteredDesk.jpg');
queryROI = [130 175 330 365];

figure
```

```
imshow(queryImage)
rectangle('Position',queryROI,'EdgeColor','yellow')
```



You can also use the `imrect` function to select an ROI interactively. For example, `queryROI = getPosition(imrect)`

Find images that contain the object.

```
imageIDs = retrieveImages(queryImage,imageIndex,'ROI',queryROI)
```

```
imageIDs = 12×1 uint32 column vector

    1
   11
    6
   12
    2
    3
    8
    5
   14
   13
   10
   16
```

Display the best match.

```
bestMatch = imageIDs(1);

figure
imshow(imageIndex.ImageLocation{bestMatch})
```



Geometric Verification Using `estgeotform2d` Function

Use the locations of visual words to verify the best search result. To rerank the search results based on geometric information, repeat this procedure for the top *N* search results.

Specify the location of the images.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","bookCovers");
bookCovers = imageDatastore(dataDir);
```

Index the image set. This process can take a few minutes.

```
imageIndex = indexImages(bookCovers);
```

```
Creating an inverted image index using Bag-Of-Features.
-------------------------------------------------------

Creating Bag-Of-Features.
-------------------------

* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 58 images...done. Extracted 29216 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 1 has the least number of strongest features: 23373.
** Using the strongest 23373 features from each of the other image categories.
```

```
* Creating a 20000 word visual vocabulary.
* Number of levels: 1
* Branching factor: 20000
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 23373
* Number of clusters          : 20000
* Initializing cluster centers...100.00%.
* Clustering...completed 5/100 iterations (~1.18 seconds/iteration)...converged in 5 iterations.

* Finished creating Bag-Of-Features

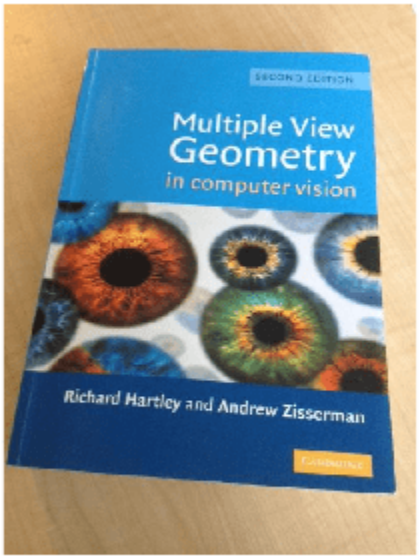

Encoding images using Bag-Of-Features.
---------------------------------------

* Encoding 58 images...done.
Finished creating the image index.
```

Select and display the query image.

```
queryDir = fullfile(dataDir,"queries",filesep);
queryImage = imread(queryDir+"query3.jpg");

figure
imshow(queryImage)
```



Retrieve the best matches. The `queryWords` output contains visual word locations information for the query image. Use this information to verify the search results.

```
[imageIDs, ~, queryWords] = retrieveImages(queryImage,imageIndex);
```

Find the best match for the query image by extracting the visual words from the image index. The image index contains the visual word information for all images in the index.

```
bestMatch = imageIDs(1);
bestImage = imread(imageIndex.ImageLocation{bestMatch});
bestMatchWords = imageIndex.ImageWords(bestMatch);
```

Generate a set of tentative matches based on visual word assignments. Each visual word in the query can have multiple matches due to the hard quantization used to assign visual words.

```
queryWordsIndex     = queryWords.WordIndex;
bestMatchWordIndex  = bestMatchWords.WordIndex;

tentativeMatches = [];
for i = 1:numel(queryWords.WordIndex)

    idx = find(queryWordsIndex(i) == bestMatchWordIndex);

    matches = [repmat(i, numel(idx), 1) idx];

    tentativeMatches = [tentativeMatches; matches];

end
```

Show the point locations for the tentative matches. There are many poor matches.

```
points1 = queryWords.Location(tentativeMatches(:,1),:);
points2 = bestMatchWords.Location(tentativeMatches(:,2),:);

figure
showMatchedFeatures(queryImage,bestImage,points1,points2,"montage")
```



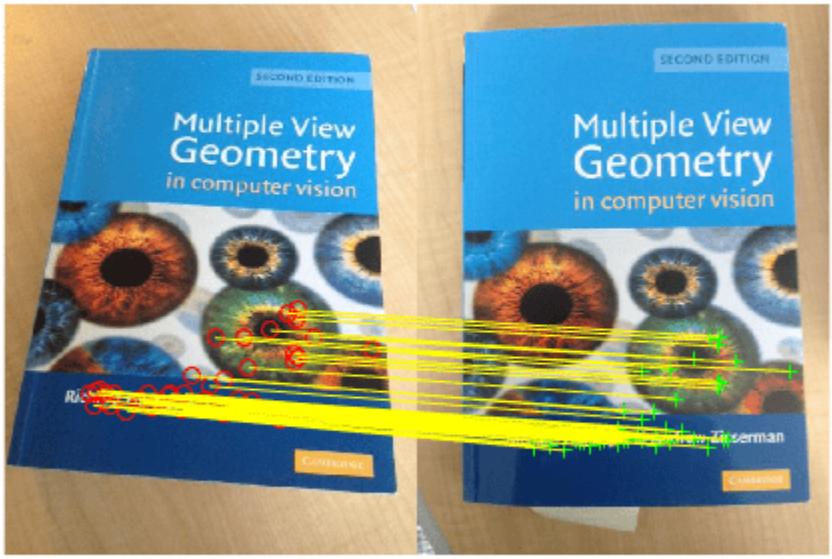
Remove poor visual word assignments using `estgeotform2d` function. Keep the assignments that fit a valid geometric transform.

```
[tform,inlierIdx] = estgeotform2d(points1,points2,"affine",MaxNumTrial=2000);
inlierPoints1 = points1(inlierIdx, :);
inlierPoints2 = points2(inlierIdx, :);
```

Rerank the search results by the percentage of inliers. Do this when the geometric verification procedure is applied to the top *N* search results. Those images with a higher percentage of inliers are more likely to be relevant.

```
percentageOfInliers = size(inlierPoints1,1)./size(points1,1);

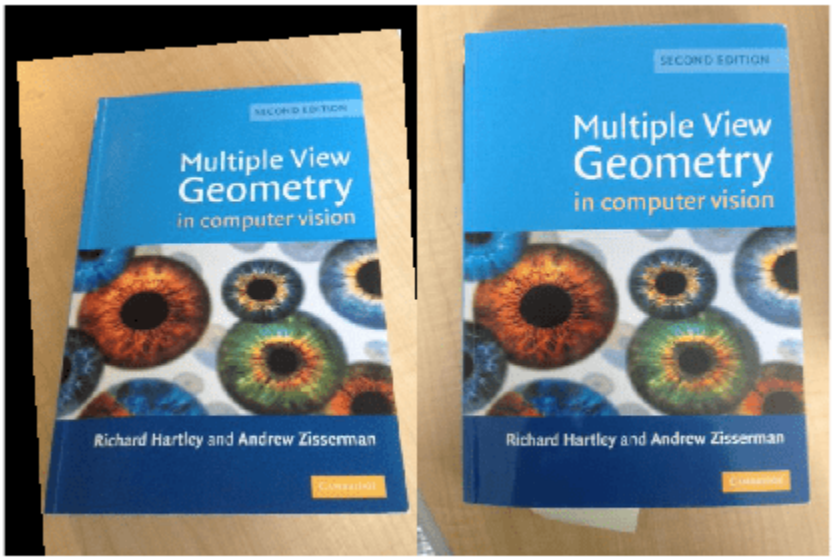
figure
showMatchedFeatures(queryImage,bestImage,inlierPoints1,...
    inlierPoints2,"montage")
```



Apply the estimated transform.

```
outputView = imref2d(size(bestImage));
Ir = imwarp(queryImage, tform, OutputView=outputView);

figure
imshowpair(Ir,bestImage,"montage")
```



Modify Search Parameters for Image Search

Use the `evaluateImageRetrieval` function to help select proper search parameters.

Create an image set.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets','cups');
imds = imageDatastore(setDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');
```

Index the image set.

```
 imageIndex = indexImages(imds,'Verbose',false);
```

Tune image search parameters.

```
imageIndex.MatchThreshold = 0.2;
imageIndex.WordFrequencyRange = [0 1]
```

```
imageIndex =
  invertedImageIndex with properties:

         ImageLocation: {6×1 cell}
            ImageWords: [6×1 vision.internal.visualWords]
         WordFrequency: [0.1667 0.1667 0.1667 0.3333 0.1667 0.1667 0.1667 0.5000 0.3333 0.1667 0
         BagOfFeatures: [1×1 bagOfFeatures]
               ImageID: [1 2 3 4 5 6]
        MatchThreshold: 0.2000
    WordFrequencyRange: [0 1]
```

```
queryImage = readimage(imds, 1);
indices = retrieveImages(queryImage,imageIndex);
```

Input Arguments

`queryImage` — Input query image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input query image, specified as either an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-D grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`imageIndex` — Image search index

`invertedImageIndex` object

Image search index, specified as an `invertedImageIndex` object. The `indexImages` function creates the `invertedImageIndex` object, which stores the data used for the image search.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumResults'`,25 sets the `'NumResults'` property to 25

`NumResults` — Maximum number of results

`20` (default) | numeric value

Maximum number of results to return, specified as the comma-separated pair consisting of '`NumResults`' and a numeric value. Set this value to `Inf` to return as many matching images as possible.

`ROI` — Query image search region

`[1 1 size(queryImage,2) size(queryImage,1)]` (default) | [*x y width height*] vector

Query image search region, specified as the comma-separated pair consisting of '`ROI`' and an [*x y width height*] vector.

`Metric` — Similarity metric

`'cosine'` (default) | `'L1'`

Similarity metric used to rank the image retrieval results, specified as `'cosine'` or `'L1'` [3] on page 3-1144.

Output Arguments

`imageIDs` — Ranked index of retrieved images

M-by-1 vector

Ranked index of retrieved images, returned as an *M*-by-1 vector. The image IDs are returned in ranked order, from the most to least similar matched image.

`scores` — Similarity metric
N-by-1 vector

Similarity metric, returned as an *N*-by-1 vector. This output contains the scores that correspond to the retrieved images in the `imageIDs` output. The scores are computed using the `Metric` property and a range from 0 to 1.

`imageWords` — Object for storing visual word assignments
`visualWords` object

Object for storing visual word assignments, returned as a `visualWords` object. The object stores the visual word assignments of `queryImage` and their locations within that image.

Version History

Introduced in R2015a

References

[1] Sivic, J. and A. Zisserman. *Video Google: A text retrieval approach to object matching in videos*. ICCV (2003) pg 1470-1477.

[2] Philbin, J., O. Chum, M. Isard, J. Sivic, and A. Zisserman. *Object retrieval with large vocabularies and fast spatial matching*. CVPR (2007).

[3] Gálvez-López, Dorian, and Juan D. Tardos. *Bags of binary words for fast place recognition in image sequences.* IEEE Transactions on Robotics 28.5 (2012): 1188-1197.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`evaluateImageRetrieval` | `imageDatastore` | `imageSet` | `bagOfFeatures` | `invertedImageIndex`

Topics
“Image Retrieval Using Customized Bag of Features”
“Image Retrieval with Bag of Visual Words”

roialign

Non-quantized ROI pooling of `dlarray` data

Syntax

```
dlY = roialign(dlX,boxes,outputSize)
dlY = roialign(dlX,boxes,outputSize,Name=Value)
```

Description

The ROI align operation pools a rectangular ROI into fixed sized bins without quantizing the grid points to the nearest pixel. The function uses bilinear interpolation to infer the value at each grid point.

Given input data of size [*H W C N*], where *C* is the number of channels and *N* is the number of observations, the pooled deep learning data has size [*h w C* `sum`(*M*)], where *h* and *w* are the specified output size. *M* is a vector of length *N* and *M*(*i*) is the number of ROIs associated with the *i*-th observation.

Note To perform ROI pooling within a `Layer` array, use `roiAlignLayer`.

This function requires Deep Learning Toolbox.

`dlY = roialign(dlX,boxes,outputSize)` performs a pooling operation along the spatial dimensions of the input `X` for each bounding box in `boxes`. The outputs, `Y`, are of size `outputSize`.

`dlY = roialign(dlX,boxes,outputSize,Name=Value)` specifies additional name-value arguments.

Examples

Perform ROI Pooling

Create a 4-D formatted `dlarray` object that simulates a batch of two RGB images.

```
X = dlarray(rand(10,10,3,2),"SSCB");
```

Specify the position and batch index of one bounding box.

```
startXY = [2 2];
endXY = [4 4];
batchIdx = 1;
rois = [startXY endXY batchIdx]';
```

Perform ROI pooling with an output size of 3-by-3.

```
Y = roialign(X,rois,[3 3])
```

```
Y =
  3(S) × 3(S) × 3(C) × 1(B) single dlarray


(:,:,1) =

    0.7464    0.3069    0.1780
    0.9212    0.8491    0.4677
    0.7303    0.9057    0.3840


(:,:,2) =

    0.3024    0.6428    0.6594
    0.1542    0.0046    0.1228
    0.6295    0.5182    0.3304


(:,:,3) =

    0.4915    0.7590    0.5035
    0.4574    0.4302    0.5453
    0.2960    0.2666    0.5389
```

Input Arguments

`dlX` — Deep learning data to pool

4-D formatted `dlarray` object

Deep learning data to pool, specified as a 4-D formatted `dlarray` object with a data format of "SSCB".

`boxes` — Bounding boxes

5-by-*N* numeric matrix

Bounding boxes, specified as a 5-by-*N* numeric matrix, where *N* is the number of bounding boxes. Each bounding box is formatted as a column vector of the form [*x_start*; *y_start*; *x_end*; *y_end*; *batchIdx*], where:

- *x_start* and *y_start* specify the (x,y) coordinates of the upper-left corner of the rectangle.
- *x_end* and *y_end* specify the (x,y) coordinates of the bottom-right corner of the rectangle.
- *batchIdx* specifies the index of the observation corresponding to the rectangle.

By default, `boxes` are in the same coordinate space and scale as the input deep learning data `dlX`.

`outputSize` — Pooled output size

vector of two positive integers

Pooled output size, specified as a vector of two positive integers `[h w]`, where `h` is the height and `w` is the width.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dlY = roialign(dlX,boxes,outputSize,ROIScale=2)` scales the input ROIs by a factor of 2

`ROIScale` — Ratio of scale of input feature map to ROI coordinates
`1` (default) | numeric scalar

Ratio of the scale of the input feature map to that of the ROI coordinates. This ratio specifies the factor used to scale input ROIs to the input feature map size.

`SamplingRatio` — Number of samples in each pooled bin
`"auto"` (default) | row vector of two positive integers

Number of samples in each pooled bin, specified as `"auto"` or a row vector of two positive integers. The two elements are the number of vertical and horizontal samples, respectively.

If you do not specify the sampling ratio, then the number of vertical samples has the default value `ceil(`*`roiHeight/outputHeight`*`)`. Likewise, the number of horizontal samples has the default value `ceil(`*`roiWidth/outputWidth`*`)`.

Data Types: `double` | `char`

Output Arguments

`dlY` — Pooled deep learning data
4-D formatted `dlarray` object

Pooled deep learning data, returned as a 4-D formatted `dlarray` object with a data format of "SSCB".

More About

ROI Align

An ROI align operation returns fixed size feature maps for every rectangular ROI within an input `dlarray`. The function first partitions an ROI into fixed sized bins of size `OutputSize` without quantizing the grid points. Each bin is further sampled at `SamplingRatio` locations. The value at each sampled point is inferred using bilinear interpolation. The average of the sampled values is returned as the output value of each pooled bin.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

When you specify a custom range of values for the `SamplingRatio` argument, you will observe a slight numerical discrepancy between the MATLAB simulation results and the generated code.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

See Also

Objects
`dlarray`

Functions
`roiAlignLayer`

Topics
“Get Started with Object Detection Using Deep Learning”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Functions with dlarray Support” (Deep Learning Toolbox)

rotationMatrixToVector

(Not recommended) Convert 3-D rotation matrix to rotation vector

Note `rotationMatrixToVector` is not recommended. Use the `rotmat2vec3d` function instead. For more information, see “Version History”.

Syntax

```
rotationVector = rotationMatrixToVector(rotationMatrix)
```

Description

`rotationVector = rotationMatrixToVector(rotationMatrix)` returns an axis-angle rotation vector that corresponds to the input 3-D rotation matrix. The function uses the Rodrigues formula for the conversion.

Examples

Convert Rotation Matrix to Rotation Vector

Create a matrix representing a 90-degree rotation about the *Z*-axis.

```
rotationMatrix = [0 1 0; -1 0 0; 0 0 1];
```

Find the equivalent rotation vector.

```
rotationVector = rotationMatrixToVector(rotationMatrix)
```

```
rotationVector = 1×3

         0         0    1.5708
```

Input Arguments

`rotationMatrix` — Rotation of camera

3-by-3 matrix

Rotation of camera, specified as a 3-by-3 matrix. You can obtain this matrix by using the `estimateExtrinsics` function.

Output Arguments

`rotationVector` — Rotation vector

three-element vector

Rotation vector, returned as a three-element vector. The vector represents the axis of rotation in 3-D, where the magnitude corresponds to the rotation angle in radians.

Data Types: `single` | `double`

Version History

Introduced in R2016a

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `rotationMatrixToVector` function uses the postmultiply convention. Although there are no plans to remove `rotationMatrixToVector` at this time, you can streamline your geometric transformation workflows by switching to the `rotmat2vec3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `rotationMatrixToVector` to `rotmat2vec3d`.
- Specify the rotation matrix as the transpose of the `rotationMatrix` argument of the `rotationMatrixToVector` function. Alternatively, you can get a rotation matrix in the correct premultiply format by using the `estimateExtrinsics` function and querying the `R` property of the returned `rigidtform3d` object.

The table shows examples of how to update your code.

Discouraged Usage	Recommended Replacement

This example converts a 3-D rotation matrix in the postmultiply convention to a rotation vector using the `rotationMatrixToVector` function.
 	
`rotVectorOld = rotationMatrixToVector(rotMatrixOld);`
	This example converts a 3-D rotation matrix in the premultiply convention to a rotation vector using the `rotmat2vec2d` function. The 3-D rotation matrix is the transpose of `rotMatrixOld`.
	
	`rotMatrixNew = rotMatrixOld';`
	`rotVectorNew = rotmat2vec2d(rotMatrixNew);`

This example obtains a 3-D rotation matrix in the postmultiply convention using the `extrinsics` function, then converts the matrix to a rotation vector using the `rotationMatrixToVector` function.
 	
`[rotMatrixOld,transVector] = extrinsics( ...`
`    imagePoints,worldPoints,intrinsics);`
`rotVectorOld = rotationMatrixToVector(rotMatrixOld);`
	This example obtains a 3-D rotation matrix in the premultiply convention using the `estimateExtrinsics` function, then converts the matrix to a rotation vector using the `rotmat2vec2d` function.
	
	`camExtrinsics = estimateExtrinsics( ...`
	`    imagePoints,worldPoints,intrinsics);`
	`rotMatrixNew = camExtrinsics.R;`
	`rotVectorNew = rotmat2vec2d(rotMatrixNew);`

References

[1] Trucco, Emanuele, and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 1998.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`rotmat2vec3d` | `rotvec2mat3d` | `estrelpose` | `triangulate` | `estimateExtrinsics`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

rotationVectorToMatrix

(Not recommended) Convert 3-D rotation vector to rotation matrix

Note `rotationVectorToMatrix` is not recommended. Use the `rotvec2mat3d` function instead. For more information, see “Version History”.

Syntax

```
rotationMatrix = rotationVectorToMatrix(rotationVector)
```

Description

`rotationMatrix = rotationVectorToMatrix(rotationVector)` returns a 3-D rotation matrix that corresponds to the input axis-angle rotation vector. The function uses the Rodrigues formula for the computation.

Examples

Convert Rotation Vector to Rotation Matrix

Create a vector representing a 90-degree rotation about the *Z*-axis.

```
rotationVector = pi/2 * [0, 0, 1];
```

Find the equivalent rotation matrix.

```
rotationMatrix = rotationVectorToMatrix(rotationVector)
```

```
rotationMatrix = 3×3

    0.0000    1.0000         0
   -1.0000    0.0000         0
         0         0    1.0000
```

Input Arguments

`rotationVector` — Rotation vector

three-element vector

Rotation vector, specified as a three-element vector. The vector represents the axis of rotation in 3-D, where the magnitude corresponds to the rotation angle in radians.

Data Types: `single` | `double`

Output Arguments

`rotationMatrix` — Rotation of camera
3-by-3 matrix

Rotation of camera, returned as a 3-by-3 matrix that corresponds to the input axis-angle rotation vector.

Version History

Introduced in R2016a

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `rotationVectorToMatrix` function uses the postmultiply convention. Although there are no plans to remove `rotationVectorToMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `rotvec2mat3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `rotationVectorToMatrix` to `rotvec2mat3d`.
- You do not need to change the input argument, `rotationVector`. The `rotvec2mat3d` function automatically returns the rotation matrix using the premultiply convention.

References

[1] Trucco, Emanuele, and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 1998.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`rotvec2mat3d` | `rotmat2vec3d` | `estrelpose` | `triangulate` | `estimateExtrinsics`

Topics
“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

rotm

Extract rotation matrix

Syntax

```
rotationMatrix = rotm(transformation)
rotationMatrix = rotm(rotation)
```

Description

`rotationMatrix = rotm(transformation)` returns the rotation matrix `rotationMatrix` from the SE(3) transformation `transformation`.

`rotationMatrix = rotm(rotation)` returns the rotation matrix `rotationMatrix` from the SO(2) or SO(3) rotation `rotation`.

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of transformation objects

Transformation, specified as a scalar `se3` object or an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformation` as an array, each element must be of the same type.

`rotation` — Rotation

`so3` object | *N*-element array of rotation objects

Rotation, specified as a scalar `so3` object, or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotation` as an array, each element must be of the same type.

Output Arguments

`rotationMatrix` — Rotation matrix

2-by-2-by-*N* array | 3-by-3-by-*N* array

Rotation matrix, returned as a 2-by-2-by-*N* array for 2-D transformations or a 3-by-3-by-*N* array for 3-D transformations. *N* is the total number of transformations.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`tform` | `trvec`

Objects
`se3` | `so3`

rotmat2vec3d

Convert 3-D rotation matrix to rotation vector

Syntax

```
rotationVector = rotmat2vec3d(rotationMatrix)
```

Description

`rotationVector = rotmat2vec3d(rotationMatrix)` returns an axis-angle rotation vector that corresponds to the input 3-D rotation matrix. The function uses the Rodrigues formula for the conversion.

Examples

Convert Rotation Matrix to Rotation Vector

Create a matrix representing a 90-degree rotation about the *Z-*axis.

```
rotationMatrix = [0 -1 0; 1 0 0; 0 0 1];
```

Find the equivalent rotation vector.

```
rotationVector = rotmat2vec3d(rotationMatrix)
```

```
rotationVector = 1×3

         0         0    1.5708
```

Input Arguments

`rotationMatrix` — Rotation of camera

3-by-3 matrix

Rotation of camera, specified as a 3-by-3 matrix. You can obtain this matrix by using the `estimateExtrinsics` function.

Output Arguments

`rotationVector` — Rotation vector

three-element vector

Rotation vector, returned as a three-element vector. The vector represents the axis of rotation in 3-D, where the magnitude corresponds to the rotation angle in radians.

Data Types: `single` | `double`

Version History

Introduced in R2022b

R2022b: Recommended over `rotationMatrixToVector`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `rotationMatrixToVector` function uses the postmultiply convention. Although there are no plans to remove `rotationMatrixToVector` at this time, you can streamline your geometric transformation workflows by switching to the `rotmat2vec3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Trucco, Emanuele, and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 1998.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`rotvec2mat3d` | `estrelpose` | `estworldpose` | `estimateExtrinsics` | `cameraParameters`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

rotvec2mat3d

Convert 3-D rotation vector to rotation matrix

Syntax

```
rotationMatrix = rotvec2mat3d(rotationVector)
```

Description

`rotationMatrix = rotvec2mat3d(rotationVector)` returns a 3-D rotation matrix that corresponds to the input axis-angle rotation vector. The function uses the Rodrigues formula for the computation.

Examples

Convert Rotation Vector to Rotation Matrix

Create a vector representing a 90-degree rotation about the *Z-*axis.

```
rotationVector = pi/2 * [0 0 1];
```

Find the equivalent rotation matrix.

```
rotationMatrix = rotvec2mat3d(rotationVector)
```

```
rotationMatrix = 3×3

    0.0000   -1.0000         0
    1.0000    0.0000         0
         0         0    1.0000
```

Input Arguments

`rotationVector` — Rotation vector

three-element vector

Rotation vector, specified as a three-element vector. The vector represents the axis of rotation in 3-D, where the magnitude corresponds to the rotation angle in radians.

Data Types: `single` | `double`

Output Arguments

`rotationMatrix` — Rotation of camera

3-by-3 matrix

Rotation of camera, returned as a 3-by-3 matrix that corresponds to the input axis-angle rotation vector.

Version History

Introduced in R2022b

R2022b: Recommended over `rotationVectorToMatrix`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `rotationVectorToMatrix` function uses the postmultiply convention. Although there are no plans to remove `rotationVectorToMatrix` at this time, you can streamline your geometric transformation workflows by switching to the `rotvec2mat3d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Trucco, Emanuele, and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 1998.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`rotmat2vec3d` | `estrelpose` | `estworldpose` | `estimateExtrinsics` | `cameraParameters`

Topics

“Evaluating the Accuracy of Single Camera Calibration”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

sceneLabelTrainingData

Create training data for scene classification from ground truth

Syntax

```
imds = sceneLabelTrainingData(gTruth)
[imds,groupInfo] = sceneLabelTrainingData(gTruth)
```

Description

`imds = sceneLabelTrainingData(gTruth)` creates image datastore `imds` from the specified ground truth.

Note This function only supports ground truth exported by the Image Labeler app.

`[imds,groupInfo] = sceneLabelTrainingData(gTruth)` also returns information about the groups of scene labels.

Input Arguments

`gTruth` — Ground truth data

`groundTruth` object

Ground truth data, specified as a `groundTruth` object.

Output Arguments

`imds` — Collection of images

`ImageDatastore` object

Collection of images extracted from the ground truth object `gTruth`, returned as an `ImageDatastore` object. The `Labels` property of the datastore has a scalar valued scene label corresponding to each image.

`groupInfo` — Information about scene label groups

structure

Information about the scene label groups, returned as a structure. The names of the fields describe scene label groups. The values in the structure are string arrays that indicate which scene label names belong to the corresponding group.

Version History

Introduced in R2022b

See Also

`pixelLabelTrainingData` | `trainFCDDAnomalyDetector`

segmentationConfusionMatrix

Confusion matrix of multi-class pixel-level image segmentation

Syntax

```
confusionMatrix = segmentationConfusionMatrix(LPred,LTruth)
```

Description

`confusionMatrix = segmentationConfusionMatrix(LPred,LTruth)` computes a confusion matrix from the predicted pixel labels `LPred` and ground truth pixel labels `LTruth`.

Examples

Calculate Confusion Matrix from Labeled Images

Load a pretrained network that performs binary segmentation of triangles against a background.

```
load("triangleSegmentationNetwork");
```

The `triangleImages` data set has 100 test images with ground truth labels. Define the location of the data set.

```
dataSetDir = fullfile(toolboxdir("vision"),"visiondata","triangleImages");
```

Get one test image and the corresponding ground truth label.

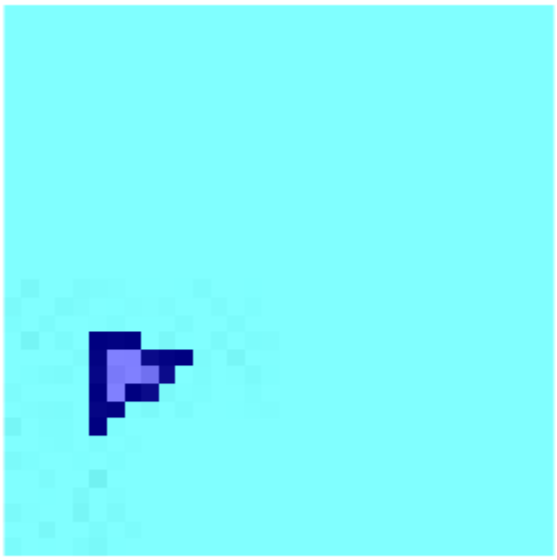
```
testImage = imread(fullfile(dataSetDir,"testImages","image_001.jpg"));
gtLabel   = imread(fullfile(dataSetDir,"testLabels","labeled_image_001.png"));
```

The label image is stored as a numeric image, so convert the ground truth label to a categorical image.

```
classNames = ["triangle" "background"];
labelIDs   = [255 0];
gtLabel = categorical(gtLabel,labelIDs,classNames);
```

Display the ground truth labels over the test image.

```
imshow(labeloverlay(testImage,gtLabel))
```



Perform semantic image segmentation.

```
predLabel = semanticseg(testImage,net,Classes=classNames);
```

Get the confusion matrix of the predicted labels and ground truth labels. The confusion matrix shows the segmentation correctly classified 21 pixels as triangle and 939 pixels as background. The confusion matrix also shows that the segmentation misclassified 64 background pixels as triangle. No triangle pixels are misclassified as background.

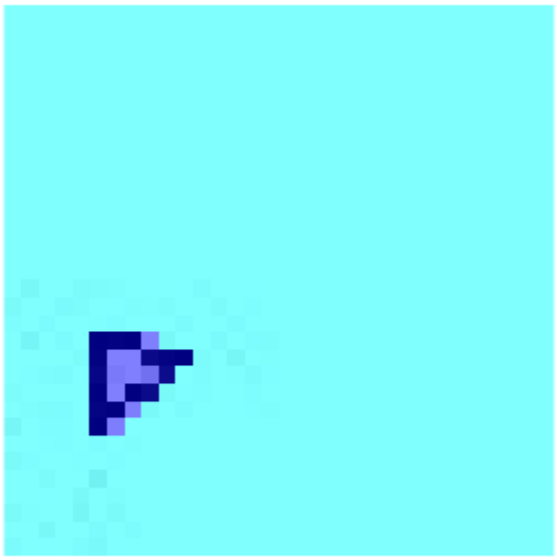
```
confusionMatrix = segmentationConfusionMatrix(predLabel,gtLabel)
```

```
confusionMatrix = 2×2

          21           0
           3        1000
```

To check the accuracy of the confusion matrix, display the predicted labels over the test image. The overlay agrees with the values in the confusion matrix.

```
imshow(labeloverlay(testImage,predLabel))
```



Input Arguments

`LPred` — Predicted pixel labels

2-D label image | 3-D label image

Predicted pixel labels after semantic segmentation, specified as a 2-D label image or 3-D label image.

Data Types: `double` | `logical` | `categorical`

`LTruth` — Ground truth pixel labels

2-D label image | 3-D label image

Ground truth pixel labels, specified as a 2-D label image or 3-D label image of the same size as the predicted pixel labels, `LPred`.

Data Types: `double` | `logical` | `categorical`

Output Arguments

`confusionMatrix` — Confusion matrix

C-by-*C* numeric matrix

Confusion matrix for the classes in the segmented images, returned as a *C*-by-*C* numeric matrix, where *C* is the number of classes in the semantic segmentation. Element (*i*,*j*) is the count of pixels known to belong to class *i* but predicted to belong to class *j*.

Tips

- You can calculate semantic segmentation metrics such as the Jaccard score and classification accuracy from the confusion matrix by using the `evaluateSemanticSegmentation` function.

- You can use this function in block-based image processing workflows such as `blockedImage`. For example, you can perform semantic segmentation and calculate the confusion matrix for a block in the same call to the `apply` function.

Version History

Introduced in R2020b

See Also

`semanticseg` | `dice` | `jaccard` | `semanticSegmentationMetrics` | `evaluateSemanticSegmentation` | `plotconfusion`

Topics

“Calculate Segmentation Metrics in Block-Based Workflow”
“Get Started with Semantic Segmentation Using Deep Learning”

segmentLidarData

Segment organized 3-D range data into clusters

Syntax

```
labels = segmentLidarData(ptCloud,distThreshold)
labels = segmentLidarData(ptCloud,distThreshold,angleThreshold)
[labels,numClusters] = segmentLidarData( ___ )
[ ___ ] = segmentLidarData( ___ ,NumClusterPoints=[1,Inf])
```

Description

`labels = segmentLidarData(ptCloud,distThreshold)` segments organized 3-D range data `ptCloud` into clusters. The function assigns an integer cluster label to each point in the point cloud and returns the cluster label of all points in `labels`.

The function groups two neighboring points into the same cluster if their Euclidean distance is less than `distThreshold` or if the angle between the sensor and two neighboring points is at least 5 degrees.

`labels = segmentLidarData(ptCloud,distThreshold,angleThreshold)` sets the angle constraint for grouping points into the same cluster to `angleThreshold`.

`[labels,numClusters] = segmentLidarData( ___ )` also returns the number of clusters.

`[ ___ ] = segmentLidarData( ___ ,NumClusterPoints=[1,Inf])` also sets the minimum and maximum number of points in each cluster, specified as a 2-element vector or as a scalar value. When you specify `NumClusterPoints` as a scalar, the maximum number of points in the cluster is unrestricted. The function sets the `labels` to `0` when clusters are outside of the specified range.

Examples

Cluster Organized Synthetic Lidar Data

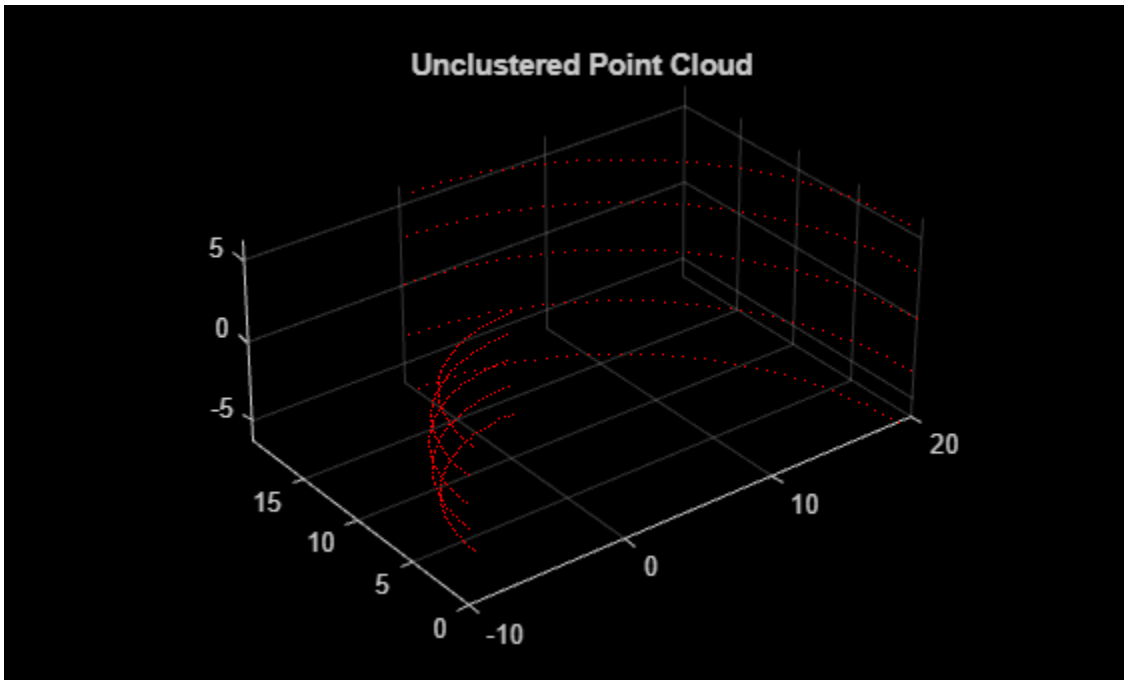
Create organized synthetic lidar data containing two objects.

```
ldr = zeros(5,100);
ldr(:,1:50) = 10;
ldr(:,51:end) = 20;
pitch = linspace(-18,18,5);
pitch = repmat(pitch',1,100);
yaw = linspace(-90,90,100);
yaw = repmat(yaw,5,1);
```

Convert to Cartesian coordinates.

```
X = ldr .* cosd(pitch) .* sind(yaw);
Y = ldr .* cosd(pitch) .* cosd(yaw);
Z = ldr .* sind(pitch);
pc = pointCloud(cat(3,X,Y,Z));
```

```
figure
pcshow(pc.Location,'r')
title('Unclustered Point Cloud')
```



Set the threshold.

```
distThreshold = 5;
```

Segment the lidar data.

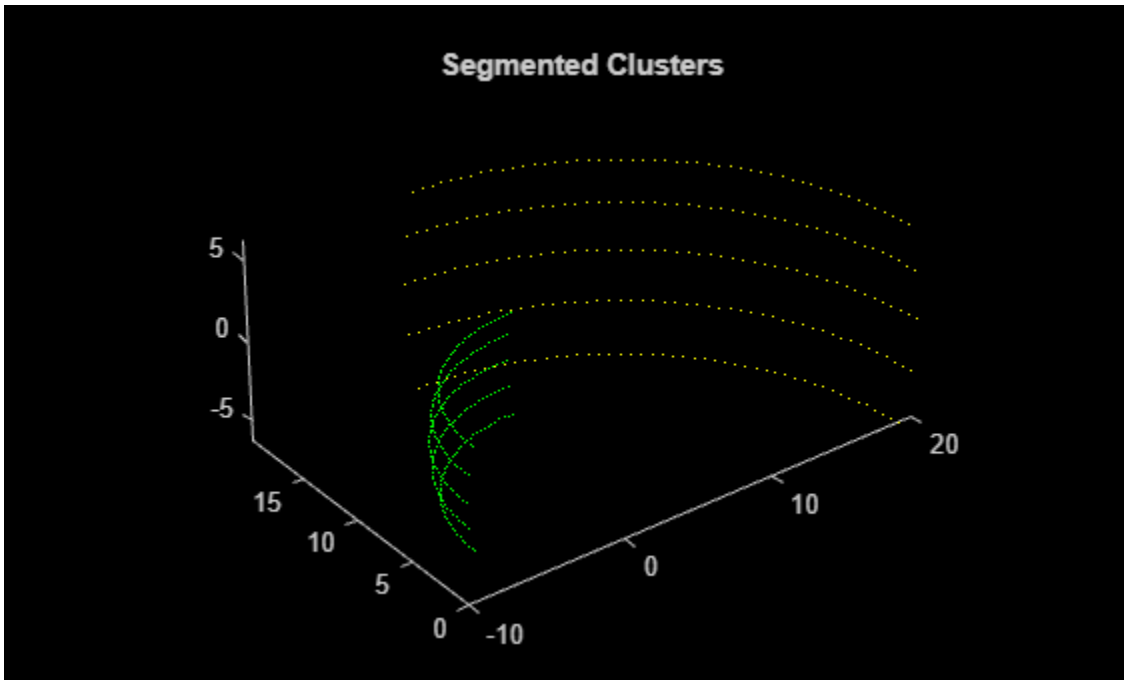
```
labels = segmentLidarData(pc,distThreshold);
```

Plot the results.

```
figure
hold on
title('Segmented Clusters')

pc1 = select(pc,find(labels == 1));
pcshow(pc1.Location,'g')

pc2 = select(pc,find(labels == 2));
pcshow(pc2.Location,'y')
```



Cluster Organized Lidar Point Cloud

Set up the PCAP file reader.

```
veloReader = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Wait for 0.3 seconds from the beginning of the file, then read the point cloud from the next frame.

```
veloReader.CurrentTime = veloReader.StartTime + seconds(0.3);
ptCloud = readFrame(veloReader);
```

Segment and remove the ground plane.

```
groundPtsIdx = segmentGroundFromLidarData(ptCloud);
ptCloudWithoutGround = select(ptCloud,~groundPtsIdx,'OutputSize','full');
```

Cluster the remaining points. Distance is in meters.

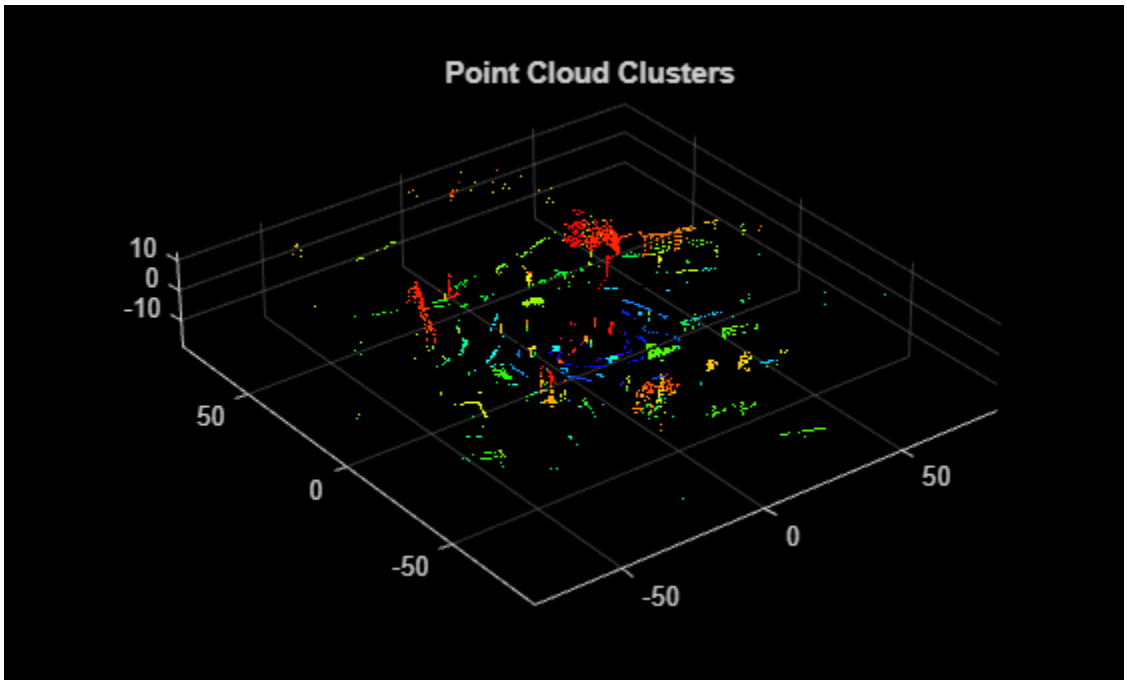
```
distThreshold = 0.5;
[labels,numClusters] = segmentLidarData(ptCloudWithoutGround,distThreshold);
```

Add an additional label for the ground plane.

```
numClusters = numClusters+1;
labels(groundPtsIdx) = numClusters;
```

Plot the labeled results. Display the ground plane in black.

```
labelColorIndex = labels+1;
pcshow(ptCloud.Location,labelColorIndex)
colormap([hsv(numClusters);[0 0 0]])
title('Point Cloud Clusters')
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object. `ptCloud` is an organized point cloud that stores point coordinates in an *M*-by-*N*-by-3 matrix. The points must be organized by pitch and yaw angles in a sequential scanning order, which is typically obtained from laser range finders and the `velodyneFileReader`.

`distThreshold` — Distance threshold

nonnegative scalar

Distance threshold in world units, specified as a nonnegative scalar. Adjacent points are grouped into the same cluster if the distance between them is less than the distance threshold. To reduce the number of output clusters, increase the value of `distThreshold`.

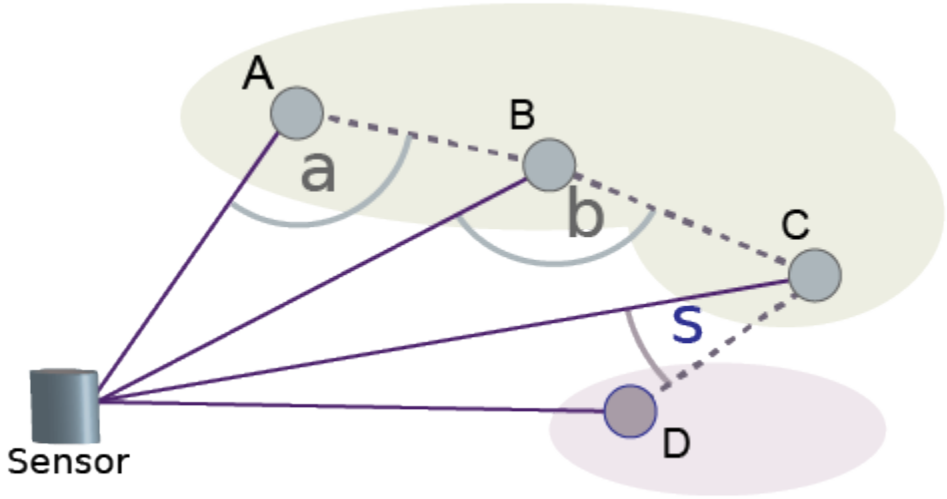
Data Types: `single` | `double`

`angleThreshold` — Angle threshold

scalar in the range [0, 180]

Angle threshold, in degrees, specified as a scalar in the range [0, 180]. To reduce the number of output clusters, decrease the value of `angleThreshold`. The function groups adjacent points into the same cluster if the angle formed by the sensor and the points is greater than the angle threshold. For example, in the figure, the function groups points A and B into the same cluster if the angle formed by the sensor, point A, and point B, is greater than `angleThreshold`. For more details, see

“Algorithms” on page 3-1170.



Data Types: `single` | `double`

Output Arguments

`labels` — Cluster labels

M-by-*N* matrix

Cluster labels of all points in the point cloud `ptCloud`, returned as an *M*-by-*N* matrix of integers. Each valid point in `ptCloud` belongs to a cluster. All points in a cluster are assigned the same integer cluster label, ranging from `1` to `numClusters`. Invalid points, such as points with `Inf` or `NaN` coordinates, are assigned the label `0`.

`numClusters` — Number of clusters

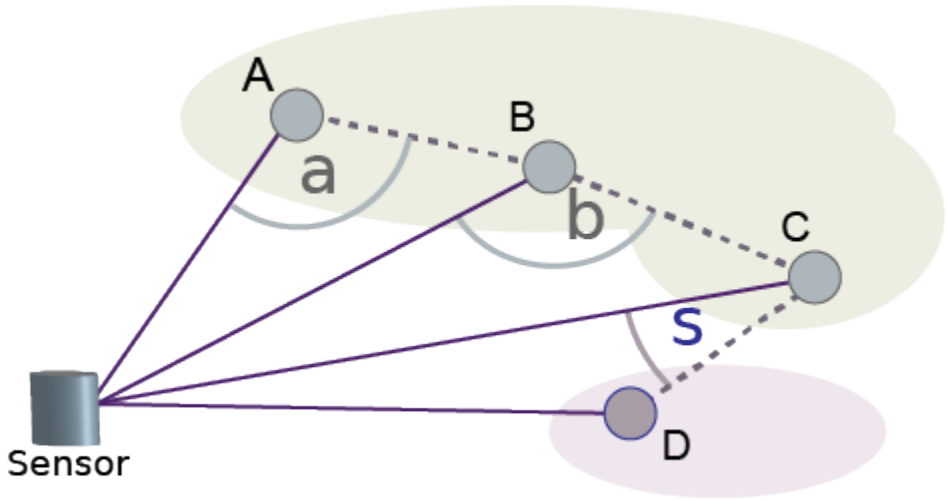
nonnegative integer

Number of clusters, returned as a positive integer. The number of clusters does not include the cluster corresponding to invalid points and excludes the label value, `0`, which is reserved for invalid points.

Algorithms

The `segmentLidarData` function uses distance and angle thresholds to cluster neighboring points. The function groups two neighboring points into the same cluster if their Euclidean distance is less than the input `distThreshold` or if the angle between the sensor and neighboring points is greater than or equal to the input `angleThreshold`. If you do not specify `angleThreshold`, the function sets this angle to 5 degrees.

For example, suppose `angleThreshold` is set to `90`. Because angles α and β in the figure are both greater than the specified threshold of 90 degrees, the function groups points A, B, and C into the same cluster. Because angle σ is less than the 90-degree threshold, the function groups point D into a separate cluster. Each angle the function uses for clustering is formed by the line from a point to the sensor and the line from that same point to the neighboring point



Version History

Introduced in R2018a

References

[1] Bogoslavskyi, I. “Efficient Online Segmentation for Sparse 3D Laser Scans.” *Journal of Photogrammetry, Remote Sensing and Geoinformation Science*. Vol. 85, Issue 1, 2017, pp. 41–52.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`pointCloud` | `pcsegdist` | `pcfitplane` | `velodyneFileReader` | `segmentGroundFromLidarData`

selectStrongestBbox

Select strongest bounding boxes from overlapping clusters using nonmaximal suppression (NMS)

Syntax

```
[selectedBbox,selectedScore] = selectStrongestBbox(bbox,score)
[selectedBbox,selectedScore,index] = selectStrongestBbox(bbox,score)
[___] = selectStrongestBbox(___,Name,Value)
```

Description

`[selectedBbox,selectedScore] = selectStrongestBbox(bbox,score)` returns selected bounding boxes that have a high confidence score. The function uses nonmaximal suppression (NMS) to eliminate overlapping bounding boxes from the `bbox` input.

`[selectedBbox,selectedScore,index] = selectStrongestBbox(bbox,score)` additionally returns the `index` vector associated with `selectedBbox`. This vector contains the indices of the selected boxes in the `bbox` input.

`[___] = selectStrongestBbox(___,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Run Nonmaximal Suppression on Bounding Boxes Using People Detector

Load the pretrained aggregate channel features (ACF) people detector.

```
peopleDetector = peopleDetectorACF();
```

Detect people in an image. Disable the default nonmaximal suppression used by the detector.

```
I = imread('visionteam1.jpg');
[bbox,score] = detect(peopleDetector,I,'SelectStrongest',false);
```

Run nonmaximal suppression with custom threshold.

```
[selectedBbox,selectedScore] = selectStrongestBbox(bbox,score,'OverlapThreshold',0.3);
```

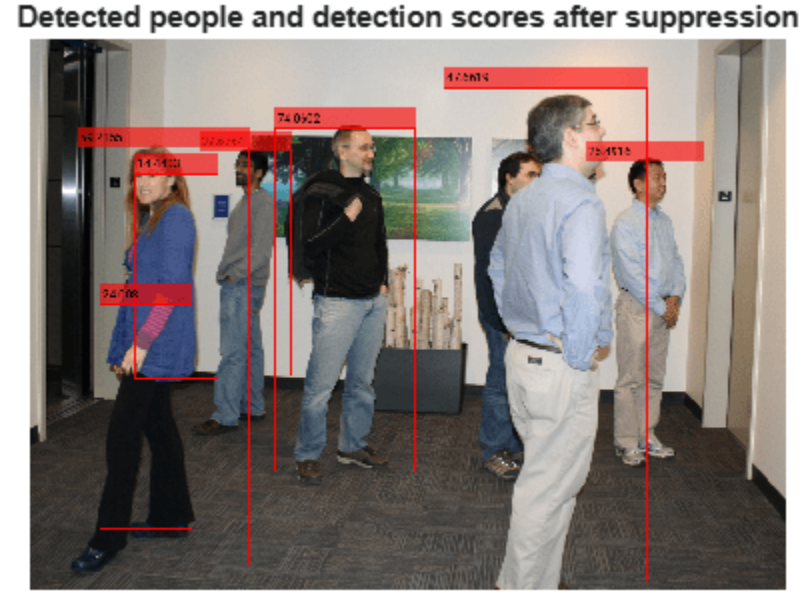
Display the results.

```
I1 = insertObjectAnnotation(I,'rectangle',bbox,score,'AnnotationColor','r');
I2 = insertObjectAnnotation(I,'rectangle',selectedBbox,selectedScore,'AnnotationColor','r');

figure, imshow(I1);
title('Detected people and detection scores before suppression');
```




```
figure, imshow(I2);
title('Detected people and detection scores after suppression');
```



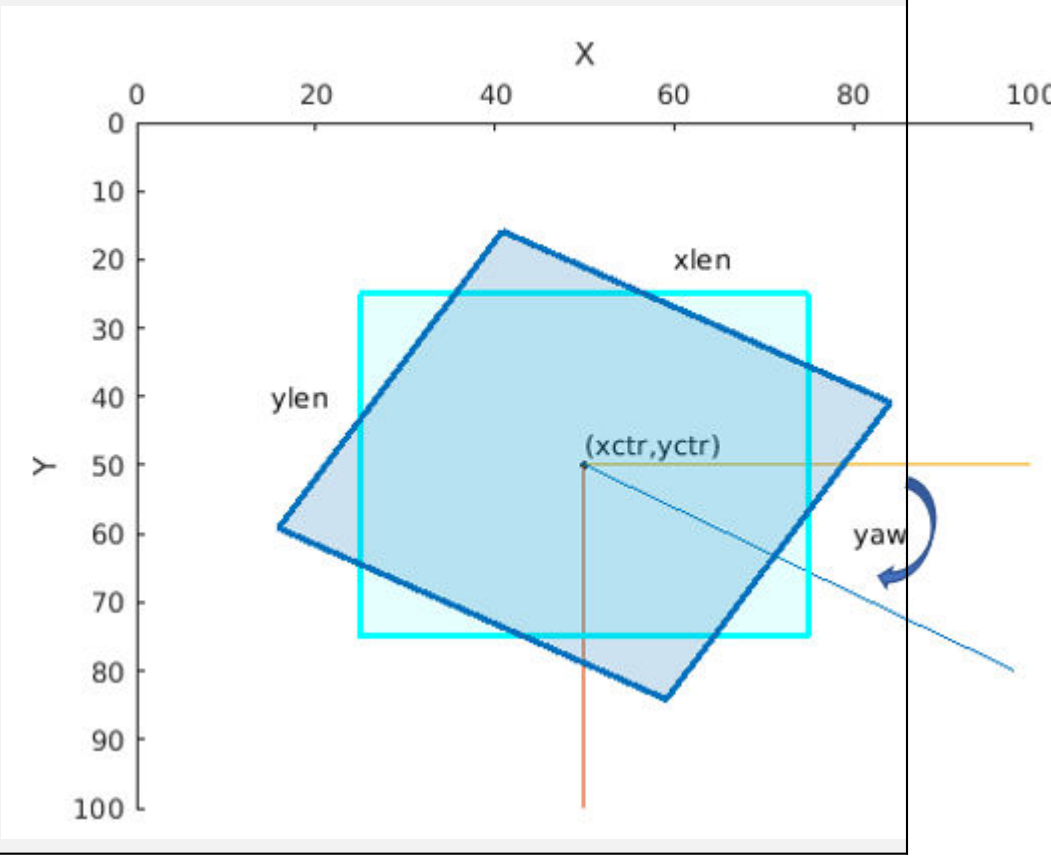
Input Arguments

bbox — Bounding boxes

M-by-4 matrix | *M*-by-5 matrix

Bounding boxes, specified as an *M*-by-4 or *M*-by-5 nonsparse numeric matrix. *M* is the number of bounding boxes. Each row of the matrix defines a bounding box as either an axis-aligned rectangle or a rotated rectangle. This table describes the format for each bounding box.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	Description
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`score` — Confidence score

M-by-1 vector

Confidence score, specified as an *M*-by-1 vector. The *M*th score corresponds to the *M*th bounding box in the `bbox` input. The `selectStrongestBbox` function uses nonmaximal suppression to eliminate overlapping bounding boxes and associate the confidence score with the boxes. A higher score represents a higher confidence in keeping the bounding box. The `score` input must be real, finite, and nonsparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

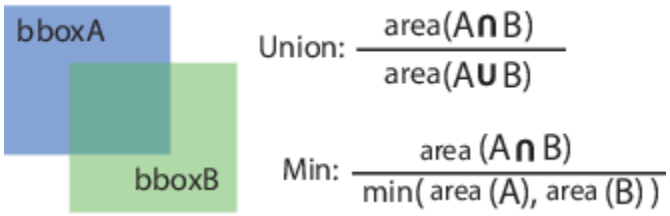
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'RatioType','Union'` sets the `'RatioType'` property to `'Union'`.

`RatioType` — Bounding box overlap ratio denominator
`'Union'` (default) | `'Min'`

Ratio type, specified as the character vector `'Union'` or `'Min'`.

- Set the ratio type to `'Union'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the area of the union of the two.
- Set the ratio type to `'Min'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the minimum area of the two bounding boxes.



Data Types: `char`

`OverlapThreshold` — Overlap ratio threshold
`0.5` (default) | scalar in the range [0 1]

Overlap ratio threshold, specified as the comma-separated pair consisting of '`OverlapThreshold`' and a scalar in the range [0 1]. When the overlap ratio is above the threshold you set, the function removes bounding boxes around the reference box. Decrease this value to reduce the number of selected bounding boxes. However, if you decrease the overlap ratio too much, you might eliminate boxes that represent objects close to each other in the image.

Data Types: `single` | `double`

`NumStrongest` — Maximum number of strongest boxes
`inf` (default) | positive scalar

Maximum number of strongest boxes, specified as the comma-separated pair consisting of `'NumStrongest'` and `inf` or a positive scalar. Use this argument to reduce processing time when you have a priori knowledge about the maximum number of boxes. Set the value to `inf` to select all the strongest, non-overlapping, bounding boxes.

Output Arguments

`selectedBbox` — Selected bounding boxes

M-by-4 matrix | *M*-by-5 matrix

Selected bounding boxes, returned as an *M*-by-4 or an *M*-by-5 matrix. The 4-element vectors represent axis-aligned rectangles and the 5-element vectors represent rotated rectangles.

The `selectedBbox` output returns the selected bounding boxes from the `bbox` input that have the highest confidence score. The function uses nonmaximal suppression to eliminate overlapping bounding boxes.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`selectedScore` — Scores of selected bounding boxes

M-by-1 vector

Scores of selected bounding boxes, returned as an *M*-by-1 vector. The *M*th score in the `selectedScore` output corresponds to the *M*th bounding box in the `selectedBbox` output.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`index` — Index of selected bounding boxes

M-by-1 vector

Index of selected bounding boxes, returned as an *M*-by-1 vector. The `index` vector contains the indices to the selected boxes in the `bbox` input.

Data Types: `double`

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation is only supported for numeric `labels`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- CUDA code generation is only supported for numeric `labels`.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- GPU Arrays is not supported for rotated rectangle bounding box inputs.

See Also

`bboxOverlapRatio` | `selectStrongestBboxMulticlass`

selectStrongestBboxMulticlass

Select strongest multiclass bounding boxes from overlapping clusters using nonmaximal suppression (NMS)

Syntax

```
selectedBboxes = selectStrongestBboxMulticlass(bboxes,scores,labels)
[selectedBboxes,selectedScores,selectedLabels,index] =
selectStrongestBboxMulticlass(bboxes,scores,labels)
[ ___ ] = selectStrongestBboxMulticlass( ___ ,Name,Value)
```

Description

`selectedBboxes = selectStrongestBboxMulticlass(bboxes,scores,labels)` returns selected bounding boxes that have high confidence scores. The function uses greedy nonmaximal suppression (NMS) to eliminate overlapping bounding boxes from the `bboxes` input, only if they have the same class label.

`[selectedBboxes,selectedScores,selectedLabels,index] = selectStrongestBboxMulticlass(bboxes,scores,labels)` additionally returns the scores, labels, and index associated with the selected bounding boxes.

`[ ___ ] = selectStrongestBboxMulticlass( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Run Multiclass Nonmaximal Suppression on Bounding Boxes Using People Detector

Create detectors using two different models. These will be used to generate multiclass detection results.

```
detectorInria = peopleDetectorACF('inria-100x41');
detectorCaltech = peopleDetectorACF('caltech-50x21');
```

Apply the detectors.

```
I = imread('visionteam1.jpg');
[bboxesInria,scoresInria] = detect(detectorInria,I,'SelectStrongest',false);
[bboxesCaltech,scoresCaltech] = detect(detectorCaltech,I,'SelectStrongest',false);
```

Create categorical labels for each the result of each detector.

```
labelsInria = repelem("inria",numel(scoresInria),1);
labelsInria = categorical(labelsInria,{'inria','caltech'});
labelsCaltech = repelem("caltech",numel(scoresCaltech),1);
labelsCaltech = categorical(labelsCaltech,{'inria','caltech'});
```

Combine results from all detectors to for multiclass detection results.

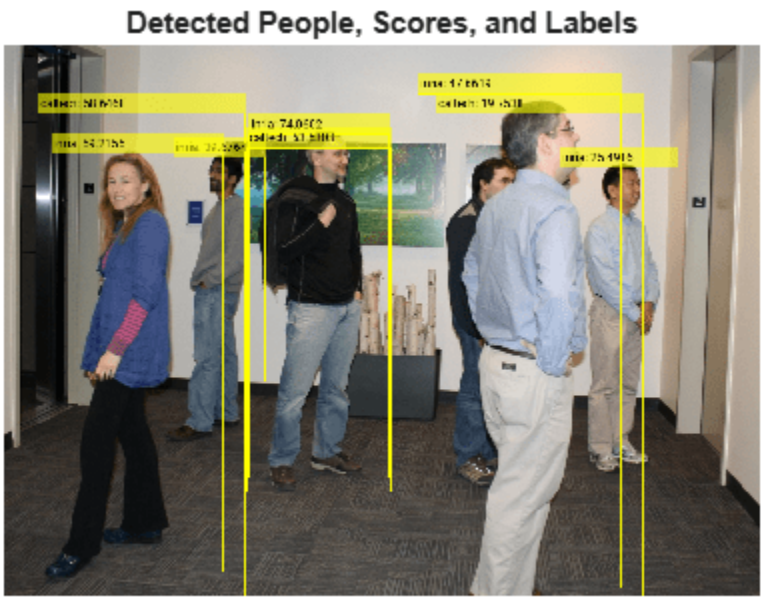
```
allBBoxes = [bboxesInria;bboxesCaltech];
allScores = [scoresInria;scoresCaltech];
allLabels = [labelsInria;labelsCaltech];
```

Run multiclass non-maximal suppression.

```
[bboxes,scores,labels] = selectStrongestBboxMulticlass(allBBoxes,allScores,allLabels,...
    'RatioType','Min','OverlapThreshold',0.65);
```

Annotate detected people.

```
annotations = string(labels) + ": " + string(scores);
I = insertObjectAnnotation(I,'rectangle',bboxes,cellstr(annotations));
imshow(I)
title('Detected People, Scores, and Labels')
```

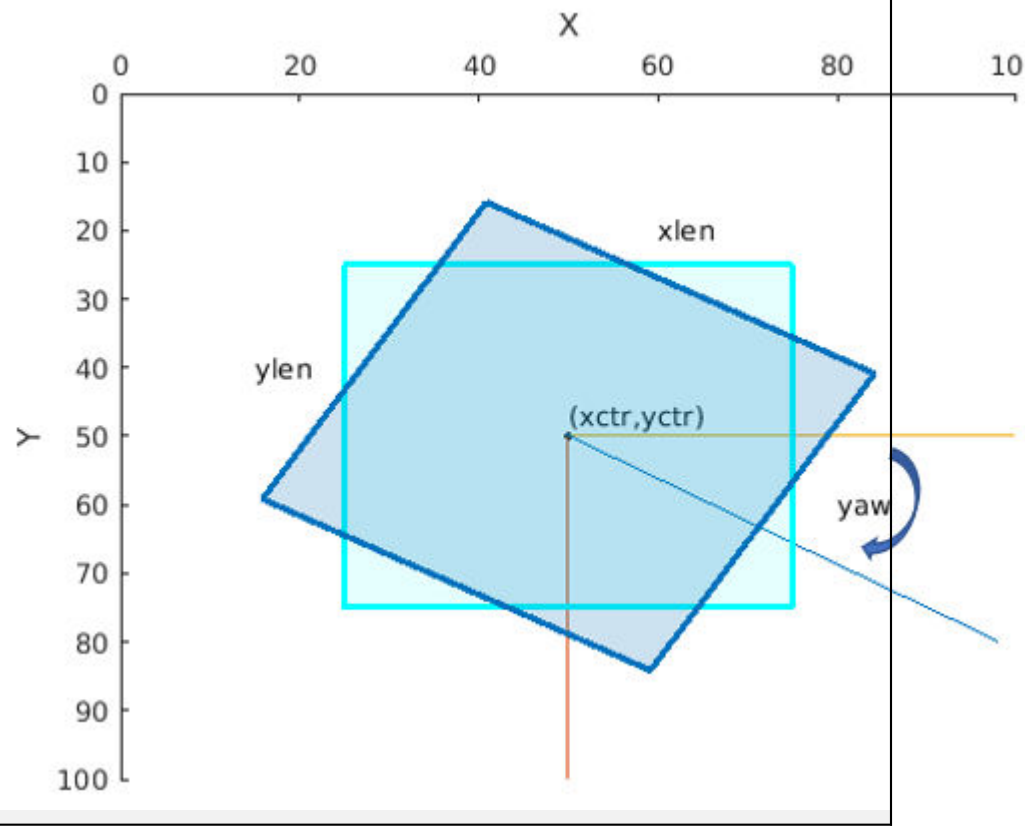


Input Arguments

bboxes — Bounding boxes

M-by-4 matrix | *M*-by-5 matrix

Bounding boxes, specified as an *M*-by-4 or *M*-by-5 nonsparse numeric matrix. *M* is the number of bounding boxes. Each row of the matrix defines a bounding box as either an axis-aligned rectangle or a rotated rectangle. This table describes the format for each bounding box.

Bounding Box	Description
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr xlen ylen yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *xlen* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *ylen* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`scores` — Confidence scores
M-by-1 vector

Confidence scores corresponding to the input bounding boxes, specified as an *M*-by-1 vector. The `selectStrongestBboxMulticlass` function uses greedy NMS to eliminate overlapping bounding boxes and associate the confidence score with the boxes. A higher score represents a higher confidence in keeping the bounding box. The `scores` input must be real, finite, and nonsparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`labels` — Labels
M-by-1 categorical vector | *M*-by-1 numeric vector

Labels corresponding to the input bounding boxes, specified as an *M*-by-1 categorical or numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `categorical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

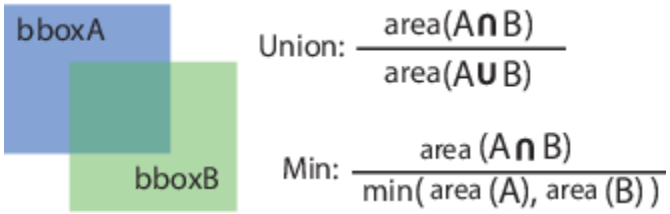
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'RatioType','Union'` sets the `'RatioType'` property to `'Union'`.

`RatioType` — Bounding box overlap ratio denominator
`'Union'` (default) | `'Min'`

Ratio type, specified as the character vector `'Union'` or `'Min'`.

- Set the ratio type to `'Union'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the area of the union of the two.
- Set the ratio type to `'Min'` to compute the ratio as the area of intersection between `bboxA` and `bboxB`, divided by the minimum area of the two bounding boxes.



Data Types: `char`

`OverlapThreshold` — Overlap ratio threshold
`0.5` (default) | scalar in the range [0 1]

Overlap ratio threshold, specified as the comma-separated pair consisting of '`OverlapThreshold`' and a scalar in the range [0 1]. When the overlap ratio is above the threshold, the function removes

bounding boxes around the reference box. Decrease the threshold to reduce the number of selected bounding boxes. However, if you decrease the threshold too much, you might eliminate boxes that represent objects close to each other in the image.

Data Types: `single` | `double`

`NumStrongest` — Maximum number of strongest boxes
`inf` (default) | positive scalar

Maximum number of strongest boxes, specified as the comma-separated pair consisting of `'NumStrongest'` and `inf` or a positive scalar. Use this argument to reduce processing time when you have a priori knowledge about the maximum number of boxes. Set the value to `inf` to select all the strongest, non-overlapping, bounding boxes.

When the `labels` input contains categorical labels, you can also specify a vector that contains the maximum number of strongest boxes for each category in the labels input. The length of the specified vector must equal the number of categories in the label.

Output Arguments

`selectedBboxes` — Selected bounding boxes
M-by-4 matrix | *M*-by-5 matrix

Selected bounding boxes, returned as an *M*-by-4 or an *M*-by-5 matrix. The 4-element vectors represent axis-aligned rectangles and the 5-element vectors represent rotated rectangles.

The `selectedBbox` output returns the selected bounding boxes from the `bbox` input that have the highest confidence score. The function uses nonmaximal suppression to eliminate overlapping bounding boxes.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`selectedScores` — Scores of selected bounding boxes
M-by-1 vector

Scores of selected bounding boxes, returned as an *M*-by-1 vector. The *M*th score in the `selectedScores` output corresponds to the *M*th bounding box in the `selectedBboxes` output. The data type of `selectedScores` matches the data type of `scores`.

`selectedLabels` — Labels of selected bounding boxes
M-by-1 categorical vector | *M*-by-1 numeric vector

Labels of selected bounding boxes, returned as an *M*-by-1 categorical or numeric vector. The *M*th label in the `selectedLabels` output corresponds to the *M*th bounding box in the `selectedBboxes` output. The data type of `selectedLabels` matches the data type of `labels`.

`index` — Index of selected bounding boxes
M-by-1 vector

Index of selected bounding boxes, returned as an *M*-by-1 vector. The `index` vector contains the indices to the selected boxes in the `bboxes` input.

Data Types: `double`

Version History

Introduced in R2018a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- GPU Arrays is not supported for rotated rectangle bounding box inputs.

See Also

`bboxOverlapRatio` | `selectStrongestBbox`

showExtrinsics

Visualize extrinsic camera parameters

Syntax

```
showExtrinsics(cameraParams)
showExtrinsics(cameraParams,viewStyle)
showExtrinsics( ___ ,Name=Value)

ax = showExtrinsics( ___ )
```

Description

`showExtrinsics(cameraParams)` renders a 3-D visualization of extrinsic parameters of a single calibrated camera, a calibrated stereo pair, or multiple calibrated cameras. The function plots a 3-D view of the calibration patterns with respect to the camera. The `cameraParams` input contains one of:

- `cameraParameters` or `stereoParameters` object which the `estimateCameraParameters` function returns.
- `fisheyeParameters` or object which the `estimateFisheyeParameters` function returns.
- `multiCameraParameters` object which the `estimateMultiCameraParameters` function returns.

The showExtrinsics function shows the 3-D view of the calibration pattern as a convex polygon containing all the detected keypoints of the pattern in the original calibration image.

`showExtrinsics(cameraParams,viewStyle)` displays visualization of the camera extrinsic parameters using the style specified by the `viewStyle` input.

`showExtrinsics( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `HighlightIndex = [1,4]` sets the highlight patterns to 1 and 4.

`ax = showExtrinsics( ___ )` returns the plot axis, using any of the preceding syntaxes.

Examples

Visualize Single Camera Extrinsic Parameters

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
  "calibration","dslr"));
imageFileNames = images.Files;
```

Detect calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate world coordinates of the corners of the squares. The square size is in millimeters.

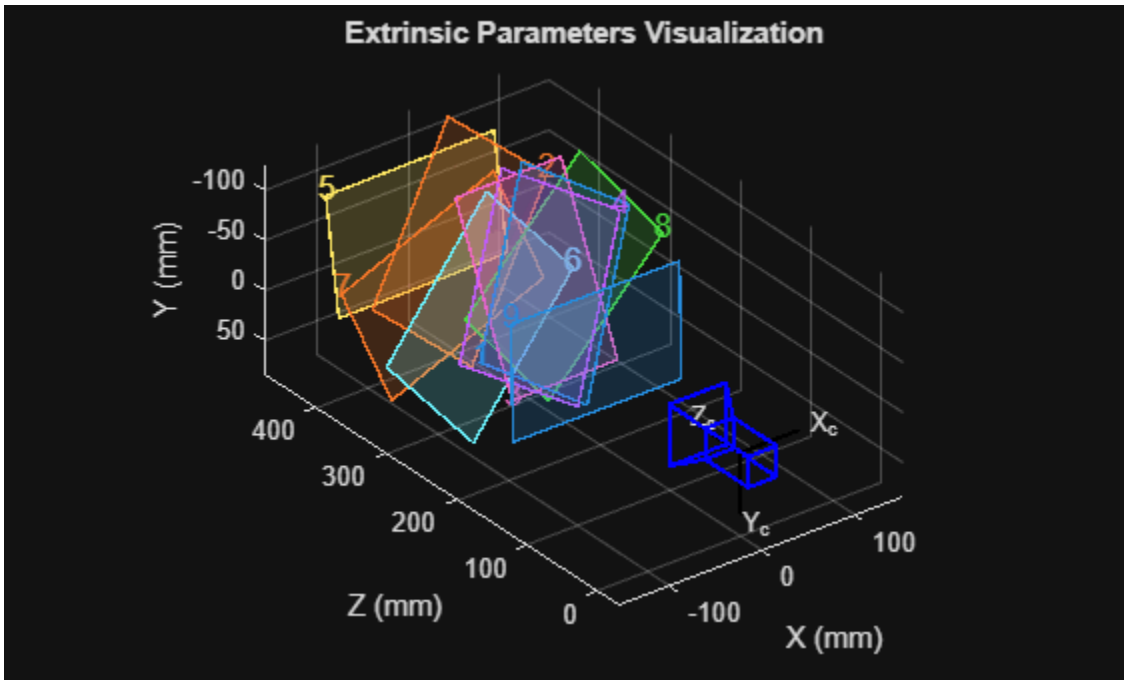
```
squareSide = 23; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSide);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = size(I,1:2);
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

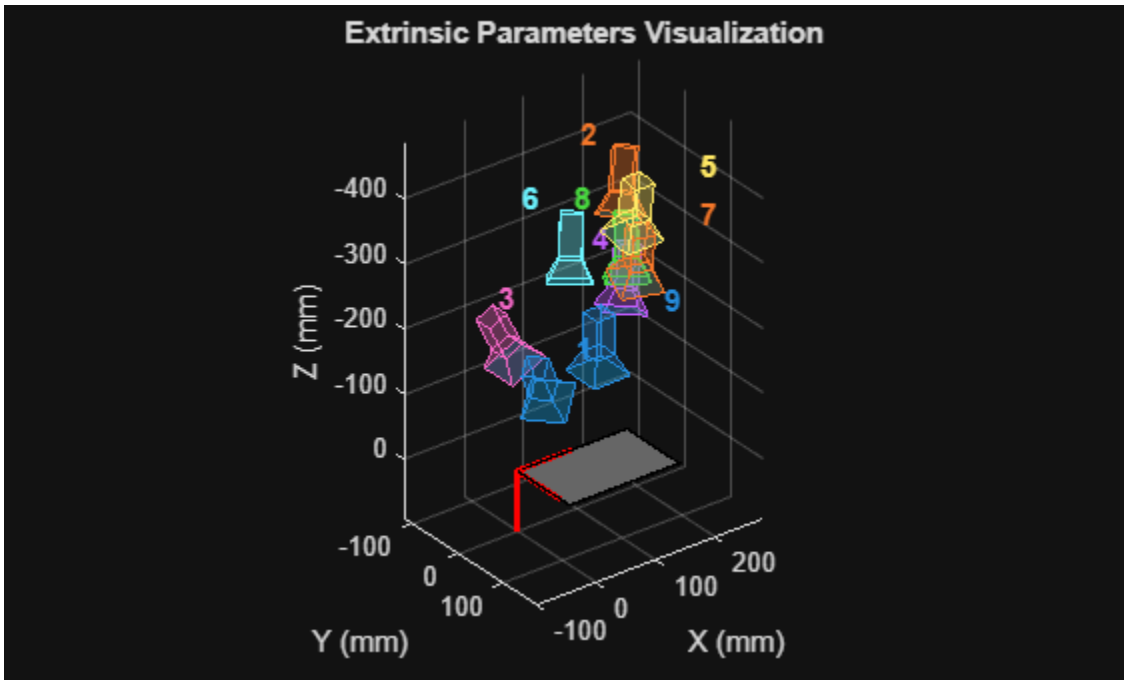
Visualize pattern locations.

```
figure
showExtrinsics(cameraParams);
```



Visualize camera locations.

```
figure
showExtrinsics(cameraParams,"patternCentric");
```



Visualize Stereo Pair of Camera Extrinsic Parameters

Specify calibration images.

```
imageDir = fullfile(toolboxdir("vision"),"visiondata",...
        "calibration","circleGrid","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
leftImageFileNames = leftImages.Files(2:2:10);
rightImageFileNames = rightImages.Files(2:2:10);
```

Define the circle grid pattern dimensions, and detect the pattern in the images.

```
patternDims = [4 11];
imagePoints = detectCircleGridPoints(leftImageFileNames,rightImageFileNames,patternDims);
```

Specify the world coordinates for the circle grid keypoints. Center distance is in millimeters.

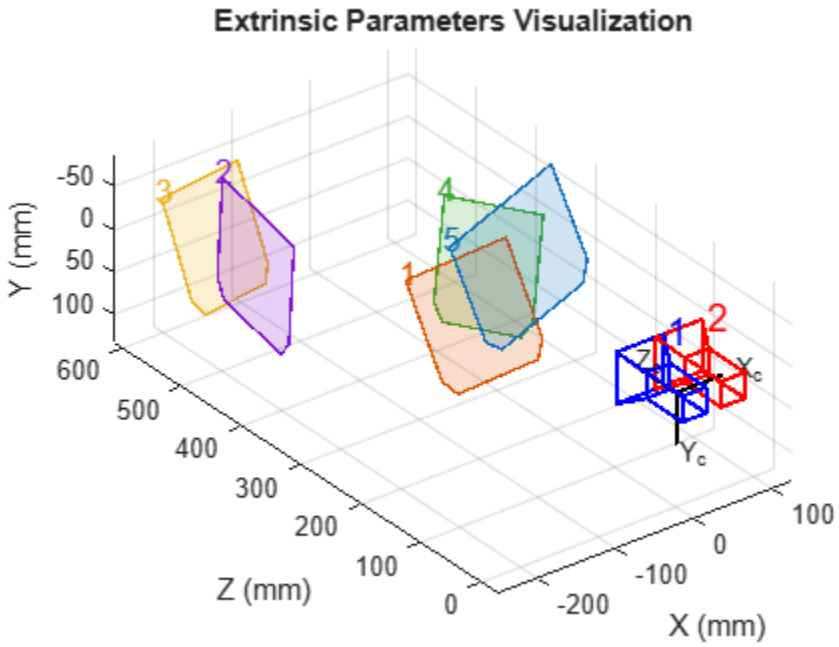
```
centerDistance = 36.5; % millimeters
worldPoints = patternWorldPoints("circle-grid-asymmetric",patternDims,centerDistance);
```

Calibrate the stereo camera system. Both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = [size(I,1),size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

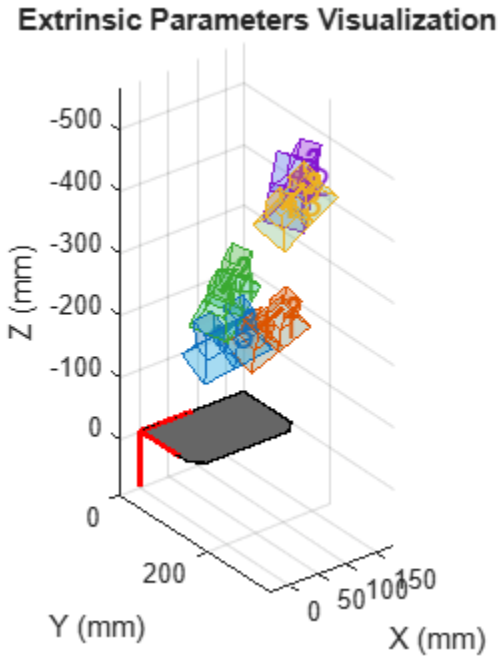
Visualize pattern locations.

```
figure
showExtrinsics(cameraParams)
```



Visualize camera locations.

```
figure
showExtrinsics(cameraParams,"patternCentric")
```



Calibrate a Multi-Camera System using a ChArUco board

Estimate the relative position and orientation of six cameras with overlapping fields of view by using calibration images that contain a single ChArUco board.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-charuco
calibImagesDir = fullfile(pwd,"overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (52 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (52 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
numKeyPoints = prod(patternDims - 1);
minMarkerId = 144;
```

Detect the key points of the ChArUco board in the calibration images.

```
imagePoints = detectPatternPoints(imageFiles,"charuco-board",patternDims, ...
    markerFamily,checkerSize,markerSize,MinMarkerID=minMarkerId);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world points for the pattern.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

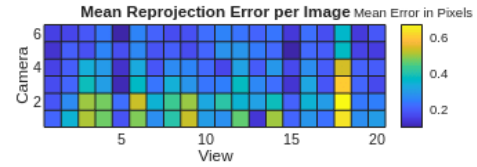
```
ld = load("sixCameraIntrinsics.mat");
```

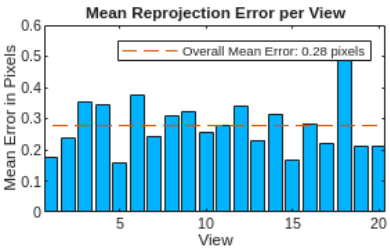
Perform multi-camera calibration.

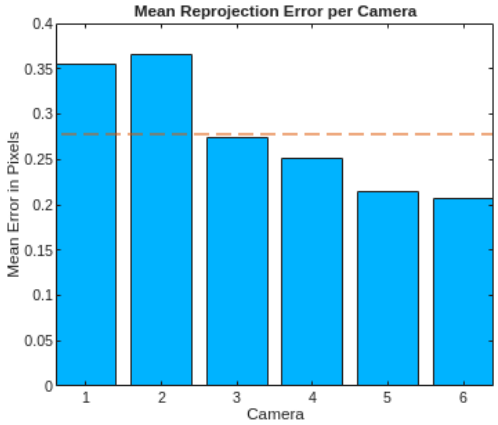
```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

Visualize the calibration accuracy.

```
figure(Position=[100,100,1000,400])
showReprojectionErrors(params)
```

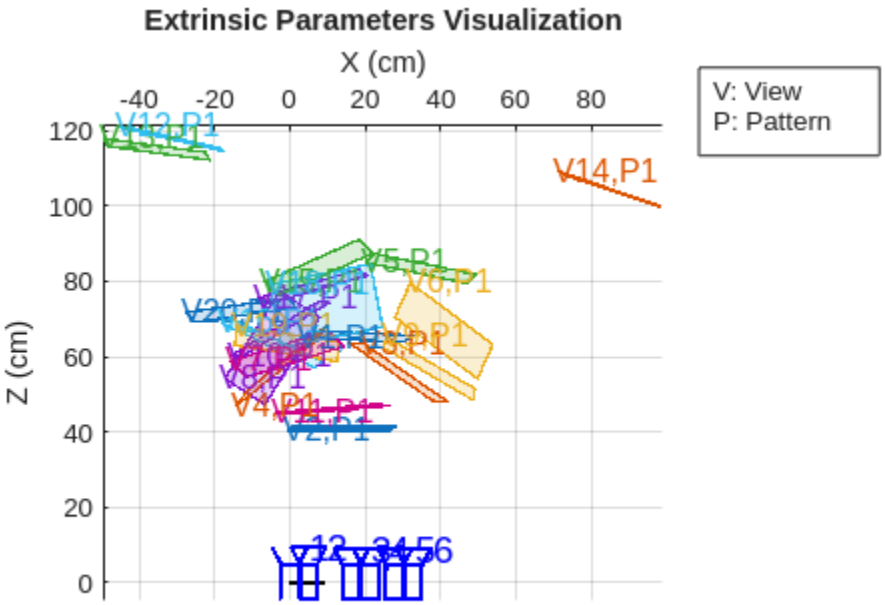






Visualize the camera extrinsic parameters.

```
figure
showExtrinsics(params)
view(2)
```



Input Arguments

`cameraParams` — Object containing parameters of single camera or stereo pair

`cameraParameters` object | `fisheyeParameters` object | `stereoParameters` object | `multiCameraParameters` object

Object containing parameters for a single, fisheye, stereo pair, or multi-camera system, specified as either a `cameraParameters`, `fisheyeParameters`, `stereoParameters`, `multiCameraParameters` object. You can create these objects using the:

- `estimateCameraParameters` function to return a `cameraParameters` or `stereoParameters` object.
- `estimateFisheyeParameters` function to return a `fisheyeParameters` object.
- `estimateMultiCameraParameters` function to return a `multiCameraParameters` object.

You can also use the Camera Calibrator app to create the `cameraParameters` input object, or use Stereo Camera Calibrator app to create the `stereoParameters` input object. See “Using the Single Camera Calibrator App” and “Using the Stereo Camera Calibrator App”.

`viewStyle` — Camera- or pattern-centric view

`"CameraCentric"` | `"PatternCentric"`

Camera or pattern-centric view, specified as `"CameraCentric"` or `"PatternCentric"`. The `viewStyle` input sets the visualization for the camera extrinsic parameters. If you keep your camera stationary while moving the calibration pattern, set `viewStyle` to `"CameraCentric"`. If the pattern is stationary while you move your camera, set it to `"PatternCentric"`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `showExtrinsics(cameraParams,HighlightIndex = [1,4])`, sets the highlight selection index to patterns 1 and 4.

`HighlightIndex` — Highlight selection index

`[]` (default) | vector | scalar

Highlight selection index, specified as a scalar or a vector of integers. When you specify this index, the opacity of one or more specified patterns is increased, making the selection stand out in contrast to the other patterns.

`CameraIndex` — Camera selection index

`[]` (default) | vector | scalar

Camera selection index, specified as a scalar or a vector of integers. This argument applies only when you set `cameraParams` to a `multiCameraParameters` object. When `CameraIndex` is empty, all cameras are displayed.

`ViewIndex` — View selection index

`[]` (default) | vector | scalar

View selection index, specified as a scalar or a vector of integers. Specify the index of camera views to display. When this value is empty, all views will be displayed.

`PatternIndex` — Pattern selection index
`[ ]` (default) | vector | scalar

Pattern selection index, specified as a scalar or a vector of integers. Specify the index of calibration patterns to display. This argument applies only when you set `cameraParams` to a `multiCameraParameters` object and `viewStyle` to `"CameraCentric"`. When this value is empty, all patterns will be displayed.

`Parent` — Output axes
current axes (default)

Output axes, specified as an axes. You can obtain the current axes handle by returning the function to an output variable:
`ax = showExtrinsics(cameraParams)`
You can also use the `gca` function to get the current axes handle.

Output Arguments

`ax` — Current axes handle
scalar value

Current axes handle, returned as a scalar value. The function returns the handle to the current axes for the current figure.

Example: `ax = showExtrinsics(cameraParams)`

Tips

- When displaying multiple patterns in the `"CameraCentric"` view, patterns corresponding to the same view are displayed with the same color and are labeled as "*V*`viewIndex`, *P*`patternIndex`". For example, patterns 1 and 2 of view 3 are labeled as "`V3,P1`" and "`V3,P2`", respectively.

  In the `"CameraCentric"` view, the cameras with the same color have overlapped fields of view.
- In the `"PatternCentric"` view, the first pattern is chosen as the reference pattern. All cameras are displayed relative to this reference and are labeled as "*V*`viewIndex`, *C*`cameraIndex`". For example, cameras 1 and 2 of view 3 are labeled as "`V3,C1`" and "`V3,C2`, respectively.

Version History

Introduced in R2014a

R2026a: New `ViewIndex` and `PatternIndex` name-value arguments

Added the `ViewIndex` and `PatternIndex` name-value arguments to support multiple calibration patterns.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`estimateMultiCameraParameters` | `plotCamera` | `estimateCameraParameters` | `showReprojectionErrors` | `undistortImage` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `patternWorldPoints` | `detectPatternPoints`

Objects
`multiCameraParameters` | `cameraParameters` | `stereoParameters` | `fisheyeParameters`

Topics
“Using the Single Camera Calibrator App”
“3-D Motion Reconstruction Using Multiple Cameras”

showMatchedFeatures

Display corresponding feature points

Syntax

```
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2)
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,method)

showMatchedFeatures(___,PlotOptions, {MarkerStyle1, MarkerStyle2, LineStyle})
showMatchedFeatures(___,Name=Value)

H = showMatchedFeatures(___)
```

Description

`showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2)` displays an overlay of images `I1` and `I2` with a color-coded plot of corresponding points connected by a line. `matchedPoints1` and `matchedPoints2` contain the coordinates of corresponding points in `I1` and `I2`.

`showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,method)` displays images `I1` and `I2` using the visualization style specified by the `method` parameter.

`showMatchedFeatures(___,PlotOptions, {MarkerStyle1, MarkerStyle2, LineStyle})` lets you specify custom plot options in a cell array containing three values. The `MarkerStyle1`, `MarkerStyle2`, and `LineStyle` values correspond to the marker symbol in `I1`, marker symbol in `I2`, and the line style and color. The `LineSpec` syntax of the `plot` function defines each of the specifiers.

`showMatchedFeatures(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `showMatchedFeatures(__,PlotOptions={"d","+","g"})` sets the marker style to a diamond for `I1`, a plus sign for `I2`, and the color to green.

`H = showMatchedFeatures(___)` returns the handle to the image object returned by `showMatchedFeatures`.

Examples

Find Corresponding Points Between Two Images Using Harris Features

Read images.

```
I1 = im2gray(imread("parkinglot_left.png"));
I2 = im2gray(imread("parkinglot_right.png"));
```

Detect Harris features.

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
```

Extract features.

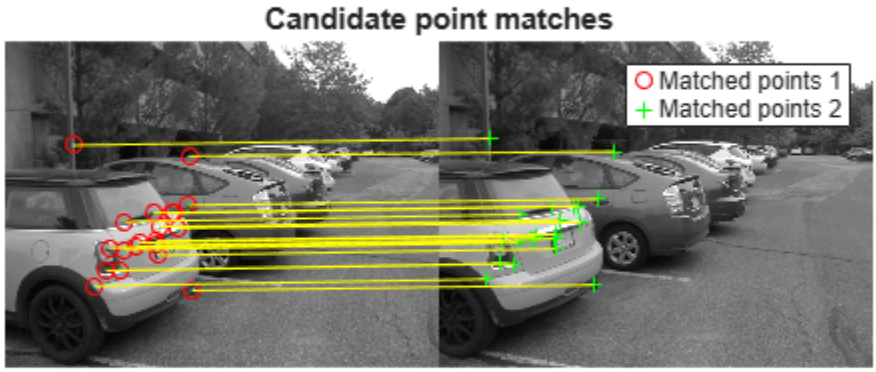
```
[f1,vpts1] = extractFeatures(I1,points1);
[f2,vpts2] = extractFeatures(I2,points2);
```

Match features.

```
indexPairs = matchFeatures(f1,f2);
matchedPoints1 = vpts1(indexPairs(1:20,1));
matchedPoints2 = vpts2(indexPairs(1:20,2));
```

Visualize candidate matches.

```
figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,"montage");
title("Candidate point matches");
legend("Matched points 1","Matched points 2");
```



Display Corresponding Points Between Two Rotated and Scaled Images

Read images.

```
I1 = imread("cameraman.tif");
I2 = imresize(imrotate(I1,-20), 1.2);
```

Detect SURF features.

```
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);
```

Extract features.

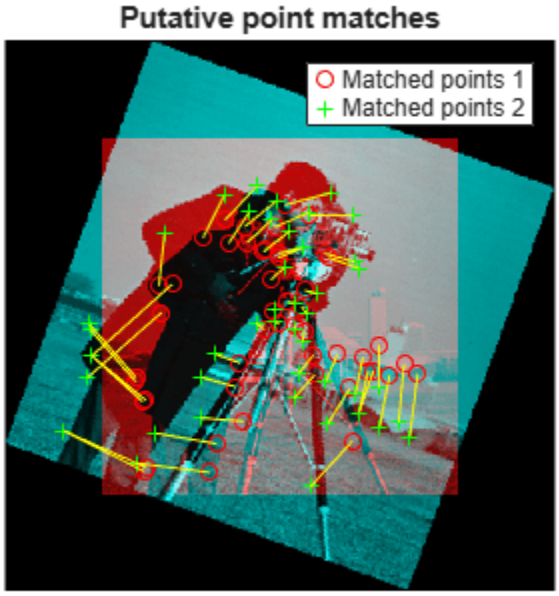
```
[f1,vpts1] = extractFeatures(I1,points1);
[f2,vpts2] = extractFeatures(I2,points2);
```

Match features.

```
indexPairs = matchFeatures(f1,f2);
matchedPoints1 = vpts1(indexPairs(:,1));
matchedPoints2 = vpts2(indexPairs(:,2));
```

Visualize candidate matches.

```
figure;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2);
title("Putative point matches");
legend("Matched points 1","Matched points 2");
```



Input Arguments

`I1` — Input image

numeric array

Input image one, specified as a numeric array.

`I2` — Input image

numeric array

Input image two, specified as a numeric array.

`matchedPoints1` — Coordinates of points

M-by-2 matrix | point feature object

Coordinates of points in image one, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`matchedPoints2` — Coordinates of points
M-by-2 matrix | point feature object

Coordinates of points in image two, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

`method` — Display method
`falsecolor` (default) | `blend` | `montage`

Display style method, specified as one of the following:

`falsecolor`	Overlay the images by creating a composite red-cyan image showing `I1` as red and `I2` as cyan.
`blend`	Overlay `I1` and `I2` using alpha blending.
`montage`	Place `I1` and `I2` next to each other in the same image.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `showMatchedFeatures(__,PlotOptions={"d","+","g"})` sets the marker style to a diamond for `I1`, a plus sign for `I2`, and the color to green.

`PlotOptions` — Marker, line style, and color
`{"ro","g+","y-"}` (default) | cell array of character vectors | string array

Marker, line style, and color options, specified as a cell array of character vectors or a string array. The three values {*MarkerStyle1*, *MarkerStyle2*, *LineStyle*}, correspond to the marker symbol in `I1`, the marker symbol in `I2`, and the line style and color. The `LineSpec` syntax of the `plot` function defines each of the specifiers.

`Parent` — Output axes
`axes` graphics object

Output axes for displaying visualization, specified as an `axes` graphics object.

Output Arguments

`H` — Handle to image object
handle

Handle to image object, displayed by `showMatchedFeatures`.

Version History

Introduced in R2012b

See Also

Functions
`matchFeatures` | `estgeotform2d` | `legend` | `imshowpair`

Objects
`SIFTPoints` | `MSERRegions` | `ORBPoints` | `BRISKPoints` | `cornerPoints`

Topics
“Point Feature Types”

showPointCloud

Plot 3-D point cloud

Syntax

```
showPointCloud
```

Description

`showPointCloud` was renamed to `pcshow`. Please use `pcshow` in place of `showPointCloud`.

Version History

Introduced in R2014b

R2020a: `showPointCloud` renamed to `pcshow`
Behavior changed in R2020a

`showPointCloud` was renamed to `pcshow`. Please use `pcshow` in place of `showPointCloud`.

pcshow

Plot 3-D point cloud

Syntax

```
pcshow(ptCloud)

pcshow(xyzPoints)
pcshow(xyzPoints,color)
pcshow(xyzPoints,colorMap)
pcshow(filename)

pcshow( ___ ,Name=Value)

ax = pcshow( ___ )
```

Description

`pcshow(ptCloud)` displays points using the locations and colors stored in the point cloud object.

To visualize a large point cloud (greater than one million points), use the `pcviewer` object.

`pcshow(xyzPoints)` displays points specified by the `xyzPoints` matrix.

`pcshow(xyzPoints,color)` displays points contained in the `xyzPoints` matrix, with colors specified by `color`.

`pcshow(xyzPoints,colorMap)` displays points contained in the `xyzPoints` matrix, with colors specified by `colorMap`.

`pcshow(filename)` displays the point cloud stored in the file specified by `filename`.

`pcshow( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination input of arguments from previous syntaxes. For example, `ViewPlane="XY"` sets the plane from which to visualize the point cloud as the *xy*-plane.

`ax = pcshow( ___ )` returns the plot axes.

Examples

Plot Spherical Point Cloud with Texture Mapping

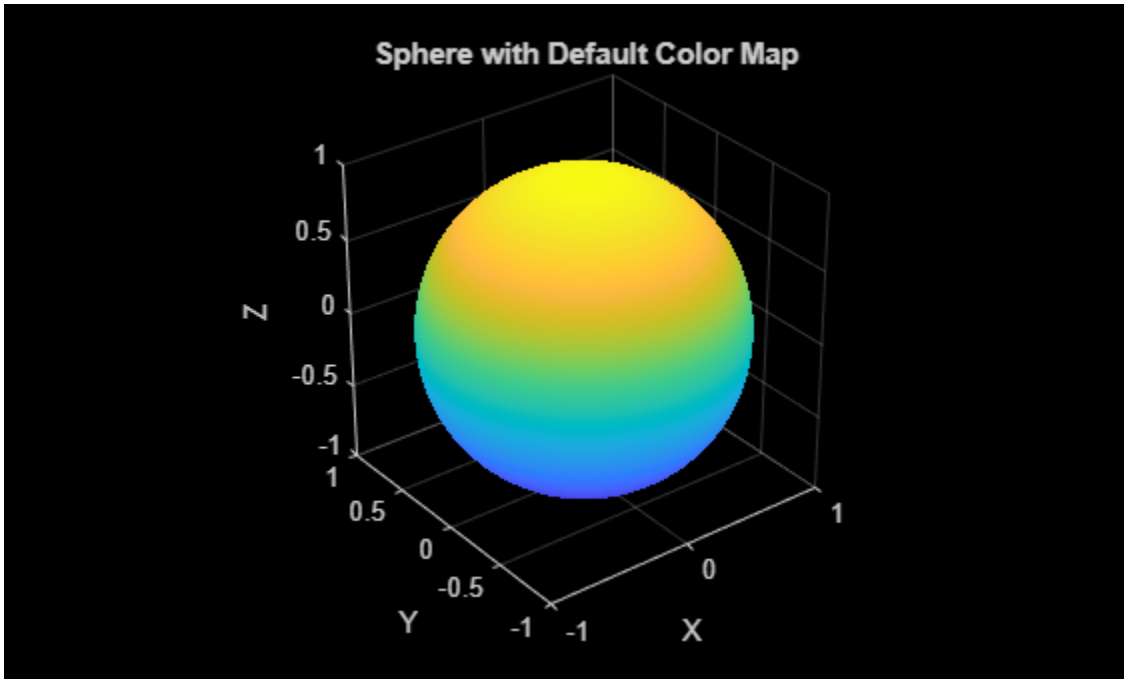
Generate a sphere consisting of 600-by-600 faces.

```
numFaces = 600;
[x,y,z] = sphere(numFaces);
```

Plot the sphere using the default color map.

```
figure;
pcshow([x(:) y(:) z(:)])
```

```
title("Sphere with Default Color Map")
xlabel("X")
ylabel("Y")
zlabel("Z")
```



Load and display an image for texture mapping.

```
I = im2double(imread("visionteam1.jpg"));
imshow(I)
```

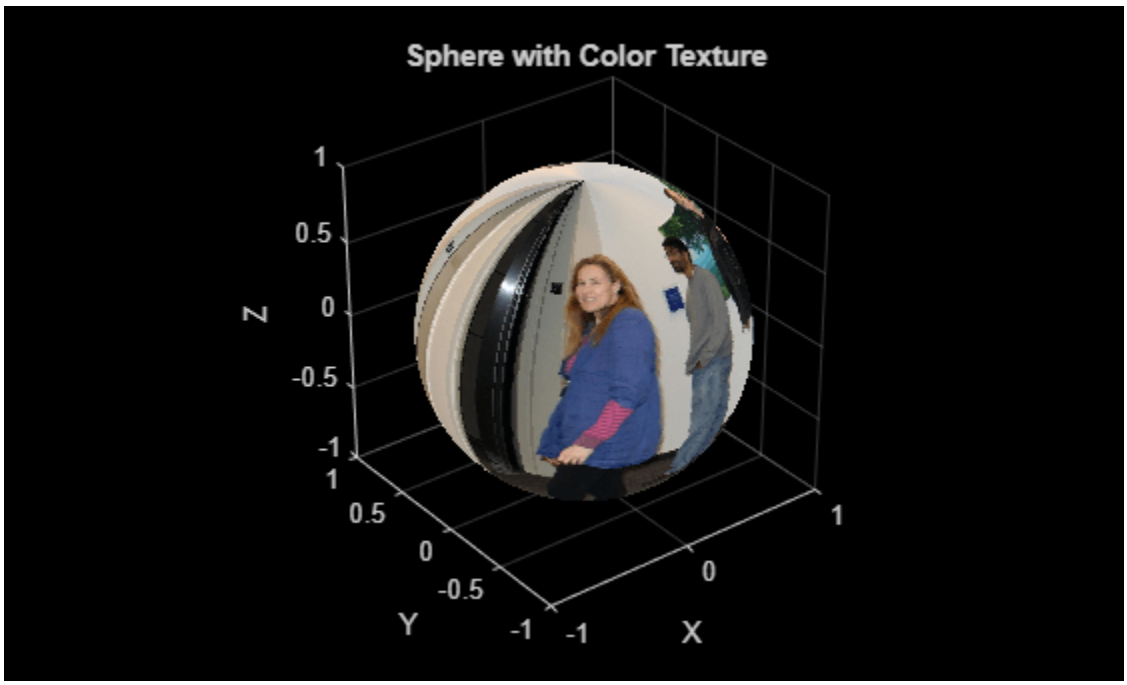


Resize and flip the image in preparation for mapping the coordinates.

```
J = flipud(imresize(I,size(x)));
```

Plot the sphere with the image as a color texture.

```
pcshow([x(:) y(:) z(:)],reshape(J,[],3))
title("Sphere with Color Texture")
xlabel("X")
ylabel("Y")
zlabel("Z")
```



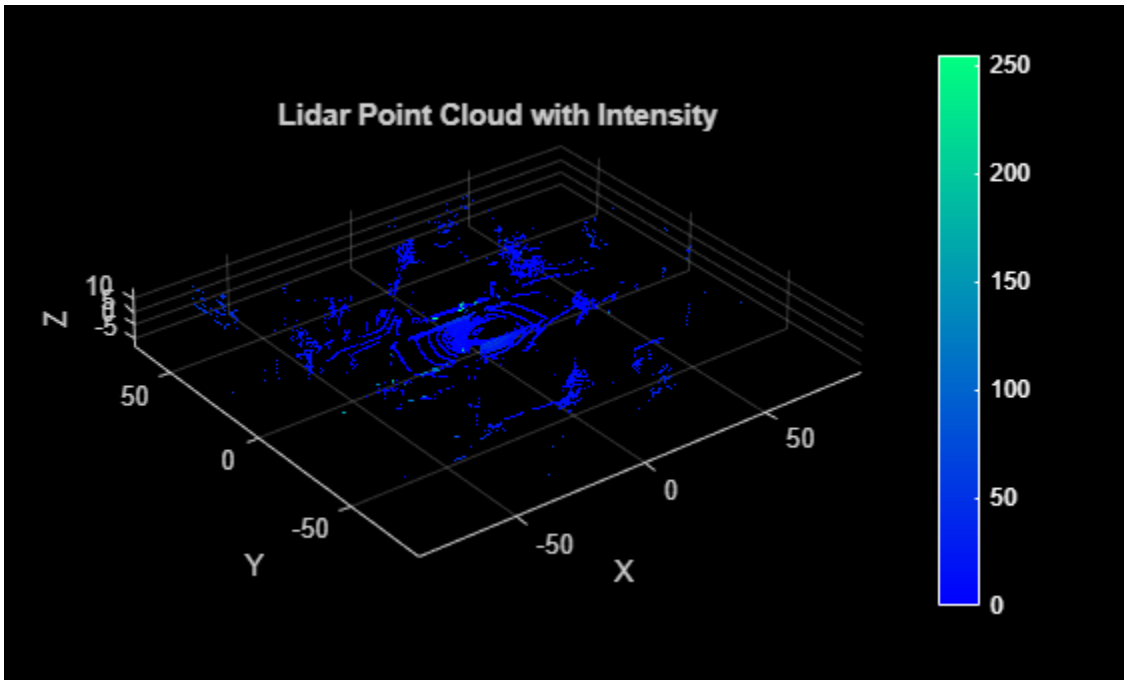
Plot Lidar Point Cloud Based On Intensity

Load an organized lidar point cloud that includes intensity data.

```
ld = load("drivingLidarPoints.mat");
```

Plot the point cloud and label the figure. By default, the intensity is mapped to the `jet` colormap.

```
figure
pcshow(ld.ptCloud, ColorSource="Intensity")
title("Lidar Point Cloud with Intensity")
xlabel("X")
ylabel("Y")
zlabel("Z")
colorbar(Color=[1 1 1])
colormap("winter")
```



Input Arguments

`ptCloud` — Point cloud

`pointCloud` object

Point cloud, specified as a `pointCloud` object. The object contains the locations, intensities, and RGB colors to render the point cloud.

`filename` — Point cloud filename

character vector | string scalar

Point cloud filename, specified as a character vector or a string scalar. The file must be of a type supported by `pcread`. The `pcshow` function calls `pcread` to read the point cloud from the file, but does not store the data in the MATLAB workspace.

`xyzPoints` — Point cloud *x*-, *y*-, and *z*- locations

M-by-3 numeric matrix | *M*-by-*N*-by-3 numeric array

Point cloud *x*-, *y*-, and *z*- locations, specified as either an *M*-by-3 numeric matrix or an *M*-by-*N*-by-3 numeric array. The `xyzPoints` argument contains *M* or *M*-by-*N* points in the form [*x*,*y*,*z*]. The *z* values, which generally correspond to depth or elevation, determine the color of each point. When you do not specify the input color, the function maps each *z* value to a color in the current colormap.

`color` — Color for points in point cloud

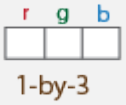
RGB triplet | short name of color | long name of color | array of color names | cell array of color names | *M*-by-3 array | *M*-by-*N*-by-3 matrix

Color for points in the point cloud, specified as an RGB triplet, an *M*-by-3 matrix, an *M*-by-*N*-by-3 array, a short color name, a long color name, array of color names, or a cell array of color names. For details on color values, see the Color Value on page 3-1212 table.

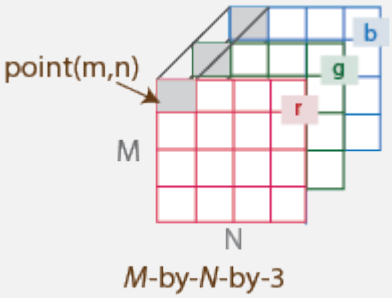
- For the `single` or `double` data type, each `color` RGB value must be in the range [0, 1].
- For the `uint8` data type, each `color` RGB value must be in the range [0, 255].
- The function stores the `color` value as either an *M*-by-3 matrix or *M*-by-*N*-by-3 array. If you specify the color as a string or a 1-by-3 vector, the function converts it to one of these dimensions.

Color	Format	Example

Specify one color for all points	short or long color name	`"r"`
		
		`"red"`

	1-by-3 vector (RGB triplet)	 `[255 0 0]`
Specify a color for each point	string array of character vectors	`["red","yellow","blue"]`

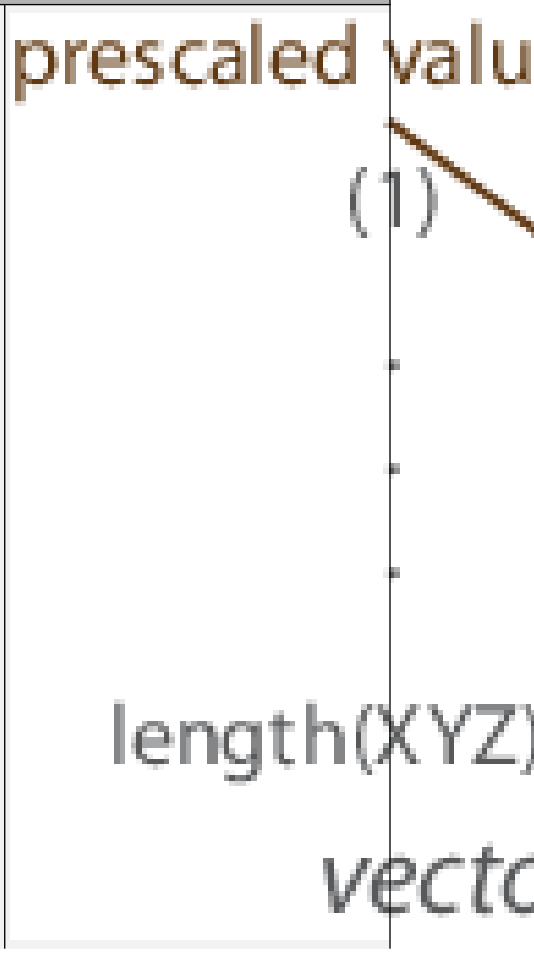
	M-by-3 matrix, in which each row is an RGB triplet	`255 0 0`
		`255 0 0`
		`0 255 255`
		

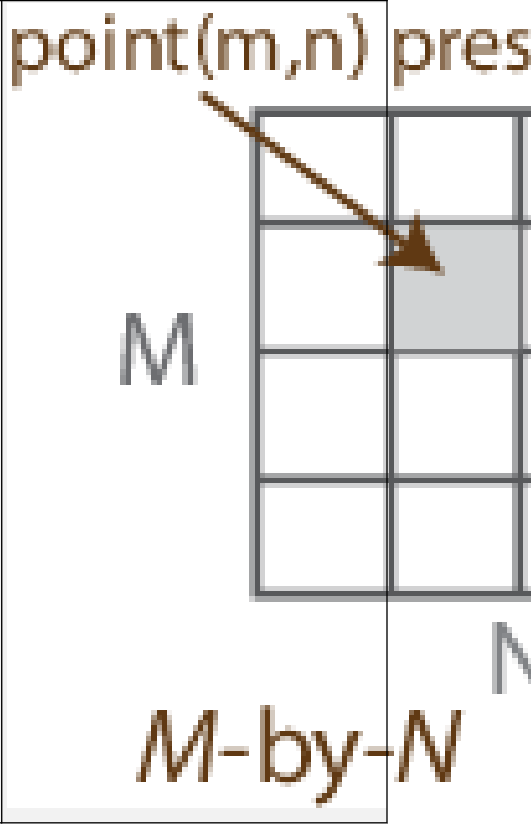
	M-by-*N*-by-3 array for an organized point cloud, containing RGB values for each point, where *M*-by-*N* is the size of the point cloud.	

`colorMap` — Point cloud color map

M-by-1 vector | *M*-by-*N* matrix

Point cloud color map, specified as an *M*-by-1 vector or an *M*-by-*N* matrix.

Points Input	Color Selection	Valid Values of color	
`xyzPoints`	Different color for each point	Vector or *M*-by-*N* matrix. The matrix must contain values that are linearly mapped to a color in the current `colormap`.	

Points Input	Color Selection	Valid Values of color	
			

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ViewPlane="XY"` sets the plane to the x-y axes from which to visualize the point cloud.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`MarkerSize` — Diameter of marker

`6` (default) | positive scalar

Diameter of the marker, specified as a positive scalar. The value specifies the approximate diameter of the point marker. MATLAB graphics defines the unit as points. A marker size larger than 6 can reduce rendering performance.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`BackgroundColor` — Background color

`[0 0 0]` (black) (default) | RGB triplet | hexadecimal color code | character vector

Background color, specified as an RGB triplet, hexadecimal color code, or a character vector that specifies a long or short color name.

Data Types: `cell` | `char` | `uint8` | `uint16` | `int16` | `double` | `single`

`VerticalAxis` — Vertical axis
`"Z"` (default) | `"X"` | `"Y"`

Vertical axis, specified as `"X"`, `"Y"`, or `"Z"`. This value sets the selected axis of the point cloud in the vertical direction.

`VerticalAxisDir` — Vertical axis direction
`"Up"` (default) | `"Down"`

Vertical axis direction, specified as `"Up"` or `"Down"`.

`Projection` — Camera projection for 3-D views
`"perspective"` (default) | `"orthographic"`

Camera projection for 3-D views, specified as one of these values:

- `"perspective"` — Projects the viewing volume as the frustum of a pyramid (a pyramid whose apex has been cut off parallel to the base). Objects further from the camera appear smaller. Distance causes foreshortening, which enables you to perceive the depth of 3-D objects. This projection type is useful when you want to display realistic views of real objects. Perspective projection does not preserve the relative dimensions of objects. Instead, it displays a distant line segment smaller than a nearer line segment of the same length. Lines that are parallel in the data might not appear parallel in the scene.
- `"orthographic"` — This projection type maintains the correct relative dimensions of graphics objects regarding the distance of a given point from the viewer. Relative distance from the camera does not affect the size of objects. Lines that are parallel in the data parallel appear parallel on the screen. This projection type is useful when it is important to maintain the actual size of objects and the angles between objects.

`ViewPlane` — Plane to visualize point cloud
`"auto"` (default) | `"XY"` | `"YX"` | `"XZ"` | `"ZX"` | `"YZ"` | `"ZY"`

Plane to visualize point cloud, specified as `"auto"`, `"XY"`, `"YX"`, `"XZ"`, `"ZX"`, `"YZ"`, or `"ZY"`. The `ViewPlane` argument sets the line of sight from the camera, centered in the selected plane, to the center of the plot.

`ColorSource` — Point cloud property for colormap
`"auto"` (default) | `"X"` | `"Y"` | `"Z"` | `"Intensity"` | `"Color"` | `"Row"` | `"Column"`

Point cloud property for colormap, specified as `"auto"`,`"X"`, `"Y"`, `"Z"`, `"Intensity"`, `"Color"`, `"Row"`, `"Column"`. The `pcshow` function uses the value of `ColorSource` to assign a colormap when rendering the point cloud. The values use these properties:

- `"X"` — *x*-coordinates in the `Location` property of the input point cloud.
- `"Y"` — *y*-coordinates in the `Location` property of the input point cloud.
- `"Z"` — *z*-coordinates in the `Location` property of the input point cloud.
- `"Intensity"` — `Intensity` property of the input point cloud.
- `"Color"` — `Color` property of the input point cloud.
- `"Row"` — Row values of an organized point cloud.
- `"Column"` — Column values of an organized point cloud.

When you set `ColorSource` to `"auto"`, the function sets the colormap depending on which property values are available in the input point cloud object:

Available Point Cloud Property	Color Source
`Location` only	Maps the *z*-value to a color value in the current colormap.
`Location` and `Intensity`	Maps the *z*-value to a color value in the current colormap.
`Location` and `Color`	Maps the color values to the current colormap.
`Location`, `Intensity`, and `Color`	Maps the color values to the current colormap.

`AxesVisibility` — State of axes visibility
`"on"` or `1` (`true`) (default) | `"off"` or `0` (`false`)

State of axes visibility, specified as `"on"` or `"off"`, or as numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true`, and `"off"` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `"on"` — Display the axes and its children.
- `"off"` — Hide the axes without deleting it. You still can access the properties of an invisible axes object. (Child objects such as lines remain visible.)

`Parent` — Axes on which to display visualization
`Axes` graphics object

Axes on which to display the visualization, specified as an `Axes` object. To create an `Axes` object, use the `axes` function. To display the visualization in a new figure, leave `Parent` unspecified.

Output Arguments

`ax` — Plot axes
`Axes` graphics object

Plot axes, returned as an `axes` graphics object. Points that are represented as `NaN` or `Inf` are not included.

You can set the default center of rotation for the point cloud viewer as the axes center or as a point. Set the default behavior from the “Computer Vision Toolbox Preferences”.

A rotation axes (represented with red, blue, and green lines) appears at the center of rotation when rotating the point cloud.

More About

Axes Toolbar Operations

Keyboard shortcuts are enabled only for the rotate (default) mode. To view the axes toolbar operations, point to the figure and select one of these options. Click the Restore View icon to reset the display.



Feature	Description
Vertical Axis	Select an *X*-, *Y*-, or *Z*-direction to set the direction of that axis of the point cloud as the positive vertical direction. Use the keyboard arrow keys to move the point cloud with respect to the direction selected.
View Plane	Select the viewing angle of the point cloud figure as the *XY*, *YX*, *XZ*, *ZX*, *YZ*, or *ZY* plane.
Background Color	Display a color chart from which you can select a background color.
Axes On/Off	Turn the display axes on or off.
Projection	Select perspective or orthogonal view.
Export	• — Save the content as a tightly cropped image or PDF. • — Copy the content as an image. • — Copy the content as a vector graphic.
Brush	Enable data brushing mode.
Data Tips	View the point data values of a point by selecting it in the point cloud figure. The Data Tips display shows the *x*, *y*, and *z* values of the points. Point clouds for depth images and lidar contain these additional data properties:

Point Cloud Data Source	Data Value Properties
Depth image (RGB-D sensor)	**Color**, **Row**, and **Column**
Lidar	**Intensity**, **Range**, **Azimuth** angle, **Elevation** angle, **Row**, and **Column**

<table>
<tr><th>Feature</th><th>Description</th></tr>
<tr><td>Rotate 3D</td><td>Drag to rotate the point cloud figure. You can also right-click any point on the point cloud figure and select Rotate Around a Point to specify that point as the center of rotation for the figure. To return to axes rotation, right-click again and select Rotate Around Axes Center.

While in Rotate 3D mode, you can also select the color source for the point cloud by right-clicking anywhere in the point cloud figure and selecting Color Source, and then selecting an option from the list. The figure applies the active colormap to the point cloud based on the values in the selected source. For a point cloud of only position data, the Color Source options are X, Y, Z, and User Specified. Point clouds for depth image and lidar data include these additional color source options:

<table>
<tr><th>Point Cloud Data Source</th><th>Color Source Options</th></tr>
<tr><td>Depth image (RGB-D sensor)</td><td>Color, Row, and Column</td></tr>
<tr><td>Lidar</td><td>Intensity, Range, Azimuth angle, Elevation angle, Row, and Column</td></tr>
</table>
</td></tr>
<tr><td>Pan, Zoom in, and Zoom out</td><td>Maneuver the point cloud in the figure, or zoom the point cloud figure in or out.</td></tr>
<tr><td>Restore View</td><td>Restore the original view of the axes.</td></tr>
</table>

Move Camera Through Scene

To navigate into a point cloud and observe the scene, use these keyboard shortcuts:

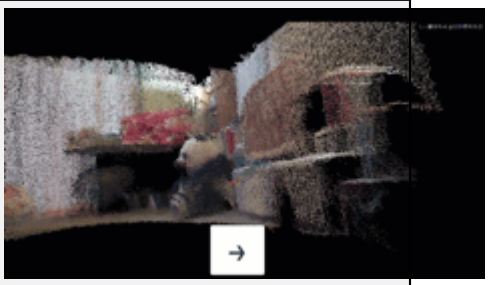
<table>
<tr><th>Action</th><th>Keyboard Shortcut</th><th>Example</th></tr>
<tr><td>Move forward or backward through the scene. The camera and the target can move.</td><td>w — Forward

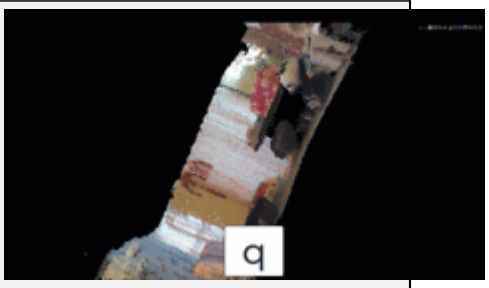
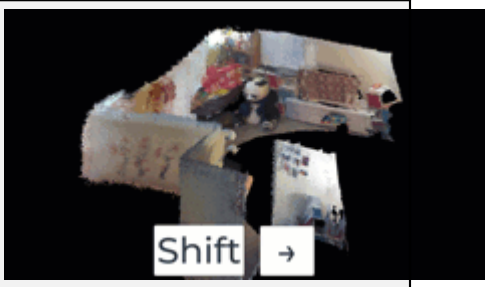
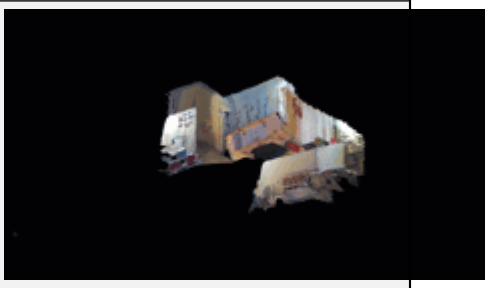
s — Backward</td><td rowspan="2">
</td></tr>
<tr><td>Move left or right through the scene. The camera and the target can move.</td><td>a — Left

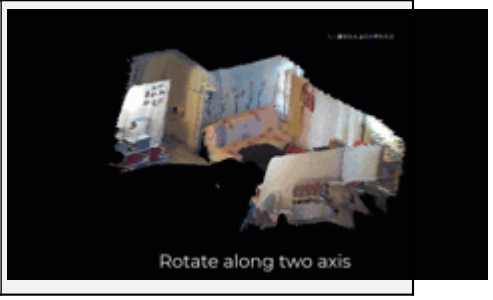
d — Right</td></tr>
<tr><td>Look around the scene (pitch/yaw rotation of the camera). Camera position is fixed, target moves.</td><td>Up arrow — Camera pitches up

Down arrow — Camera pitches down

Left arrow — Camera turns left in yaw

Right arrow — Camera turns right in yaw</td><td></td></tr>
</table>

Action	Keyboard Shortcut	Example
Roll the scene (roll rotation of the camera).	**q** — Camera moves clockwise. (Scene appears to move counterclockwise.) **e** — Camera moves counterclockwise. (Scene appears to move clockwise.)	
Rotate the scene (rotate the camera around the point cloud). Camera position changes, target is fixed.	**Shift** + **Left** or **Right** — Camera orbits horizontally around the point cloud. (Scene appears to stand still as camera moves externally around it horizontally.) **Shift** + **Up** or **Down** — Camera orbits vertically around the point cloud. (Scene appears to stand still as camera moves externally around it vertically.)	
Zoom in or out.	**z** — Zoom in **x** — Zoom out	
Look at a view plane.	1 — **XY** 2 — **YX** 3 — **XZ** 4 — **ZX** 5 — **YZ** 6 — **ZY** Camera positioned in the selected plane with a line of sight to the center of the plot.	

Action	Keyboard Shortcut	Example
Switch between rotate and pan.	**Space bar** — Press and hold the space bar to switch the interaction to pan. Release the space bar to switch back to rotate.	
Rotate in X-, Y-, and Z-dimensions.	**T** — Adds rotate interaction in the third dimension.	

Color Values

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Performance and Tips

- To improve performance, the function automatically downsamples the rendered point cloud when you rotate, pan, or zoom in or out by clicking the mouse. Downsampling is not applied for zoom in or out by using the mouse scroll mechanism. When using data tips, the figure does not apply downsampling. The downsampling occurs only for rendering the point cloud and does not affect the saved points.
- The keyboard shortcuts described in “Move Camera Through Scene” on page 3-1210 are not supported when loading a saved figure or for popped-out windows using Live Editor.
- When you reload a saved figure, any action on the figure resets the vertical axis to the *z*-axis and in the up direction.

- To visualize a large point cloud (greater than one million points), use the `pcviewer` object.

Version History

Introduced in R2015b

R2022b: Default projection changed to perspective

The default projection for the camera 3-D view display has changed from orthographic to perspective. You can use the new `Projection` name-value argument to specify the projection as `"perspective"` or `"orthographic"`. Prior to R2022b, the display rendered the point cloud in an orthographic view only.

See Also

Objects
`pcviewer` | `pointCloud` | `pcplayer` | `planeModel`

Functions
`showShape` | `pcshowpair` | `plot3` | `pcwrite` | `pcread` | `pcmerge` | `scatter3` | `reconstructScene` | `triangulate` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp`

Topics
“Structure from Motion from Two Views”
“Depth Estimation from Stereo Video”
“Choose Function to Visualize Detected Objects”
“Coordinate Systems”

pcshowpair

Visualize difference between two point clouds

Syntax

```
pcshowpair(ptCloudA,ptCloudB)

pcshowpair(ptCloudA,ptCloudB,Name=Value)

ax = pcshowpair( ___ )
```

Description

`pcshowpair(ptCloudA,ptCloudB)` creates a visualization depicting the differences between the two input point clouds. The differences are displayed using a blending of magenta for point cloud A and green for point cloud B.

`pcshowpair(ptCloudA,ptCloudB,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `pchowpair(ptCloudA,ptCloudB,BackgroundColor="yellow")`, sets the figure background color to yellow.

`ax = pcshowpair( ___ )` returns the plot axes to the visualization of the differences, using any of the preceding syntaxes.

Examples

Visualize the Difference Between Two Point Clouds

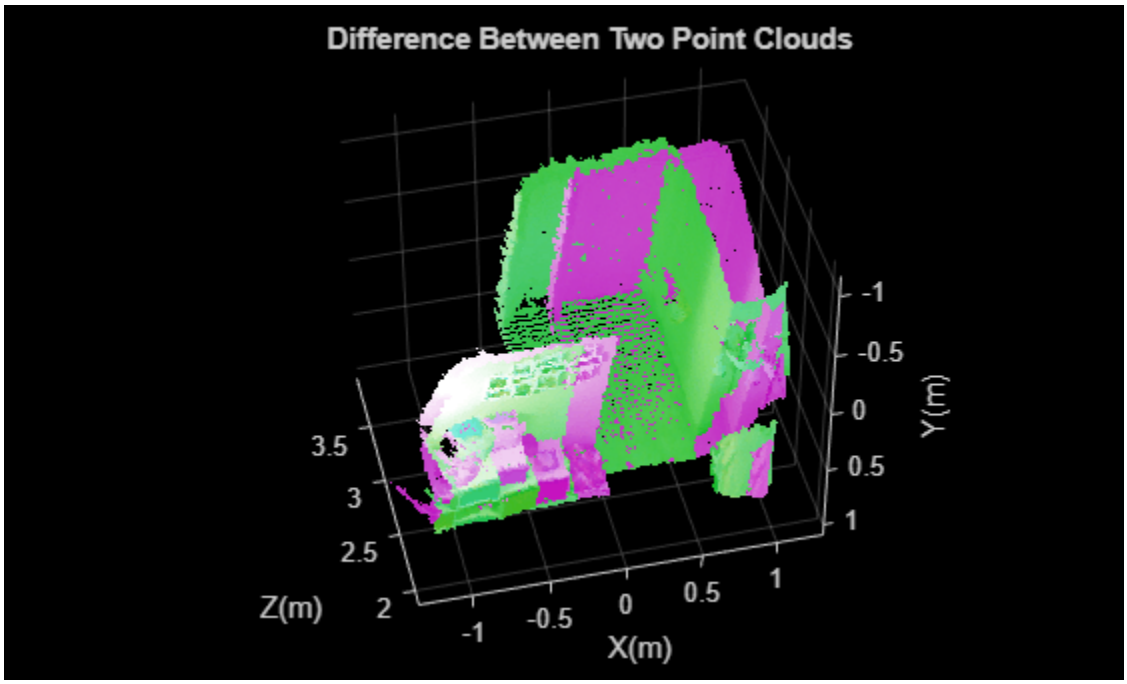
Load two point clouds that were captured using a Kinect device in a home setting.

```
load("livingRoom");

pc1 = livingRoomData{1};
pc2 = livingRoomData{2};
```

Plot and set the viewpoint of point clouds.

```
figure
pcshowpair(pc1,pc2,VerticalAxis="Y",VerticalAxisDir="Down")
title("Difference Between Two Point Clouds")
xlabel("X(m)")
ylabel("Y(m)")
zlabel("Z(m)")
```



Input Arguments

`ptCloudA` — Point cloud
`pointCloud` object

Point cloud A, specified as a `pointCloud` object. The function uses levels of magenta to represent `ptCloudA` and a pure magenta when the point cloud contains no color information.

`ptCloudB` — Point cloud
`pointCloud` object

Point cloud B, specified as a `pointCloud` object. The function uses levels of green to represent `ptCloudB` and a pure green when the point cloud contains no color information.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `pchowpair(ptCloudA,ptCloudB,BackgroundColor="yellow")`, sets the figure background color to yellow.

`MarkerSize` — Diameter of marker
`6` (default) | positive scalar

Approximate diameter of the point marker, specified as a positive scalar. The units are in points. A marker size larger than six can reduce the rendering performance.

`BackgroundColor` — Background color
`[0 0 0]` (black) (default) | RGB triplet | hexadecimal color code | character vector

Background color, specified as an RGB triplet, hexadecimal color code, or a character vector that specifies a long or short color name.

`VerticalAxis` — Vertical axis
`"Z"` (default) | `"X"` | `"Y"`

Vertical axis, specified as `"X"`, `"Y"`, or `"Z"`. When you reload a saved figure, any action on the figure resets the vertical axis to the *z*-axis.

`VerticalAxisDir` — Vertical axis direction
`"Up"` (default) | `"Down"`

Vertical axis direction, specified as `"Up"` or `"Down"`. When you reload a saved figure, any action on the figure resets the direction to the up direction.

`Projection` — Camera projection for 3-D views
`"perspective"` (default) | `"orthographic"`

Camera projection for 3-D views, specified as one of these values:

- `"perspective"` — Projects the viewing volume as the frustum of a pyramid (a pyramid whose apex has been cut off parallel to the base). Objects further from the camera appear smaller. Distance causes foreshortening, which enables you to perceive the depth of 3-D objects. This projection type is useful when you want to display realistic views of real objects. Perspective projection does not preserve the relative dimensions of objects. Instead, it displays a distant line segment smaller than a nearer line segment of the same length. Lines that are parallel in the data might not appear parallel in the scene.
- `"orthographic"` — This projection type maintains the correct relative dimensions of graphics objects regarding the distance of a given point from the viewer. Relative distance from the camera does not affect the size of objects. Lines that are parallel in the data parallel appear parallel on the screen. This projection type is useful when it is important to maintain the actual size of objects and the angles between objects.

`ViewPlane` — Plane to visualize point cloud
`"XY"` (default) | `"YX"` | `"XZ"` | `"ZX"` | `"YZ"` | `"ZY"`

Plane to visualize point cloud, specified as `"XY"`, `"YX"`, `"XZ"`, `"ZX"`, `"YZ"`, `"ZY"`. The `ViewPlane` sets the line of site from the camera, centered in the selected plane, to the center of the plot.

`ColorSource` — Point cloud property for color map
`"auto"` (default) | `"X"` | `"Y"` | `"Z"` | `"Intensity"` | `"Color"` | `"Row"` | `"Column"`

Point cloud property for color map, specified as `"auto"`,`"X"`, `"Y"`, `"Z"`, `"Intensity"`, `"Color"`, `"Row"`, `"Column"`. The `pcshow` function uses the value of `ColorSource` to assign a color map for rendering the point cloud. The values use these properties:

- `"X"` — *x* coordinates in the `Location` property of the input point cloud.
- `"Y"` — *y* coordinates in the `Location` property of the input point cloud.
- `"Z"` — *z* coordinates in the `Location` property of the input point cloud.
- `"Intensity"` — `Intensity` property of the input point cloud.

- `"Color"` — `Color` property of the input point cloud.
- `"Row"` — Using the row values of an organized point cloud.
- `"Column"` — Using the column values of an organized point cloud.

When you set `ColorSource` to `"auto"`, the function sets the color map depending on which property values are available in the input point cloud object:

Available Point Cloud Property	Color Source
`Location` only	Maps the Z-value to a color value in the current color map.
`Location` and `Intensity`	Maps the z-value to a color value in the current color map.
`Location` and `Color`	Maps the color values to the current color map.
`Location`, `Intensity`, and `Color`	Maps the color values to the current color map.

`AxesVisibility` — State of axes visibility
`"on"` or `1` (`true`) (default) | `"off"` or `0` (`false`)

State of axes visibility, specified as `"on"` or `"off"`, or as numeric or logical `0` (`false`) or `1` (`true`). A value of `"on"` is equivalent to `true`, and `"off"` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `"on"` — Display the axes and its children.
- `"off"` — Hide the axes without deleting it. You still can access the properties of an invisible axes object. (Child objects such as lines remain visible.)

`Parent` — Axes on which to display the visualization
`axes` graphics object

Axes on which to display the visualization, specified as an `Axes` object. To create an `Axes` object, use the `axes` function. To display the visualization in a new figure, leave `"Parent"` unspecified.

Output Arguments

ax — Plot axes
`axes` graphics object

Plot axes, returned as an `axes` graphics object. Points with `NaN` or `Inf` coordinates are not displayed.

You can set the default center of rotation for the point cloud viewer to rotate around the axes center or around a point. Set the default behavior from the “Computer Vision Toolbox Preferences”.

More About

Axes Toolbar Operations

Keyboard shortcuts are enabled only for the rotate (default) mode. To view the axes toolbar operations, point to the figure and select one of these options. Click the Restore View icon to reset the display.



Feature	Description
Vertical Axis	Select an *X*-, *Y*-, or *Z*-direction to set the direction of that axis of the point cloud as the positive vertical direction. Use the keyboard arrow keys to move the point cloud with respect to the direction selected.
View Plane	Select the viewing angle of the point cloud figure as the *XY*, *YX*, *XZ*, *ZX*, *YZ*, or *ZY* plane.
Background Color	Display a color chart from which you can select a background color.
Axes On/Off	Turn the display axes on or off.
Projection	Select perspective or orthogonal view.
Export	• — Save the content as a tightly cropped image or PDF. • — Copy the content as an image. • — Copy the content as a vector graphic.
Brush	Enable data brushing mode.
Data Tips	View the point data values of a point by selecting it in the point cloud figure. The Data Tips display shows the *x*, *y*, and *z* values of the points. Point clouds for depth images and lidar contain these additional data properties:

Point Cloud Data Source	Data Value Properties
Depth image (RGB-D sensor)	**Color**, **Row**, and **Column**
Lidar	**Intensity**, **Range**, **Azimuth** angle, **Elevation** angle, **Row**, and **Column**

<table>
<tr><th>Feature</th><th>Description</th></tr>
<tr><td>Rotate 3D</td><td>Drag to rotate the point cloud figure. You can also right-click any point on the point cloud figure and select Rotate Around a Point to specify that point as the center of rotation for the figure. To return to axes rotation, right-click again and select Rotate Around Axes Center.

While in Rotate 3D mode, you can also select the color source for the point cloud by right-clicking anywhere in the point cloud figure and selecting Color Source, and then selecting an option from the list. The figure applies the active colormap to the point cloud based on the values in the selected source. For a point cloud of only position data, the Color Source options are X, Y, Z, and User Specified. Point clouds for depth image and lidar data include these additional color source options:

<table>
<tr><th>Point Cloud Data Source</th><th>Color Source Options</th></tr>
<tr><td>Depth image (RGB-D sensor)</td><td>Color, Row, and Column</td></tr>
<tr><td>Lidar</td><td>Intensity, Range, Azimuth angle, Elevation angle, Row, and Column</td></tr>
</table>
</td></tr>
<tr><td>Pan, Zoom in, and Zoom out</td><td>Maneuver the point cloud in the figure, or zoom the point cloud figure in or out.</td></tr>
<tr><td>Restore View</td><td>Restore the original view of the axes.</td></tr>
</table>

Move Camera Through Scene

To navigate into a point cloud and observe the scene, use these keyboard shortcuts:

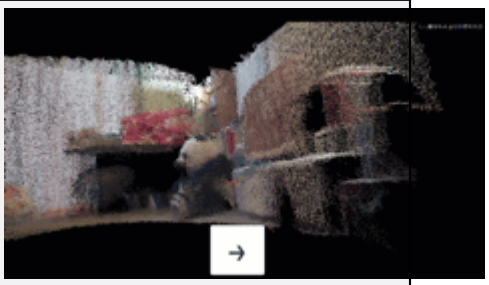
<table>
<tr><th>Action</th><th>Keyboard Shortcut</th><th>Example</th></tr>
<tr><td>Move forward or backward through the scene. The camera and the target can move.</td><td>w — Forward

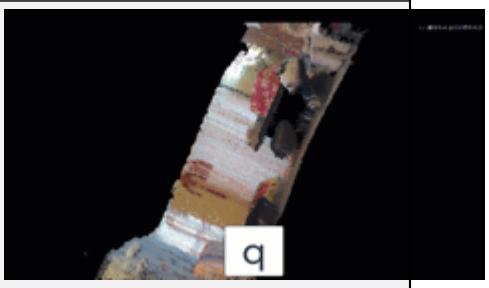
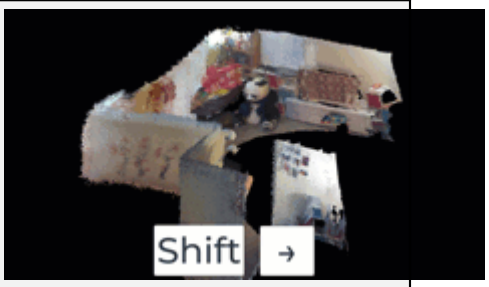
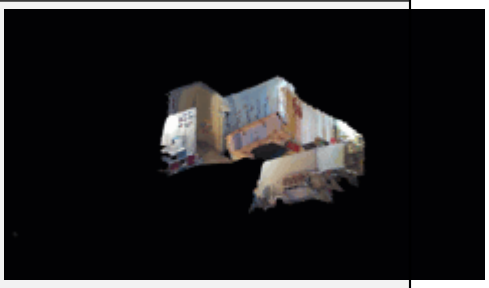
s — Backward</td><td rowspan="2">
</td></tr>
<tr><td>Move left or right through the scene. The camera and the target can move.</td><td>a — Left

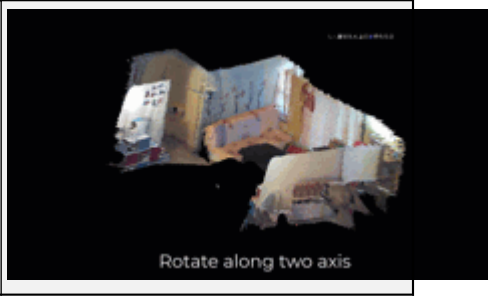
d — Right</td></tr>
<tr><td>Look around the scene (pitch/yaw rotation of the camera). Camera position is fixed, target moves.</td><td>Up arrow — Camera pitches up

Down arrow — Camera pitches down

Left arrow — Camera turns left in yaw

Right arrow — Camera turns right in yaw</td><td></td></tr>
</table>

Action	Keyboard Shortcut	Example
Roll the scene (roll rotation of the camera).	**q** — Camera moves clockwise. (Scene appears to move counterclockwise.) **e** — Camera moves counterclockwise. (Scene appears to move clockwise.)	
Rotate the scene (rotate the camera around the point cloud). Camera position changes, target is fixed.	**Shift** + **Left** or **Right** — Camera orbits horizontally around the point cloud. (Scene appears to stand still as camera moves externally around it horizontally.) **Shift** + **Up** or **Down** — Camera orbits vertically around the point cloud. (Scene appears to stand still as camera moves externally around it vertically.)	
Zoom in or out.	**z** — Zoom in **x** — Zoom out	
Look at a view plane.	1 — **XY** 2 — **YX** 3 — **XZ** 4 — **ZX** 5 — **YZ** 6 — **ZY** Camera positioned in the selected plane with a line of sight to the center of the plot.	

Action	Keyboard Shortcut	Example
Switch between rotate and pan.	**Space bar** — Press and hold the space bar to switch the interaction to pan. Release the space bar to switch back to rotate.	
Rotate in X-, Y-, and Z-dimensions.	**T** — Adds rotate interaction in the third dimension.	

Color Formats

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Performance and Tips

- To improve performance, the function automatically downsamples the rendered point cloud when you rotate, pan, or zoom in or out by clicking the mouse. Downsampling is not applied for zoom in or out by using the mouse scroll mechanism. When using data tips, the figure does not apply downsampling. The downsampling occurs only for rendering the point cloud and does not affect the saved points.
- The keyboard shortcuts described in “Move Camera Through Scene” on page 3-1210 are not supported when loading a saved figure or for popped-out windows using Live Editor.
- When you reload a saved figure, any action on the figure resets the vertical axis to the *z*-axis and in the up direction.

- To visualize a large point cloud (greater than one million points), use the `pcviewer` object.

Version History

Introduced in R2015b

R2022b: Projection display default behavior changed to perspective

The default projection for the camera 3-D view display is changed from orthographic to perspective. You can use the new `Projection` name-value argument to specify the projection as `"perspective"` or `"orthographic"`. Prior to R2022b, the display rendered the point cloud in an orthographic view only.

See Also

Objects
`pcviewer` | `pointCloud` | `pcplayer` | `planeModel`

Functions
`showShape` | `pcshow` | `plot3` | `pcwrite` | `pcread` | `pcmerge` | `scatter3` | `reconstructScene` | `triangulate` | `pcdownsample` | `pcfitplane` | `pcdenoise` | `pcregistericp`

Topics
“Structure from Motion from Two Views”
“Depth Estimation from Stereo Video”
“Choose Function to Visualize Detected Objects”
“Coordinate Systems”

showReprojectionErrors

Visualize calibration errors

Syntax

```
showReprojectionErrors(cameraParams)
showReprojectionErrors(cameraParams,viewStyle)
showReprojectionErrors( ___ ,Name=Value)

ax = showReprojectionErrors( ___ )
```

Description

`showReprojectionErrors(cameraParams)` displays a bar graph that represents the calibration accuracy for a single camera, a stereo pair, a fisheye camera, or a multi-camera system. For non-multi-camera systems, the bar graph shows the mean reprojection error for each image. For multi-camera systems, the graph is divided into three sub-plots, displaying the mean reprojection error per image, per view, and per camera. A view represents a specific position where the calibration pattern is observed by the cameras. The `cameraParams` object contains one of:

- `cameraParameters` or `stereoParameters` object which the `estimateCameraParameters` function returns.
- `fisheyeParameters` or object which the `estimateFisheyeParameters` function returns.
- `multiCameraParameters` object which the `estimateMultiCameraParameters` function returns.

`showReprojectionErrors(cameraParams,viewStyle)` displays the reprojection errors using the visualization style specified by the `viewStyle` input. The `viewStyle` argument is not supported for a multi-camera system.

`showReprojectionErrors( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `HighlightIndex = [1,4]` sets the highlight patterns to 1 and 4.

`ax = showReprojectionErrors( ___ )` returns the plot axis, using any of the preceding syntaxes.

Examples

Visualize Reprojection Errors for a Single Camera

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
  "calibration","dslr"));
imageFileNames = images.Files;
```

Detect calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate world coordinates of the corners of the squares. The square size is in millimeters.

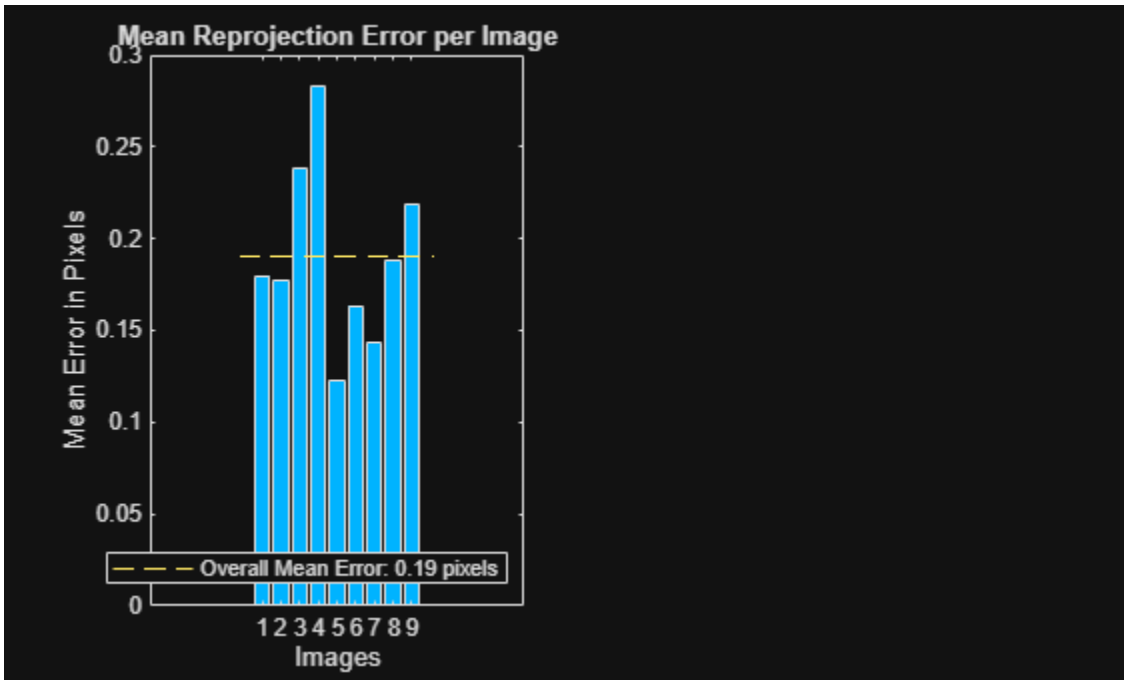
```
squareSize = 23; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,1);
imageSize = [size(I, 1), size(I, 2)];
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

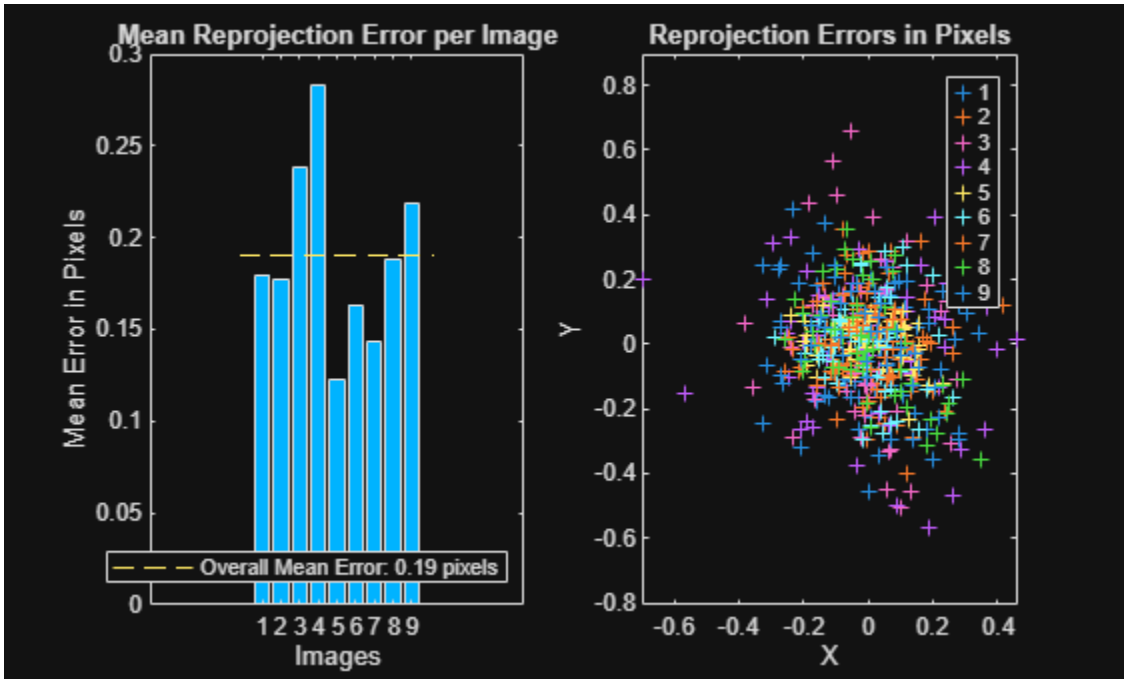
Visualize the errors as a bar graph.

```
subplot(1,2,1);
showReprojectionErrors(params);
```



Visualize the errors as a scatter plot.

```
subplot(1,2,2);
showReprojectionErrors(params,"ScatterPlot");
```



Visualize Reprojection Errors for a Stereo Pair of Cameras

Specify calibration images

```
imageDir = fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","stereo");
leftImages = imageDatastore(fullfile(imageDir,"left"));
rightImages = imageDatastore(fullfile(imageDir,"right"));
```

Detect the checkerboards.

```
[imagePoints,patternDims] = detectCheckerboardPoints(...
     leftImages.Files,rightImages.Files);
```

Specify world coordinates of checkerboard keypoints. The square size is in millimeters.

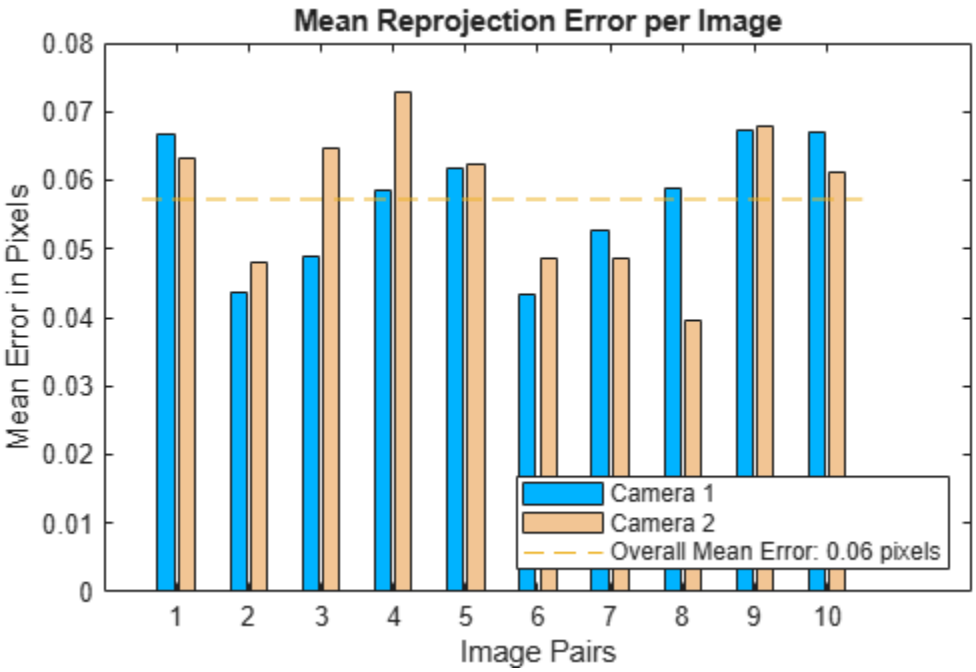
```
squareSize = 108; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the stereo camera system. Here both cameras have the same resolution.

```
I = readimage(leftImages,1);
imageSize = size(I,1:2);
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

Visualize calibration accuracy.

```
showReprojectionErrors(params);
```



Calibrate a Multi-Camera System using a ChArUco board

Estimate the relative position and orientation of six cameras with overlapping fields of view by using calibration images that contain a single ChArUco board.

Download the calibration images.

```
calibImagesURL = "https://www.mathworks.com/supportfiles/vision/data/overlapping-cameras-charuco
calibImagesDir = fullfile(pwd,"overlapping-cameras-charuco");
calibImagesZip = fullfile(pwd,"overlapping-cameras-charuco.zip");
if ~exist(calibImagesZip,"file")
    disp("Downloading calibration images (52 MB)...")
    websave(calibImagesZip,calibImagesURL);
end
```

```
Downloading calibration images (52 MB)...
```

```
if ~exist(calibImagesDir,"dir")
    unzip(calibImagesZip,pwd)
end
```

Specify calibration image filenames for each camera.

```
numCameras = 6;
camDirPrefix = "Cam00";
imageFiles = cell(1,numCameras);
for i = 1:numCameras
    camDir = fullfile(calibImagesDir,camDirPrefix+i);
    imds = imageDatastore(camDir);
    imageFiles{i} = imds.Files;
```

```
end
imageFiles = [imageFiles{:}];
```

Define the ChArUco board properties. Specify checker size and marker size in centimeters.

```
markerFamily = "DICT_6X6_1000";
patternDims = [5 5];
markerSize = 6.8;   % in cm
checkerSize = 9.15; % in cm
numKeyPoints = prod(patternDims - 1);
minMarkerId = 144;
```

Detect the key points of the ChArUco board in the calibration images.

```
imagePoints = detectPatternPoints(imageFiles,"charuco-board",patternDims, ...
    markerFamily,checkerSize,markerSize,MinMarkerID=minMarkerId);
```

```
[==================================================] 100%
Elapsed time: 00:00:09
Estimated time remaining: 00:00:00
```

Generate the world points for the pattern.

```
worldPoints = patternWorldPoints("charuco-board",patternDims,checkerSize);
```

Load the intrinsic parameters of the six cameras. These parameters have been estimated using the “Using the Single Camera Calibrator App”.

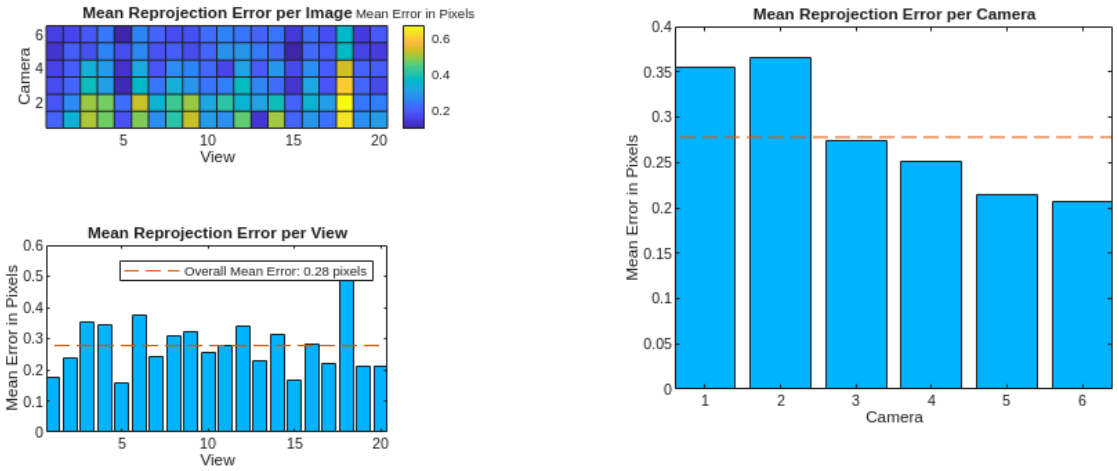
```
ld = load("sixCameraIntrinsics.mat");
```

Perform multi-camera calibration.

```
params = estimateMultiCameraParameters(imagePoints,worldPoints,ld.intrinsics,WorldUnits="cm");
```

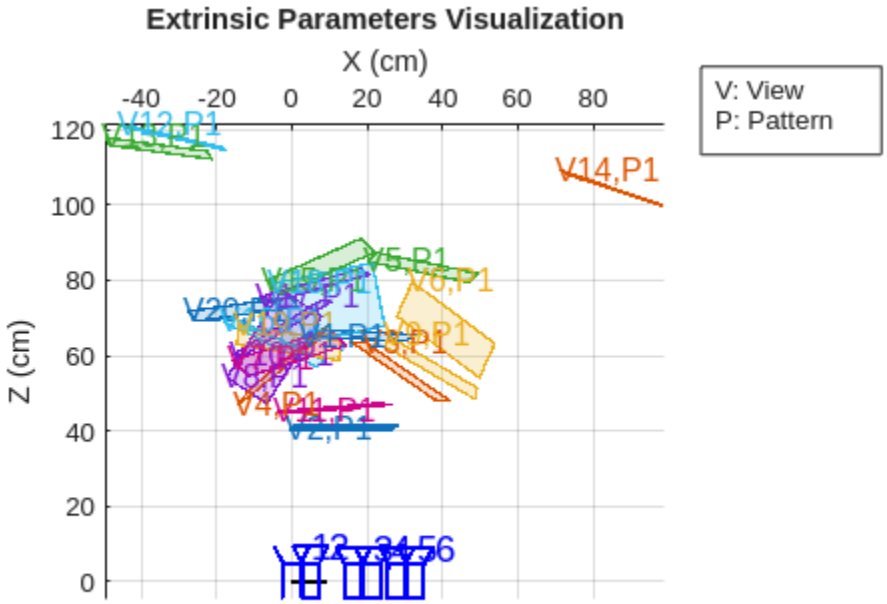
Visualize the calibration accuracy.

```
figure(Position=[100,100,1000,400])
showReprojectionErrors(params)
```



Visualize the camera extrinsic parameters.

```
figure
showExtrinsics(params)
view(2)
```



Input Arguments

`cameraParams` — Object containing parameters of single camera or stereo pair
`cameraParameters` object | `fisheyeParameters` object | `stereoParameters` object | `multiCameraParameters` object

Object containing parameters for a single, fisheye, stereo pair, or multi-camera system, specified as either a `cameraParameters`, `fisheyeParameters`, `stereoParameters`, `multiCameraParameters` object. You can create these objects using the:

- `estimateCameraParameters` function to return a `cameraParameters` or `stereoParameters` object.
- `estimateFisheyeParameters` function to return a `fisheyeParameters` object.
- `estimateMultiCameraParameters` function to return a `multiCameraParameters` object.

You can also use the Camera Calibrator app to create the `cameraParameters` input object, or use Stereo Camera Calibrator app to create the `stereoParameters` input object. See “Using the Single Camera Calibrator App” and “Using the Stereo Camera Calibrator App”.

`viewStyle` — Plot view
`"BarGraph"` | `"ScatterPlot"`

Plot view, specified as either `"BarGraph"` or `"ScatterPlot"`. The `viewStyle` input sets the visualization for the camera extrinsic parameters. Set `viewStyle` to `"BarGraph"` to display the

mean error per image as a bar graph. Set `viewStyle` to `"ScatterPlot"` to display the error for each point as a scatter plot. The `"ScatterPlot"` option applies only to the single camera case. This argument is not supported for a multi-camera system.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `showReprojectionErrors(cameraParams,HighlightIndex = [1,4])`, sets the highlight selection index to patterns 1 and 4.

`HighlightIndex` — Highlight selection index

`[]` (default) | vector of integers | scalar

Highlight selection index, specified as a scalar or a vector of integers. When you set the `viewStyle` to `"BarGraph"`, the function highlights the bars corresponding to the selected images. When you set the `viewStyle` to `"ScatterPlot"`, the function highlights the points corresponding to the selected images with circle markers. This argument is not supported for a multi-camera system.

`Parent` — Output axes

current axes (default) | scalar value

Output axes, specified as the comma-separated pair consisting of '`Parent`' and a scalar value. Specify output axes to display the visualization. You can obtain the current axes handle by returning the function to an output variable:
`ax = showReprojectionErrors(cameraParams)`
You can also use the `gca` function to get the current axes handle.

Example: `showReprojectionErrors(cameraParams,'Parent',ax)`

Output Arguments

`ax` — Current axes handle

scalar value

Current axes handle, returned as a scalar value. The function returns the handle to the current axes for the current figure.

Example: `ax = showReprojectionErrors(cameraParams)`

Version History

Introduced in R2014a

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions
estimateMultiCameraParameters | plotCamera | estimateCameraParameters | showExtrinsics | undistortImage | detectCheckerboardPoints | generateCheckerboardPoints | patternWorldPoints | detectPatternPoints

Objects
multiCameraParameters | cameraParameters | stereoParameters | fisheyeParameters

Topics
“Using the Single Camera Calibrator App”
“3-D Motion Reconstruction Using Multiple Cameras”

stereoAnaglyph

Create red-cyan anaglyph from stereo pair of images

Syntax

```
J = stereoAnaglyph(I1,I2)
```

Description

`J = stereoAnaglyph(I1,I2)` combines images `I1` and `I2` into a red-cyan anaglyph. When the inputs are rectified stereo images, you can view the output image with red-blue stereo glasses to see the stereo effect.

Examples

Create 3-D Stereo Display

Load parameters for a calibrated stereo pair of cameras.

```
load('webcamsSceneReconstruction.mat')
```

Load a stereo pair of images.

```
I1 = imread('sceneReconstructionLeft.jpg');
I2 = imread('sceneReconstructionRight.jpg');
```

Rectify the stereo images.

```
[J1, J2] = rectifyStereoImages(I1, I2, stereoParams);
```

Create the anaglyph.

```
A = stereoAnaglyph(J1, J2);
```

Display the anaglyph. Use red-blue stereo glasses to see the stereo effect.

```
figure; imshow(A);
```



Input Arguments

`I1` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-*D* grayscale image

Input image `I1`, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-*D* grayscale image. `I1` and `I2` must be real, finite, and nonsparse, and the images must be the same size. If the images are not the same size, use `imfuse` to pad the smaller image dimension with zeros before creating the anaglyph.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`I2` — Input image

M-by-*N* 2-*D* grayscale image | *M*-by-*N*-by-3 truecolor image

Input image `I2`, specified as an *M*-by-*N*-by-3 truecolor image or an *M*-by-*N* 2-*D* grayscale image. `I1` and `I2` must be real, finite, and nonsparse, and the images must be the same size. If the images are not the same size, use `imfuse` to pad the smaller image dimension with zeros before creating the anaglyph.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Output Arguments

`J` — Stereo anaglyph output image

M-by-*N*-by-3 truecolor image

Stereo anaglyph output image, returned as an *M*-by-*N*-by-3 truecolor image. Output image `J` is the same size as input images `I1` and `I2`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`rectifyStereoImages` | `reconstructScene` | `estimateUncalibratedRectification` | `imfuse` | `imshowpair`

Topics

“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”

tform

Extract homogeneous transformation

Syntax

```
transformationMatrix = tform(transformation)
transformationMatrix = tform(rotation)
```

Description

`transformationMatrix = tform(transformation)` extracts the homogeneous transformation matrix `transformationMatrix` that corresponds to the SE(3) transformation `transformation`.

`transformationMatrix = tform(rotation)` creates a homogeneous transformation matrix `transformationMatrix`, with zero translation, that corresponds to the SO(3) rotation `rotation`.

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of transformation objects

Transformation, specified as a scalar `se3` object or an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformation` as an array, each element must be of the same type.

`rotation` — Rotation

`so3` object | *N*-element array of rotation objects

Rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotation` as an array, each element must be of the same type.

Output Arguments

`transformationMatrix` — Homogeneous transformation matrix

3-by-3-by-*N* array | 4-by-4-by-*N* array

Homogeneous transformation matrix, returned as a 4-by-4-by-*N* array for `se3` and `so3` objects. *N* is the total number of transformations.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`rotm` | `trvec`

Objects
`se3` | `so3`

trainCascadeObjectDetector

Train cascade object detector model

Syntax

```
trainCascadeObjectDetector(outputXMLFilename,positiveInstances,
negativeImages)
trainCascadeObjectDetector(outputXMLFilename,'resume')

trainCascadeObjectDetector( ___ ,Name=Value)
```

Description

`trainCascadeObjectDetector(outputXMLFilename,positiveInstances, negativeImages)` writes a trained cascade detector XML file named, `outputXMLFilename`. The file name must include an XML extension. For a more detailed explanation on how this function works, refer to “Get Started with Cascade Object Detector”.

`trainCascadeObjectDetector(outputXMLFilename,'resume')` resumes an interrupted training session. The `outputXMLFilename` input must match the output file name from the interrupted session. All arguments saved from the earlier session are reused automatically.

`trainCascadeObjectDetector( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ObjectTrainingSize=[100,100]` sets the height and width of objects during training.

Examples

Train Stop Sign Detector

Load the positive samples data from a MAT file. The file contains the ground truth, specified as table of bounding boxes for several object categories. The ground truth was labeled and exported from the Image Labeler app.

```
load("stopSignsAndCars.mat");
```

Prefix the fullpath to the stop sign images.

```
stopSigns = fullfile(toolboxdir("vision"),"visiondata",stopSignsAndCars{:,1});
```

Create datastores to load the ground truth data for stop signs.

```
imds = imageDatastore(stopSigns);
blds = boxLabelDatastore(stopSignsAndCars(:,2));
```

Combine the image and box label datastores.

```
positiveInstances = combine(imds,blds);
```

Add the image folder path to the MATLAB® path.

```
imDir = fullfile(matlabroot,"toolbox","vision","visiondata","stopSignImages");
addpath(imDir);
```

Specify a folder for negative images.

```
negativeFolder = fullfile(matlabroot,"toolbox","vision","visiondata","nonStopSigns");
```

Create an imageDatastore object containing negative images.

```
negativeImages = imageDatastore(negativeFolder);
```

Train a cascade object detector called "`stopSignDetector.xml`" using HOG features. NOTE: The command can take a few minutes to run.

```
trainCascadeObjectDetector("stopSignDetector.xml",positiveInstances,negativeFolder,FalseAlarmRate
```

```
Automatically setting ObjectTrainingSize to [35, 32]
Using at most 42 of 42 positive samples per stage
Using at most 84 negative samples per stage

--cascadeParams--
Training stage 1 of 3
[..................................................................]
Used 42 positive and 84 negative samples
Time to train stage 1: 0 seconds

Training stage 2 of 3
[..................................................................]
Used 42 positive and 84 negative samples
Time to train stage 2: 1 seconds

Training stage 3 of 3
[..................................................................]
Used 42 positive and 84 negative samples
Time to train stage 3: 3 seconds

Training complete
```

Use the newly trained classifier to detect a stop sign in an image.

```
detector = vision.CascadeObjectDetector("stopSignDetector.xml");
```

Read the test image.

```
img = imread("stopSignTest.jpg");
```

Detect a stop sign in the test image.

```
bbox = step(detector,img);
```

Insert bounding box rectangles and return the marked image.

```
detectedImg = insertObjectAnnotation(img,"rectangle",bbox,"stop sign");
```

Display the detected stop sign.

```
figure
imshow(detectedImg)
```



Remove the image folder from the path.

```
rmpath(imDir);
```

Input Arguments

`positiveInstances` — Positive samples

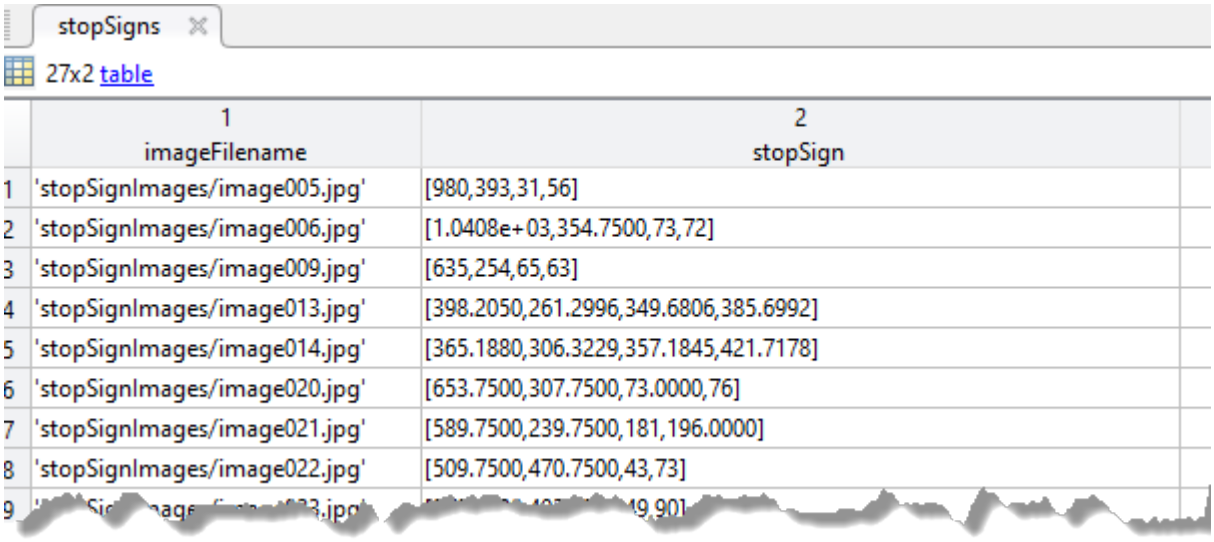
datastore | table

Positive samples, specified as a datastore or a two-column table.

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with at least two columns. The table describes the data contained in the columns:

Images	*boxes*	*labels (optional)*
Cell vector of grayscale or RGB images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	Cell array that contains an *M*-element categorical vector containing object class names. All categorical data returned by the datastore must contain the same categories. When you provide this data, the function uses the class label to fill the `ClassificationModel` property of the trained detector, specified as a `vision.CascadeObjectDetector` object. Otherwise, the class labels are not required for training because the cascade object detector is a single class detector.

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.


stopSigns

27x2 table

	1 imageFilename	2 stopSign
1	'stopSignImages/image005.jpg'	[980,393,31,56]
2	'stopSignImages/image006.jpg'	[1.0408e+03,354.7500,73,72]
3	'stopSignImages/image009.jpg'	[635,254,65,63]
4	'stopSignImages/image013.jpg'	[398.2050,261.2996,349.6806,385.6992]
5	'stopSignImages/image014.jpg'	[365.1880,306.3229,357.1845,421.7178]
6	'stopSignImages/image020.jpg'	[653.7500,307.7500,73.0000,76]
7	'stopSignImages/image021.jpg'	[589.7500,239.7500,181,196.0000]
8	'stopSignImages/image022.jpg'	[509.7500,470.7500,43,73]
9		

`negativeImages` — Negative images

`ImageDatastore` object | cell array | character vector | string scalar

Negative images, specified as an `ImageDatastore` object, a path to a folder containing images, or as a cell array of image file names. Because the images are used to generate negative samples, they must not contain any objects of interest. Instead, they should contain backgrounds associated with the object.

`outputXMLFilename` — Trained cascade detector file name
character vector | string scalar

Trained cascade detector file name, specified as a character vector or a string scalar with an XML extension. For example, '`stopSignDetector.xml`'.

Data Types: `char`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `FeatureType='Haar'` specifies Haar for the type of features to use.

`ObjectTrainingSize` — Object size for training
`'Auto'` (default) | two-element vector

Training object size, specified as either a two-element [*height*, *width*] vector or as `'Auto'`. Before training, the function resizes the positive and negative samples to `ObjectTrainingSize` in pixels. If you select `'Auto'`, the function determines the size automatically based on the median width-to-height ratio of the positive instances. For optimal detection accuracy, specify an object training size close to the expected size of the object in the image. However, for faster training and detection, set the object training size to be smaller than the expected size of the object in the image.

Data Types: `char` | `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NegativeSamplesFactor` — Negative sample factor
`2` (default) | real-valued scalar

Negative sample factor, specified as a real-valued scalar. The number of negative samples to use at each stage is equal to

`NegativeSamplesFactor` × [*the number of positive samples used at each stage*].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumCascadeStages` — Number of cascade stages
`20` (default) | positive integer

Number of cascade stages to train, specified as a positive integer. Increasing the number of stages may result in a more accurate detector but also increases training time. More stages can require more training images, because at each stage, some number of positive and negative samples are eliminated. This value depends on the values of `FalseAlarmRate` and `TruePositiveRate`. More stages can also enable you to increase the `FalseAlarmRate`. See the “Get Started with Cascade Object Detector” tutorial for more details.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FalseAlarmRate` — Acceptable false alarm rate
`0.5` (default) | value in the range (0 1]

Acceptable false alarm rate at each stage, specified as a value in the range (0 1]. The false alarm rate is the fraction of negative training samples incorrectly classified as positive samples.

The overall false alarm rate is calculated using the `FalseAlarmRate` per stage and the number of cascade stages, `NumCascadeStages`:

$FalseAlarmRate^{NumCascadeStages}$

Lower values for `FalseAlarmRate` increase complexity of each stage. Increased complexity can achieve fewer false detections but can result in longer training and detection times. Higher values for `FalseAlarmRate` can require a greater number of cascade stages to achieve reasonable detection accuracy.

Data Types: `single` | `double`

`TruePositiveRate` — Minimum true positive rate
`0.995` (default) | value in the range (0,1]

Minimum true positive rate required at each stage, specified as a value in the range (0 1]. The true positive rate is the fraction of correctly classified positive training samples.

The overall resulting target positive rate is calculated using the `TruePositiveRate` per stage and the number of cascade stages, `NumCascadeStages`:

$TruePositiveRate^{NumCascadeStages}$

Higher values for `TruePositiveRate` increase complexity of each stage. Increased complexity can achieve a greater number of correct detections but can result in longer training and detection times.

Data Types: `single` | `double`

`FeatureType` — Feature type
`'HOG'` (default) | `'LBP'` | `'Haar'`

Feature type, specified as one of the following:
`'Haar'`[1] — Haar-like features
`'LBP'`[2] — Local binary patterns
`'HOG'`[3] — Histogram of oriented gradients

The function allocates a large amount of memory, especially the Haar features. To avoid running out of memory, use this function on a 64-bit operating system with a sufficient amount of RAM.

Data Types: `char`

Tips

- Training a good detector requires thousands of training samples. Processing time for a large amount of data varies, but it is likely to take hours or even days. During training, the function displays the time it took to train each stage in the MATLAB command window.
- The OpenCV HOG parameters used in this function are:

- Numbins: 9
- CellSize = `[8 8]`
- BlockSize = `[4 4]`
- BlockOverlap = `[2 2]`
- UseSignedOrientation = `false`

Version History

Introduced in R2013a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

References

[1] Viola, P., and M. Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features.” *Proceedings of the 2001 IEEE Computer Society Conference*. CVPR 2001, 1:I-511-I–518. Kauai, HI, USA: IEEE Comput. Soc, 2001.

[2] Ojala, T., M. Pietikainen, and T. Maenpaa. “Multiresolution Gray-scale and Rotation Invariant Texture Classification With Local Binary Patterns.” In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 7: 971–87, 2002. *DOI.org (Crossref)*, https://doi.org/10.1109/TPAMI.2002.1017623.

[3] Dalal, N., and B. Triggs. “Histograms of Oriented Gradients for Human Detection.” In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, 1:886–93. San Diego, CA, USA: IEEE, 2005. *DOI.org (Crossref)*, https://doi.org/10.1109/CVPR.2005.177.

[4] Lienhart, R., Kuranov, A., Pisarevsky, V.. “Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection” DAGM 2003. *Lecture Notes in Computer Science*. 2781:297-304. Springer, 2003. *DOI.org (Crossref)*, https://doi.org/10.1007/978-3-540-45243-0_39.

See Also

Apps
Image Labeler | Video Labeler

Functions
`insertObjectAnnotation` | `imrect` | `vision.CascadeObjectDetector`

Topics
“Image Category Classification Using Bag of Features”
“Get Started with the Image Labeler”
“Get Started with Cascade Object Detector”
“Multiple Object Tracking”

External Websites
Cascade Training GUI

trainImageCategoryClassifier

Train an image category classifier

Syntax

```
classifier = trainImageCategoryClassifier(imds,bag)
classifier = trainImageCategoryClassifier(imds,bag,Name,Value)
```

Description

`classifier = trainImageCategoryClassifier(imds,bag)` returns an image category classifier. The classifier contains the number of categories and the category labels for the input `imds` images. The function trains a support vector machine (SVM) multiclass classifier using the input `bag`, a `bagOfFeatures` object.

You must have a Statistics and Machine Learning Toolbox license to use this function.

This function supports parallel computing using multiple MATLAB workers. Enable parallel computing using the “Computer Vision Toolbox Preferences” dialog. To open Computer Vision Toolbox preferences, on the **Home** tab, in the **Environment** section, click **Preferences**. Select Computer Vision Toolbox.

`classifier = trainImageCategoryClassifier(imds,bag,Name,Value)` returns a `classifier` object with optional input properties specified by one or more `Name,Value` pair arguments.

Examples

Train, Evaluate, and Apply Image Category Classifier

Load two image categories.

```
setDir  = fullfile(toolboxdir('vision'),'visiondata','imageSets');
imds = imageDatastore(setDir,'IncludeSubfolders',true,'LabelSource',...
    'foldernames');
```

Split the data set into a training and test data. Pick 30% of images from each set for the training data and the remainder 70% for the test data.

```
[trainingSet,testSet] = splitEachLabel(imds,0.3,'randomize');
```

Create bag of visual words.

```
bag = bagOfFeatures(trainingSet);
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: books
* Image category 2: cups
* Selecting feature point locations using the Grid method.
```

```
* Extracting SURF features from the selected feature point locations.
** The GridStep is [8 8] and the BlockWidth is [32 64 96 128].

* Extracting features from 4 images...done. Extracted 76800 features.

* Keeping 80 percent of the strongest features from each category.

* Creating a 500 word visual vocabulary.
* Number of levels: 1
* Branching factor: 500
* Number of clustering steps: 1

* [Step 1/1] Clustering vocabulary level 1.
* Number of features          : 61440
* Number of clusters          : 500
* Initializing cluster centers...100.00%.
* Clustering...completed 55/100 iterations (~0.92 seconds/iteration)...converged in 55 iterations

* Finished creating Bag-Of-Features
```

Train a classifier with the training sets.

```
categoryClassifier = trainImageCategoryClassifier(trainingSet,bag);
```

```
Training an image category classifier for 2 categories.
--------------------------------------------------------
* Category 1: books
* Category 2: cups

* Encoding features for 4 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Evaluate the classifier using test images. Display the confusion matrix.

```
confMatrix = evaluate(categoryClassifier,testSet)
```

```
Evaluating image category classifier for 2 categories.
-------------------------------------------------------

* Category 1: books
* Category 2: cups

* Evaluating 8 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


             PREDICTED
KNOWN    | books   cups
-------------------------
books    | 0.75    0.25
cups     | 0.25    0.75

* Average Accuracy is 0.75.
```

```
confMatrix = 2×2

    0.7500    0.2500
    0.2500    0.7500
```

Find the average accuracy of the classification.

```
mean(diag(confMatrix))
```

```
ans =
0.7500
```

Apply the newly trained classifier to categorize new images.

```
img = imread(fullfile(setDir,'cups','bigMug.jpg'));
[labelIdx, score] = predict(categoryClassifier,img);
```

```
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
```

Display the classification label.

```
categoryClassifier.Labels(labelIdx)
```

```
ans = 1×1 cell array
    {'cups'}
```

Input Arguments

`imds` — Images

`imageDatastore` object

Images specified as an `imageDatastore` object.

`bag` — Bag of features

`bagOfFeatures` object

Bag of features, specified as a `bagOfFeatures` object. The object contains a visual vocabulary of extracted feature descriptors from representative images of each image category.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Verbose',true` sets `'Verbose'` to the logical `true`.

`Verbose` — Enable progress display to screen

`true` (default) | false

Enable progress display to screen, specified as the comma-separated pair consisting of 'Verbose' and the logical true or false.

LearnerOptions — Classifier options
default values of templateSVM function

Classifier options, specified as the comma-separated pair consisting of 'LearnerOptions' and the learner options output returned by the templateSVM function.

Example 3.2. Example

To adjust the regularization parameter of templateSVM and to set a custom kernel function, use the following syntax:

```
opts = templateSVM('BoxConstraint',1.1,'KernelFunction','gaussian');
classifier = trainImageCategoryClassifier(imds,bag,'LearnerOptions',opts);
```

Output Arguments

classifier — Image category classifier
imageCategoryClassifier object

Image category classifier, returned as an imageCategoryClassifier object. The function trains a support vector machine (SVM) multiclass classifier using the error correcting output codes (ECOC) framework.

Version History

Introduced in R2014b

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

References

[1] Csurka, G., C. R. Dance, L. Fan, J. Willamowski, and C. Bray *Visual Categorization with Bag of Keypoints*, Workshop on Statistical Learning in Computer Vision, ECCV 1 (1-22), 1-2.

Extended Capabilities

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set 'UseParallel' to true or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

imageSet | bagOfFeatures | imageCategoryClassifier | fitcecoc | templateSVM

Topics
“Image Category Classification Using Bag of Features”
“Image Classification with Bag of Visual Words”

transform

Apply rigid body transformation to points

Syntax

```
tpoints = transform(transformation,points)
tpoints = transform(rotation,points)
tpoints = transform( ___ ,isCol=format)
```

Description

`tpoints = transform(transformation,points)` applies the rigid body transformation `transformation` to the input points `points`, and returns the transformed points `tpoints`.

`tpoints = transform(rotation,points)` applies the rotation `rotation` to the input points `points`, and returns the transformed points `tpoints`.

`tpoints = transform( ___ ,isCol=format)` sets the expected format of the input points `points` to be either column-wise or row-wise by using the logical flag `format` in addition to the input arguments from the previous syntax.

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of transformation objects

Transformation, specified as a scalar `se3` object or an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformation` as an array, each element must be of the same type.

`rotation` — Rotation

`so3` object | *N*-element array of rotation objects

Rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotation` as an array, each element must be of the same type.

`points` — Points to transform

N-by-*D*-by-*M* array | *D*-by-*N*-by-*M* array

Points to transform, specified as an *N*-by-*D*-by-*M* array, where:

- *D* is the dimension of the transformation, defined as 2 for 2-D transformations and 3 for 3-D transformations.
- *N* is the total number of input points to transform.
- *M* is the total number of transforms to perform on each point.

For 2-D transformations and rotations, each row specifies a point in the form [*X Y*]. For 3-D transformations and rotations, each row specifies a point in the form [*X Y Z*].

If you specify `format` as `true`, then you must specify `points` as a *D*-by-*N*-by-*M* array, where each column specifies a point.

Data Types: `single` | `double`

`format` — Point format
`false` or `0` (default) | `true` or `1`

Point format, specified as a logical `0` (`false`) or `1` (`true`). If you specify this argument as true, you must specify the points in points as columns. Otherwise, specify points as rows.

Example: `isCol=true`

Data Types: `logical`

Output Arguments

`tpoints` — Transformed points
N-by-*D*-by-*M* array | *D*-by-*N*-by-*M* array

Transformed points, returned as an *N*-by-*D*-by-*M* array, where:

- *D* is the dimension of the transformation, defined as 2 for 2-D transformations and rotations and 3 for 3-D transformations or rotations.
- *N* is the total number of input points to transform.
- *M* is the total number of transforms to perform on each point.

For 2-D transformations and rotations, each row specifies a point in the form [*X Y*]. For 3-D transformations and rotations, each row specifies a point in the form [*X Y Z*].

If you specify `format` as `true`, `tpoints` is returned as a *D*-by-*N*-by-*M* array, where each column specifies a point.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dist` | `interp` | `normalize`

Objects
`se3` | `so3`

Image Labeler

Label images for computer vision applications

Description

The Image Labeler app enables you to label ground truth data in a collection of images. Using the app, you can:

- Define axis-aligned or rotated rectangular regions of interest (ROI) labels, line ROI labels, pixel ROI labels, polygon ROI labels, point ROI labels, projected cuboid ROI labels, and scene labels. Use these labels to interactively label your ground truth data.
- Use built-in automation algorithms, such as Segment Anything Model or Grounding DINO, to label your ground truth data.
- Write, import, and use your own custom automation algorithm to automatically label ground truth. See “Create Custom Automation Algorithm for Labeling”.
- Evaluate the performance of your label automation algorithms using a visual summary. See “View Summary of Ground Truth Labels”.
- Export the labeled ground truth as a `groundTruth` object. You can use this object for system verification or for training an object detector or semantic segmentation network. See “Training Data for Object Detection and Semantic Segmentation”.

The Image Labeler app also enables you to create an individual- or a team labeling project. To launch the Image Labeler app, see “Open the Image Labeler App” on page 3-0 .



After launching the Image Labeler app, select one of these options to create a new labeling project.

- New Individual Project — Create a labeling project for yourself. To get started using an individual labeling project, see “Get Started with the Image Labeler”.
- New Team Project — Create a labeling project for a team with multiple users. To get started with a team-based project, see “Get Started with Team-Based Labeling”.

The Image Labeler app supports all image file formats supported by the `imread` function and additionally supports the Digital Imaging and Communication in Medicine (DICOM) format including the ability to load multiframe data such as an ultrasound video. To read additional file formats supported by the Image Labeler app, you can create an `imageDatastore` and use the `ReadFcn` property. To label 2-D or 3-D medical image data stored in the DICOM, Neuroimaging Informatics Technology Initiative (NIfTI), or nearly raw raster data (NRRD) file formats, use the Medical Image Labeler.

When loading images, if an image has a dimension larger than 8000 pixels or is a multiresolution image, the **Image Labeler** app offers you the option to convert the image into a blocked image. A *blocked image* consists of a large image that has been divided into smaller blocks that can fit in memory. Once the Image Labeler converts the large image into a blocked, you can process it in the app as you would any other image. While using blocked images enables you to process images in the app that you might not otherwise be able to, there are some limitations. For more information, see "Label Very Large High-Resolution Images Using Image Labeler".

Open the Image Labeler App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `imageLabeler`.

Programmatic Use

`imageLabeler` opens a new session of the app, enabling you to label ground truth data in images.

`imageLabeler(imageFolder)` opens the app and loads all the images from the folder named `imageFolder`.

The images in the folder can be unordered and can vary in size. To label a video, or a set of ordered images that resemble a video, use the Video Labeler app instead.

`imageLabeler(imageDatastore)` opens the app and reads all of the images from an `imageDatastore` object. The `ReadFcn` property of the `imageDatastore` object specifies how to read the data.

For example, to open the app with a collection of stop sign images:

```
stopSignsFolder = fullfile(toolboxdir("vision"),"visiondata","stopSignImages");
imds = imageDatastore(stopSignsFolder)
imageLabeler(imds)
```

`imageLabeler(projectFile)` opens the app and loads a saved app session from the project file, `projectFile`. The `projectFile` input contains the path and name of the `.prj`-file corresponding to the saved project.

To load a labeling session from a MAT file exported by a legacy version of the app (as opposed to a project file), use this syntax, specifying the path and name of the MAT file.

`imageLabeler(gTruth)` opens the app and loads a `groundTruth` object. The ground truth object data source must be an image collection or an `imageDatastore`.

More About

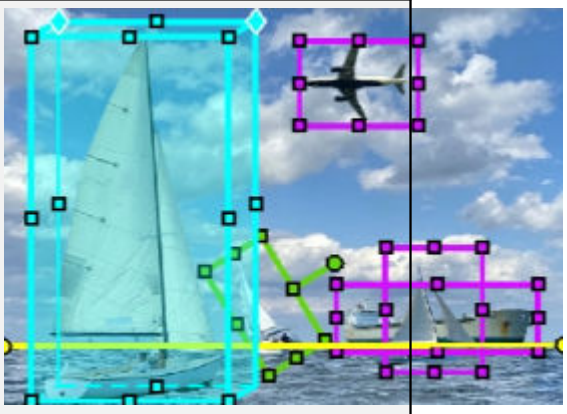
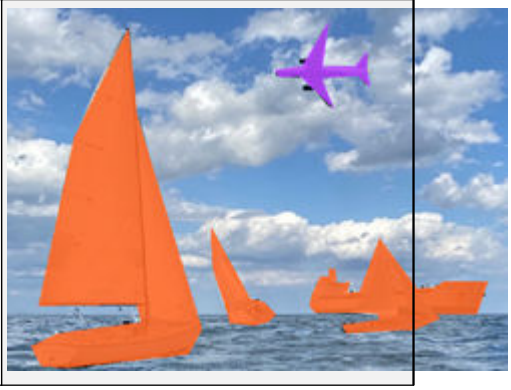
ROI Labels, Sublabels, and Attributes

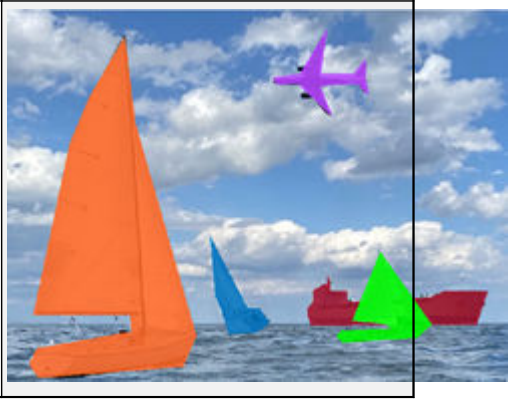
On the left side of the app, the **ROI Labels** pane contains the region of interest (ROI) label definitions that you can mark on the frames. You can create label definitions directly from this pane.

Alternatively, you can create label definitions programmatically by using a `labelDefinitionCreator` object and then import these label definitions into an app session.

Selecting ROI Label and Drawing Tool for Your Application

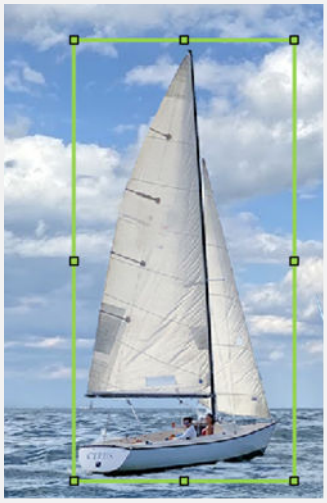
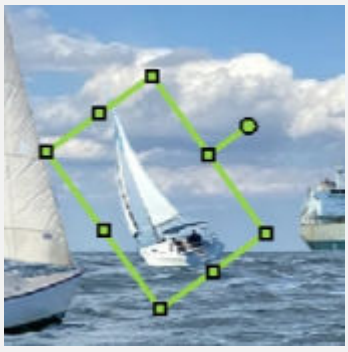
You can use labeled data to train or validate algorithms such as image classifiers, object detectors, and semantic and instance segmentation networks. Consider your application when choosing a labeling drawing tool to create ROI labels. The figure below shows labeling techniques for four different applications.

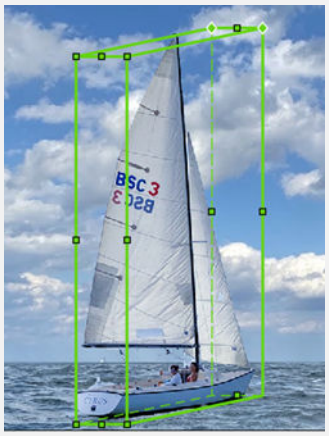
Application	Label Types	Example
Image Classification	N/A	 Boat, Plane
Object Detection	Line Rectangle Rotated Rectangle Projected Cuboid	
Semantic Segmentation	Pixel label Polygon	

Application	Label Types	Example
Instance Segmentation	Polygon	

ROI Labels

An ROI label is a label that corresponds to a region of interest (ROI) in a signal frame. The table describes the supported label types.

ROI Label	Creation	Scene
`Rectangle`	• Automatically create rectangular ROI labels (bounding boxes) by specifying descriptive text to the Grounding DINO vision-language model. To learn more, see “Automatically Label Ground Truth Using Vision-Language Model”. • Automatically create rectangular ROI labels using the Segment Anything Model (SAM) tool. • Draw rectangular ROI labels around objects using manual selection.	
`Rotated Rectangle`	Draw rotated rectangle ROI labels (bounding boxes) around objects.	

ROI Label	Creation	Scene
`Projected cuboid`	Draw cuboidal ROI labels (3-D bounding boxes).	
`Line`	Draw linear ROI labels to represent lines. To draw a line ROI, use two or more points.	
`Pixel label`	Assign labels to pixels for semantic segmentation: • Label pixels automatically using the SAM. For an example, see “Automatically Label Ground Truth Using Segment Anything Model”. • Label pixels manually using polygons, brushes, or flood fill. For more information, see “Label Pixels for Semantic Segmentation”	

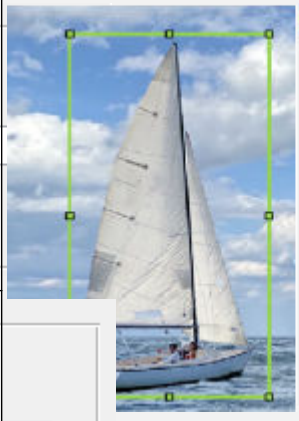
ROI Label	Creation	Scene
`Polygon`	Draw polygon ROI labels and label distinct instances of the same class using manual selection or the automatic SAM tool. For more information, see “Label Objects Using Polygons for Instance Segmentation”.	
`Point`	Draw point ROI labels for keypoint detection in objects.	 

ROI Sublabels

An ROI sublabel is an ROI label that belongs to a parent label. Use ROI sublabels to provide a greater level of detail about the ROIs in your labeled ground truth data. For example, a **vehicle** label might contain **headlight**, **licensePlate**, and **wheel** sublabels. You can create sublabels for rectangle, polygon, line, and projected cuboid labels. For more details about sublabels, see “Use Sublabels and Attributes to Label Ground Truth Data”.

ROI Attributes

An ROI attribute specifies additional information about an ROI label or sublabel. For example, in an ocean scene, attributes might include the type or color of a boat. The table describes the supported attribute types.

Attribute Type	Sample Attribute Definition	Sample Default Values
`Numeric Value`	Attributes numSails 2	
`String`	Attributes boatColor white	
`Logical`	Attributes underway True	
`List`	Attribute Name boatType List List Items (Each item must appear on a new line) Rowboat Sailboat Motorboat Cargo Tanker	

For more details on attributes, see “Use Sublabels and Attributes to Label Ground Truth Data”.

Algorithms

You can use label automation algorithms to speed up labeling within the app. To create your own label automation algorithm to use within the app, see “Create Custom Automation Algorithm for Labeling”. You can also use one of the built-in algorithms by following these steps:

1. Import the data you want to label, and create at least one label definition.
2. On the app toolstrip, click **Select Algorithm** and select one of the built-in automation algorithms.

3 Click **Automate**, and then follow the automation instructions in the right pane of the automation window.

ACF People Detector

Detect and label people using aggregate channel features (ACF). This algorithm is based on the `peopleDetectorACF` function. To use this algorithm, you must define at least one rectangle ROI label. You do not need to draw any ROI labels.

To help improve the algorithm results, first click **Settings**. You can change any of these settings.

- The pretrained people detector model that the algorithm uses — The `'inria-100x41'` model was trained using the INRIA person data set. The `'caltech-50x21'` model was trained using the Caltech Pedestrian data set.
- The overlap ratio threshold, from 0 to 1, for detecting people — When rectangle ROIs overlap by more than this threshold, the algorithm discards one of the ROIs.
- The classification score threshold for detecting people — Increase the score to increase the prediction confidence of the algorithm. Rectangles with scores below this threshold are discarded.

ACF Vehicle Detector

Detect and label vehicles using aggregate channel features (ACF). This algorithm is based on the `vehicleDetectorACF` function. To use this algorithm, you must define at least one rectangle ROI label. You do not need to draw any ROI labels.

To help improve the algorithm results, first click **Settings**. You can change any of these settings.

- The pretrained vehicle detector model that the algorithm uses — The `'full-view'` model was trained using unoccluded images of the front, rear, left, and right sides of vehicles. The `'front-rear-view'` model was trained using images of only the front and rear sides of the vehicle.
- The overlap ratio threshold, from 0 to 1, for detecting vehicles — When rectangle ROIs overlap by more than this threshold, the algorithm discards one of the ROIs.
- The classification score threshold for detecting vehicles — Increase the score to increase the prediction confidence of the algorithm. Rectangles with scores below this threshold are discarded.

You can also configure the detector with a calibrated monocular camera by importing a `monoCamera` object into the MATLAB workspace. Specify the length and width ranges of the vehicle in world units, such as meters.

Version History

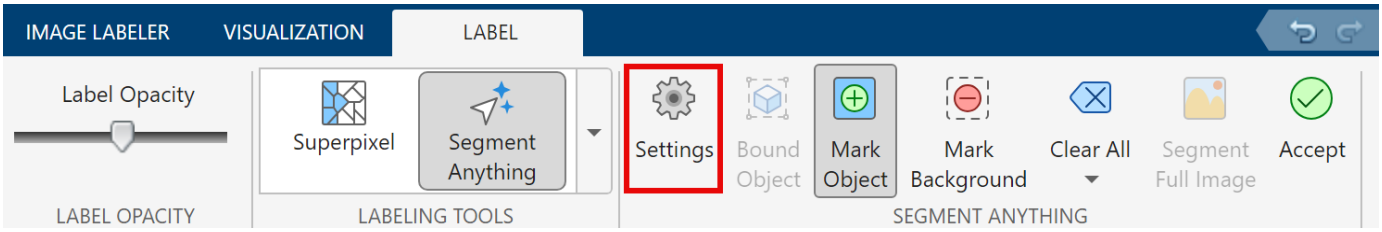
Introduced in R2018a

R2026a: Label ground truth using Segment Anything Model 2 (SAM 2)

By default, the **Segment Anything** tool now uses the Segment Anything Model 2 (SAM 2). By selecting the Settings in the **Segment Anything** section of the app toolstrip, you can specify one of these SAM model types.

- Segment Anything Model 2 - Large *(default)*
- Segment Anything Model 2 - Base

- Segment Anything Model 2 - Small
- Segment Anything Model 2 - Tiny
- Segment Anything Model - Base



This functionality requires the Image Processing Toolbox Model for Segment Anything Model 2 add-on. You can install the add-on from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”. The functionality also requires a Deep Learning Toolbox license.

R2026a: Automatically create rectangle ROI labels using Grounding DINO Vision-Language Model

Using the Grounding DINO vision-language model in the Image Labeler app, you can now automatically create rectangle ground truth ROI labels for object detection by specifying descriptive text. To use Grounding DINO for rectangle ROI labeling, select the **Grounding DINO** tool on the **Label** tab of the app toolstrip.

R2025a: Create rectangle and polygon ROI labels, and ROI-based pixel labels, using Segment Anything Model (SAM)

Using the Segment Anything Model (SAM) in the Image Labeler app, you can now interactively:

- Create rectangle ROI labels for object detection. To use the SAM for rectangle ROI labeling, select the **Segment Anything** tool on the **Label** tab of the app toolstrip.
- Create polygon ROI labels for instance segmentation. To use the SAM for polygon ROI labeling, select the **Segment Anything** tool on the **Label** tab of the app toolstrip. To learn more, see “Label Objects Using Polygons for Instance Segmentation”.
- Create pixel labels by drawing an ROI around an area to segment. To use the SAM for pixel labeling, select the **Segment Anything** tool on the **Label** tab of the Image Labeler app toolstrip. For an example, see “Label Pixels for Semantic Segmentation”.

This functionality requires the Image Processing Toolbox Model for Segment Anything Model add-on. You can install the add-on from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”. The functionality also requires Deep Learning Toolbox.

R2024b: Segment and label ground truth for semantic segmentation using Segment Anything Model (SAM)

Use the **Segment Anything** pixel labeling tool within the Image Labeler app to automatically segment images using the Segment Anything Model (SAM) and create pixel labels. This functionality requires the Image Processing Toolbox Model for Segment Anything Model add-on. You can install the add-on from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”. The functionality also requires Deep Learning Toolbox.

Assign pixel labels to objects by segmenting objects in an image with just a few clicks. Alternatively, using the **Segment Full Image** option of the **Segment Anything** tool, segment the entire image into regions and click on objects to easily create pixel labels for all the objects in a scene. For an example, see “Automatically Label Ground Truth Using Segment Anything Model”. To learn more about the SAM, see “Get Started with Segment Anything Model for Image Segmentation”.

See Also

Apps
Video Labeler | Ground Truth Labeler | Lidar Labeler | Medical Image Labeler

Functions
`pixelLabelTrainingData` | `objectDetectorTrainingData` | `polyToBlockedImage` | `groundTruthToOpenLabel`

Objects
`groundTruth` | `groundTruthDataSource` | `labelDefinitionCreator`

Topics
“Choose an App to Label Ground Truth Data”
“Get Started with the Image Labeler”
“Get Started with Team-Based Labeling”
“Label Objects Using Polygons for Instance Segmentation”
“Automatically Label Ground Truth Using Vision-Language Model”
“Automatically Label Ground Truth Using Segment Anything Model”
“Label Pixels for Semantic Segmentation”
“Label Very Large High-Resolution Images Using Image Labeler”
“Share and Store Labeled Ground Truth Data”
“Create Custom Automation Algorithm for Labeling”
“View Summary of Ground Truth Labels”
“Keyboard Shortcuts and Mouse Actions for Image Labeler”
“Training Data for Object Detection and Semantic Segmentation”
“Convert Image Labeler Polygons to Labeled Blocked Image for Semantic Segmentation”

Video Labeler

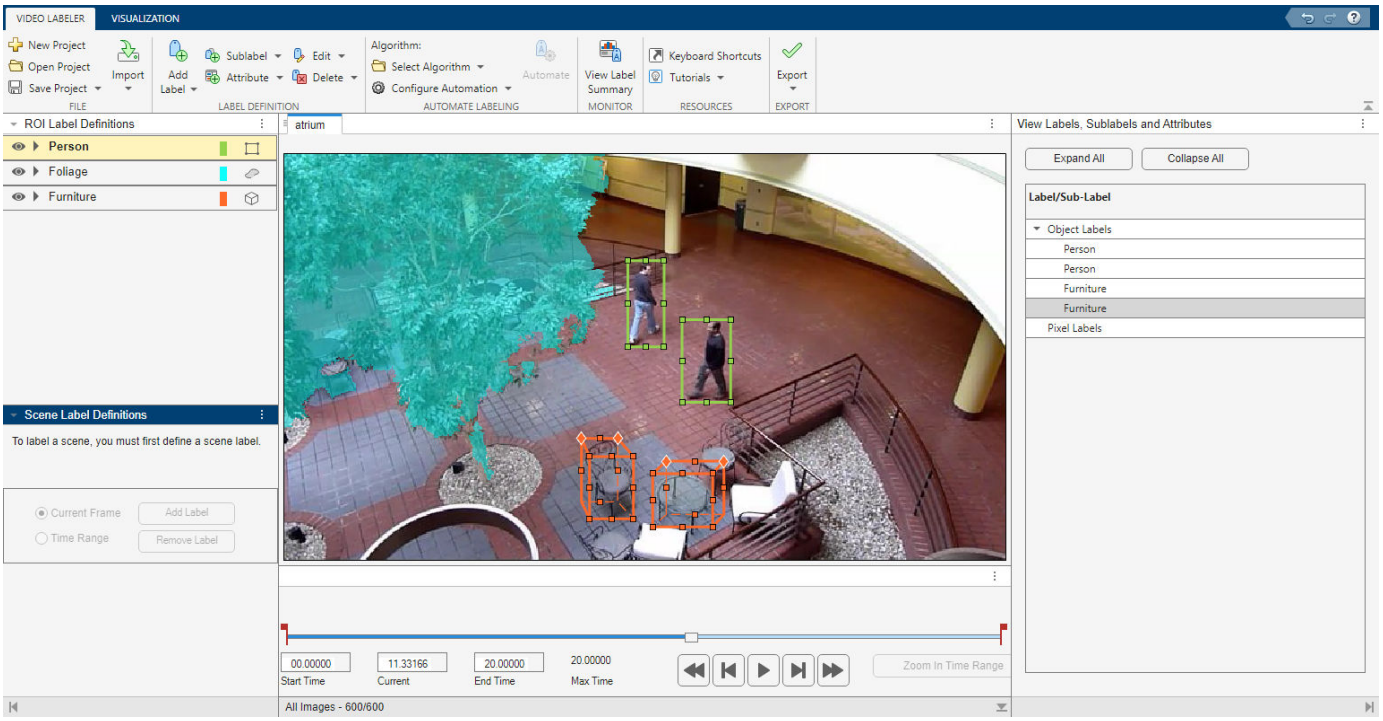
Label video for computer vision applications

Description

The Video Labeler app enables you to label ground truth data in a video, in an image sequence, or from a custom data source reader. Using the app, you can:

- Define axis-aligned and rotated rectangular regions of interest (ROI) labels, projected cuboid ROI labels, line ROI labels, polygon ROI labels, point ROI labels, pixel ROI labels, and scene labels. Use these labels to interactively label your ground truth data.
- Use built-in detection or tracking algorithms to label your ground truth data. Use the built-in Segment Anything Model or Grounding DINO tools to automatically label ground truth.
- Write, import, and use your own custom automation algorithm to automatically label ground truth. See “Create Custom Automation Algorithm for Labeling”.
- Evaluate the performance of your label automation algorithms using a visual summary. See “View Summary of Ground Truth Labels”.
- Export the labeled ground truth as a `groundTruth` object. You can use this object for system verification or for training an object detector or semantic segmentation network. See “Training Data for Object Detection and Semantic Segmentation”.

To learn more about this app, see “Get Started with the Video Labeler”.



Open the Video Labeler App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `videoLabeler`.

Examples

- "Get Started with the Video Labeler"

Programmatic Use

`videoLabeler` opens a new session of the app, enabling you to label ground truth data in a video or image sequence.

`videoLabeler(videoFileName)` opens the app and loads the input video. The video file must have an extension supported by `VideoReader`.

Example: `videoLabeler('vipmen.avi')`

`videoLabeler(imageSeqFolder)` opens the app and loads the image sequence from the input folder. An image sequence is an ordered set of images that resemble a video.

`imageSeqFolder` must be a string scalar or character vector that specifies the folder containing the image files. The image files must have extensions supported by `imformats` and are loaded in the order returned by the `dir` function.

`videoLabeler(imageSeqFolder,timestamps)` opens the app and loads a sequence of images with their corresponding timestamps. `timestamps` must be a `duration` vector of the same length as the number of images in the sequence.

For example, load a sequence of images and their corresponding timestamps into the app.

```
imageDir = fullfile(toolboxdir('vision'),'visiondata','NewTsukuba');
timeStamps = seconds(1:150);
videoLabeler(imageDir,timeStamps)
```

`videoLabeler(gtSource)` opens the app and loads the data source and corresponding timestamps from a `groundTruthDataSource` object, `gtSource`. To generate this object for a custom data source, you can specify a custom reader function. For details, see "Use Custom Image Source Reader for Labeling".

`groundTruthLabeler(projectFile)` opens the app and loads a saved app session from the project file, `projectFile`. The `projectFile` input contains the path and name of the `.prj`-file corresponding to the saved project.

To load a labeling session from a MAT file exported by a legacy version of the app (as opposed to a project file), use this syntax, specifying the path and name of the MAT file.

`videoLabeler(gTruth)` opens the app and loads a `groundTruth` object. You do not need to add a video before loading the object. The ground truth object data source must be an image sequence, video, or a custom data source.

Limitations

- The built-in automation algorithms support the automation of rectangular ROI labels and line ROI labels only. When you select a built-in algorithm and click **Automate**, scene labels, pixel ROI labels, polyline ROI labels, sublabels, and attributes are not imported into the automation session. To automate the labeling of these features, create a custom automation algorithm. See “Create Custom Automation Algorithm for Labeling”.
- Pixel ROI labels do not support sublabels or attributes.
- The Label Summary window does not support sublabels or attributes

More About

ROI Labels, Sublabels, and Attributes

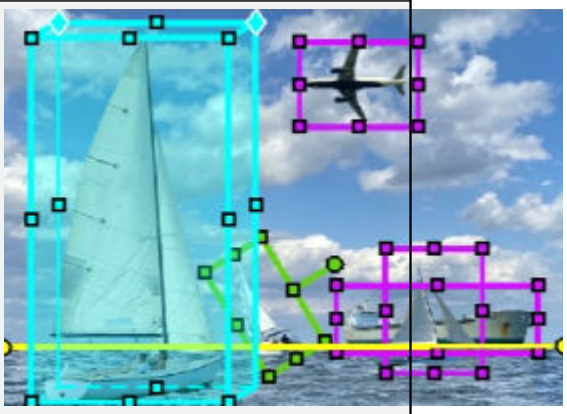
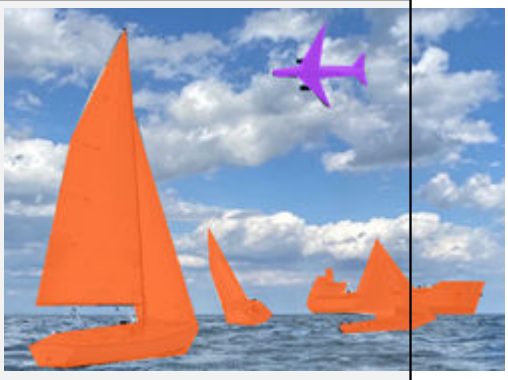
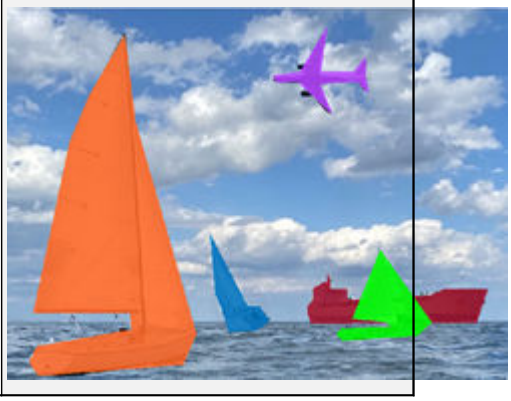
On the left side of the app, the **ROI Labels** pane contains the region of interest (ROI) label definitions that you can mark on the frames. You can create label definitions directly from this pane. Alternatively, you can create label definitions programmatically by using a `labelDefinitionCreator` object and then import these label definitions into an app session.

The app supports the definition of ROI labels, sublabels, and attributes.

Selecting ROI Label and Drawing Tool for Your Application

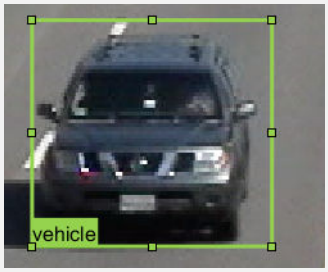
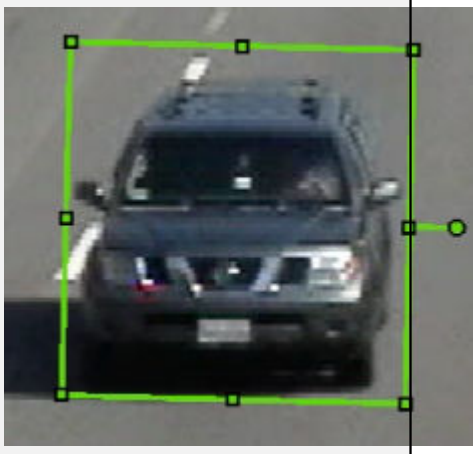
You can use labeled data to validate or train algorithms such as image classifiers, object detectors, and semantic and instance segmentation networks. Consider your application when choosing a labeling drawing tool to create ROI labels. The figure below shows labeling techniques for four different applications.

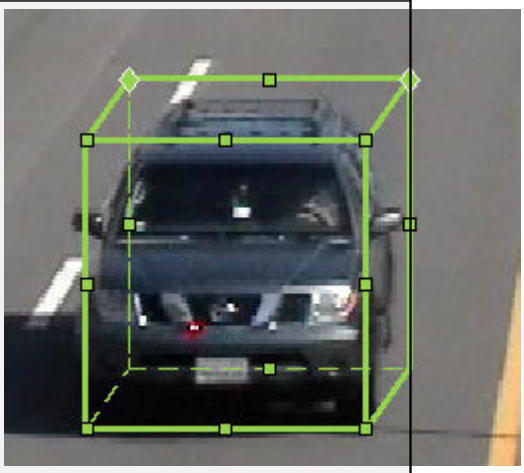
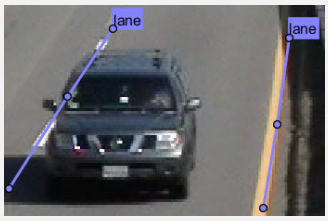
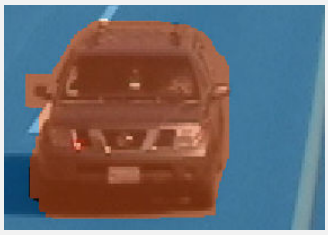
Application	Label Types	Example
Image Classification	N/A	 Boat, Plane

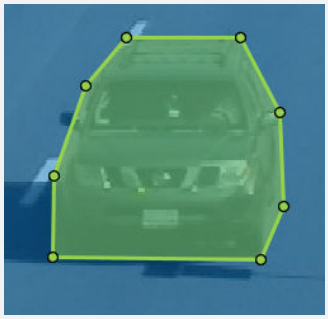
Application	Label Types	Example
Object Detection	Line Rectangle Rotated Rectangle Projected Cuboid	
Semantic Segmentation	Pixel label Polygon	
Instance Segmentation	Polygon	

ROI Labels

An ROI label is a label that corresponds to a region of interest (ROI) in a signal frame. The table describes the supported label types.

ROI Label	Description	Scene
`Rectangle`	Create axis-aligned or rotated rectangular ROI labels around objects using one of these options: • **Draw Rectangle** – Manual ROI selection • **Segment Anything** – Segment Anything Model (SAM) automatic ROI labeling tool. To learn more, see “Automatically Label Ground Truth Using Segment Anything Model”. • **Grounding DINO** – Grounding DINO automatic ROI labeling tool. To learn more, see “Automatically Label Ground Truth Using Vision-Language Model”.	Vehicles, pedestrians, road signs `Rectangle`: 
`Rotated Rectangle`	Draw rotated rectangular ROI labels around object.	`Rotated Rectangle`  :

ROI Label	Description	Scene
`Projected cuboid`	Draw cuboidal ROI labels (3-D bounding boxes).	
`Line`	Draw linear ROI labels to represent lines. To draw a polyline ROI, use two or more points.	Lane boundaries, guard rails, road curbs 
`Pixel label`	Assign labels to pixels for semantic segmentation: • Label pixels automatically using the SAM. For an example, see “Automatically Label Ground Truth Using Segment Anything Model”. • Label pixels manually using polygons, brushes, or flood fill. For more information, see “Label Pixels for Semantic Segmentation”	Vehicles, road surface, trees, pavement 

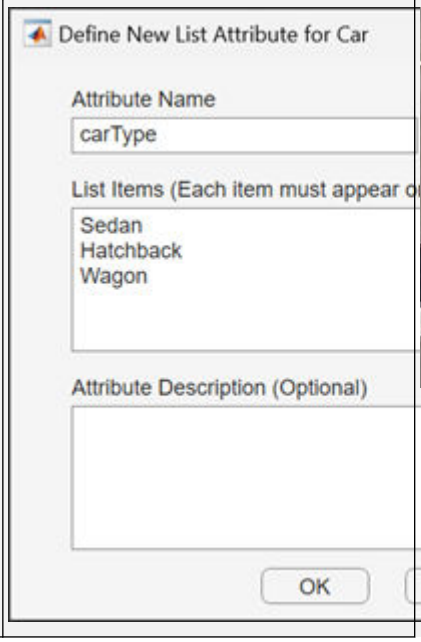
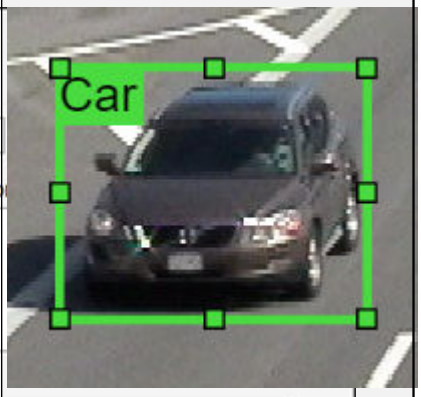
ROI Label	Description	Scene
`Polygon`	Draw polygon ROI labels and label distinct instances of the same class using manual selection or the automatic SAM tool. For more information on drawing polygon ROI labels for instance and semantic segmentation networks, see “Label Objects Using Polygons for Instance Segmentation”	Vehicles, road surface, trees, pavement 

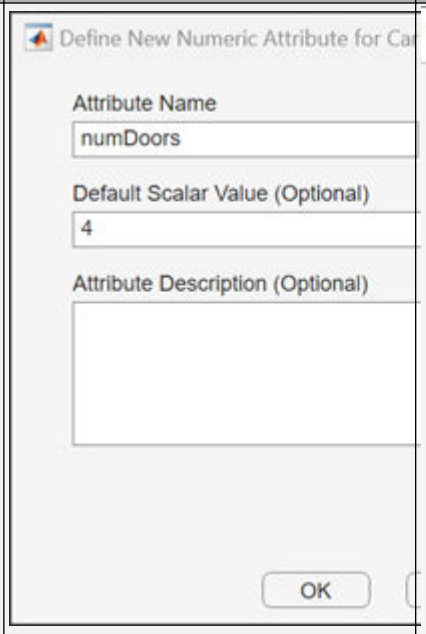
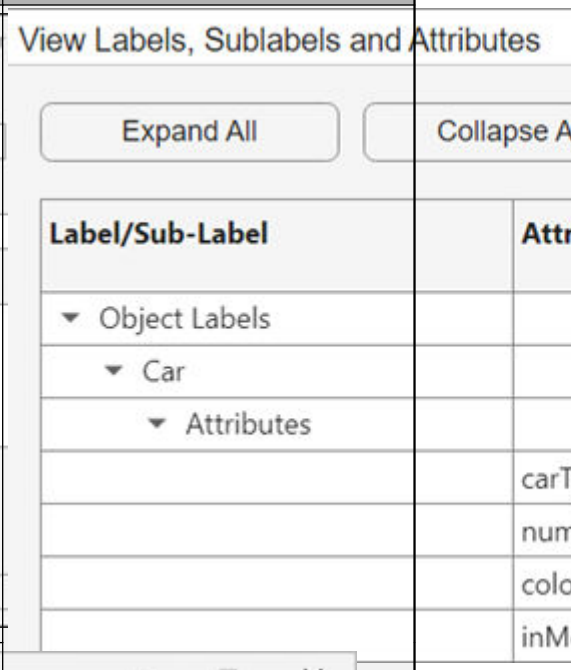
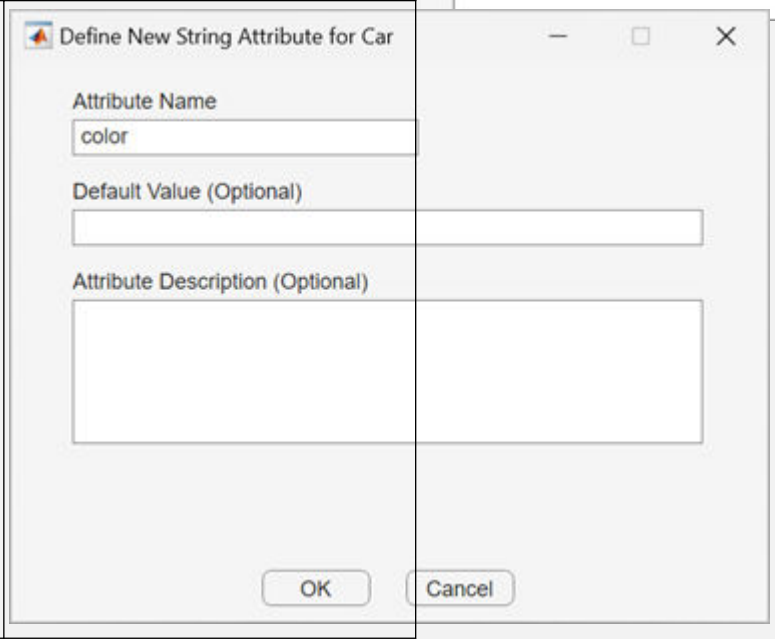
ROI Sublabels

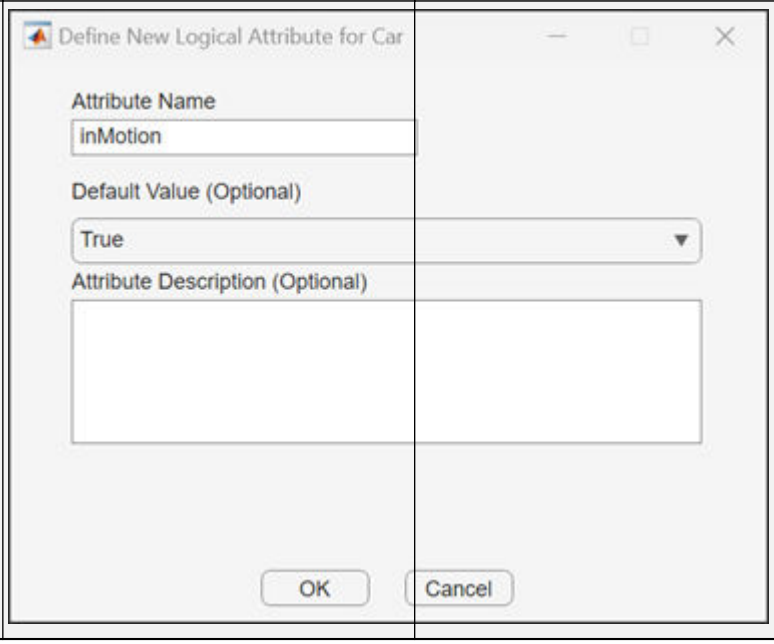
An ROI sublabel is an ROI label that belongs to a parent label. Use ROI sublabels to provide a greater level of detail about the ROIs in your labeled ground truth data. For example, a **vehicle** label might contain **headlight**, **licensePlate**, and **wheel** sublabels. You can create sublabels only for all ROI label types with the exception of pixel labels. For more details about sublabels, see “Use Sublabels and Attributes to Label Ground Truth Data”.

ROI Attributes

An ROI attribute specifies additional information about an ROI label or sublabel. For example, in a driving scene, attributes might include the type or color of a vehicle. The table describes the supported attribute types.

Attribute Type	Sample Attribute Definition	Sample Default Values
`List`		

Attribute Type	Sample Attribute Definition	Sample Default Values
`Numeric`		
`String`		

Attribute Type	Sample Attribute Definition	Sample Default Values
`Logical`		

For more details about attributes, see “Use Sublabels and Attributes to Label Ground Truth Data”.

Tips

- To avoid having to relabel ground truth with new labels, organize the labeling scheme you want to use before marking your ground truth.

Algorithms

You can use label automation algorithms to speed up labeling within the app. To create your own label automation algorithm to use within the app, see “Create Custom Automation Algorithm for Labeling”. You can also use one of the built-in algorithms by following these steps:

1. Import the data you want to label, and create at least one label definition.
2. On the app toolstrip, click **Select Algorithm** and select one of the built-in automation algorithms.
3. Click **Automate**, and then follow the automation instructions in the right pane of the automation window.

ACF People Detector

Detect and label people using aggregate channel features (ACF). This algorithm is based on the `peopleDetectorACF` function. To use this algorithm, you must define at least one rectangle ROI label. You do not need to draw any ROI labels.

To help improve the algorithm results, first click **Settings**. You can change any of these settings.

- The pretrained people detector model that the algorithm uses — The `'inria-100x41'` model was trained using the INRIA person data set. The `'caltech-50x21'` model was trained using the Caltech Pedestrian data set.

- The overlap ratio threshold, from 0 to 1, for detecting people — When rectangle ROIs overlap by more than this threshold, the algorithm discards one of the ROIs.
- The classification score threshold for detecting people — Increase the score to increase the prediction confidence of the algorithm. Rectangles with scores below this threshold are discarded.

Point Tracker

Track and label one or more axis-aligned or rotated rectangle ROI labels over short intervals by using the Kanade-Lucas-Tomasi (KLT) algorithm. This algorithm is based on the `vision.PointTracker` System object. To use this algorithm, you must define at least one rectangle ROI label, but you do not need to draw any ROI labels.

To change the feature detector used to obtain the initial points for tracking, click **Settings**. This table shows the feature detector options.

Feature Detector	Description	Equivalent Function
`Minimum Eigen Value`	Detect corners by using the minimum eigenvalue algorithm.	`detectMinEigenFeatures`
`Harris`	Detect corners by using the Harris–Stephens algorithm.	`detectHarrisFeatures`
`FAST`	Detect corners by using the features from accelerated segment test (FAST) algorithm.	`detectFASTFeatures`
`BRISK`	Detect features by using the binary robust invariant scalable keypoints (BRISK) algorithm.	`detectBRISKFeatures`
`KAZE`	Detect features by using nonlinear diffusion to construct a scale space of an image, and then detecting multiscale corner features (KAZE features) from that scale space.	`detectKAZEFeatures`
`SURF`	Detect blob features by using the speeded-up robust features (SURF) algorithm.	`detectSURFFeatures`
`MSER`	Detect regions by using the maximally stable extremal regions (MSER) algorithm.	`detectMSERFeatures`

Temporal Interpolator

Estimate axis-aligned or rotated rectangle ROIs or projected cuboid ROIs between frames by interpolating the ROI locations across the time range. To use this algorithm, you must draw a rectangle ROI or a projected cuboid ROI on a minimum of two frames: one at the beginning of the interval and one at the end of the interval. The interpolation algorithm estimates and draws ROIs in the intermediate frames.

Consider a video with 10 frames. The first frame has a rectangle ROI centered at [5, 5]. The 10th frame has a rectangle ROI centered at [25, 25]. At each frame, the algorithm moves the ROI 2 pixels in the *x*-direction and 2 pixels in the *y*-direction. Therefore, the algorithm centers the ROI at [7, 7] in the second frame, [9, 9] in the third frame, and so on, up to [23, 23] in the second-to-last frame.

ACF Vehicle Detector

Detect and label vehicles using aggregate channel features (ACF). This algorithm is based on the `vehicleDetectorACF` function. To use this algorithm, you must define at least one rectangle ROI label. You do not need to draw any ROI labels.

To help improve the algorithm results, first click **Settings**. You can change any of these settings.

- The pretrained vehicle detector model that the algorithm uses — The `'full-view'` model was trained using unoccluded images of the front, rear, left, and right sides of vehicles. The `'front-rear-view'` model was trained using images of only the front and rear sides of the vehicle.
- The overlap ratio threshold, from 0 to 1, for detecting vehicles — When rectangle ROIs overlap by more than this threshold, the algorithm discards one of the ROIs.
- The classification score threshold for detecting vehicles — Increase the score to increase the prediction confidence of the algorithm. Rectangles with scores below this threshold are discarded.

You can also configure the detector with a calibrated monocular camera by importing a `monoCamera` object into the MATLAB workspace. Specify the length and width ranges of the vehicle in world units, such as meters.

Lane Boundary Detector (requires Automated Driving Toolbox)

Detect and label lane boundaries using an estimated bird’s-eye-view projected image. To use this algorithm, you must define at least one line ROI label. You do not need to draw any ROI labels. To detect lane boundaries, the algorithm follows these steps:

1. It makes an initial guess at the placement of the lane boundaries in the image.
2. It transforms the ROI around the lanes into a bird's-eye view image to make the lanes parallel and remove distortion.
3. It uses this image to segment the lane boundaries.

To help improve the algorithm results, first click **Settings**. You can change any of these settings.

- The placement of the lane lines for generating the bird's-eye view image
- The ROI around the lanes, which you can expand to include more than just the ego lane boundaries in the image
- The pixel width of detected lane boundaries in the image

You can also change the number of lane boundaries that you want to detect. The default number of lane boundaries is 2.

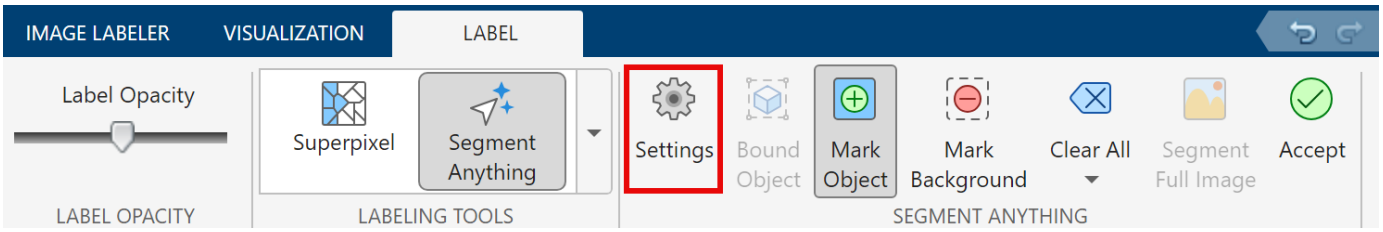
Version History

Introduced in R2018b

R2026a: Label ground truth using Segment Anything Model 2 (SAM 2)

By default, the **Segment Anything** tool now uses the Segment Anything Model 2 (SAM 2). By selecting the Settings in the **Segment Anything** section of the app toolstrip, you can specify one of these SAM model types.

- Segment Anything Model 2 - Large *(default)*
- Segment Anything Model 2 - Base
- Segment Anything Model 2 - Small
- Segment Anything Model 2 - Tiny
- Segment Anything Model - Base



This functionality requires the Image Processing Toolbox Model for Segment Anything Model 2 add-on. You can install the add-on from Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”. The functionality also requires a Deep Learning Toolbox license.

R2025a: Create pixel, rectangle, and polygon ROI labels using Segment Anything Model (SAM)

Using the Segment Anything Model (SAM) in the Video Labeler, you can now interactively:

- Create pixel labels for semantic segmentation
- Create rectangle ROI labels for object detection
- Create polygon ROI labels for instance segmentation

To use the SAM for labeling, select the **Segment Anything** tool on the **Label** tab of the Video Labeler toolstrip.

This functionality requires the Image Processing Toolbox Model for Segment Anything Model support package. You can install the support package from the Add-On Explorer. For more information about installing add-ons, see “Get and Manage Add-Ons”.

See Also

Apps
Image Labeler | Ground Truth Labeler | Lidar Labeler

Functions
`objectDetectorTrainingData` | `pixelLabelTrainingData`

Objects
`groundTruth` | `groundTruthDataSource` | `labelDefinitionCreator`

Topics
“Get Started with the Video Labeler”
“Choose an App to Label Ground Truth Data”
“Automatically Label Ground Truth Using Vision-Language Model”
“Use Custom Image Source Reader for Labeling”

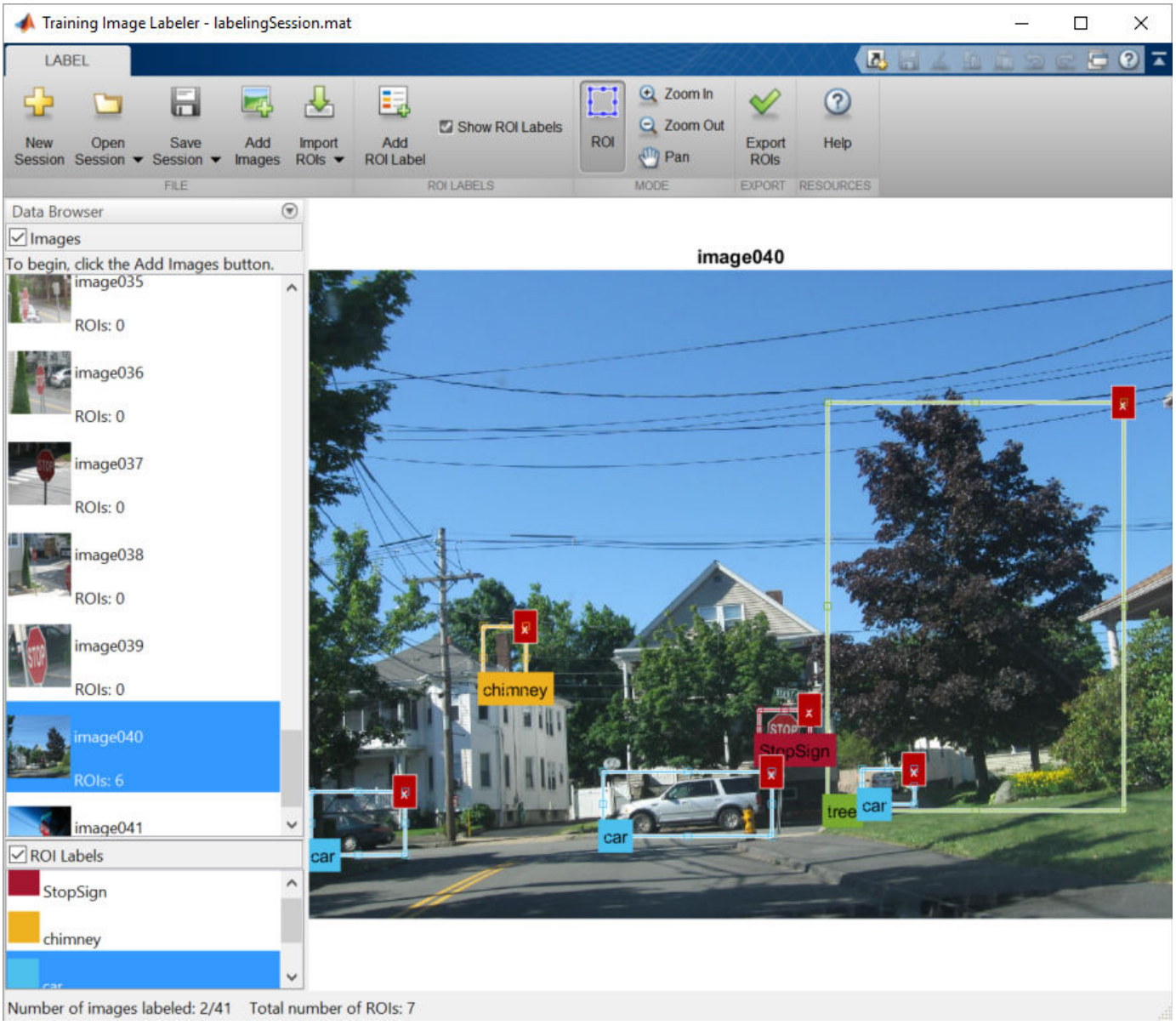
“Use Sublabels and Attributes to Label Ground Truth Data”
“Label Pixels for Semantic Segmentation”
“Label Objects Using Polygons for Instance Segmentation”
“Create Custom Automation Algorithm for Labeling”
“Share and Store Labeled Ground Truth Data”
“Keyboard Shortcuts and Mouse Actions for Video Labeler”
“Training Data for Object Detection and Semantic Segmentation”

Training Image Labeler

Label images for training a classifier

Description

The Image Labeler app replaces the Training Image Labeler app. Use the Image Labeler instead.



Open the Training Image Labeler App

- MATLAB command prompt: Enter `trainingImageLabeler`.

Note The `trainingImageLabeler` function opens the Image Labeler app.

Examples

Open Training Image Labeler App

Type `trainingImageLabeler` on the MATLAB command line or select it from the MATLAB desktop **Apps** tab.

Note The `trainingImageLabeler` function opens the Image Labeler app.

Programmatic Use

`trainingImageLabeler` invokes an app for labeling ground truth data in images. This app allows you to interactively specify rectangular Regions of Interest (ROIs). The ROIs define locations of objects, which are used to train a classifier. It outputs training data in a format supported by the `trainCascadeObjectDetector` function. The function trains a model to use with the `vision.CascadeObjectDetector` detector.

Note The `trainingImageLabeler` function opens the Image Labeler app.

Version History

Introduced in R2014a

See Also

Image Labeler | `insertObjectAnnotation` | `imrect` | `trainCascadeObjectDetector` | `vision.CascadeObjectDetector` | `imageDatastore`

Topics

“Get Started with the Image Labeler”
“Get Started with Cascade Object Detector”

triangulate

3-D locations of undistorted matching points in stereo images

Syntax

```
worldPoints = triangulate(matchedPoints1,matchedPoints2,stereoParams)
worldPoints = triangulate(matchedPoints1,matchedPoints2,camProjection1,
camProjection2)
[worldPoints,reprojectionErrors] = triangulate( ___ )
[worldPoints,reprojectionErrors,validIndex] = triangulate( ___ )
```

Description

`worldPoints = triangulate(matchedPoints1,matchedPoints2,stereoParams)` returns the 3-D locations of matching pairs of undistorted image points from two stereo images.

`worldPoints = triangulate(matchedPoints1,matchedPoints2,camProjection1, camProjection2)` returns the 3-D locations of the matching pairs in a world coordinate system. These locations are defined by camera projection matrices.

`[worldPoints,reprojectionErrors] = triangulate( ___ )` additionally returns reprojection errors for the world points using any of the input arguments from previous syntaxes.

`[worldPoints,reprojectionErrors,validIndex] = triangulate( ___ )` additionally returns the indices of valid and invalid world points. Valid points are located in front of the cameras.

Examples

Measure Real-World Distances to Objects Using a Stereo-Camera

Estimate the real-world distance to a face in an image by applying triangulation techniques to stereo-camera data.

Load stereo parameters.

```
load('webcamsSceneReconstruction.mat');
```

Read in the stereo pair of images.

```
I1 = imread('sceneReconstructionLeft.jpg');
I2 = imread('sceneReconstructionRight.jpg');
```

Undistort the images.

```
I1 = undistortImage(I1,stereoParams.CameraParameters1);
I2 = undistortImage(I2,stereoParams.CameraParameters2);
```

Detect a face in both images.

```
faceDetector = vision.CascadeObjectDetector;
face1 = faceDetector(I1);
face2 = faceDetector(I2);
```

Find the center of the face.

```
center1 = face1(1:2) + face1(3:4)/2;
center2 = face2(1:2) + face2(3:4)/2;
```

Compute the distance from camera 1 to the face.

```
point3d = triangulate(center1, center2, stereoParams);
distanceInMeters = norm(point3d)/1000;
```

Display the detected face and distance.

```
distanceAsString = sprintf('%0.2f meters', distanceInMeters);
I1 = insertObjectAnnotation(I1,'rectangle',face1,distanceAsString,'FontSize',18);
I2 = insertObjectAnnotation(I2,'rectangle',face2, distanceAsString,'FontSize',18);
I1 = insertShape(I1,'filled-rectangle',face1);
I2 = insertShape(I2,'filled-rectangle',face2);

imshowpair(I1, I2, 'montage');
```



Input Arguments

`matchedPoints1` — Coordinates of corresponding points

M-by-2 matrix | point feature object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

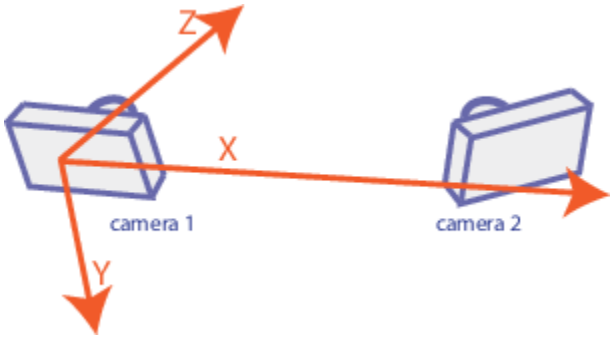
matchedPoints2 — Coordinates of corresponding points
M-by-2 matrix | point feature object

Coordinates of corresponding points in image one, specified as an *M*-by-2 matrix of *M* number of [*x* *y*] coordinates, or as one of the point feature objects described in “Point Feature Types”.

stereoParams — Camera parameters for stereo system
`stereoParameters` object

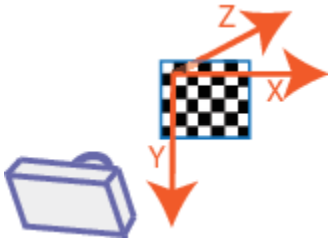
Camera parameters for stereo system, specified as a `stereoParameters` object. The object contains the intrinsic, extrinsic, and lens distortion parameters of the stereo camera system. You can use the `estimateCameraParameters` function to estimate camera parameters and return a `stereoParameters` object.

When you pass a `stereoParameters` object to the function, the origin of the world coordinate system is located at the optical center of camera 1. The *x*-axis points to the right, the *y*-axis points down, and the *z*-axis points away from the camera.



camProjection1 — Projection matrix for camera 1
3-by-4 matrix

Projection matrix for camera 1, specified as a 3-by-4 matrix. The matrix maps a 3-D point in homogeneous coordinates onto the corresponding point in the image from the camera. This input describes the location and orientation of camera 1 in the world coordinate system. `camProjection1` must be a real and nonsparse numeric matrix. You can obtain the camera matrix using the `cameraProjection` function.

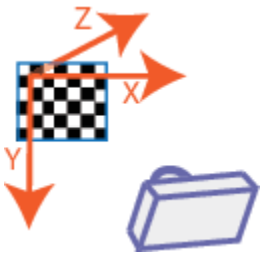


Camera projection matrices, passed to the function, define the world coordinate system.

camProjection2 — Projection matrix for camera 2
3-by-4 matrix

Projection matrix for camera 2, specified as a 3-by-4 matrix. The matrix maps a 3-D point in homogeneous coordinates onto the corresponding point in the image from the camera. This input

describes the location and orientation of camera 2 in the world coordinate system. `camProjection2` must be a real and nonsparse numeric matrix. You can obtain the camera matrix using the `cameraProjection` function.



Camera projection matrices, passed to the function, define the world coordinate system.

Output Arguments

`worldPoints` — 3-D locations of matching pairs of undistorted image points

M-by-3 matrix

3-D locations of matching pairs of undistorted image points, returned as an *M*-by-3 matrix. The matrix contains *M* number of [*x y z*] locations of matching pairs of undistorted image points from two stereo images.

When you specify the camera geometry using `stereoParams`, the world point coordinates are relative to the optical center of camera 1.

When you specify the camera geometry using `camProjection1` and `camProjection2`, the world point coordinates are defined by the camera matrices.

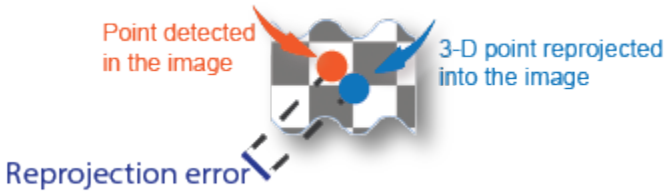
The function returns `worldPoints` as data type `double` when `matchedPoints1` and `matchedPoints2` are of data type `double`. Otherwise, the function returns `worldPoints` as data type `single`.

Data Types: `single` | `double`

`reprojectionErrors` — Reprojection errors

M-by-1 vector

Reprojection errors, returned as an *M*-by-1 vector. The function projects each world point back into both images. Then, in each image, the function calculates the reprojection error as the distance between the detected and the reprojected point. The `reprojectionErrors` vector contains the average reprojection error for each world point.



`validIndex` — Validity of world points

M-by-1 logical vector

Validity of world points, returned as an *M*-by-1 logical vector. Valid points, denoted as a logical `1` (`true`), are located in front of the cameras. Invalid points, denoted as a logical `0` (`false`), are located behind the cameras.

The validity of a world point with respect to the position of a camera is determined by projecting the world point onto the image using the camera matrix and homogeneous coordinates. The world point is valid if the resulting scale factor is positive.

Tips

The `triangulate` function does not account for lens distortion. You can undistort the images using the `undistortImage` function before detecting the points. Alternatively, you can undistort the points themselves using the `undistortPoints` function.

Version History

Introduced in R2014b

R2022b: Supports premultiply geometric transformation convention

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `camProjection1` and `camProjection2` arguments as 3-by-4 matrices calculated by the `cameraProjection` function, which uses the premultiply convention.

Although you can still specify `camProjection1` and `camProjection2` as 4-by-3 matrices calculated by the `cameraMatrix` function, this function is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed., 312-313. Cambridge, UK ; New York: Cambridge University Press, 2003.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`triangulateMultiview` | `reconstructScene` | `estimateCameraParameters` | `cameraProjection` | `undistortImage` | `undistortPoints` | `estrelpose`

Objects
`stereoParameters` | `cameraParameters`

Topics
“Structure from Motion from Two Views”
“Code Generation for Depth Estimation from Stereo Video”
“Coordinate Systems”
“Point Feature Types”

triangulateMultiview

3-D locations of world points matched across multiple images

Syntax

```
worldPoints = triangulateMultiview(pointTracks,cameraPoses,intrinsics)
[worldPoints,reprojectionErrors] = triangulateMultiview( ___ )
[worldPoints,reprojectionErrors,validIndex] = triangulateMultiview( ___ )
```

Description

`worldPoints = triangulateMultiview(pointTracks,cameraPoses,intrinsics)` returns the locations of 3-D world points that correspond to points matched across multiple images taken with calibrated cameras. `pointTracks` specifies an array of matched points. `cameraPoses` and `intrinsics` specify camera pose information and intrinsics, respectively. The function does not account for lens distortion.

`[worldPoints,reprojectionErrors] = triangulateMultiview( ___ )` additionally returns the mean reprojection error for each 3-D world point using all input arguments in the prior syntax.

`[worldPoints,reprojectionErrors,validIndex] = triangulateMultiview( ___ )` additionally returns the indices of valid and invalid world points. Valid points are located in front of the cameras.

Examples

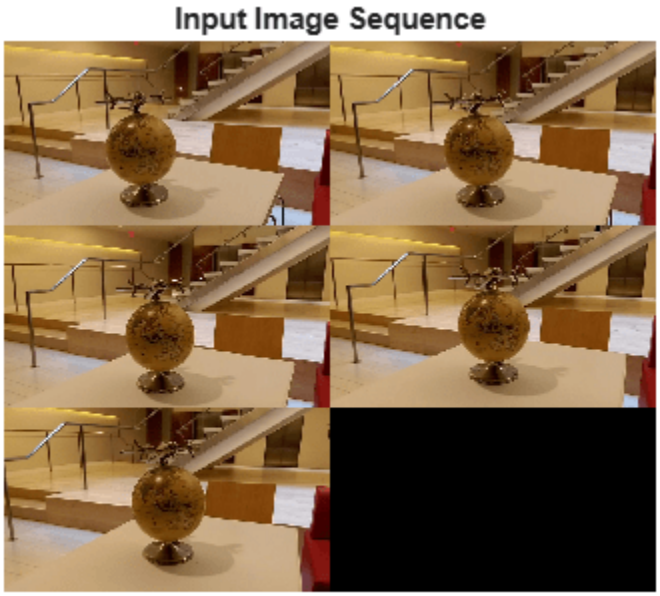
Reconstruct Scene from Sequential Views

This example shows how to reconstruct 3-D structure from a sequence of images. To process non-sequential images, follow the steps in the “Structure from Motion from Multiple Views” example.

Read and display the image sequence.

```
% Use imageDatastore to get a list of all image file names in a directory
imageDir = fullfile(toolboxdir("vision"), "visiondata",  "structureFromMotion");
imds = imageDatastore(imageDir);

% Display the images
figure
montage(imds.Files);
title("Input Image Sequence");
```




```
% Load camera intrinsics
data = load(fullfile(imageDir, "cameraParams.mat"));
intrinsics = data.cameraParams.Intrinsics;

% Undistort the images.
images = cell(1, numel(imds.Files));
for viewId = 1:numel(imds.Files)
    images{viewId} = undistortImage(readimage(imds, viewId), intrinsics);
end

% Convert the image to grayscale.
I = im2gray(images{1});

% Detect SIFT features. Use an ROI to eliminate spurious
% features around the edges of the image.
border = 50;
roi = [border, border, size(I, 2)- 2*border, size(I, 1)- 2*border];
prevPoints   = detectSIFTFeatures(I, ROI=roi);

% Extract features. Using 'Upright' features improves matching, as long as
% the camera motion involves little or no in-plane rotation.
[prevFeatures, prevPoints] = extractFeatures(I, prevPoints);

% Create an empty imageviewset object to manage the data associated with each
% view.
vSet = imageviewset;

% Add the first view. Place the camera associated with the first view
% and the origin, oriented along the Z-axis.
viewId = 1;
pose1  = rigidtform3d;
vSet = addView(vSet, viewId, pose1, Points=prevPoints, Features=prevFeatures);
```

```
% Process the rest of the views
for viewId = 2:numel(images)
    % Convert the image to grayscale.
    I = im2gray(images{viewId});

    % Detect, extract and match features.
    currPoints   = detectSIFTFeatures(I, ROI=roi);
    [currFeatures, currPoints] = extractFeatures(I, currPoints);

    % Match features between the previous and current image.
    indexPairs = matchFeatures(prevFeatures, currFeatures, MatchThreshold=20, MaxRatio=0.2, Uniqu

    if viewId == 2
        % Select matched points
        matchedPoints1 = prevPoints(indexPairs(:, 1));
        matchedPoints2 = currPoints(indexPairs(:, 2));

        % Estimate the camera pose of current view relative to the previous view.
        % The pose is computed up to scale, meaning that the distance between
        % the cameras in the previous view and the current view is set to 1.
        [currPose, inlierIndex] = helperEstimateRelativePose(...
            matchedPoints1, matchedPoints2, intrinsics);

        indexPairs = indexPairs(inlierIndex, :);
    else
        % Triangulate points from the previous two views, and find the
        % corresponding points in the current view.
        [worldPoints, imagePoints] = helperFind3Dto2DCorrespondences(vSet, ...
            intrinsics, indexPairs, currPoints);

        % Estimate the world camera pose for the current view.
        currPose = estworldpose(imagePoints, worldPoints, intrinsics, MaxReprojectionError=3);
    end

    % Add the current view to the view set.
    vSet = addView(vSet, viewId, currPose, Points=currPoints, Features=currFeatures);

    % Store the point matches between the previous and the current views.
    vSet = addConnection(vSet, viewId-1, viewId, Matches=indexPairs);

    % Find point tracks across all views.
    tracks = findTracks(vSet);

    % Get the table containing camera poses for all views.
    camPoses = poses(vSet);

    % Triangulate initial locations for the 3-D world points.
    xyzPoints = triangulateMultiview(tracks, camPoses, intrinsics);

    % Refine the 3-D world points and camera poses.
    [xyzPoints, camPoses, reprojectionErrors] = bundleAdjustment(xyzPoints, ...
        tracks, camPoses, intrinsics, FixedViewId=1, ...
        PointsUndistorted=true, AbsoluteTolerance=1e-6);

    % Store the refined camera poses.
    vSet = updateView(vSet, camPoses);
```

```
    prevFeatures = currFeatures;
end
```

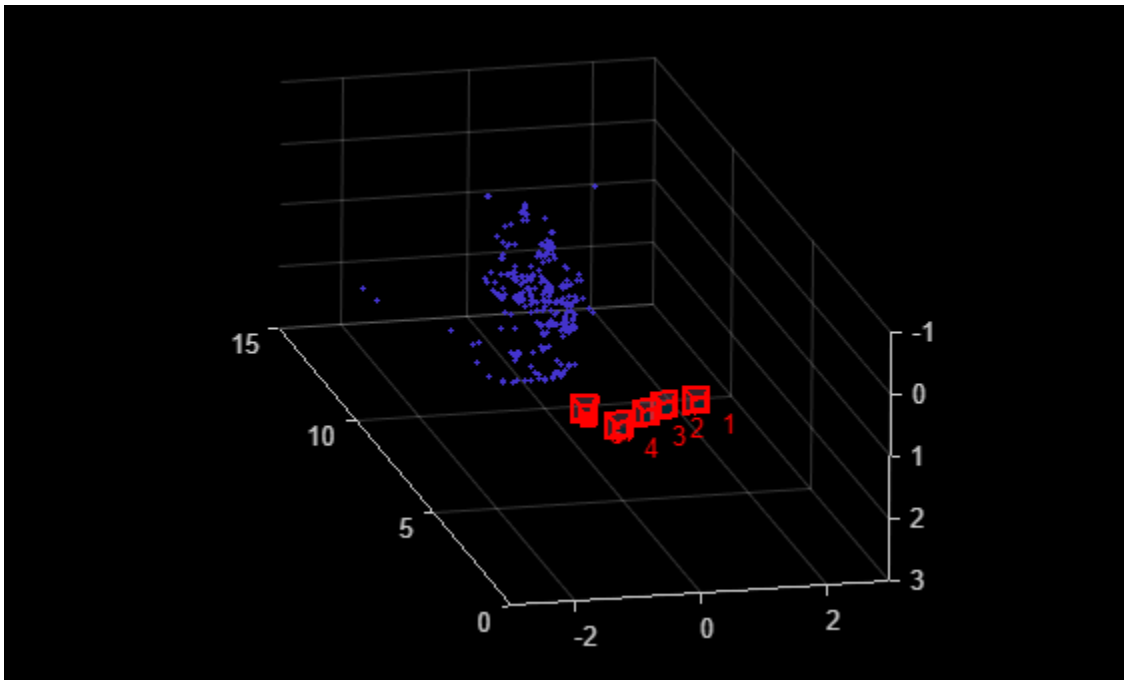
Display the camera poses and 3-D world points.

```
% Display camera poses
figure;
plotCamera(camPoses, Size=0.2);
hold on

% Exclude noisy 3-D points
goodIdx = reprojectionErrors < 5;

% Display the dense 3-D world points
pcshow(xyzPoints(goodIdx, :), VerticalAxis="y", VerticalAxisDir="down", MarkerSize=40);
grid on
hold off

% Specify the viewing volume
loc1 = camPoses.AbsolutePose(1).Translation;
xlim([loc1(1)-3, loc1(1)+3]);
ylim([loc1(2)-1, loc1(2)+3]);
zlim([loc1(3), loc1(3)+15]);
camorbit(0, -30);
```



Input Arguments

`pointTracks` — Matched points across multiple images

N-element array of `pointTrack` objects

Matched points across multiple images, specified as an *N*-element array of `pointTrack` objects. Each element contains two or more points that match across multiple images.

`cameraPoses` — Camera pose information

two-column table

Camera pose information, specified as a two-column table. You can obtain `cameraPoses` from an `imageviewset` object by using the `poses` object function.

Column	Description
`ViewID`	View identifier in the `pointTrack` object, specified as an integer.
`AbsolutePose`	Absolute pose of the view, specified as a `rigidtform3d` object.

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | *M*-element vector of `cameraIntrinsics` objects

Camera intrinsics, specified as a `cameraIntrinsics` object or an *M*-element vector of `cameraIntrinsics` objects. *M* is the number of camera poses. When all images are captured by the same camera, specify one `cameraIntrinsics` object. When images are captured by different cameras, specify a vector.

Output Arguments

`worldPoints` — 3-D world points

N-by-3 matrix

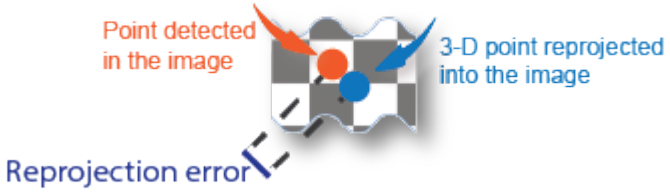
3-D world points, returned as an *N*-by-3 matrix. Each row represents one 3-D world point and is of the form [*x*, *y*, *z*]. *N* is the number of 3-D world points.

Data Types: `single` | `double`

`reprojectionErrors` — Reprojection errors

N-element vector

Reprojection errors, returned as an *N*-element vector. To calculate reprojection errors, first the function projects each world point back into each image. Then, in each image, the function calculates the distance between the detected and the reprojected point. Each element of the `reprojectionErrors` output is the average reprojection error for the corresponding world point in the `worldPoints` output.



`validIndex` — Validity of world points

M-by-1 logical vector

Validity of world points, returned as an *M*-by-1 logical vector. Valid points, denoted as a logical `1` (`true`), are located in front of the cameras. Invalid points, denoted as logical `0` (`false`), are located behind the cameras.

The validity of a world point with respect to the position of a camera is determined by projecting the world point onto the image using the camera matrix and homogeneous coordinates. The world point is valid if the resulting scale factor is positive.

Tips

Before detecting the points, correct the images for lens distortion by using by using the `undistortImage` function. Alternatively, you can directly undistort the points by using the `undistortPoints` function.

Version History

Introduced in R2016a

R2022b: Supports premultiply geometric transformation convention
Behavior changed in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. Accordingly, you can now specify the `AbsolutePose` field of the `cameraPoses` argument as a `rigidtform3d` object, which uses the premultiply convention. You can continue to estimate the camera poses using the `poses` object function of the `imageviewset` object, which has been updated to support the premultiply convention.

Although you can still specify `AbsolutePose` as a `rigid3d` object, this object is not recommended because it uses the postmultiply convention. You can streamline your geometric transformation workflows by switching to the premultiply geometric transformation convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

You can no longer specify `cameraPoses` as a three-column table with fields `ViewID`, `Orientation`, and `Location`. This table format is not supported because there is not enough information to determine whether the 3-by-3 rotation matrix in the `Orientation` field is intended for the premultiply or postmultiply convention.

References

[1] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

See Also

Apps
Camera Calibrator

Functions
`undistortImage` | `estimateCameraParameters` | `bundleAdjustment` | `bundleAdjustmentMotion` | `bundleAdjustmentStructure` | `undistortPoints` | `estrelpose`

Objects
`cameraParameters` | `pointTrack` | `imageviewset` | `rigidtform3d`

Topics
“3-D Motion Reconstruction Using Multiple Cameras”
“Structure from Motion from Two Views”
“Structure from Motion from Multiple Views”
“Code Generation for Depth Estimation from Stereo Video”
“What Is Structure from Motion?”
“Coordinate Systems”

trvec

Extract translation vector

Syntax

```
translationVector = trvec(transformation)
```

Description

`translationVector = trvec(transformation)` extracts the translation vector `translationVector` of the SE(3) transformation `transformation`.

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of transformation objects

Transformation, specified as a scalar `se3` object or an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformation` as an array, each element must be of the same type.

Output Arguments

`translationVector` — Translation vector

N-by-2 matrix | *N*-by-3 matrix

Translation vector, returned as an *N*-by-2 matrix for `se2` objects or an *N*-by-3 matrix for `se3` objects. *N* is the total number of transformations or rotations, and each row is a translation vector in the form [*X Y*] for 2-D transformations or [*X Y Z*] for 3-D transformations.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`rotm` | `tform`

Objects

`se3`

undistortImage

Correct image for lens distortion

Syntax

```
[J,camIntrinsics] = undistortImage(I,intrinsics)
[J,camIntrinsics] = undistortImage(I,intrinsics,interp)
[J,camIntrinsics] = undistortImage( ___ ,Name=Value)
```

Description

`[J,camIntrinsics] = undistortImage(I,intrinsics)` removes the lens distortion in the input image `I` using the camera parameters defined in `intrinsics`. The function returns the corrected image `J` and the undistorted camera intrinsic parameters, `camIntrinsics`.

`[J,camIntrinsics] = undistortImage(I,intrinsics,interp)` specifies the interpolation method for the function to use on the input image.

`[J,camIntrinsics] = undistortImage( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `(FillValues=[0 0.4470 0.7410])` sets the output pixel values that are outside the input image boundaries to a blue color.

Examples

Correct Image for Lens Distortion

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
    "calibration","mono"));
```

Detect calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate world coordinates of the corners of the squares. The square size is in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

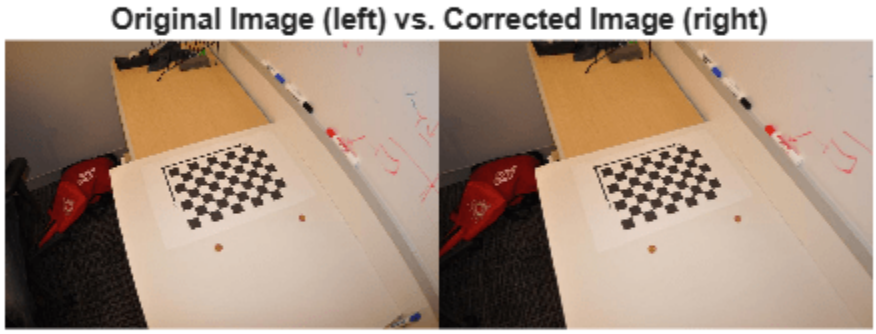
Calibrate the camera.

```
I = readimage(images,1);
imageSize = size(I,1:2);
cameraParams = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
```

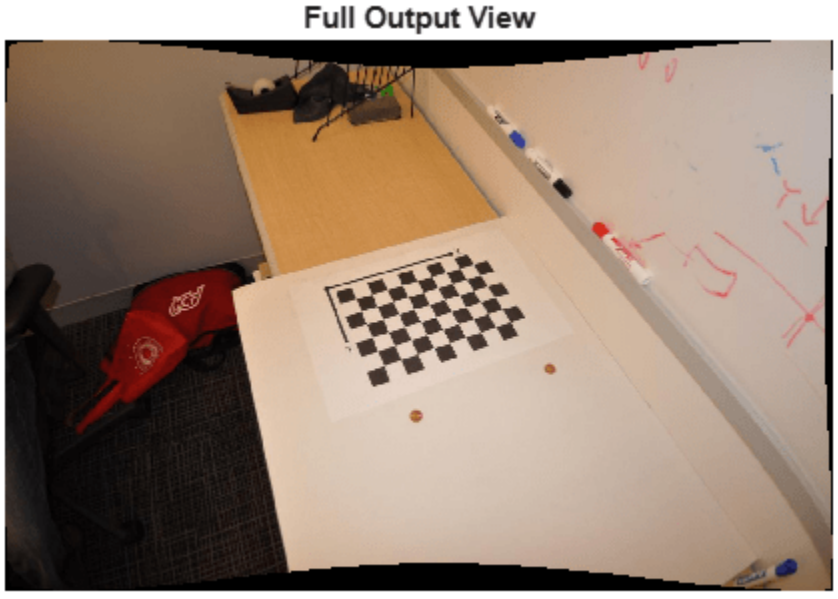
Remove lens distortion and display results.

```
I = images.readimage(1);
J1 = undistortImage(I,cameraParams);
```

```
figure;
imshowpair(I,J1,"montage");
title("Original Image (left) vs. Corrected Image (right)");
```




```
J2 = undistortImage(I,cameraParams,OutputView="full");
figure;
imshow(J2);
title("Full Output View");
```



Undistort Checkerboard Points

Create an imageDatastore object containing calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
   "calibration","mono"));
imageFileNames = images.Files;
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate world coordinates of the corners of the squares. The square size is in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,10);
imageSize = [size(I, 1), size(I, 2)];
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
intrinsics = params.Intrinsics;
```

Load an image and detect the checkerboard points.

```
points = detectCheckerboardPoints(I);
```

Undistort the points

```
undistortedPoints = undistortPoints(points,intrinsics);
```

Undistort the image.

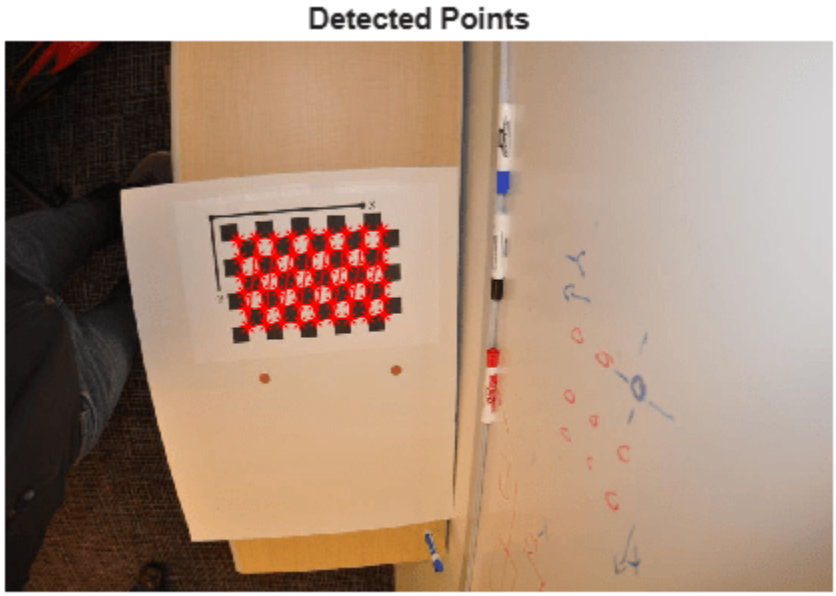
```
[J, newIntrinsics] = undistortImage(I,intrinsics,OutputView="full");
```

Translate undistorted points

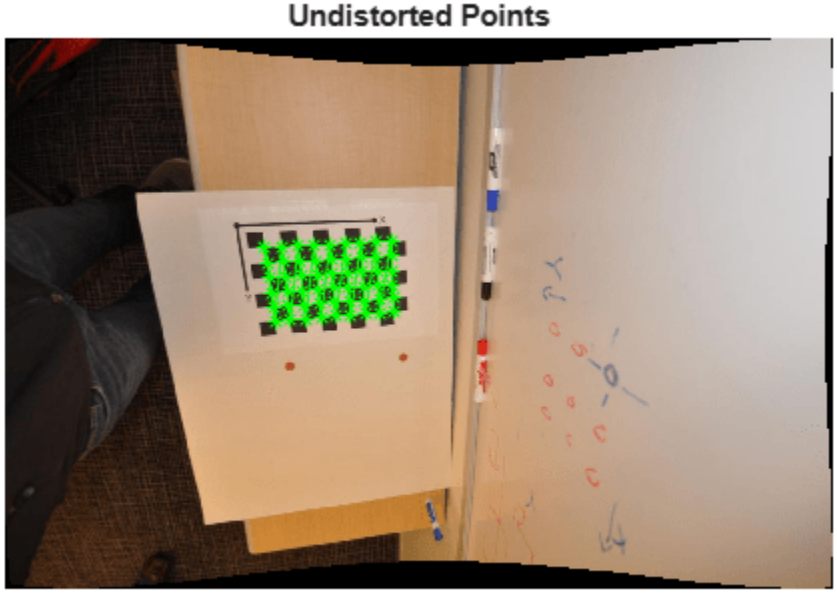
```
newOrigin = intrinsics.PrincipalPoint - newIntrinsics.PrincipalPoint;
undistortedPoints = [undistortedPoints(:,1) - newOrigin(1), ...
                     undistortedPoints(:,2) - newOrigin(2)];
```

Display the results

```
figure;
imshow(I);
hold on;
plot(points(:,1),points(:,2),"r*-");
title("Detected Points");
hold off;
```




```
figure;
imshow(J);
hold on;
plot(undistortedPoints(:,1),undistortedPoints(:,2),"g*-");
title("Undistorted Points");
hold off;
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image, specified in either *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`intrinsics` — Camera intrinsic parameters

`cameraParameters` object | `cameraIntrinsics` object | `cameraIntrinsicsKB` object

Camera intrinsic parameters, specified as a `cameraParameters`, `cameraIntrinsics`, `cameraIntrinsicsKB` object.

`interp` — Interpolation method

`"linear"` (default) | `"nearest"` | `"cubic"`

Interpolation method to use on the input image, specified as `"linear"`, `"nearest"` , or `"cubic"`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `(FillValues=[0 0.4470 0.7410])` sets the output pixel values that are outside the input image boundaries to a blue color.

`FillValues` — Output pixel fill values

`0` (default) | scalar | 3-element vector

Output pixel fill values, specified as an array containing one or more fill values. When the corresponding inverse transformed location in the input image lies completely outside the input image boundaries, you use the fill values for output pixels. When you use a 2-D grayscale input image, you must set the `FillValues` to scalar. When you use a truecolor, `FillValues` can be a scalar or a 3-element vector of RGB values.

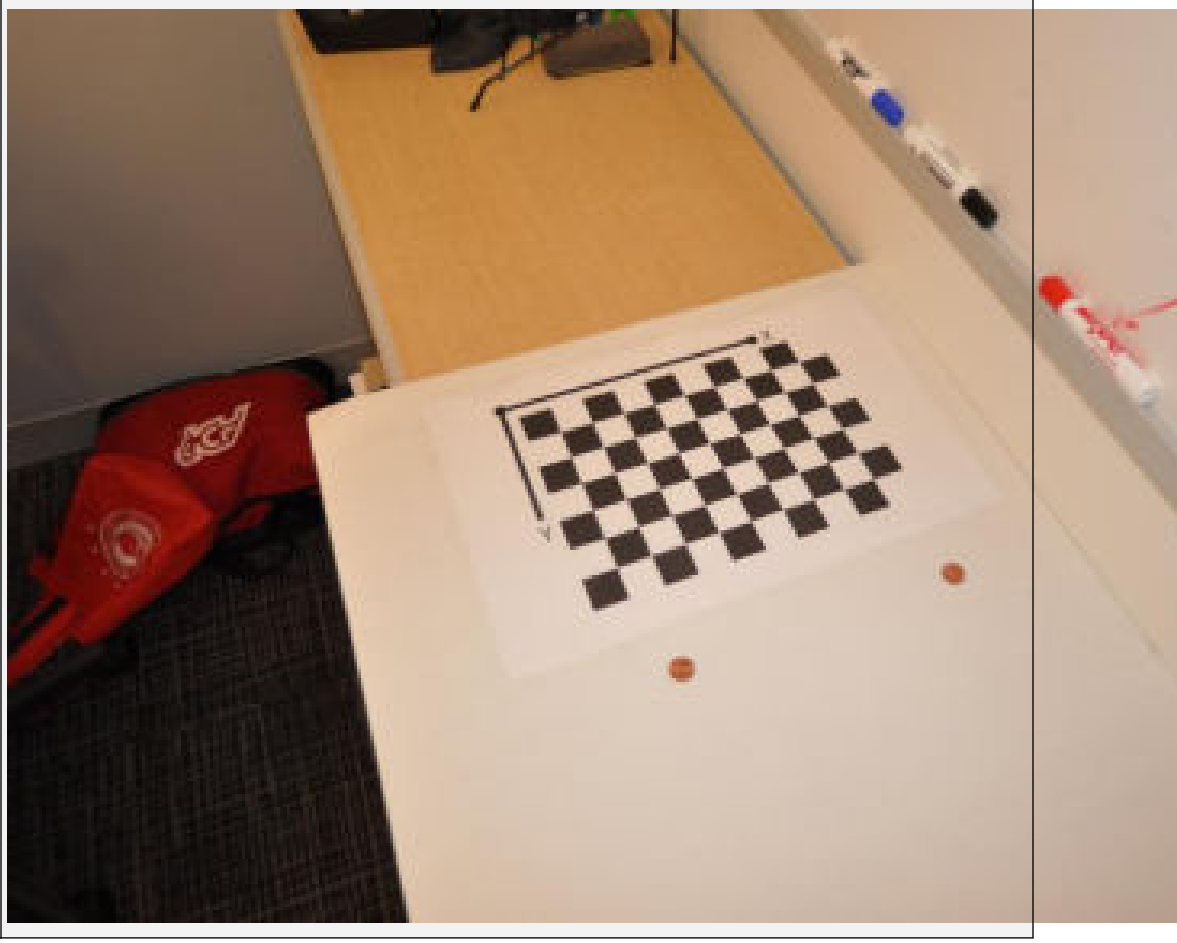
`OutputView` — Size of output image

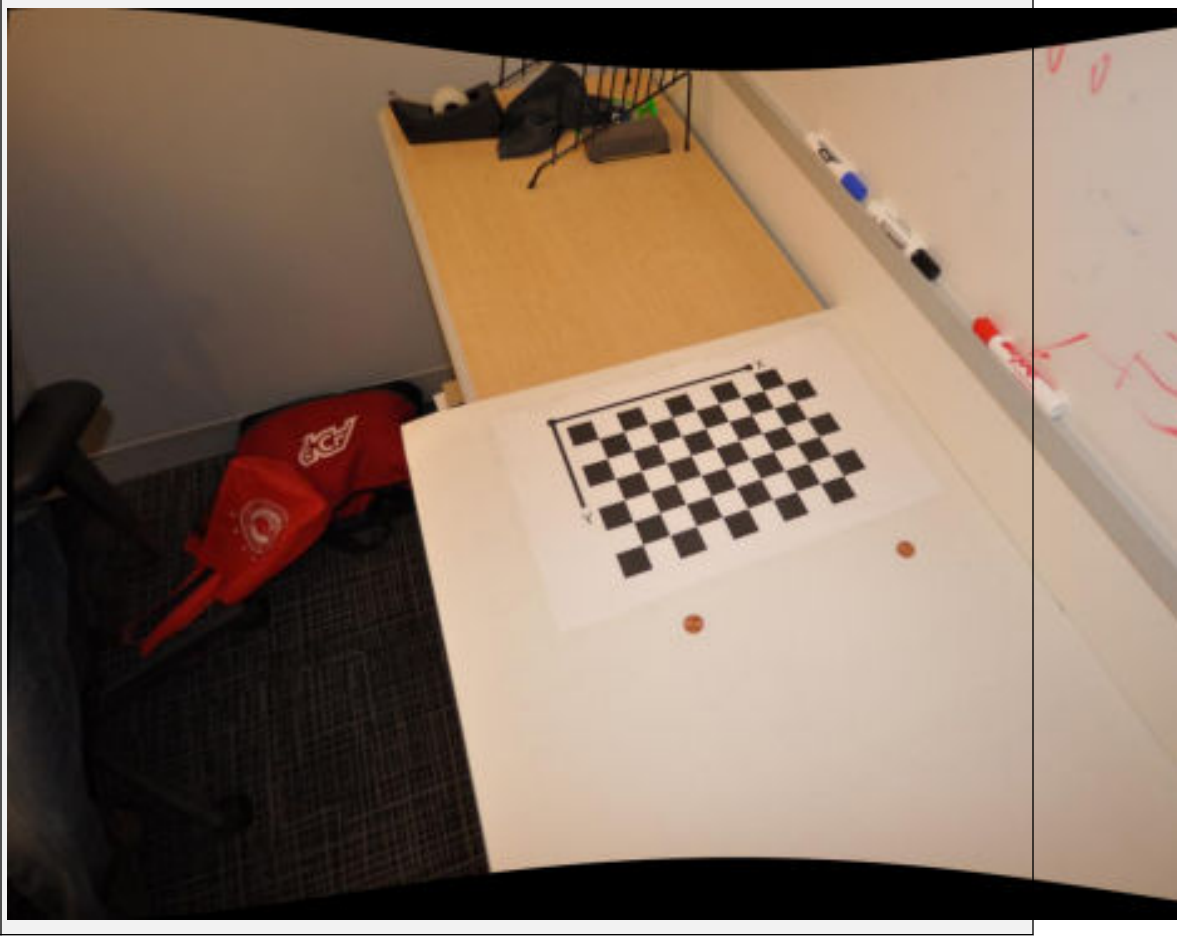
`"same"` (default) | `"full"` | `"valid"`

Size of output image, specified as `"same"`, `"full"`, or `"valid"`. When you set the property to `"same"`, the function sets the output image to match the size of the input image. When you set the property to `"full"`, the output includes all pixels from the input image. When you set the property to `"valid"`, the function crops the output image to contain only valid pixels.

For the input image:



OutputView	Output Image
`"same"`	Match the size of the input image. 

OutputView	Output Image
`"full"`	All pixels from the input image. 

OutputView	Output Image
`"valid"`	Only valid pixels from the input image. 

Output Arguments

`J` — Undistorted image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Undistorted image, returned in either *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`camIntrinsics` — Camera intrinsic parameters

`cameraIntrinsics` object

Camera intrinsic parameters, returned as a `cameraIntrinsics` object. The object contains the intrinsic parameters and lens distortion coefficients for the camera, which are used to produce an output image with lens distortion removed.

It is recommended to use the `camIntrinsics` output in any photogrammetry application that uses the undistorted image `J`, such as stereo camera calibration, multi-camera calibration, lidar-camera calibration, triangulation, and similar tasks.

Version History

Introduced in R2014a

R2024a: Returns camera intrinsic parameters object
Behavior changed in R2024a

This release replaces the second optional output argument. The function now returns a `cameraIntrinsics` object in the second argument. Prior to this release, the function returned a 2-element vector `newOrigin`. You can still calculate the new origin by subtracting the output intrinsics principal point from the intrinsics input principal point using this code:

```
newOrigin = intrinsics.PrincipalPoint - newIntrinsics.PrincipalPoint
```

R2024a: Added support for the `cameraIntrinsicsKB` object

Support added for the `cameraIntrinsicsKB` object as the `intrinsics` input.

R2025a: Generate CUDA code using GPU Coder

The `undistortImage` function now supports the generation of CUDA code (requires GPU Coder).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Use in a MATLAB Function block is not supported.
- `'interp'` and `'OutputView'` must be compile-time constants.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`estimateCameraParameters` | `triangulate` | `undistortPoints` | `estimateExtrinsics`

Objects
`cameraIntrinsicsKB` | `cameraParameters` | `cameraIntrinsics` | `stereoParameters`

Topics
“What Is Camera Calibration?”
“Fisheye Calibration Basics”
“Code Generation for Depth Estimation from Stereo Video”

undistortPoints

Correct point coordinates for lens distortion

Syntax

```
undistortedPoints = undistortPoints(points,cameraParams)
[undistortedPoints,reprojectionErrors] = undistortPoints(points,cameraParams)
```

Description

`undistortedPoints = undistortPoints(points,cameraParams)` returns point coordinates corrected for lens distortion. This function uses numeric nonlinear least-squares optimization.

`[undistortedPoints,reprojectionErrors] = undistortPoints(points,cameraParams)` additionally returns the errors used to evaluate the accuracy of undistorted points.

Examples

Undistort Checkerboard Points

Create an imageDatastore object containing calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
   "calibration","mono"));
imageFileNames = images.Files;
```

Detect the calibration pattern.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate world coordinates of the corners of the squares. The square size is in millimeters.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
I = readimage(images,10);
imageSize = [size(I, 1), size(I, 2)];
params = estimateCameraParameters(imagePoints,worldPoints,ImageSize=imageSize);
intrinsics = params.Intrinsics;
```

Load an image and detect the checkerboard points.

```
points = detectCheckerboardPoints(I);
```

Undistort the points

```
undistortedPoints = undistortPoints(points,intrinsics);
```

Undistort the image.

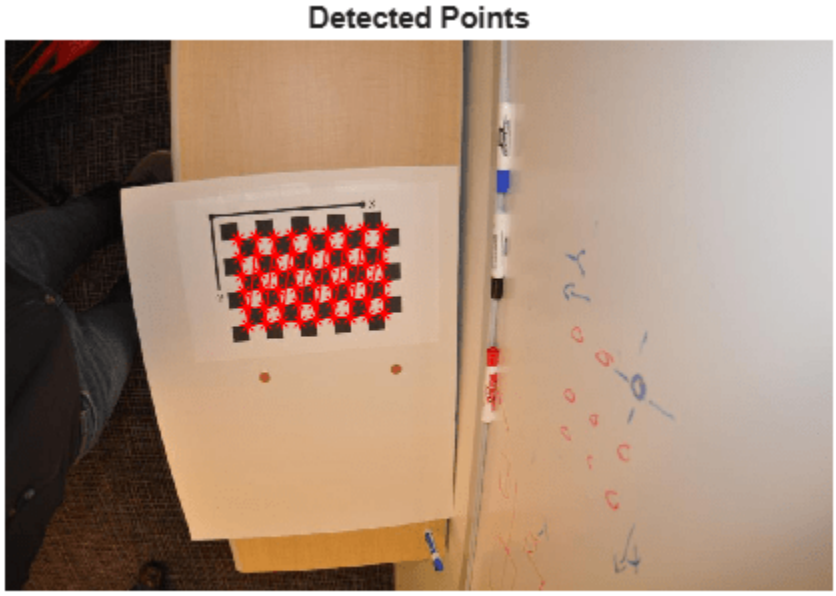
```
[J, newIntrinsics] = undistortImage(I,intrinsics,OutputView="full");
```

Translate undistorted points

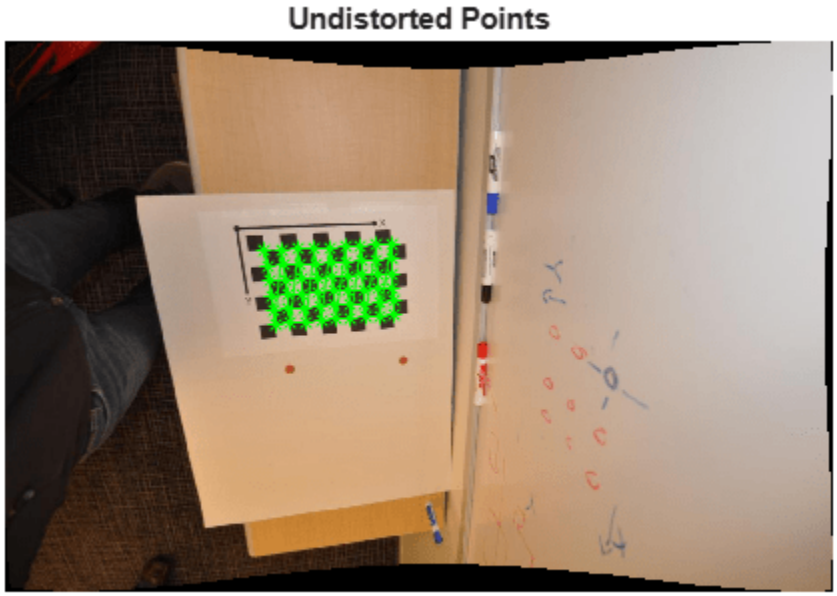
```
newOrigin = intrinsics.PrincipalPoint - newIntrinsics.PrincipalPoint;
undistortedPoints = [undistortedPoints(:,1) - newOrigin(1), ...
                     undistortedPoints(:,2) - newOrigin(2)];
```

Display the results

```
figure;
imshow(I);
hold on;
plot(points(:,1),points(:,2),"r*-");
title("Detected Points");
hold off;
```




```
figure;
imshow(J);
hold on;
plot(undistortedPoints(:,1),undistortedPoints(:,2),"g*-");
title("Undistorted Points");
hold off;
```



Input Arguments

`points` — Input points

M-by-2 matrix

Input points, specified an *M*-by-2 matrix of *M* number of [x y] coordinates.

`cameraParams` — Object for storing camera parameters

`cameraParameters` object | `cameraIntrinsics` object | `cameraIntrinsicsKB` object

Camera parameters, specified as a `cameraParameters`, `cameraIntrinsics`, or a `cameraIntrinsicsKB` object. You can return the `cameraParameters` object using the `estimateCameraParameters` function. The `cameraParameters` object contains the intrinsic, extrinsic, and lens distortion parameters of a camera.

Output Arguments

`undistortedPoints` — Undistorted points

M-by-2 matrix

Undistorted points, returned as an *M*-by-2 matrix. The `undistortedPoints` output contains *M* [*x*,*y*] point coordinates corrected for lens distortion. When you input points as `double`, the function outputs `undistortedPoints` as `double`. Otherwise, it outputs `undistortedPoints` as single.

Data Types: `single` | `double`

`reprojectionErrors` — Reprojection errors

M-by-1 vector

Reprojection errors, returned as an *M*-by-1 vector. You can use the errors to evaluate the accuracy of undistorted points. The function computes the errors by applying distortion to the undistorted points,

and then taking the distances between the result and the corresponding input points. The `reprojectionErrors` output is in pixels.

Version History

Introduced in R2014b

R2024a: Added support for the `cameraIntrinsicsKB` object

Support added for the `cameraIntrinsicsKB` object as the `cameraParams` input.

R2026a: Code generation support for `cameraIntrinsicsKB` object input

Starting in R2026a, you can use a `cameraIntrinsicsKB` object as input when generating code for the `undistortPoints` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`estimateCameraParameters` | `triangulate` | `undistortImage` | `estimateExtrinsics` | `cameraProjection` | `estimateFisheyeParameters` | `plotCamera` | `img2world2d` | `world2img` | `extr2pose` | `pose2extr`

Objects
`cameraIntrinsicsKB` | `cameraParameters` | `cameraIntrinsics` | `stereoParameters` | `fisheyeIntrinsics`

Topics
“What Is Camera Calibration?”
“Fisheye Calibration Basics”
“Code Generation for Depth Estimation from Stereo Video”

visionlib

Open top-level Computer Vision Toolbox Simulink library

Syntax

```
visionlib
```

Description

`visionlib` opens the top-level Computer Vision Toolbox block library.

Examples

Open Computer Vision Toolbox Block Library

View and gain access to the Computer Vision Toolbox blocks.

```
visionlib
```

Alternatives

To view and gain access to the Computer Vision Toolbox blocks using the Simulink library browser:

- Type `simulink` at the MATLAB command line, and then expand the Computer Vision Toolbox node in the library browser.
- Click the Simulink icon  from the MATLAB desktop or from a model.

Version History

Introduced in R2011a

visionSupportPackages

Start Installer to download, install, or uninstall Computer Vision Toolbox data

Syntax

```
visionSupportPackages
```

Description

`visionSupportPackages` launches the Support Package Installer, which you can use to download, install, or uninstall support packages for Computer Vision Toolbox.

Computer Vision Toolbox Support Packages
“Install OCR Language Data Files”
“Install and Use Computer Vision Toolbox Interface for OpenCV in MATLAB”

Examples

Start Computer Vision Toolbox installer

```
visionSupportPackages
```

Version History

Introduced in R2014b

See Also

Topics
“Install Computer Vision Toolbox Add-on Support Files”

ocvStructToKeyPoints

Convert MATLAB feature points struct to OpenCV `KeyPoint` vector

C++ Syntax

```
#include "opencvmex.hpp"
void ocvStructToKeyPoints(const mxArray * in,cv::vector<cv::KeyPoint>
&keypoints);
```

Arguments

`in`

Pointer to a MATLAB structure, `mxArray`, that represents a point feature. Format:

Field Name	Field Requirement	Field Data Type
`Location`	Required	Single
`Scale`	Required	Single
`Metric`	Required	Single
`Orientation`	Optional	Single
`Octave`	Optional	int32
`Misc`	Optional	int32

Description

The `ocvStructToKeyPoints` function converts a point feature data structure from a MATLAB struct to an OpenCV's KeyPoint vector.

See Also

`mxArray`, `ocvKeyPointsToStruct`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxGpuArrayToGpuMat_{DataType}

Create `cv::gpu::GpuMat` from `mxArray` containing GPU data.

C++ Syntax

Note This function will be removed in a future release.

```
#include "opencvgpumex.hpp"
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_double(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_single(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_uint8(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_uint16(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_uint32(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_int8(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_int16(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_int32(const mxArray * in);
cv::Ptr<cv::gpu::GpuMat> ocvMxGpuArrayToGpuMat_bool(const mxArray * in);
```

Arguments

`in`

Pointer to a MATLAB struct, `mxArray`, containing GPU data. Supported data types:

`real_T` (`double`)	`real32_T` (`single`)	`uint8_T` (`uint8`)
`uint16_T` (`uint16`)	`uint32_T` (`uint32`)	`int8_T` (`int8`)
`int16_T` (`int16`)	`int32_T` (`int32`)	`boolean_T` (`bool`)

Returns

OpenCV smart pointer (`cv::Ptr`) to a `cv::gpu::GpuMat` object.

Description

The `ocvMxGpuArrayToGpuMat_{DataType}` function creates a `cv::gpu::GpuMat` object from an `mxArray` containing GPU data. This function requires the Parallel Computing Toolbox software.

See Also

`mxArray`, `ocvMxGpuArrayFromGpuMat_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxGpuArrayFromGpuMat_{DataType}

Create an `mxArray` from `cv::gpu::GpuMat` object

C++ Syntax

Note This function will be removed in a future release.

```
#include "opencvgpumex.hpp"
mxArray * ocvMxGpuArrayFromGpuMat_double(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_single(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_uint8(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_uint16(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_uint32(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_int8(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_int16(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_int32(const cv::gpu::GpuMat & in);
mxArray * ocvMxGpuArrayFromGpuMat_bool(const cv::gpu::GpuMat & in)
```

Arguments

`in`

Reference to OpenCV `cv::gpu::GpuMat` object.

Returns

Pointer to a MATLAB struct, `mxArray`, containing GPU data. Supported data types:

`real_T` (`double`)	`real32_T` (`single`)	`uint8_T` (`uint8`)
`uint16_T` (`uint16`)	`uint32_T` (`uint32`)	`int8_T` (`int8`)
`int16_T` (`int16`)	`int32_T` (`int32`)	`boolean_T` (`bool`)

Description

The `ocvMxGpuArrayFromGpuMat` function creates an `mxArray` from a `cv::gpu::GpuMat` object. `GpuMat` supports 2-D arrays only. This function requires the Parallel Computing Toolbox software.

See Also

`mxArray`, `ocvMxGpuArrayToGpuMat_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayToSize_{DataType}

Convert 2-element `mxArray` to `cv::Size`.

C++ Syntax

```
#include "opencvmex.hpp"
cv::Size ocvMxArrayToSize_single(const mxArray * in, bool rcInput = true);
cv::Size ocvMxArrayToSize_int32(const mxArray * in, bool rcInput = true);
```

Arguments

`in`

Pointer to a MATLAB `mxArray` having 2 elements. Supported data types:

`single`
`int32`

`rcInput`

Boolean flag that indicates if input `mxArray` is of the format [*r c*] or [*x y*].

rcInput	**in**
`true` (default)	[*r c*] (*height, width*)
`false`	[*x y*] (*width, height*)

Returns

OpenCV `cv::Size`

Description

The `ocvMxArrayToSize_{DataType}` function converts a 2-element `mxArray` to `cv::Size`. Empty input (`[]`) returns `cv::Size(0,0)`;

See Also

`mxArray`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayToMat_{DataType}

Convert column major `mxArray` to row major `cv::Mat` for generic matrix

C++ Syntax

```
#include "opencvmex.hpp"
void ocvMxArrayToMat_double(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_single(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_uint8(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_uint16(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_uint32(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_int8(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_int16(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_int32(const mxArray *in, cv::Mat &out);
void ocvMxArrayToMat_bool(const mxArray *in, cv::Mat &out);
cv::Ptr<cv::Mat> ocvMxArrayToMat_double(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_single(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_uint8(const mxArray *in, const bool copyData
= true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_uint16(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_uint32(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_int8(const mxArray *in, const bool copyData
= true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_int16(const mxArray *in, const bool copyData
= true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_int32(const mxArray *in, const bool copyData
= true);
cv::Ptr<cv::Mat> ocvMxArrayToMat_uint8(const mxArray *in, const bool copyData
= true);
```

Arguments

`in`

Pointer to a MATLAB struct, `mxArray`, having column major data. The data can be *n*-channel matrices. Supported data types:

`real_T` (`double`)	`uint8_T` (`uint8`)	`uint32_T` (`uint32`)	`int16_T` (`int16`)
`real32_T` (`single`)	`uint16_T` (`uint16`)	`int8_T` (`int8`)	`int32_T` (`int32`)

`copyData`

Boolean flag to copy data from `mxArray` to the `Mat` object.

- `true` (default) — The function transposes and interleaves (for RGB images) column major `mxArray` data into a row major `cv::Mat` object.

- `false` — No data copies from the `mxArray` to the `Mat` object. The function creates a new `Mat` wrapper and uses it to point to the `mxArray` data. Because OpenCV is row-based and MATLAB is column-based, the columns of the `mxArray` become the rows of the `Mat` object. If the image is 2-D, then `copyData` is `false`.

`out`

Reference to OpenCV `cv::Mat` with row major data.

Returns

The functions that set `copyData` return an OpenCV smart pointer (`cv::Ptr`) to a `cv::Mat` object.

Description

The `ocvMxArrayToMat_{DataType}` function applies to two C++ implementations. One set returns `void` and the other set returns an OpenCV smart pointer. The functions that return `void` reallocate memory for the `cv::Mat` if needed.

The `ocvMxArrayToMat_{DataType}` transposes and interleaves column major `mxArray` data into row major `cv::Mat`. This matrix conversion is a generic routine for any number of channels.

See Also

`mxArray`, `ocvMxArrayToImage_{DataType}`, `ocvMxArrayFromMat_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayToImage_{DataType}

Convert column major `mxArray` to row major `cv::Mat` for image

C++ Syntax

```
#include "opencvmex.hpp"
void ocvMxArrayToImage_double(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_single(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_uint8(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_uint16(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_uint32(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_int8(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_int16(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_int32(const mxArray *in, cv::Mat &out);
void ocvMxArrayToImage_bool(const mxArray *in, cv::Mat &out);
cv::Ptr<cv::Mat> ocvMxArrayToImage_double(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_single(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_uint8(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_uint16(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_uint32(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_int8(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_int16(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_int32(const mxArray *in, const bool
copyData = true);
cv::Ptr<cv::Mat> ocvMxArrayToImage_bool(const mxArray *in, const bool
copyData = true);
```

Arguments

`in`

Pointer to a MATLAB struct, `mxArray`, having column major data that represents a 2-D or 3-D image. Supported data types:

`real_T` (`double`)	`uint8_T` (`uint8`)	`uint32_T` (`uint32`)	`int16_T` (`int16`)
`real32_T` (`single`)	`uint16_T` (`uint16`)	`int8_T` (`int8`)	`int32_T` (`int32`)

`copyData`

Boolean flag to copy data from `mxArray` to the `Mat` object.

- `true` (default) — The function transposes and interleaves (for RGB images) column major `mxArray` data into a row major `cv::Mat` object.

- `false` — No data copies from the `mxArray` to the `Mat` object. The function creates a new `Mat` wrapper and uses it to point to the `mxArray` data. Because OpenCV is row-based and MATLAB is column-based, the columns of the `mxArray` become the rows of the `Mat` object. If the image is 2-D, then `copyData` is `false`.

`out`

Reference to OpenCV `cv::Mat` with row major data.

Returns

The functions that set `copyData` return an OpenCV smart pointer (`cv::Ptr`) to a `cv::Mat` object.

Description

The `ocvMxArrayToImage_{DataType}` function applies to two C++ implementations. One set returns `void` and the other set returns an OpenCV smart pointer. The functions that return `void` reallocate memory for the `cv::Mat` if needed.

The `ocvMxArrayToImage_{DataType}` transposes and interleaves column major `mxArray` data into row major `cv::Mat`. The `ocvMxArrayToImage_{DataType}` function supports 2-D and 3-D images.

These functions are not a generic matrix conversion routine. For 3-D images, they take into account that the OpenCV format uses BGR ordering and manipulate the data to comply with that formatting.

See Also

`mxArray`, `ocvMxArrayToMat_{DataType}`, `ocvMxArrayFromImage_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayToCvRect

Convert a MATLAB struct representing a rectangle to an OpenCV `CvRect`

C++ Syntax

```
#include "opencvmex.hpp"
CvRect ocvMxArrayToCvRect(const mxArray *in);
```

Arguments

`in`

Pointer to a MATLAB structure, `mxArray`, that represents a rectangle. The structure must have four scalar-valued fields: `x`, `y`, `width`, and `height`. The (`x`, `y`) fields represent the upper-left corner of the rectangle.

Returns

OpenCV `CvRect`.

Description

The `ocvMxArrayToCvRect` function converts a rectangle data structure from a MATLAB struct to an OpenCV KeyPoint vector.

See Also

`mxArray`, `ocvCvRectToMxArray`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayFromVector_{DataType}

Convert numeric `vectorT` to `mxArray`

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvMxArrayFromVector_double(const std::vector<real_T> &v);
mxArray *ocvMxArrayFromVector_single(const std::vector<real32_T> &v);
mxArray *ocvMxArrayFromVector_uint8(const std::vector<uint8_T> &v);
mxArray *ocvMxArrayFromVector_uint16(const std::vector<uint16_T> &v);
mxArray *ocvMxArrayFromVector_uint32(const std::vector<uint32_T> &v);
mxArray *ocvMxArrayFromVector_int8(const std::vector<int8_T> &v);
mxArray *ocvMxArrayFromVector_int16(const std::vector<int16_T> &v);
mxArray *ocvMxArrayFromVector_int32(const std::vector<int32_T> &v);
mxArray *ocvMxArrayFromVector_bool(const std::vector<boolean_T> &v);
```

Arguments

v

Reference to `vector<DataType>`. Supported data types:

`real_T`	`real32_T`	`uint8_T`
`uint16_T`	`uint32_T`	`int8_T`
`int16_T`	`int32_T`	`boolean_T`

Returns

Pointer to a MATLAB struct `mxArray`.

Description

The `ocvMxArrayFromVector_{DataType}` function converts numeric `std::vector<DataType>` to an `mxArray`.

See Also

`mxArray`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayFromPoints2f

Converts `vector<cv::Point2f>` to `mxArray`

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvMxArrayFromPoints2f(const std::vector<cv::Point2f> &points);
```

Arguments

`points`

Reference to OpenCV `vector<cv::Point2f>`.

Returns

Pointer to a MATLAB `mxArray`.

Description

The `ocvMxArrayFromPoints2f` function converts `std::vector<cv::Point2f>` to an `mxArray`.

See Also

“C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayFromMat_{DataType}

Convert row major `cv::Mat` to column major `mxArray` for generic matrix

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvMxArrayFromMat_double(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_single(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_uint8(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_uint16(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_uint32(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_int8(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_int16(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_int32(const cv::Mat &in);
mxArray *ocvMxArrayFromMat_bool(const cv::Mat &in);
```

Arguments

`in`

Reference to OpenCV `cv::Mat` with row major data.

Returns

Pointer to a MATLAB struct, `mxArray`, having column major data. Supported data types:

`real_T` (`double`)	`uint8_T` (`uint8`)	`uint32_T` (`uint32`)	`int16_T` (`int16`)
`real32_T` (`single`)	`uint16_T` (`uint16`)	`int8_T` (`int8`)	`int32_T` (`int32`)

Description

The `ocvMxArrayFromMat_{DataType}` function creates an `mxArray` from a `cv::Mat` object. The `mxArray` contains column major data and `cv::Mat` contains row major data. This matrix conversion is a generic routine for any number of channels.

See Also

`mxArray`, `ocvMxArrayToImage_{DataType}`, `ocvMxArrayFromImage_{DataType}`, `ocvMxArrayToMat_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvMxArrayFromImage_{DataType}

Convert row major `cv::Mat` to column major `mxArray` for image

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvMxArrayFromImage_double(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_single(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_uint8(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_uint16(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_uint32(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_int8(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_int16(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_int32(const cv::Mat &in);
mxArray *ocvMxArrayFromImage_bool(const cv::Mat &in);
```

Arguments

`in`

Reference to OpenCV `cv::Mat` with row major data.

Returns

Pointer to a MATLAB struct, `mxArray`, with column major data. Supported data types:

`real_T` (`double`)	`uint8_T` (`uint8`)	`uint32_T` (`uint32`)	`int16_T` (`int16`)
`real32_T` (`single`)	`uint16_T` (`uint16`)	`int8_T` (`int8`)	`int32_T` (`int32`)

Description

The `ocvMxArrayFromImage_{DataType}` function creates an `mxArray` from a `cv::Mat` object. The `mxArray` contains column major data and the `cv::Mat` contains row major data.

This function is not a generic matrix conversion routine. For 3-D images, it takes into account that the OpenCV format uses BGR ordering and manipulates the data to comply with that formatting.

See Also

`mxArray`, `ocvMxArrayToImage_{DataType}`, `ocvMxArrayFromMat_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvKeyPointsToStruct

Convert OpenCV `KeyPoint` vector to MATLAB struct

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvKeyPointsToStruct(cv::vector<cv::KeyPoint> &in);
```

Arguments

`in`

Reference to an OpenCV's KeyPoint vector.

Returns

Pointer to a MATLAB structure `mxArray` that represents a point feature.

Format:

Field Name	Field Requirement	Field Data Type
`Location`	Required	Single
`Scale`	Required	Single
`Metric`	Required	Single
`Orientation`	Optional	Single
`Octave`	Optional	int32
`Misc`	Optional	int32

Description

The `ocvKeyPointsToStruct` function converts a point feature data structure from an OpenCV KeyPoint vector to a MATLAB struct.

See Also

`mxArray`, `ocvStructToKeyPoints`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvCvRectToMxArray

Convert OpenCV `CvRect` to a MATLAB struct

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvCvRectToMxArray(const CvRect *in);
```

Arguments

`in`

Pointer to OpenCV `CvRect`.

Returns

Pointer to a MATLAB structure, `mxArray`, that represents a rectangle. The structure must have four scalar-valued fields, `x`, `y`, `width`, and `height`. The (`x`, `y`) fields represent the upper-left corner of the rectangle.

Description

The `ocvCvRectToMxArray` function converts a rectangle data structure from an OpenCV KeyPoint vector to a MATLAB struct.

See Also

`mxArray`, `ocvMxArrayToCvRect`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvCvRectToBoundingBox_{DataType}

Convert `vector<cv::Rect>` to *M*-by-4 `mxArray` of bounding boxes

C++ Syntax

```
#include "opencvmex.hpp"
mxArray * ocvCvRectToBoundingBox_double(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_single(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_uint8(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_uint16(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_uint32(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_int8(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_int16(const std::vector<cv::Rect> & rects);
mxArray * ocvCvRectToBoundingBox_int32(const std::vector<cv::Rect> & rects);
```

Arguments

`rects`

Reference to OpenCV `vector<cv::Rect>`.

Returns

Pointer to a MATLAB `mxArray` having *M*-by-4 elements. Supported data types:

`real_T` (`double`)	`uint8_T` (`uint8`)	`uint32_T` (`uint32`)	`int16_T` (`int16`)
`real32_T` (`single`)	`uint16_T` (`uint16`)	`int8_T` (`int8`)	`int32_T` (`int32`)

Description

The `ocvCvRectToBoundingBox_{DataType}` function converts `vector<cv::Rect>` to an *M*-by-4 `mxArray` of bounding boxes.

See Also

`mxArray`, `ocvCvBox2DToMxArray`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvCvBox2DToMxArray

Convert OpenCV `CvBox2D` to a MATLAB struct

C++ Syntax

```
#include "opencvmex.hpp"
mxArray *ocvCvBox2DToMxArray(const CvBox2D *in);
```

Arguments

`in`

Pointer to OpenCV `CvBox2D`.

Returns

Pointer to a MATLAB structure, `mxArray`, that represents a rectangle. The structure must have five scalar-valued fields: `x_center`, `y_center`, `width`, `height`, and `angle`. The (`x_center`, `y_center`) fields represent the center of the rectangle.

Description

The `ocvCvBox2DToMxArray` function converts a rectangle data structure from an OpenCV `CvBox2D` to a MATLAB struct.

See Also

`mxArray`, `ocvCvRectToBoundingBox_{DataType}`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

ocvCheckFeaturePointsStruct

Check that MATLAB struct represents feature points

C++ Syntax

```
#include "opencvmex.hpp"
void ocvCheckFeaturePointsStruct(const mxArray *in);
```

Arguments

`in`

Pointer to a MATLAB structure, `mxArray`, that represents point feature. Format:

Field Name	Field Requirement	Field Data Type
`Location`	Required	Single
`Scale`	Required	Single
`Metric`	Required	Single
`Orientation`	Optional	Single
`Octave`	Optional	int32
`Misc`	Optional	int32

Description

The `ocvCheckFeaturePointsStruct` function performs the key point struct checker.

See Also

`mxArrayocvStructToKeyPoints`, `ocvKeyPointsToStruct`, “C Matrix API”, “Write C Functions Callable from MATLAB (MEX Files)”

Version History

Introduced in R2015a

createMat

Create MATLAB interface object for OpenCV `Mat` class

Syntax

```
[ocvMat,ocvInputArray] = createMat(img)
[ocvMat,ocvArray] = createMat(img,arrayType)
[ocvMat,ocvOutputArray] = createMat
[ocvMat,ocvArray] = createMat(arrayType)
```

Description

`[ocvMat,ocvInputArray] = createMat(img)` creates the MATLAB interface objects for the OpenCV `Mat` class and the associated `InputArray` class. Use these interface objects to pass an image as input to the OpenCV functions.

`[ocvMat,ocvArray] = createMat(img,arrayType)` creates the MATLAB interface objects for the OpenCV `Mat` class and the associated `InputArray`, `OutputArray`, or `InputOutputArray` class specified by `arrayType`.

`[ocvMat,ocvOutputArray] = createMat` creates an empty interface object for the OpenCV `Mat` class and the associated `OutputArray` class. Use this interface to write the output returned by an OpenCV function.

`[ocvMat,ocvArray] = createMat(arrayType)` creates an empty interface object for the OpenCV `Mat` class and the associated `InputArray`, `OutputArray`, or `InputOutputArray` class specified by `arrayType`.

Examples

Create MATLAB Interface Object for OpenCV `Mat` and Associated Array Classes

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Create MATLAB Interface Object for OpenCV `Mat` and `InputArray` Class

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create a MATLAB interface object for the OpenCV `Mat` class and store the image data. The function also creates an interface object for the OpenCV `InputArray` class that is constructed from the `Mat` class.

```
[ocvMat,ocvArray] = createMat(img);
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `Mat` class.

```
ocvMat
```

```
ocvMat =
  MatND with properties:

        flags: 1124024320
         dims: 2
         rows: 256
         cols: 256
    allocator: [1×1 clib.opencv.cv.MatAllocator]
            u: [1×1 clib.opencv.cv.UMatData]
         size: [1×1 clib.opencv.cv.MatSize]
         step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_InputArray with no properties.
```

Use the `getImage` utility function to read the image stored in the `InputArray` class. Display the image.

```
outImg = getImage(ocvArray);
figure
imshow(outImg)
```



Create MATLAB Interface Object for OpenCV `Mat` and `OutputArray` Classes

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Create a MATLAB interface object for the OpenCV `Mat` class. The function also creates an interface object for the OpenCV `OutputArray` class that is constructed from the `Mat` class.

```
[ocvMat,ocvArray] = createMat;
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `Mat` class.

```
ocvMat
```

```
ocvMat =
  MatND with properties:

        flags: 1124007936
         dims: 0
         rows: 0
         cols: 0
    allocator: [1×1 clib.opencv.cv.MatAllocator]
            u: [1×1 clib.opencv.cv.UMatData]
         size: [1×1 clib.opencv.cv.MatSize]
         step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_OutputArray with no properties.
```

Specify Array Class to Construct from OpenCV `Mat` Class

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create a MATLAB interface object for the OpenCV `Mat` class. Specify the array type as `"InputOutput"` for the function to construct the OpenCV `InputOutputArray` class from the `Mat` class.

```
arrayType = "InputOutput";
[ocvMat,ocvArray] = createMat(img,arrayType);
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `Mat` class.

```
ocvMat
```

```
ocvMat =
  MatND with properties:

        flags: 1124024320
         dims: 2
         rows: 256
```

```
        cols: 256
   allocator: [1×1 clib.opencv.cv.MatAllocator]
           u: [1×1 clib.opencv.cv.UMatData]
        size: [1×1 clib.opencv.cv.MatSize]
        step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_InputOutputArray with no properties.
```

Input Arguments

`img` — Input image

M-by-*N* matrix | *M*-by-*N*-by-3 array

Input image, specified as an *M*-by-*N* matrix representing an intensity image or *M*-by-*N*-by-3 array representing a color image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16`

`arrayType` — Type of array

`"Input"` | `"Output"` | `"InputOutput"`

Type of array to associate with the MATLAB interface object for the OpenCV `Mat` class, specified as: `"Input"`, `"Output"`, or `"InputOutput"`.

Data Types: `char` | `string`

Output Arguments

`ocvMat` — *n*-dimensional dense array class

`MatND` interface object

n-dimensional dense array class, returned as a `MatND` interface object. This interface object is a representation of the OpenCV class `cv::Mat`.

`ocvInputArray` — Proxy class for passing inputs

`InputArray` interface object

Proxy class for passing inputs to OpenCV functions, returned as an `InputArray` interface object. This interface object is a representation of the OpenCV class `cv::_InputArray`.

`ocvArray` — Proxy class for inputs or outputs

`InputArray` interface object | `InputOutputArray` interface object | `OutputArray` interface object

Proxy class for inputs or outputs, returned as one of these values:

- `InputArray` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::_InputArray`. This value is returned if `arrayType` is specified as `"Input"`.

- `InputOutputArray` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::_InputOutputArray`. This value is returned if `arrayType` is specified as `"InputOutput"`.
- `OutputArray` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::_OutputArray`. This value is returned if `arrayType` is specified as `"Output"`.

`ocvOutputArray` — Proxy class for writing output
`OutputArray` interface object

Proxy class for writing outputs from OpenCV functions, returned as an `OutputArray` interface object. This MATLAB interface object is a representation of the OpenCV class `cv::_OutputArray`.

Version History

Introduced in R2021b

See Also

`createUMat` | `getImage` | `getBasePtr` | `keyPointsToStruct` | `rectToBbox`

Topics

“Use Prebuilt MATLAB Interface to OpenCV”

createUMat

Create MATLAB interface object for OpenCV `UMat` class

Syntax

```
[ocvUMat,ocvInputArray] = createUMat(img)
[ocvUMat,ocvArray] = createUMat(img,arrayType)
[ocvUMat,ocvOutputArray] = createUMat
[ocvUMat,ocvArray] = createUMat(arrayType)
```

Description

`[ocvUMat,ocvInputArray] = createUMat(img)` creates the MATLAB interface objects for the OpenCV `UMat` class and the associated `InputArray` class. Use these interface objects to pass an image as input to the OpenCV functions.

`[ocvUMat,ocvArray] = createUMat(img,arrayType)` creates the MATLAB interface objects for the OpenCV `UMat` class and the associated `InputArray`, `OutputArray`, or `InputOutputArray` class specified by `arrayType`.

`[ocvUMat,ocvOutputArray] = createUMat` creates an empty interface object for the OpenCV `UMat` class and the associated `OutputArray` class. Use this interface to write the output returned by an OpenCV function.

`[ocvUMat,ocvArray] = createUMat(arrayType)` creates an empty interface object for the OpenCV `UMat` class and the associated `InputArray`, `OutputArray`, or `InputOutputArray` class specified by `arrayType`.

Examples

Create MATLAB Interface Object for OpenCV `UMat` and Associated Array Classes

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Create MATLAB Interface Object for OpenCV `UMat` and `InputArray` Class

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create a MATLAB interface object for the OpenCV `UMat` class and store the image data. The function also creates an interface object for the OpenCV `InputArray` class that is constructed from the `UMat` class.

```
[ocvMat,ocvArray] = createUMat(img);
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `UMat` class.

```
ocvMat
```

```
ocvMat =
  UMat with properties:

         flags: 1124024320
          dims: 2
          rows: 256
          cols: 256
     allocator: [1×1 clib.opencv.cv.MatAllocator]
    usageFlags: USAGE_DEFAULT
             u: [1×1 clib.opencv.cv.UMatData]
        offset: 0
          size: [1×1 clib.opencv.cv.MatSize]
          step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_InputArray with no properties.
```

Use the `getImage` utility function to read the image stored in the `InputArray` class. Display the image.

```
outImg = getImage(ocvArray);
figure
imshow(outImg)
```



Create MATLAB Interface Object for OpenCV `UMat` and `OutputArray` Classes

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Create a MATLAB interface object for the OpenCV `UMat` class. The function also creates an interface object for the OpenCV `OutputArray` class that is constructed from the `UMat` class.

```
[ocvMat,ocvArray] = createUMat;
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `UMat` class.

```
ocvMat
```

```
ocvMat =
  UMat with properties:

         flags: 1124007936
          dims: 0
          rows: 0
          cols: 0
     allocator: [1×1 clib.opencv.cv.MatAllocator]
    usageFlags: USAGE_DEFAULT
             u: [1×1 clib.opencv.cv.UMatData]
        offset: 0
          size: [1×1 clib.opencv.cv.MatSize]
          step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_OutputArray with no properties.
```

Specify Array Class to Construct from OpenCV `UMat` Class

Add the prebuilt MATLAB interface to OpenCV package name to the import list.

```
import vision.opencv.util.*
```

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create a MATLAB interface object for the OpenCV `UMat` class. Specify the array type as `"InputOutput"` for the function to construct the OpenCV `InputOutputArray` class from the `UMat` class.

```
arrayType = "InputOutput";
[ocvMat,ocvArray] = createUMat(img,arrayType);
```

Display and inspect the properties of the MATLAB interface object for the OpenCV `UMat` class.

```
ocvMat
```

```
ocvMat =
  UMat with properties:

         flags: 1124024320
          dims: 2
          rows: 256
          cols: 256
     allocator: [1×1 clib.opencv.cv.MatAllocator]
    usageFlags: USAGE_DEFAULT
             u: [1×1 clib.opencv.cv.UMatData]
        offset: 0
          size: [1×1 clib.opencv.cv.MatSize]
          step: [1×1 clib.opencv.cv.MatStep]
```

Inspect the type of array class returned by the function.

```
ocvArray
```

```
ocvArray =
  x_InputOutputArray with no properties.
```

Input Arguments

`img` — Input image

M-by-*N* matrix | *M*-by-*N*-by-3 array

Input image, specified as an *M*-by-*N* matrix representing an intensity image or *M*-by-*N*-by-3 array representing a color image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16`

`arrayType` — Type of array

`"Input"` | `"Output"` | `"InputOutput"`

Type of array to associate with the MATLAB interface object for OpenCV `UMat` class, specified as : `"Input"`, `"Output"`, or `"InputOutput"`.

Data Types: `char` | `string`

Output Arguments

`ocvUMat` — Unified *n*-dimensional dense array class

`UMat` interface object

Unified *n*-dimensional dense array class, returned as a `UMat` interface object. This MATLAB interface object is a representation of the OpenCV class `cv::UMat`.

`ocvInputArray` — Proxy class for passing inputs

`InputArray` interface object

Proxy class for passing inputs to OpenCV functions, returned as an `InputArray` interface object. This MATLAB interface object is a representation of the OpenCV class `cv::_InputArray`.

ocvArray — Proxy class for inputs or outputs
`InputArray` interface object | `InputOutputArray` interface object | `OutputArray` interface object

Proxy class for inputs or outputs, returned as one of these values:

- `InputArray` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::_InputArray`. This value is returned if `arrayType` is specified as `"Input"`.
- `InputOutputArray` interface object — This MATLAB interface object is a representation of the OpenCV class, `cv::_InputOutputArray`. This value is returned if `arrayType` is specified as `"InputOutput"`.
- `OutputArray` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::_OutputArray`. This value is returned if `arrayType` is specified as `"Output"`.

ocvOutputArray — Proxy class for writing output
`OutputArray` interface object

Proxy class for writing outputs from OpenCV functions, returned as an `OutputArray` interface object. This MATLAB interface object is a representation of the OpenCV class `cv::_OutputArray`.

Version History

Introduced in R2021b

See Also

`rectToBbox` | `getImage` | `getBasePtr` | `createMat` | `keyPointsToStruct`

Topics
“Use Prebuilt MATLAB Interface to OpenCV”

getBasePtr

Namespace: `vision.opencv.util`

Create MATLAB interface object for OpenCV base class

Syntax

```
output = getBasePtr(ptr)
```

Description

`output = getBasePtr(ptr)` creates a MATLAB interface object for an OpenCV base class pointed by a smart pointer `ptr`. You can use the interface object to access the public member functions and public attributes of an OpenCV base class directly from MATLAB.

Examples

Perform Histogram Equalization by Using OpenCV in MATLAB

Perform histogram equalization of an image by using the prebuilt MATLAB interface to the OpenCV function `cv::CLAHE`. The `cv::CLAHE` function is a base class, and you must create a MATLAB interface object by using the `getBasePtr` utility function to access the public methods and attributes of the base class.

Add the MATLAB interface to OpenCV package names to the import list.

```
import clib.opencv.*;
import vision.opencv.util.*;
```

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create the MATLAB interface objects for the OpenCV `MatND` and `InputArray` classes to store the input image.

```
[inputMat,inputArray] = createMat(img);
```

Create the MATLAB interface objects for the OpenCV `MatND` and `OutputArray` classes to write the output image returned by the OpenCV function.

```
[outputMat,outputArray] = createMat;
```

Create a MATLAB interface object to represent the smart pointer of the OpenCV CLAHE class `cv::CLAHE`.

```
ocvPtr = cv.createCLAHE;
```

Create a MATLAB interface object for the base class `cv::CLAHE`.

```
basePtr = getBasePtr(ocvPtr);
```

Set the value of clip limit to 4.

```
basePtr.setClipLimit(4);
```

Perform contrast-limited adaptive histogram equalization by using the `apply` method of the `cv::CLAHE` base class.

```
basePtr.apply(inputArray,outputArray);
```

Read the enhanced output image.

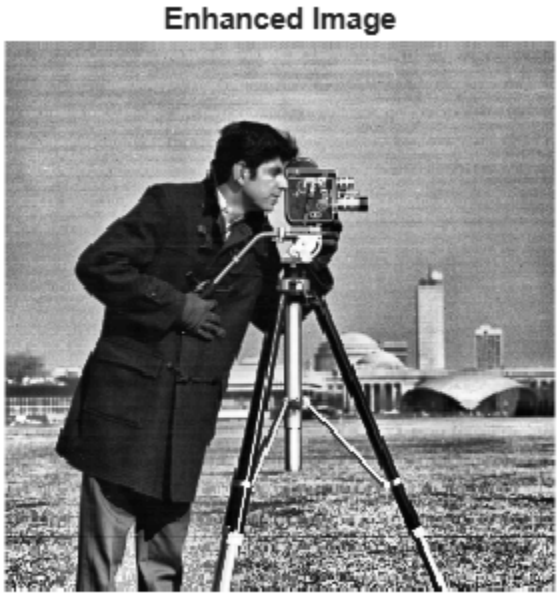
```
enhanced = getImage(outputArray);
```

Display the original input and the enhanced output image.

```
figure
imshow(img)
title("Input Image")
```




```
figure
imshow(enhanced)
title("Enhanced Image")
```



Input Arguments

`ptr` — OpenCV smart pointer

`Ptr_cv_<_T>` interface object

OpenCV smart pointer, specified as a `Ptr_cv_<_T>` interface object. This MATLAB interface object is a representation of an OpenCV smart pointer `cv::Ptr<_T>`.

Output Arguments

`output` — Representation of OpenCV base class

MATLAB interface object

Representation of OpenCV base class, returned as a MATLAB interface object. You can use the object to directly access the public member functions and public attributes of an OpenCV base class.

Version History

Introduced in R2021b

See Also

`rectToBbox` | `getImage` | `createMat` | `keyPointsToStruct` | `createUMat`

Topics

“Use Prebuilt MATLAB Interface to OpenCV”

getImage

Namespace: `vision.opencv.util`

Read images stored in MATLAB interface object for OpenCV class

Syntax

```
outputImg = getImage(ocvInput)
```

Description

`outputImg = getImage(ocvInput)` reads the image stored in a MATLAB interface object for an OpenCV class and writes it to the specified output variable `outputImg` in the MATLAB workspace.

Examples

Read Image Stored in OpenCV `Mat` Class

Add the MATLAB interface to OpenCV package names to the import list.

```
import clib.opencv.*
import vision.opencv.util.*
```

Read an image into the MATLAB workspace.

```
img = imread("cameraman.tif");
```

Create a MATLAB interface object for the OpenCV `InputOutput` array class.

```
[ocvMat,ocvArray] = createMat(img);
```

Read the image stored in the `InputOutput` interface object.

```
output = getImage(ocvArray);
```

Display the output image.

```
figure
imshow(output)
```



Input Arguments

`ocvInput` — MATLAB interface object for OpenCV class

`MatND` interface object | `UMat` interface object | `InputArray` interface object | `InputOutputArray` interface object | `OutputArray` interface object

MATLAB interface object for an OpenCV class, specified as one of these values:

- `MatND` interface object — Created by using the `createMat` function and represents the OpenCV `cv::Mat` class.
- `UMat` interface object — Created by using the `createUMat` function and represents the OpenCV `cv::UMat` class.
- `InputArray` interface object — Created by using the `createMat` or `createUMat` function and represents the OpenCV `cv::_InputArray` class. The `InputArray` interface object passes array inputs to the OpenCV functions.
- `InputOutputArray` interface object — Created by using the `createMat` or `createUMat` function and represents the OpenCV `cv::_InputOutputArray` class. The `InputOutputArray` interface object passes array inputs to the OpenCV functions and writes the array outputs returned by the OpenCV functions.
- `OutputArray` interface object — Created by using the `createMat` or `createUMat` function and represents the OpenCV `cv::_OutputArray` class. The `OutputArray` interface object contains the array outputs returned by the OpenCV functions.

Output Arguments

`outputImg` — Output image

M-by-*N* matrix | *M*-by-*N*-by-3 array

Output image read from a MATLAB interface object, returned as an *M*-by-*N* matrix for intensity images and *M*-by-*N*-by-3 array for color images.

Version History

Introduced in R2021b

See Also

`createMat` | `createUMat`

Topics

“Use Prebuilt MATLAB Interface to OpenCV”

keyPointsToStruct

Namespace: `vision.opencv.util`

Convert OpenCV `KeyPoints` object to MATLAB structure

Syntax

```
mlstruct = keyPointsToStruct(keypoints)
```

Description

`mlstruct = keyPointsToStruct(keypoints)` converts an OpenCV `KeyPoints` object to a MATLAB structure.

Examples

Detect Keypoints in Image Using OpenCV in MATLAB

This example shows how to use the prebuilt MATLAB interface for the OpenCV function `cv::Fast` in MATLAB to detect keypoints in an image. Additionally, use the `keyPointsToStruct` utility function to write the keypoints returned by the OpenCV `cv::Fast` function to a MATLAB structure.

Add the MATLAB interface to OpenCV package names to the import list.

```
import clib.opencv.*;
import vision.opencv.util.*;
```

Read an image into the MATLAB workspace.

```
img = imread("elephant.jpg");
```

Create MATLAB interface objects for the OpenCV `MatND` and `InputArray` classes to store the input image.

```
[inputMat,inputArray] = createMat(img);
```

Create a MATLAB interface object for the OpenCV `KeyPoint` vector by using the `clibArray` function.

```
keyPointsVec = clibArray("clib.opencv.cv.KeyPoint",0);
```

Specify the parameters for computing keypoints using the FAST detector.

```
threshold = 100;
nonmaxSuppression = true;
```

Compute keypoints in the image by calling the OpenCV function `cv::FAST` in MATLAB.

```
cv.FAST(inputArray,keyPointsVec,threshold,nonmaxSuppression);
```

Convert the `KeyPoints` object returned by the OpenCV function into a MATLAB structure.

```
mlstruct = keyPointsToStruct(keyPointsVec);
```

Inspect the fields in the output MATLAB structure.

```
mlstruct
```

```
mlstruct = struct with fields:
       Location: [48×2 double]
          Scale: [48×1 double]
         Metric: [48×1 double]
           Misc: [48×1 double]
    Orientation: [48×1 double]
```

Display the input image and plot the detected keypoints.

```
figure
imshow(img)
hold on
plot(mlstruct.Location(:,1),mlstruct.Location(:,2),"*r")
hold off
```



Input Arguments

`keypoints` — OpenCV `KeyPoints` class

MATLAB interface object

OpenCV `KeyPoints` class, specified as a MATLAB interface object. This interface object is a representation of the `KeyPoints` class returned by any of the OpenCV functions for keypoint detection.

Output Arguments

`mlstruct` — Keypoints detected using OpenCV function

structure

Keypoints detected using the OpenCV function, returned as a MATLAB structure with fields `Location`, `Scale`, `Metric`, `Misc`, and `Orientation`.

Fields	Description
`Location`	*x* and *y*-coordinates of the keypoints.
`Scale`	Diameter of the neighborhood region around the keypoints.
`Metric`	Strength of the keypoints.
`Orientation`	Orientation of the keypoints.

Version History

Introduced in R2021b

See Also

`rectToBbox` | `getImage` | `getBasePtr` | `createUMat` | `createMat` | `clibArray`

Topics

“Use Prebuilt MATLAB Interface to OpenCV”

rectToBbox

Namespace: `vision.opencv.util`

Compute bounding boxes from outputs returned by MATLAB interface object for OpenCV `Rect` class

Syntax

```
bbox = rectToBbox(input)
```

Description

`bbox = rectToBbox(input)` computes bounding box values from the outputs returned by a MATLAB interface object for OpenCV `Rect` class. The OpenCV `Rect` class creates rectangles on an input image. This function maps the zero-based indexing in OpenCV to one-based indexing in MATLAB.

Examples

Draw Rectangle on Image Using OpenCV in MATLAB

Draw a rectangle on an image by using the prebuilt MATLAB interface for the OpenCV function `cv::rectangle`. Compute the corresponding bounding box values in MATLAB to use to crop the image region.

Add the MATLAB interface to OpenCV package names to the import list.

```
import clib.opencv.*;
import vision.opencv.util.*;
```

Read an image into the MATLAB workspace.

```
img = imread("highway.png");
```

Create interface objects for the OpenCV `Mat` and `InputOutputArray` classes to store the input image.

```
[inputMat,ocvArray] = createMat(img,"InputOutput");
```

Define Rectangular Region

Call the OpenCV function `cv::Rect2i` using MATLAB, and specify the coordinates and dimensions for the rectangle. Display the values.

```
rec = cv.Rect2i(140,60,100,100)
```

```
rec =
  Rect2i with properties:

         x: 140
         y: 60
     width: 100
```

```
    height: 100
```

Draw Rectangle and Display Results

Specify the properties of the line to use for drawing the rectangle. Set these values:

- Color of the line to red. To set this value, use the OpenCV function `cv::Scalar`.
- Thickness of the line to `2`.
- Line type to `4`.
- Shift to `0`.

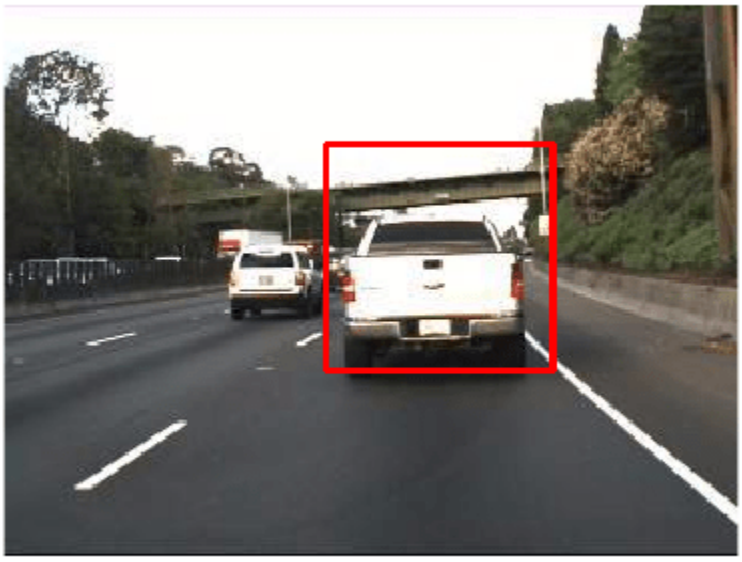
```
color = cv.Scalar(255,0,0);
thickness = 2;
lineType = 4;
shift = 0;
```

Draw the defined rectangle on the image by using the OpenCV function `cv::rectangle`.

```
cv.rectangle(ocvArray,rec,color,thickness,lineType,shift);
```

Read and display the output image.

```
outputImg = getImage(ocvArray);
figure
imshow(outputImg)
```



Compute Bounding Box

Use the utility function `rectToBbox` to compute the bounding box value from the values returned by OpenCV function `cv::Rect2i`.

```
bbox = rectToBbox(rec);
```

Display the bounding box values. Notice that the bounding box values in the MATLAB workspace have one-based indexing.

```
bbox
```

```
bbox = 1×4 int32 row vector

   141    61   100   100
```

Crop the region within the bounding box by using the `imcrop` function and display the cropped image.

```
croppedImg = imcrop(img,bbox);
figure
imshow(croppedImg)
```



Input Arguments

`input` — OpenCV class for rectangles

`Rect__unsignedChar_` interface object | `Rect2d` interface object | `Rect2f` interface object | `Rect2i` interface object

OpenCV class for rectangles, specified as one of these values:

- `Rect__unsignedChar_` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::Rect_<unsigned char>`.
- `Rect2d` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::Rect_<double>`.
- `Rect2f` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::Rect_<float>`.

- `Rect2i` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::Rect_<int>`.
- `Rect2i` interface object — This MATLAB interface object is a representation of the OpenCV class `cv::Rect_<int>`.

Output Arguments

bbox — Bounding box values
four-element row vector

Bounding box values, returned as a four-element row vector of the form [`x y width height`].

Version History

Introduced in R2021b

See Also

`createMat` | `getImage` | `createUMat` | `getBasePtr`

Topics
“Use Prebuilt MATLAB Interface to OpenCV”

times, .*

Element-wise transformation or rotation multiplication

Syntax

```
transformationC = transformationA.*transformationB
rotationC = rotationA.*rotationB
```

Description

`transformationC = transformationA.*transformationB` multiplies transformations element-by-element by multiplying each element of transformation `transformationA` with the corresponding element of transformation `transformationB` and returns the product, transformation `transformationC`.

`rotationC = rotationA.*rotationB` multiplies rotations element-by-element by multiplying each element of rotation `rotationA` with the corresponding element of rotation `rotationB` and returns the product, rotation `rotationC`.

Input Arguments

`transformationA` — First transformation

`se3` object | *N*-element array of transformation objects

First transformation, a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationA` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`transformationB` — Last transformation

`se3` object | *N*-element array of transformation objects

Last transformation, specified as a scalar `se3` object or as an *N*-element array of transformation objects. *N* is the total number of transformations.

If you specify `transformationB` as an array, each element must be of the same type.

Either `transformationA` or `transformationB` must be a scalar transformation object of the same type. For example, if `transformationA` is an array of `se3` objects, `transformationB` must be a scalar `se3` object.

`rotationA` — First rotation

`so3` object | *N*-element array of rotation objects

First rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationA` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `so3` objects, `rotationB` must be a scalar `so3` object.

`rotationB` — Last rotation
`so3` object | *N*-element array of rotation objects

Last rotation, specified as a scalar `so3` object or as an *N*-element array of rotation objects. *N* is the total number of rotations.

If you specify `rotationB` as an array, each element must be of the same type.

Either `rotationA` or `rotationB` must be a scalar rotation object of the same type. For example, if `rotationA` is an array of `se3` objects, `rotationB` must be a scalar `se3` object.

Output Arguments

`transformationC` — Transformation product
`se3` object | *N*-element array of transformation objects

Transformation product, returned as a scalar `se3` object or as an *N*-element array of the same transformation type as `transformationA` and `transformationB`. *N* is the length of the longer argument between `transformationA` and `transformationB` and each row represents the product between `transformationA` and `transformationB`.

`rotationC` — Rotation product
`so3` object | *N*-element array of rotation objects

Rotation product, returned as a scalar `so3` object or as an *N*-element array of the same rotation type as `rotationA` and `rotationB`. *N* is the length of the longer argument between `rotationA` and `rotationB` and each row represents the product between `rotationA` and `rotationB`.

See Also

Functions
`mrdivide, /` | `mtimes, *`

Objects
`se3` | `so3`

trainSSDObjectDetector

Train SSD deep learning object detector

Syntax

```
trainedDetector = trainSSDObjectDetector(trainingData,detector,options)

trainedDetector = trainSSDObjectDetector(trainingData,checkpoint,options)

trainedDetector = trainSSDObjectDetector( ___ ,Name=Value)
[trainedDetector,info] = trainSSDObjectDetector( ___ )
```

Description

Train a Detector

`trainedDetector = trainSSDObjectDetector(trainingData,detector,options)` trains a single shot multibox detector (SSD) using deep learning. You can train an SSD detector to detect multiple object classes. Use this syntax to train either an untrained or pretrained SSD object detection network. You can also use this syntax to fine-tune a network with additional training data or to perform more training iterations to improve detector accuracy.

This function requires that you have Deep Learning Toolbox. It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

Resume Training a Detector

`trainedDetector = trainSSDObjectDetector(trainingData,checkpoint,options)` resumes training from a detector checkpoint.

Additional Properties

`trainedDetector = trainSSDObjectDetector( ___ ,Name=Value)` uses additional options specified by one or more name-value arguments and any of the previous inputs.

`[trainedDetector,info] = trainSSDObjectDetector( ___ )` also returns information on the training progress, such as training loss and accuracy, for each iteration.

Examples

Train SSD Object Detector

This example shows how to train an SSD object detector on a vehicle data set. The example then uses the trained detector for detecting vehicles in an image.

Load the training data into the workspace.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Specify the directory in which training samples are stored. Add full path to the filenames in training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an image datastore using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a box label datastore using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the datastores.

```
ds = combine(imds,blds);
```

Specify a base network.

```
baseNetwork = imagePretrainedNetwork("resnet50");
```

Specify the names of the classes to detect.

```
classNames = "vehicle";
```

Specify the anchor boxes to use for training the network.

```
anchorBoxes = { ...
    [30 60; 60 30; 50 50; 100 100], ...
    [40 70; 70 40; 60 60; 120 120]};
```

Specify the names of the feature extraction layers to connect to the detection subnetwork.

```
layersToConnect = ["activation_22_relu" "activation_40_relu"];
```

Create an SSD object detector by using the `ssdObjectDetector` function.

```
detector = ssdObjectDetector(baseNetwork,classNames,anchorBoxes, ...
    DetectionNetworkSource=layersToConnect);
```

Specify the training options.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.001, ...
    MiniBatchSize=16, ...
    Verbose=true, ...
    MaxEpochs=30, ...
    Shuffle="never", ...
    VerboseFrequency=10);
```

Train the SSD object detector.

```
[detector,info] = trainSSDObjectDetector(ds,detector,options);
```

```
*************************************************************************
Training an SSD Object Detector for the following object classes:

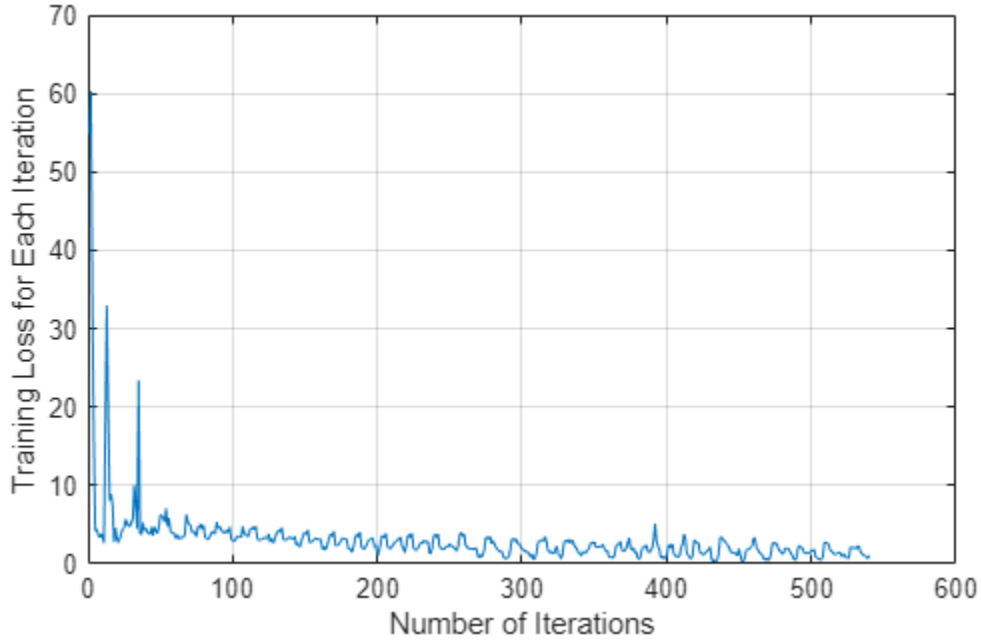
* vehicle
```

```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Lea
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate
|=========================================================================================================
|       1 |           1 |       00:00:00 |      54.8573 |       43.62% |         2.56 |
|       1 |          10 |       00:00:04 |       3.5709 |       98.90% |         1.65 |
|       2 |          20 |       00:00:07 |       3.2212 |       99.93% |         0.99 |
|       2 |          30 |       00:00:11 |       5.3296 |       98.89% |         1.51 |
|       3 |          40 |       00:00:15 |       4.5304 |       99.76% |         1.15 |
|       3 |          50 |       00:00:18 |       6.2441 |       98.47% |         1.28 |
|       4 |          60 |       00:00:22 |       3.6098 |       98.95% |         1.26 |
|       4 |          70 |       00:00:25 |       5.0838 |       98.81% |         1.23 |
|       5 |          80 |       00:00:29 |       4.9031 |       98.84% |         1.47 |
|       5 |          90 |       00:00:32 |       4.5612 |       99.91% |         0.99 |
|       6 |         100 |       00:00:36 |       3.0531 |       98.90% |         1.13 |
|       7 |         110 |       00:00:40 |       3.4591 |       99.95% |         0.74 |
|       7 |         120 |       00:00:43 |       3.1390 |       99.10% |         1.02 |
|       8 |         130 |       00:00:47 |       3.7369 |       99.77% |         1.14 |
|       8 |         140 |       00:00:50 |       3.3740 |       99.07% |         0.99 |
|       9 |         150 |       00:00:54 |       3.9861 |       98.99% |         1.35 |
|       9 |         160 |       00:00:57 |       3.1168 |       99.24% |         0.90 |
|      10 |         170 |       00:01:01 |       4.1509 |       98.92% |         1.40 |
|      10 |         180 |       00:01:04 |       1.8926 |       99.97% |         0.55 |
|      11 |         190 |       00:01:08 |       1.9993 |       99.13% |         0.88 |
|      12 |         200 |       00:01:11 |       1.0826 |       99.98% |         0.41 |
|      12 |         210 |       00:01:15 |       2.2213 |       99.37% |         0.78 |
|      13 |         220 |       00:01:18 |       2.4177 |       99.84% |         0.99 |
|      13 |         230 |       00:01:22 |       2.7282 |       99.32% |         0.81 |
|      14 |         240 |       00:01:26 |       3.7748 |       99.20% |         1.20 |
|      14 |         250 |       00:01:29 |       2.5727 |       99.33% |         0.77 |
|      15 |         260 |       00:01:33 |       3.8397 |       99.15% |         1.23 |
|      15 |         270 |       00:01:36 |       0.7277 |       99.99% |         0.43 |
|      16 |         280 |       00:01:40 |       2.9342 |       99.22% |         0.78 |
|      17 |         290 |       00:01:43 |       0.8218 |       99.99% |         0.39 |
|      17 |         300 |       00:01:47 |       1.9056 |       99.53% |         0.70 |
|      18 |         310 |       00:01:50 |       1.3405 |       99.93% |         0.71 |
|      18 |         320 |       00:01:54 |       1.5981 |       99.42% |         0.79 |
|      19 |         330 |       00:01:57 |       2.9586 |       99.30% |         1.02 |
|      19 |         340 |       00:02:01 |       1.2590 |       99.43% |         0.66 |
|      20 |         350 |       00:02:05 |       2.8346 |       99.27% |         1.04 |
|      20 |         360 |       00:02:08 |       0.9523 |       99.98% |         0.49 |
|      21 |         370 |       00:02:12 |       1.5545 |       99.26% |         0.74 |
|      22 |         380 |       00:02:16 |       0.7113 |       99.99% |         0.50 |
|      22 |         390 |       00:02:19 |       2.0585 |       99.58% |         0.61 |
|      23 |         400 |       00:02:23 |       1.0087 |       99.93% |         0.55 |
|      23 |         410 |       00:02:26 |       2.1091 |       99.48% |         0.71 |
|      24 |         420 |       00:02:30 |       2.9050 |       99.34% |         0.95 |
|      24 |         430 |       00:02:33 |       2.1874 |       99.44% |         0.64 |
|      25 |         440 |       00:02:37 |       2.9181 |       99.30% |         0.91 |
|      25 |         450 |       00:02:40 |       1.9445 |       99.94% |         0.39 |
|      26 |         460 |       00:02:44 |       3.2013 |       99.20% |         0.71 |
|      27 |         470 |       00:02:48 |       0.3986 |       99.98% |         0.37 |
|      27 |         480 |       00:02:51 |       1.3009 |       99.57% |         0.58 |
|      28 |         490 |       00:02:55 |       1.0920 |       99.96% |         0.60 |
|      28 |         500 |       00:02:58 |       1.7258 |       99.58% |         0.65 |
```

```
|      29 |         510 |       00:03:02 |       2.7426 |       99.42% |          0.86 |
|      29 |         520 |       00:03:05 |       1.4956 |       99.63% |          0.59 |
|      30 |         530 |       00:03:09 |       2.0561 |       99.39% |          0.85 |
|      30 |         540 |       00:03:12 |       1.0817 |       99.98% |          0.78 |
|=========================================================================================================
Training finished: Max epochs completed.
Detector training complete.
*********************************************************************
```

Verify the training accuracy by inspecting the training loss for each iteration.

```
figure
plot(info.TrainingLoss)
grid on
xlabel("Number of Iterations")
ylabel("Training Loss for Each Iteration")
```



Read a test image.

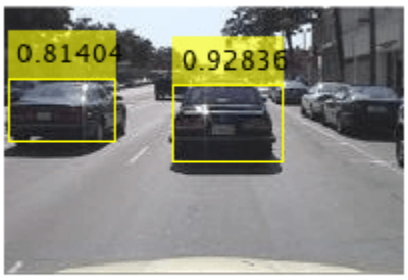
```
img = imread("detectcars.png");
```

Detect vehicles in the test image by using the trained SSD object detector.

```
[bboxes,scores] = detect(detector,img);
```

Display the detection results.

```
if(~isempty(bboxes))
    img = insertObjectAnnotation(img,"rectangle",bboxes,scores);
end
figure
imshow(img)
```



Input Arguments

`trainingData` — Labeled ground truth images

datastore

Labeled ground truth images, specified as a datastore or a table.

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns. The second column must contain the bounding boxes, and the third column must contain the labels. The first column can contain image or point cloud data.

data	*boxes*	*labels*
The first column must be images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	The third column must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

  For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

`detector` — Untrained or pretrained SSD object detector

`ssdObjectDetector` object

Untrained or pretrained SSD object detector, specified as a `ssdObjectDetector` object.

`options` — Training options

`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Note The trainSSDObjectDetector function does not support these training options:

- Datastore inputs are not supported when you set the `DispatchInBackground` training option to `true`.
- Datastore inputs are not supported if the `Shuffle` training option is set to `"never"` when the `ExecutionEnvironment` training option is `"multi-gpu"`. For more information about using datastore for parallel training, see “Preprocess Data in the Background or in Parallel” (Deep Learning Toolbox).

`checkpoint` — Saved detector checkpoint
`ssdObjectDetector` object

Saved detector checkpoint, specified as an `ssdObjectDetector` object. To periodically save a detector checkpoint during training, specify `CheckpointPath`. To control how frequently check points are saved see the `CheckPointFrequency` and `CheckPointFrequencyUnit` training options.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, if the `CheckpointPath` property of the object specified by `options` is `"/checkpath"`, you can load a checkpoint MAT file by using this code.

```
data = load("/checkpath/ssd_checkpoint__216__2018_11_16__13_34_30.mat");
checkpoint = data.detector;
```

The name of the MAT file includes the iteration number and timestamp of when the detector checkpoint was saved. The detector is saved in the `detector` variable of the file. Pass this file back into the trainSSDObjectDetector function:

```
ssdDetector = trainSSDObjectDetector(trainingData,checkpoint,options);
```

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"PositiveOverlapRange",[0.5 1]` sets the vertical axis direction to up.

`PositiveOverlapRange` — Range of bounding box overlap ratios
`[0.5 1]` (default) | two-element vector

Range of bounding box overlap ratios between `0` and `1`, specified as a two-element vector. Anchor boxes that overlap with ground truth bounding boxes within the specified range are used as positive training samples. The function computes the overlap ratio using the intersection-over-union between two bounding boxes.

`NegativeOverlapRange` — Range of bounding box overlap ratios
`[0 0.5]` (default) | two-element vector

Range of bounding box overlap ratios between `0` and `1`, specified as a two-element vector. Anchor boxes that overlap with ground truth bounding boxes within the specified range are used as negative

training samples. The function computes the overlap ratio using the intersection-over-union between two bounding boxes.

`ExperimentManager` — Detector training experiment monitoring
`"none"` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see “Train Object Detectors in Experiment Manager”.

Information monitored during training:

- Training loss at each iteration.
- Training accuracy at each iteration.
- Training root mean square error (RMSE) for the box regression layer.
- Learning rate at each iteration.

Validation information when the training `options` input contains validation data:

- Validation loss at each iteration.
- Validation accuracy at each iteration.
- Validation RMSE at each iteration.

Output Arguments

`trainedDetector` — Trained SSD multibox object detector
`ssdObjectDetector` object

Trained SSD object detector, returned as `ssdObjectDetector` object. You can train a SSD object detector to detect multiple object classes.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with eight fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration is calculated as the sum of regression loss and classification loss. To compute the regression loss, the `trainSSDObjectDetector` function uses smooth L1 loss function. To compute the classification loss the `trainSSDObjectDetector` function uses the softmax and binary cross-entropy loss function.
- `TrainingAccuracy` — Training set accuracy at each iteration.
- `TrainingRMSE` — Training root mean squared error (RMSE) is the RMSE calculated from the training loss at each iteration.
- `BaseLearnRate` — Learning rate at each iteration.
- `ValidationLoss` — Validation loss at each iteration.
- `ValidationAccuracy` — Validation accuracy at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.

- `FinalValidationLoss` — Final validation loss at end of the training.
- `FinalValidationRMSE` — Final validation RMSE at end of the training.

Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are assigned as `NaN`. The struct contains `ValidationLoss`, `ValidationAccuracy`, `ValidationRMSE`, `FinalValidationLoss`, and `FinalValidationRMSE` fields only when `options` specifies validation data.

Version History

Introduced in R2020a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2022a: `LayerGraph` input is not recommended
Not recommended starting in R2022a

Starting in R2022a, use of `LayerGraph` object to specify SSD object detection network as input to the `trainSSDObjectDetector` is not recommended.

The syntax `trainSSDObjectDetector(trainingData,net,options)`, specifying the input SSD detection network `net` as a `LayerGraph` object will be removed in a future release.

If your SSD object detection network is a `LayerGraph` object, configure the network as a `ssdObjectDetector` object by using the `ssdObjectDetector` function. Then, use the `ssdObjectDetector` object as input to the `trainSSDObjectDetector` function for training.

References

[1] W. Liu, E. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Fu, and A.C. Berg. "SSD: Single Shot MultiBox Detector." *European Conference on Computer Vision (ECCV)*, Springer Verlag, 2016

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainSSDObjectDetector` function uses a GPU if one is available. You can specify the hardware that the `trainSSDObjectDetector` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainingOptions` | `objectDetectorTrainingData`

Objects
`ssdObjectDetector` | `boxLabelDatastore`

Topics
“Object Detection Using SSD Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using Single Shot Multibox Detector”
“Train Object Detectors in Experiment Manager”
“Getting Started with SSD Multibox Detection”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainFasterRCNNObjectDetector

(Not recommended) Train Faster R-CNN deep learning object detector

Note `trainFasterRCNNObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Syntax

```
trainedDetector = trainFasterRCNNObjectDetector(trainingData,network,options)
[trainedDetector,info] = trainFasterRCNNObjectDetector( ___ )

trainedDetector = trainFasterRCNNObjectDetector(trainingData,checkpoint,
options)

trainedDetector = trainFasterRCNNObjectDetector(trainingData,detector,
options)

trainedDetector = trainFasterRCNNObjectDetector( ___ ,Name,Value)
```

Description

Train a Detector

`trainedDetector = trainFasterRCNNObjectDetector(trainingData,network,options)` trains a Faster R-CNN (regions with convolution neural networks) object detector using deep learning. You can train a Faster R-CNN detector to detect multiple object classes.

This function requires that you have Deep Learning Toolbox. It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[trainedDetector,info] = trainFasterRCNNObjectDetector( ___ )` also returns information on the training progress, such as training loss and accuracy, for each iteration.

Resume Training a Detector

`trainedDetector = trainFasterRCNNObjectDetector(trainingData,checkpoint, options)` resumes training from a detector checkpoint.

Fine-Tune a Detector

`trainedDetector = trainFasterRCNNObjectDetector(trainingData,detector, options)` continues training a Faster R-CNN object detector with additional fine-tuning options. Use this syntax with additional training data or to perform more training iterations to improve detector accuracy.

Additional Properties

`trainedDetector = trainFasterRCNNObjectDetector( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments and any of the previous inputs.

Examples

Train Faster R-CNN Vehicle Detector

Load training data.

```
data = load('fasterRCNNVehicleTrainingData.mat');

trainingData = data.vehicleTrainingData;

trainingData.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
    trainingData.imageFilename);
```

Randomly shuffle data for training.

```
rng(0);
shuffledIdx = randperm(height(trainingData));
trainingData = trainingData(shuffledIdx,:);
```

Create an image datastore using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a box label datastore using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the datastores.

```
ds = combine(imds, blds);
```

Set up the network layers.

```
lgraph = layerGraph(data.detector.Network);
```

Configure training options.

```
 options = trainingOptions('sgdm', ...
      'MiniBatchSize', 1, ...
      'InitialLearnRate', 1e-3, ...
      'MaxEpochs', 7, ...
      'VerboseFrequency', 200, ...
      'CheckpointPath', tempdir);
```

Train detector. Training will take a few minutes. Adjust the NegativeOverlapRange and PositiveOverlapRange to ensure training samples tightly overlap with ground truth.

```
detector = trainFasterRCNNObjectDetector(ds, lgraph, options, ...
        'NegativeOverlapRange',[0 0.3], ...
        'PositiveOverlapRange',[0.6 1]);
```

```
*************************************************************************
Training a Faster R-CNN Object Detector for the following object classes:

* vehicle

Training on single GPU.
Initializing input data normalization.
```

```
|=========================================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  RPN Mini
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |     Accu
|=========================================================================================================
|       1 |           1 |       00:00:00 |       0.8771 |       97.30% |         0.83 |
|       1 |         200 |       00:01:15 |       0.5324 |      100.00% |         0.15 |
|       2 |         400 |       00:02:40 |       0.4732 |      100.00% |         0.15 |
|       3 |         600 |       00:04:03 |       0.4776 |       97.14% |         0.09 |
|       3 |         800 |       00:05:23 |       0.5269 |       97.44% |         0.18 |
|       4 |        1000 |       00:06:44 |       0.9749 |      100.00% |              |
|       5 |        1200 |       00:08:07 |       1.1952 |       97.62% |         0.13 |
|       5 |        1400 |       00:09:24 |       0.6577 |      100.00% |              |
|       6 |        1600 |       00:10:46 |       0.6951 |      100.00% |              |
|       7 |        1800 |       00:12:08 |       0.5341 |       96.08% |         0.09 |
|       7 |        2000 |       00:13:26 |       0.3333 |      100.00% |         0.12 |
|       7 |        2065 |       00:13:52 |       1.0564 |      100.00% |              |
|=========================================================================================================
Detector training complete.
*********************************************************************
```

Test the Faster R-CNN detector on a test image.

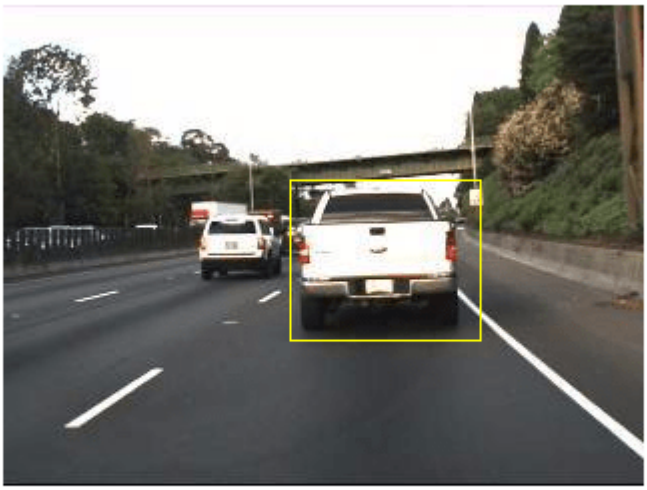
```
img = imread('highway.png');
```

Run the detector.

```
[bbox, score, label] = detect(detector,img);
```

Display detection results.

```
detectedImg = insertShape(img,'Rectangle',bbox);
figure
imshow(detectedImg)
```



Input Arguments

`trainingData` — Labeled ground truth

datastore | table

Labeled ground truth, specified as a datastore or a table.

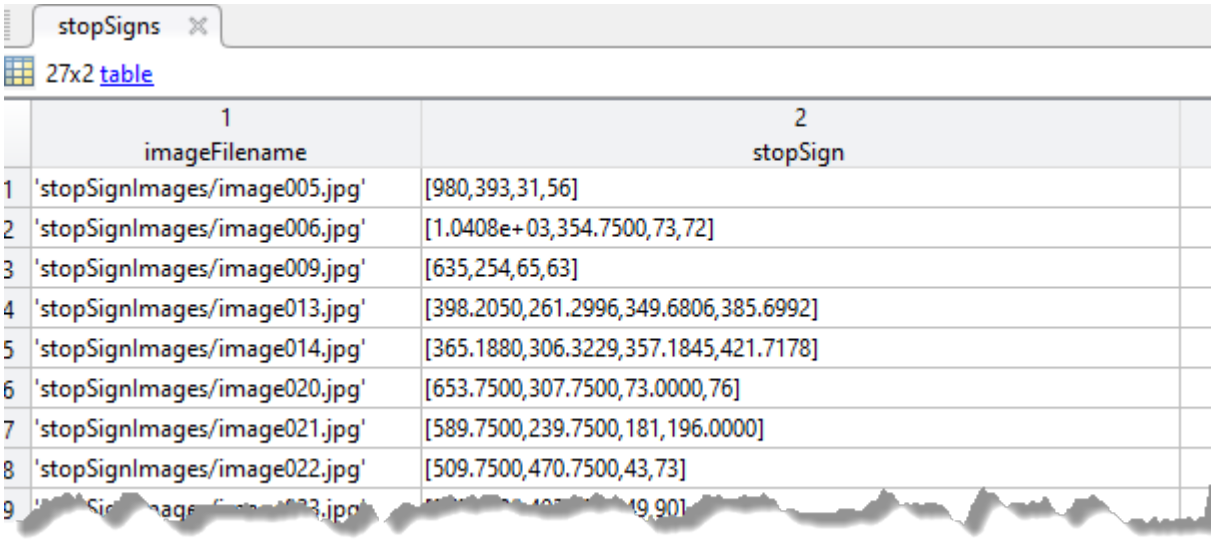
Each bounding box must be in the format [*x y width height*].

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns. The second column must contain the bounding boxes, and the third column must contain the labels. The first column can contain image or point cloud data.

data	***boxes***	***labels***
The first column must be images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	The third column must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

  For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.



`network` — Network

`SeriesNetwork` object | array of `Layer` objects | `LayerGraph` object | network name

Network, specified as a `SeriesNetwork`, an array of `Layer` objects, a `layerGraph` object, or by the network name. The network is trained to classify the object classes defined in the `trainingData` table. To use `SeriesNetwork`, `Layer`, and `layerGraph` objects, you must have Deep Learning Toolbox.

- When you specify the network as a `SeriesNetwork`, an array of `Layer` objects, or by the network name, the function transforms the network into a Faster R-CNN network by adding a region proposal network (RPN), an ROI max pooling layer, and new classification and regression layers to support object detection. Additionally, the `GridSize` property of the ROI max pooling layer is set to the output size of the last max pooling layer in the network.
- The array of `Layer` objects must contain a classification layer that supports the number of object classes, plus a background class. Use this input type to customize the learning rates of each layer. An example of an array of `Layer` objects follows:

```
layers = [imageInputLayer([28 28 3])
        convolution2dLayer([5 5],10)
        reluLayer()
        fullyConnectedLayer(10)
        softmaxLayer()
        classificationLayer()];
```

- When you specify the network as `SeriesNetwork` object, `Layer` array, or by the network name, the weights for additional convolution and fully connected layers are initialized to `'narrow-normal'`. The function adds these weights to create the network.
- The network name must be one of the following valid network names. You must also install the corresponding add-on.

Network Name	Feature Extraction Layer Name	ROI Pooling Layer OutputSize	Description
`alexnet`	`'relu5'`	[6 6]	Last max pooling layer is replaced by ROI max pooling layer
`vgg16`	`'relu5_3'`	[7 7]	
`vgg19`	`'relu5_4'`		
`squeezenet`	`'fire5-concat'`	[14 14]	ROI pooling layer is inserted after the feature extraction layer.
`resnet18`	`'res4b_relu'`		
`resnet50`	`'activation_40_relu'`		
`resnet101`	`'res4b22_relu'`		
`googlenet`	`'inception_4d-output'`		
`mobilenetv2`	`'block_13_expand_relu'`		
`inceptionv3`	`'mixed7'`	[17 17]	
`inceptionresnetv2`	`'block17_20_ac'`		

- The `LayerGraph` object must be a valid Faster R-CNN object detection network. You can use the `fasterRCNNLayers` function to create a `LayerGraph` object to train a custom Faster R-CNN network.

  Tip If your network is a `DAGNetwork`, use the `layerGraph` function to convert the network to a `LayerGraph` object. Then, create a custom Faster R-CNN network as described by the "Create Faster R-CNN Object Detection Network" example.

For more information on creating a Faster R-CNN network, see "Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN".

`options` — Training options
`trainingOptions` output

Training options, returned by the `trainingOptions` function (requires Deep Learning Toolbox). To specify solver and other options for network training, use `trainingOptions`.

Note `trainFasterRCNNObjectDetector` does not support these training options:

- Datastore inputs are not supported when you set the `DispatchInBackground` training option to `true`.

Additionally, the function does not support the following training options if you use a combined datastore input:

- `'once'` and `'every-epoch'` values for `'Shuffle'` argument
- `'parallel'` and `'multi-gpu'` values for `'ExecutionEnvironment'` argument

`checkpoint` — Saved detector checkpoint
`fasterRCNNObjectDetector` object

Saved detector checkpoint, specified as a `fasterRCNNObjectDetector` object. To periodically save a detector checkpoint during training, specify `CheckpointPath`. To control how frequently check points are saved see the `CheckPointFrequency` and `CheckPointFrequencyUnit` training options.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, if the `'CheckpointPath'` property of `options` is `'/tmp'`, load a checkpoint MAT file using:

```
data = load('/tmp/faster_rcnn_checkpoint__105__2016_11_18__14_25_08.mat');
```

The name of the MAT file includes the iteration number and a timestamp indicating when the detector checkpoint was saved. The detector is saved in the `detector` variable of the file. Pass this file back into the `trainFasterRCNNObjectDetector` function:

```
frcnn = trainFasterRCNNObjectDetector(stopSigns,...
                            data.detector,options);
```

`detector` — Previously trained Faster R-CNN object detector
`fasterRCNNObjectDetector` object

Previously trained Faster R-CNN object detector, specified as a `fasterRCNNObjectDetector` object. Use this syntax to continue training a detector with additional training data or to perform more training iterations to improve detector accuracy.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"PositiveOverlapRange",[0.75 1]`

`TrainingMethod` — Training method
`'end-to-end'` (default) | `'four-step'`

Training method, specified as `'end-to-end'` or `'four-step'`.

- `'end-to-end'` — Simultaneously train the region proposal and region classification subnetworks.
- `'four-step'` — Separately train the region proposal and region classification subnetworks in four steps.

`PositiveOverlapRange` — Bounding box overlap ratios for positive training samples
`[0.5 1]` (default) | 2-by-2 matrix | 4-by-2 matrix

Bounding box overlap ratios for positive training samples, specified as one of the following:

- A 2-element vector that specifies an identical overlap ratio for all four training stages.
- A 2-by-2 matrix, used only for the end-to-end training method. The first row of the matrix defines the overlap ratios for the region proposal subnetwork. The second row defines the overlap ratios for the region classification subnetwork.
- A 4-by-2 matrix, used only for the four-step training method. Each row of the matrix specifies the overlap ratio for each of the four training stages.

Values are in the range [0,1]. Region proposals that overlap with ground truth bounding boxes within the specified range are used as positive training samples.

The overlap ratio used for both the `PositiveOverlapRange` and `NegativeOverlapRange` is defined as:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

A and B are bounding boxes.

`NegativeOverlapRange` — Bounding box overlap ratios for negative training samples
`[0.1 0.5]` (default) | 2-element vector | 4-by-2 matrix

Bounding box overlap ratios for negative training samples, specified as one of the following.

- A 2-element vector that specifies the overlap ratio.

- A 2-by-2 matrix, used only for the end-to-end training method. The first row of the matrix defines the overlap ratios for the region proposal subnetwork. The second row defines the overlap ratios for the region classification subnetwork.
- A 4-by-2 matrix, used only for the four-step training method. Each row of the matrix specifies the overlap ratio for each of the four training stages.

Values are in the range [0,1]. Region proposals that overlap with the ground truth bounding boxes within the specified range are used as negative training samples.

The overlap ratio used for both the `PositiveOverlapRange` and `NegativeOverlapRange` is defined as:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

A and B are bounding boxes.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | positive integer | two-element vector

Maximum number of strongest region proposals to use for generating training samples, specified as a positive integer. Reduce this value to speed up processing time at the cost of training accuracy. To use all region proposals, set this value to `Inf`.

`NumRegionsToSample` — Number of region proposals
128 (default) | integer | 1-by-2 vector | 1-by-4 vector

Number of region proposals to randomly sample from each training image, specified as an integer, 1-by-2 vector, or a 1-by-4 vector. Use the 1-by-2 vector for end-to-end training. Use the 1-by-4 vector for the four-step training. Reduce the number of regions to sample to reduce memory usage and speed up training. Reducing the value can also decrease training accuracy.

When you set '`TrainingMethod`' to `'end-to-end'`, the number of region proposals can be set to a 1-by-2 vector. The first element of the vector must be the number of regions sampled for the region proposal subnetwork. The second element must be the number of regions sampled for the region classification subnetwork.

When you set `TrainingMethod` to `'four-step'`, the number of region proposals can be set to a 1-by-4 vector. The i^{th} element specifies the number of regions to sample for the i^{th} training step.

`SmallestImageDimension` — Length of smallest image dimension
`[]` (default) | positive integer

Length of the smallest image dimension, either width or height, specified as a positive integer. Training images are resized such that the length of the shortest dimension is equal to the specified integer. By default, training images are not resized. Resizing training images helps reduce computational costs and memory used when training images are large. Typical values range from 400–600 pixels.

Dependencies

- The `SmallestImageDimension` property supports only table input training data. To resize the input data of a datastore input, use the `transform` function.

`MinBoxSizes` — Minimum anchor box sizes

`'auto'` (default) | *m*-by-2 matrix

Minimum anchor box sizes for building the anchor box pyramid of the region proposal network (RPN), specified as an *m*-by-2 matrix. Each row defines the [*height width*] of an anchor box.

The default `'auto'` setting uses the minimum size and the median aspect ratio from the bounding boxes for each class in the ground truth data. To remove redundant box sizes, the function keeps boxes that have an intersection-over-union value that is less than or equal to 0.5. This behavior ensures that the minimum number of anchor boxes is used to cover all the object sizes and aspect ratios.

When anchor boxes are computed based on `MinBoxSizes`, the i^{th} anchor box size is:

```
round(MinBoxSizes(i,:) .* BoxPyramidScale ,^ (0:NumBoxPyramidLevels-1)')
```

Dependencies

- You cannot use this property if you specify the network as a `LayerGraph` object or if you resume training from a detector checkpoint.
- The `MinBoxSizes` property supports only input training in table format. To estimate anchor boxes for a datastore input, use the `estimateAnchorBoxes` function.

`BoxPyramidScale` — Anchor box pyramid scale

2 (default) | scalar

Anchor box pyramid scale factor used to successively upscale anchor box sizes, specified as a scalar. Recommended values are from 1 through 2. Increase this value for faster results. Decrease the number for greater accuracy.

Dependencies

- The `BoxPyramidScale` property supports only input training data in table format. To estimate anchor boxes for a datastore input, use the `estimateAnchorBoxes` function.

`NumBoxPyramidLevels` — Number of anchor box pyramid levels

`'auto'` (default) | scalar

Number of levels in an anchor box pyramid, specified as a scalar. Select a value that ensures that the multiscale anchor boxes are comparable in size to the size of objects in the ground truth data.

The default setting `'auto'` selects the number of levels based on the size of objects within the ground truth data. The number of levels is selected such that it covers the range of object sizes.

Dependencies

- The `NumBoxPyramidLevels` property supports only input training data in table format. To estimate anchor boxes for a datastore input, use the `estimateAnchorBoxes` function.

`FreezeBatchNormalization` — Frozen batch normalization

`true` (default) | `false`

Frozen batch normalization during training, specified as `true` or `false`. The value indicates whether to freeze the input layers to the network during training. Set this value to `true` if you are training with a small mini-batch size. Small batch sizes result in poor estimates of the batch mean and variance, which are required for effective batch normalization.

If you do not specify a value for `FreezeBatchNormalization`, the function sets the property to:

- `true` if the `MiniBatchSize` name-value argument for the `trainingOptions` function is less than 8.
- `false` if the `MiniBatchSize` name-value argument for the `trainingOptions` function is greater than or equal to 8.

You must specify a value for `FreezeBatchNormalization` to override this default behavior.

`ExperimentManager` — Detector training experiment monitoring
`'none'` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see "Train Object Detectors in Experiment Manager".

Information monitored during training:

- Training loss at each iteration.
- Training accuracy at each iteration.
- Training root mean square error (RMSE) for the box regression layer.
- Learning rate at each iteration.

Validation information when the training `options` input contains validation data:

- Validation loss at each iteration.
- Validation accuracy at each iteration.
- Validation RMSE at each iteration.

Output Arguments

`trainedDetector` — Trained Faster R-CNN object detector
`fasterRCNNObjectDetector` object

Trained Faster R-CNN object detector, returned as a `fasterRCNNObjectDetector` object.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with eight fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration is the mean squared error (MSE) calculated as the sum of localization error, confidence loss, and classification loss. For more information about the training loss function, see "YOLO v2 Training Loss" on page 3-1402.
- `TrainingAccuracy` — Training set accuracy at each iteration.
- `TrainingRMSE` — Training root mean squared error (RMSE) is the RMSE calculated from the training loss at each iteration.
- `BaseLearnRate` — Learning rate at each iteration.

- `ValidationLoss` — Validation loss at each iteration.
- `ValidationAccuracy` — Validation accuracy at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `FinalValidationLoss` — Final validation loss at end of the training.
- `FinalValidationRMSE` — Final validation RMSE at end of the training.

Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are assigned as `NaN`. The struct contains `ValidationLoss`, `ValidationAccuracy`, `ValidationRMSE`, `FinalValidationLoss`, and `FinalValidationRMSE` fields only when `options` specifies validation data.

Tips

- To accelerate data preprocessing for training, `trainFastRCNNObjectDetector` automatically creates and uses a parallel pool based on your parallel preference settings. For more details about setting these preferences, see parallel preference settings. Using parallel computing preferences requires Parallel Computing Toolbox.
- VGG-16, VGG-19, ResNet-101, and Inception-ResNet-v2 are large models. Training with large images can produce "out-of-memory" errors. To mitigate these errors, try one or more of these options:
  - Reduce the size of your images by using the '`SmallestImageDimension`' argument.
  - Decrease the value of the '`NumRegionsToSample`' name-value argument.
- This function supports transfer learning. When you input a `network` by name, such as `'resnet50'`, then the function automatically transforms the network into a valid Faster R-CNN network model based on the pretrained `resnet50` model. Alternatively, manually specify a custom Faster R-CNN network by using the `LayerGraph` extracted from a pretrained DAG network. For more details, see “Create Faster R-CNN Object Detection Network”.
- This table describes how to transform each named network into a Faster R-CNN network. The feature extraction layer name specifies the layer for processing by the ROI pooling layer. The ROI output size specifies the size of the feature maps output by the ROI pooling layer.

Network Name	Feature Extraction Layer Name	ROI Pooling Layer OutputSize	Description
`alexnet`	`'relu5'`	[6 6]	Last max pooling layer is replaced by ROI max pooling layer
`vgg16`	`'relu5_3'`	[7 7]	
`vgg19`	`'relu5_4'`		
`squeezenet`	`'fire5-concat'`	[14 14]	
`resnet18`	`'res4b_relu'`		ROI pooling layer is inserted after the feature extraction layer.
`resnet50`	`'activation_40_relu'`		
`resnet101`	`'res4b22_relu'`		
`googlenet`	`'inception_4d-output'`		

Network Name	Feature Extraction Layer Name	ROI Pooling Layer OutputSize	Description
`mobilenetv2`	`'block_13_expand_relu'`		
`inceptionv3`	`'mixed7'`	[17 17]	
`inceptionresnetv2`	`'block17_20_ac'`		

For information on modifying how a network is transformed into a Faster R-CNN network, see “Design an R-CNN, Fast R-CNN, and a Faster R-CNN Model”.

- During training, multiple image regions are processed from the training images The number of image regions per image is controlled by the `NumRegionsToSample` property. The `PositiveOverlapRange` and `NegativeOverlapRange` properties control which image regions are used for training. Positive training samples are those that overlap with the ground truth boxes by 0.6 to 1.0, as measured by the bounding box intersection-over-union metric (IoU). Negative training samples are those that overlap by 0 to 0.3. Choose values for these properties by testing the trained detector on a validation set.

Overlap Values	Description
`PositiveOverlapRange` set to `[0.6 1]`	Positive training samples are set equal to the samples that overlap with the ground truth boxes by 0.6 to 1.0, measured by the bounding box IoU metric.
`NegativeOverlapRange` set to `[0 0.3]`	Negative training samples are set equal to the samples that overlap with the ground truth boxes by 0 to 0.3.

  If you set `PositiveOverlapRange` to `[0.6 1]`, then the function sets the positive training samples equal to the samples that overlap with the ground truth boxes by 0.6 to 1.0, measured by the bounding box IoU metric. If you set `NegativeOverlapRange` to `[0 0.3]`, then the function sets the negative training samples equal to the samples that overlap with the ground truth boxes by 0 to 0.3.
- Use the `trainingOptions` function to enable or disable verbose printing.

Version History

Introduced in R2017a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These

object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

R2019b: Default Training Method
Behavior changed in R2019b

Starting in R2019b, by default, the `trainFasterRCNNObjectDetector` function uses the end-to-end method for training a detector.

In previous releases, the default training method used the "four-step" method. To preserve compatibility, set the `TrainingMethod` property to `'four-step'`.

References

[1] Ren, S., K. He, R. Girschick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *Advances in Neural Information Processing Systems*. Vol. 28, 2015.

[2] Girshick, R. "Fast R-CNN." *Proceedings of the IEEE International Conference on Computer Vision*, 1440-1448. Santiago, Chile: IEEE, 2015.

[3] Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580-587. Columbus, OH: IEEE, 2014.

[4] Zitnick, C. L., and P. Dollar. "Edge Boxes: Locating Object Proposals from Edges." *Computer Vision-ECCV 2014*, 391-405. Zurich, Switzerland: ECCV, 2014.

Extended Capabilities

Automatic Parallel Support
Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainingOptions` | `objectDetectorTrainingData` | `estimateAnchorBoxes` | `fasterRCNNLayers`

Objects
`maxPooling2dLayer` | `Layer` | `averagePooling2dLayer` | `fasterRCNNObjectDetector` | `boxLabelDatastore`

Topics
“Object Detection Using Faster R-CNN Deep Learning”

“Train Object Detectors in Experiment Manager”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainMaskRCNN

Train Mask R-CNN network to perform instance segmentation

Syntax

```
trainedDetector = trainMaskRCNN(trainingData,network,options)
trainedDetector = trainMaskRCNN(trainingData,network,options,Name=Value)
[trainedDetector,info] = trainMaskRCNN(trainingData,network,options)
```

Description

`trainedDetector = trainMaskRCNN(trainingData,network,options)` trains a Mask R-CNN network. A trained Mask R-CNN network object can perform instance segmentation to detect and segment multiple object classes. This syntax supports transfer learning on a pretrained Mask R-CNN network and training an uninitialized Mask R-CNN network.

This function requires that you have Deep Learning Toolbox. It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`trainedDetector = trainMaskRCNN(trainingData,network,options,Name=Value)` uses additional options specified by one or more name-value arguments.

`[trainedDetector,info] = trainMaskRCNN(trainingData,network,options)` also returns information on the training progress, such as training loss and accuracy, for each iteration.

Input Arguments

`trainingData` — Labeled ground truth

datastore

Labeled ground truth training data, specified as a datastore. Your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array with four columns. This table describes the format of each column.

data	*boxes*	*labels*	*masks*
RGB image that serves as a network input, specified as an *H*-by-*W*-by-3 numeric array.	Bounding boxes, specified as *M*-by-4 matrices, where *M* is the number of objects within the image. Each bounding box has the format [*x y width height*], where [*x*, *y*] represent the top-left coordinates of the bounding box.	Object class names, specified as an *M*-by-1 categorical vector. All categorical data returned by the datastore must contain the same categories.	Binary masks, specified as a logical array of size *H*-by-*W*-by-*M*. Each mask is the segmentation of one instance in the image.

You can create a datastore that returns data in the required format using these steps:

1. Create an `imageDatastore` that returns RGB image data
2. Create a `boxLabelDatastore` that returns bounding box data and instance labels as a two-element cell array
3. Create an `imageDatastore` and specify a custom read function that returns mask data as a binary matrix
4. Combine the three datastores using the `combine` function

For more information, see “Getting Started with Mask R-CNN for Instance Segmentation”.

`network` — Mask R-CNN network to train
`maskrcnn` object

Mask R-CNN network to train, specified as a `maskrcnn` object.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function. You must set the `ResetInputNormalization` property as `false`.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `trainedDetector = trainMaskRCNN(trainingData,network,options,NumRegionsToSample=64)` samples 64 region proposals from each training image

`PositiveOverlapRange` — Bounding box overlap ratios for positive training samples
`[0.5 1]` (default) | two-element numeric vector

Bounding box overlap ratios for positive training samples, specified as a two-element numeric vector with values in the range [0, 1]. Region proposals that overlap with ground truth bounding boxes within the specified range are used as positive training samples.

The overlap ratio for bounding boxes A and B is:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

`NegativeOverlapRange` — Bounding box overlap ratios for negative training samples
`[0.1 0.5]` (default) | two-element numeric vector

Bounding box overlap ratios for negative training samples, specified as a two-element numeric vector with values in the range [0, 1]. Region proposals that overlap with the ground truth bounding boxes within the specified range are used as negative training samples.

The overlap ratio for bounding boxes A and B is:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

`NumStrongestRegions` — Maximum number of strongest region proposals
`1000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals to use for generating training samples, specified as a positive integer. Reduce this value to speed up processing time at the cost of training accuracy. To use all region proposals, set this value to `Inf`.

`NumRegionsToSample` — Number of region proposals
`128` (default) | positive integer

Number of region proposals to randomly sample from each training image, specified as a positive integer. Reduce the number of regions to sample to reduce memory usage and speed up training. Reducing the value can also decrease training accuracy.

`FreezeSubNetwork` — Subnetworks to freeze
`"none"` (default) | `"backbone"` | `"rpn"` | `["backbone" "rpn"]`

Subnetworks to freeze during training, specified as one of these values:

- `"none"` — Do not freeze subnetworks
- `"backbone"` — Freeze the feature extraction subnetwork, including the layers following the ROI align layer
- `"rpn"` — Freeze the region proposal subnetwork
- `["backbone" "rpn"]` — Freeze both the feature extraction and the region proposal subnetworks

The weight of layers in frozen subnetworks does not change during training.

`ExperimentManager` — Training experiment monitor
`"none"` (default) | `experiments.Monitor` object

Training experiment monitor, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots.

Information monitored during training:

- Training loss at each iteration

- Training accuracy at each iteration
- Training root mean square error (RMSE) for the box regression layer
- Training loss for the mask segmentation branch
- Learning rate at each iteration

Validation information when the training `options` input contains validation data:

- Validation loss at each iteration
- Validation accuracy at each iteration
- Validation RMSE at each iteration
- Validation loss for the mask segmentation branch

Output Arguments

`trainedDetector` — Trained Mask R-CNN network
`maskrcnn` object

Trained Mask R-CNN network, returned as a `maskrcnn` object.

`info` — Training progress information
structure

Training progress information, returned as a structure. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The loss is the combination of the region proposal network (RPN), classification, regression and mask loss used to train the Mask R-CNN network.
- `TrainingRPNLoss` — Total RPN loss at the end of each iteration.
- `TrainingRMSE` — Training root mean squared error (RMSE) for the box regression layer at the end of each iteration.
- `TrainingMaskLoss` — Training cross-entropy loss for the mask segmentation branch at the end of each iteration.
- `LearnRate` — Learning rate at each iteration.
- `ValidationLoss` — Validation loss at each iteration.
- `ValidationRPNLoss` — Validation RPN loss at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `ValidationMaskLoss` — Validation cross-entropy loss for the mask segmentation branch at each iteration.

Each field is a numeric vector with one element per training iteration. Values that are not calculated at a specific iteration are assigned as `NaN`. The structure contains the `ValidationLoss`, `ValidationRPNLoss`, `ValidationRMSE`, and `ValidationMaskLoss` fields only when `options` specifies validation data.

Tips

- The `trainMaskRCNN` function has a high GPU memory requirement. It is recommended to train a Mask R-CNN network with at least 12 GB of available GPU memory.

- To reduce the training memory consumption, try reducing the `InputSize` property of the `network` argument or the `NumRegionsToSample` name-value argument.
- When you want to perform transfer learning on a data set with similar content to the COCO data set, freezing the feature extraction and region proposal subnetworks can help the network training converge faster.

Version History

Introduced in R2022a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2022b: Supports training options with plots

When you specify `options`, the `Plots` property of the `trainingOptions` object can now have a value other than `"none"`. Before, `trainMaskRCNN` supported only the value of `"none"`.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainMaskRCNN` function uses a GPU if one is available. You can specify the hardware that the `trainMaskRCNN` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

`maskrcnn` | `posemaskrcnn` | `estimateAnchorBoxes` | Experiment Manager

Topics
“Perform Instance Segmentation Using Mask R-CNN”
“Getting Started with Mask R-CNN for Instance Segmentation”

trainPoseMaskRCNN

Train Pose Mask R-CNN network to perform pose estimation

Syntax

```
net = trainPoseMaskRCNN(trainingData,network,trainingStage,options)
[net,info] = trainPoseMaskRCNN(trainingData,network,trainingStage,options)
[___] = trainPoseMaskRCNN(___,Name=Value)
```

Description

`net = trainPoseMaskRCNN(trainingData,network,trainingStage,options)` trains a Pose Mask R-CNN network to perform six-degrees-of-freedom (6-DoF) pose estimation for multiple object classes.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Pose Mask R-CNN 6-DoF Object Pose Estimation. To use this functionality in parallel, you must have a license for Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[net,info] = trainPoseMaskRCNN(trainingData,network,trainingStage,options)` also returns information on the training progress, such as training loss and accuracy, for each iteration.

`[___] = trainPoseMaskRCNN(___,Name=Value)` specifies options using name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `NumRegionsToSample=64` specifies for the `trainPoseMaskRCNN` function to sample 64 region proposals from each training image.

Input Arguments

`trainingData` — Labeled training data

datastore

Labeled training data, specified as a datastore. Your data must be set up so that reading the datastore using the `read` or `readall` function returns a cell array with seven columns. This table describes the format of each column.

Input Image Data	Depth Image Data	Bounding Boxes	Labels	Masks	6-DoF Object Poses	Camera Intrinsic Parameters
RGB image, stored as an *H*-by-*W*-by-3 numeric array, or grayscale image, stored as an *H*-by-*W* numeric matrix.	Depth image, stored as an *H*-by-*W* numeric matrix.	Bounding boxes, stored as an *M*-by-4 matrix. *M* is the number of objects detected in the image. Each row of the matrix specifies the bounding box of the corresponding object in the form [*x y width height*], where *x* and *y* are the position of the top-left corner of the bounding box.	Object class names, stored as an *M*-by-1 categorical vector. All categorical data returned by the datastore must contain the same categories.	Binary masks, stored as an *H*-by-*W*-by-*M*. Each page of the array specifies the binary mask of the corresponding detected object.	6-DoF object poses, stored as an *M*-by-1 vector of `rigidtform3d` objects.	Camera intrinsic parameters, stored as a scalar `cameraIntrinsics` object. All images in the training data set must have the same camera intrinsic parameters.

`network` — Pose Mask R-CNN pose estimation network to train
`posemaskrcnn` object

Pose Mask R-CNN pose estimation network to train, specified as a `posemaskrcnn` object.

`trainingStage` — Training stage
`"mask"` | `"pose-and-mask"` | `"pose-refinement"`

Training stage that specifies the losses to apply during training, specified as one of these options:

- `"mask"` — Applies classification, bounding box regression, and segmentation mask losses.
- `"pose-and-mask"` — Applies classification, bounding box regression, segmentation mask, rotation, and translation losses.
- `"pose-refinement"` — Applies classification, bounding box regression, segmentation mask, rotation, and translation losses. This option calculates rotation loss using a ShapeMatch loss which has been optimized for object symmetry [1]. When you specify `trainingStage` as `"pose-refinement"`, you must specify the `ReferencePointClouds` name-value argument. To improve performance during this stage, increase the value of the `RotationLossWeight` name-value argument.

You must train a Pose Mask R-CNN pose estimation network in two stages. First, train the network on only the instance segmentation task in the `"mask"` stage. Then, train the output network on the pose

estimation task in the `"pose-and-mask"` stage. Optionally, you can train the network in the third `"pose-refinement"` stage to refine 6-DoF rotation predictions.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `trainPoseMaskRCNN(trainingData,network,options,trainingMode,NumRegionsToSample=64)` samples 64 region proposals from each training image.

`PositiveOverlapRange` — Bounding box overlap ratios for positive training samples
`[0.5 1]` (default) | two-element numeric vector

Bounding box overlap ratios for positive training samples, specified as a two-element numeric vector with values in the range [0, 1]. The function compares region proposals to ground truth bounding boxes, and uses the proposals with overlap ratios within the specified range as positive training samples.

Given two bounding boxes, A and B, the overlap ratio is:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

`NegativeOverlapRange` — Bounding box overlap ratios for negative training samples
`[0.1 0.5]` (default) | two-element numeric vector

Bounding box overlap ratios for negative training samples, specified as a two-element numeric vector with values in the range [0, 1]. The function compares region proposals to ground truth bounding boxes, and uses the proposals with overlap ratios within the specified range as negative training samples.

Given two bounding boxes, A and B, the overlap ratio is:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

`NumStrongestRegions` — Maximum number of strongest region proposals
`1000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals to use for generating training samples, specified as a positive integer. Reduce this value to speed up processing time at the cost of training accuracy. To use all region proposals, set this value to `Inf`.

`NumRegionsToSample` — Number of region proposals
128 (default) | positive integer

Number of region proposals to randomly sample from each training image, specified as a positive integer. Specifying a smaller number of regions to sample can reduce memory usage and increase training speed, but can also decrease training accuracy.

`FreezeSubNetwork` — Subnetworks to freeze
`"none"` (default) | `"backbone"` | `"rpn"` | `["backbone" "rpn"]`

Subnetworks to freeze during training, specified as one of these values:

- `"none"` — Do not freeze subnetworks.
- `"backbone"` — Freeze the feature extraction subnetwork, including the layers following the ROI align layer.
- `"rpn"` — Freeze the region proposal subnetwork.
- `["backbone" "rpn"]` — Freeze both the feature extraction and the region proposal subnetworks.

The weight of layers in frozen subnetworks does not change during training.

`ExperimentMonitor` — Training experiment monitor
`"none"` (default) | `experiments.Monitor` object

Training experiment monitor, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots.

Information monitored during training:

- Training loss at each iteration
- Training accuracy at each iteration
- Training root mean square error (RMSE) for the box regression layer
- Training loss for the mask segmentation branch
- Training rotation loss at each iteration
- Training translation loss at each iteration
- Learning rate at each iteration

Validation information when the training `options` input argument contains validation data:

- Validation loss at each iteration
- Validation accuracy at each iteration
- Validation RMSE at each iteration
- Validation loss for the mask segmentation branch
- Validation rotation loss at each iteration
- Validation translation loss at each iteration

`TranslationLossWeight` — Translation loss weight
1 (default) | positive scalar

Translation loss weight, specified as a positive scalar. Adjust the `TranslationLossWeight` value so that the translation loss `TrainingTranslationLoss` after weighting is the same order of magnitude as the values of `TrainingRPNLoss`, `TrainingRMSE`, `TrainingClassLoss`, and `TrainingMaskLoss`.

`RotationLossWeight` — Rotation loss weight
1 (default) | positive scalar

Rotation loss weight, specified as a positive scalar. Adjust the `RotationLossWeight` value so that the rotation loss `TrainingRotationLoss` after weighting is the same order of magnitude as the values of `TrainingRPNLoss`, `TrainingRMSE`, `TrainingClassLoss`, and `TrainingMaskLoss`.

`BoxLossWeight` — Bounding box regression loss weight
1 (default) | positive scalar

Bounding box regression loss weight, specified as a positive scalar. In most cases, train the Pose Mask R-CNN network with the default value.

`ClassLossWeight` — Classification loss weight
1 (default) | positive scalar

Classification loss weight, specified as a positive scalar. In most cases, train the Pose Mask R-CNN network with the default value.

`MaskLossWeight` — Mask loss weight
1 (default) | positive scalar

Mask loss weight to use for instance segmentation, specified as a positive scalar. In most cases, train the Pose Mask R-CNN network with the default value.

`RPNLossWeight` — Region proposal network (RPN) training loss weight
1 (default) | positive scalar

Region proposal network (RPN) training loss weight, specified as a positive scalar. In most cases, train the Pose Mask R-CNN network with the default value.

`ReferencePointClouds` — Reference point clouds
`[]` (default) | *N*-by-1 array of `pointCloud` objects

Reference point clouds, specified as an *N*-by-1 array of `pointCloud` objects. *N* is the number of object classes. The order of the point clouds corresponds to the order of ground truth labels.

Output Arguments

`net` — Trained Pose Mask R-CNN pose estimation network
`posemaskrcnn` object

Trained Pose Mask R-CNN pose estimation network, returned as a `posemaskrcnn` object.

`info` — Training progress information
structure

Training progress information, returned as a structure. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The loss is the combination of the RPN, classification, regression, and mask segmentation losses used to train the Pose Mask R-CNN pose estimation network.
- `TrainingRPNLoss` — Total RPN loss at the end of each iteration.
- `TrainingRMSE` — Training RMSE for the box regression layer at the end of each iteration.
- `TrainingClassLoss` — Training cross-entropy loss for the classification layer at the end of each iteration.
- `TrainingMaskLoss` — Training cross-entropy loss for the mask segmentation branch at the end of each iteration.
- `TrainingRotationLoss` — Training rotation prediction branch loss at the end of each iteration.
- `TrainingTranslationLoss` — Training regression loss for the translation prediction branch at the end of each iteration.
- `LearnRate` — Learning rate at each iteration.
- `ValidationLoss` — Validation loss at each iteration.
- `ValidationRPNLoss` — Validation RPN loss at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `ValidationClassLoss` — Validation classification loss at each iteration.
- `ValidationMaskLoss` — Validation cross-entropy loss for the mask segmentation branch at each iteration.
- `ValidationRotationLoss` — Validation rotation loss at each iteration.
- `ValidationTranslationLoss` — Validation translation loss at each iteration.

Each field contains a numeric vector with one element per training iteration. If the function does not calculate a metric for a specific iteration, it assigns a value of `NaN` for that iteration. The structure contains the `ValidationLoss`, `ValidationRPNLoss`, `ValidationRMSE`, and `ValidationMaskLoss` fields only when `options` specifies validation data.

Tips

- The `trainPoseMaskRCNN` function has a high GPU memory requirement. It is recommended to train a Pose Mask R-CNN network with at least 12 GB of available memory.
- To reduce memory consumption during training, you can decrease the value of the `NumRegionsToSample` name-value argument to limit the number of proposals from the region proposal stage. Note that this also reduces accuracy and increases convergence time.

Version History

Introduced in R2024a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2025a: Train Post Mask R-CNN network in three stages to refine 6-DoF rotation predictions

Specify the `trainingMode` input argument as `"pose-refinement"` to use the ShapeMatch loss during an optional third training stage to refine 6-DoF rotation predictions. When you specify `trainingStage` as `"pose-refinement"`, you must also specify the `ReferencePointClouds` name-value argument.

References

[1] Xiang, Yu, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes.” In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, 2018. doi:10.15607/RSS.2018.XIV.019.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainPoseMaskRCNN` function uses a GPU if one is available. You can specify the hardware that the `trainPoseMaskRCNN` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

`posemaskrcnn` | `predictPose` | `trainingOptions` | `trainMaskRCNN` | `maskrcnn` | `boxLabelDatastore` | Experiment Manager

Topics

“Perform 6-DoF Pose Estimation for Bin Picking Using Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Getting Started with Mask R-CNN for Instance Segmentation”

trainReidentificationNetwork

Train re-identification (ReID) deep learning network

Syntax

```
trainedReID = trainReidentificationNetwork(trainingData,reID,options)
[trainedReID,info] = trainReidentificationNetwork( ___ )
[ ___ ] = trainReidentificationNetwork( ___ ,Name=Value)
```

Description

`trainedReID = trainReidentificationNetwork(trainingData,reID,options)` trains the specified ReID network `reID` to output appearance feature vectors. The `options` input specifies the network training parameters.

Note This functionality requires Deep Learning Toolbox.

`[trainedReID,info] = trainReidentificationNetwork( ___ )` returns information on training progress, such as training loss, for each iteration using the input arguments from the previous syntax.

`[ ___ ] = trainReidentificationNetwork( ___ ,Name=Value)` specifies options using one or more name-value arguments, in addition to any combination of arguments from previous syntaxes. For example, `FreezeBackbone=false` specifies not to freeze the backbone of the network during training.

Input Arguments

`trainingData` — Training data

`imageDatastore` object | datastore

Training data of RGB or grayscale images, specified as an `imageDatastore` object with a populated `Labels` property, or a datastore whose `read` function returns a 2-by-*B* cell array, where *B* is the number of images in the datastore. Each row of the cell array is of the form {*Image Class*}.

Image	Class
RGB image, stored as an *H*-by-*W*-by-3 numeric array, or grayscale image, stored as an *H*-by-*W* matrix.	String or a categorical cell vector which contains the object class name for the corresponding input image in *Image*. All categorical data returned by the datastore must have the same categories.

`reID` — Re-identification network

`reidentificationNetwork` object

Re-identification network, specified as a `reidentificationNetwork` object.

options — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:
`trainReidentificationNetwork(trainingData,reID,options,FreezeBackbone=false)` specifies not to freeze the backbone of the network during training.

LossFunction — Loss function
`"additive-margin-softmax"` (default) | `"cross-entropy"` | `"cosine-softmax"`

Loss function to use in the ReID network, specified as `"additive-margin-softmax"`, `"cross-entropy"`, or `"cosine-softmax"`. To train the ReID network using data that contains very distinct objects, with large variations in shape, color, or texture, use cross-entropy loss, `"cross-entropy"`. To train the ReID network using data that contains both subtle and distinct objects, use additive-margin-softmax loss, `"additive-margin-softmax"` or cosine-softmax loss, `"cosine-softmax"`.

FreezeBackbone — Freeze backbone of ReID network
`true` or `1` (default) | `false` or `0`

Freeze the backbone of the ReID network during training, specified as a numeric or logical `true` (`1`) or `false` (`0`). When `true`, the function freezes the ReID network up to the feature layer, as specified by the `FeatureLayer` property of the `reidentificationNetwork` object. When `false`, the classification layers of the ReID network remain unfrozen, and training speed is slower.

Tip If you adjust the input size of the backbone network, or use a backbone network that has not been pretrained, set `FreezeBackbone` to `false`.

ExperimentMonitor — ReID training experiment monitoring
`[]` (default) | `experiments.Monitor` object

ReID training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, or produce training plots. For more information on using this app, see the “Train Object Detectors in Experiment Manager” example.

The app monitors this information during training:

- Training loss at each iteration
- Learning rate at each iteration

- Validation loss at each iteration, if the `options` input contains validation data

`Margin` — Margin in loss function
`0.35` (default) | nonnegative scalar

Margin in the loss function, specified as a non-negative scalar in the range `[0, 1]`. To specify this argument, you must specify `LossFunction` as `"additive-margin-softmax"`.

`Scale` — Scale in loss function
`30` (default) | positive scalar

Scale in the loss function, specified as a positive scalar. To specify this argument, you must specify `LossFunction` as `"additive-margin-softmax"`.

Output Arguments

`trainedReID` — Trained ReID network
`reidentificationNetwork` object

Trained ReID network, returned as a `reidentificationNetwork` object. The trained network outputs feature vectors of the size specified by its `FeatureLength` property.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with these fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration
- `BaseLearnRate` — Learning rate at each iteration
- `ValidationLoss` — Validation loss at each iteration.
- `FinalValidationLoss` — Final validation loss at the end of the training

Each field contains a numeric vector with one element per training iteration. If the function does not calculate a metric for a specific iteration, it assigns a value of `NaN` for that iteration. The structure contains the `ValidationLoss` and `FinalValidationLoss` fields only when `options` specifies validation data.

Tips

- To improve ReID accuracy, increase the number of images you use to train the network. You can expand the training data set using data augmentation. For information on how to apply data augmentation for preprocessing, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).

Version History

Introduced in R2024a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

Apps
Video Labeler

Functions
`evaluateReidentificationNetwork` | `extractReidentificationFeatures` | `trainingOptions`

Objects
`reidentificationNetwork`

Topics
“Reidentify People Throughout a Video Sequence Using ReID Network”
“Automate Ground Truth Labeling for Object Tracking and Re-Identification”
“Convert Ground Truth Labeling Data for Object Re-Identification”
“Compare Classification Network Architectures Using Experiment” (Deep Learning Toolbox)
“Custom Training Loops” (Deep Learning Toolbox)
“Set Up Parameters and Train Convolutional Neural Network” (Deep Learning Toolbox)
“Datastores for Deep Learning” (Deep Learning Toolbox)

plot

Plot re-identification (ReID) metrics

Syntax

```
plot(metrics,type)
plot(metrics,type,Name=Value)
```

Description

`plot(metrics,type)` plots the ReID metrics `metrics`.

`plot(metrics,type,Name=Value)` specifies options using one or more name-value arguments. For example, `LineColor="r"` plots the specified metric type with a red line.

Input Arguments

`metrics` — ReID metrics

`reidentificationMetrics` object

ReID metrics, specified as a `reidentificationMetrics` object.

`type` — Type of metric to plot

`"cmc"` | `"cmc-per-object"` | `"precision-recall"`

Type of metric to plot, specified as one of these options:

- `"cmc"` — Cumulative matching characteristic (CMC) curve over the entire data set
- `"cmc-per-object"` — Cumulative matching characteristic (CMC) curve per object class
- `"precision-recall"` — Precision-recall curve

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `plot(metrics,"precision-recall",LineColor="r")` plots the precision-recall curve with a red line.

`ShowLegend` — Option to display legend

`true` or `1` | `false` or `0`

Option to display legend, specified as a numeric or logical `1` (`true`) or `0` (`false`). If you specify the plot type as `"cmc-per-object"` and there are 10 or fewer object classes, `ShowLegend` is `true` by default and the `plot` object function displays the legend.

`Parent` — Figure axes

`gca` (default) | `Axes` object

Figure axes, specified as an `Axes` object.

`LineColor` — Plot line color
`[0 0 0]` (default) | short or long color name | RGB triplet | three-column matrix of RGB triplets

Plot line color, specified as one or more 1-by-3 RGB triplets, or a long or short color name. If you specify the plot type as `"cmc-per-object"`, you can specify a different color for each line by using a matrix in which each row is an RGB triplet. To specify one color for all lines, specify `LineColor` as a color name or a single RGB triplet.

`LineWidth` — Plot line width
2 (default) | positive integer

Plot line width, in pixels, specified as a positive integer.

Version History

Introduced in R2024a

See Also

`evaluateReidentificationNetwork` | `reidentificationMetrics` | `reidentificationNetwork`

Topics
“Reidentify People Throughout a Video Sequence Using ReID Network”

trainSOLOV2

Train SOLOv2 network to perform instance segmentation

Syntax

```
trainedDetector = trainSOLOV2(trainingData,network,options)
[trainedDetector,info] = trainSOLOV2(trainingData,network,options)
[___] = trainSOLOV2(___,Name=Value)
```

Description

`trainedDetector = trainSOLOV2(trainingData,network,options)` trains a SOLOv2 network to perform instance segmentation. A trained SOLOv2 network object can perform instance segmentation to detect and segment multiple object classes. This syntax supports transfer learning on a pretrained SOLOv2 network, as well as training an uninitialized SOLOv2 network.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation. You can install the Computer Vision Toolbox Model for SOLOv2 Instance Segmentation from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[trainedDetector,info] = trainSOLOV2(trainingData,network,options)` also returns information on the training progress, such as the training loss for each iteration.

`[___] = trainSOLOV2(___,Name=Value)` specifies network training options using name-value arguments, in addition to any combination of output arguments from previous syntaxes. For example, `FreezeSubNetwork="none"` specifies not to freeze subnetworks during training.

Input Arguments

`trainingData` — Labeled ground truth training data

datastore

Labeled ground truth training data, specified as a datastore. You must set up your data so that calling the `read` and `readall` functions on the datastore returns a cell array with four columns. This table describes the format of each cell in each column.

Data	Boxes	Labels	Masks
RGB or grayscale image that serves as a network input, specified as an *H*-by-*W*-by-3 or *H*-by-*W*-by-1 numeric array, respectively.	Bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.	Object class names, specified as an *M*-by-1 categorical vector, where *M* is the number of object instances in the image. All categorical data read from the datastore must contain the same categories.	Binary masks, specified as a logical array of size *H*-by-*W*-by-*M*, where *M* is the number of boxes in the image. Each channel is a mask, and each mask is the segmentation of one object instance in the image.

You can create a datastore that returns data in the required format using these steps:

1. Create an `ImageDatastore` that returns RGB or grayscale image data.
2. Create a `boxLabelDatastore` that returns bounding box data and instance labels as a two-element cell array.
3. Create an `ImageDatastore` and specify a custom read function that returns mask data as a binary matrix.
4. Combine the three datastores using the `combine` function.

For more information, see “Get Started with SOLOv2 for Instance Segmentation”.

`network` — SOLOv2 instance segmentation network to train
`solov2` object

SOLOv2 instance segmentation network to train, specified as a `solov2` object.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `trainedDetector = trainSOLOV2(trainingData,network,options,FreezeSubNetwork="none")` specifies not to freeze subnetworks during training.

`FreezeSubNetwork` — Subnetworks to freeze
`"backbone"` (default) | `"none"` | `"backboneAndNeck"`

Subnetworks to freeze during training, specified as one of these values:

- `"none"` — Do not freeze subnetworks.
- `"backbone"` — Freeze the feature extraction subnetwork, including the layers following the region of interest (ROI) align layer.
- `"backboneAndNeck"` — Freeze the feature extraction subnetwork, as well as the path aggregation network used to mix backbone features at different scales.

The weight of layers in frozen subnetworks does not change during training.

`ExperimentMonitor` — Network training experiment monitoring
`"none"` (default) | `experiments.Monitor` object

Network training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used for training, and produce training plots. For more information on using this app, see the “Train Object Detectors in Experiment Manager” example.

The app monitors this information during training:

- Training loss at each iteration
- Learning rate at each iteration

When the `options` input contains validation data, the app also monitors validation loss at each iteration.

Output Arguments

`trainedDetector` — Trained SOLOv2 instance segmentation model
`solov2` object

Trained SOLOv2 instance segmentation model, returned as a `solov2` object.

`info` — Training progress information
structure

Training progress information, returned as a structure with these fields. Each field corresponds to a stage of training.

- `Epoch` — Epoch number.
- `Iteration` — Iteration number.
- `TimeElapsed` — Time elapsed since the start of training.
- `LearnRate` — Learning rate at each iteration.
- `TrainingLoss` — Training loss at each iteration. The loss is the combination of the region proposal network (RPN), classification, regression and mask loss used to train the SOLOv2 network.
- `OutputNetworkIteration` — Iteration number that corresponds to the returned trained network.

Each field contains a numeric vector with one element per training iteration. For information that the function does not calculate at a specific iteration, the value is `NaN`. If `solov2` specifies validation data, the `info` structure also contains a `ValidationLoss` field.

Tips

- For cases in which the network training does not converge, try specifying the `GradientThreshold` argument when calling the `trainingOptions` function.
- When you want to perform transfer learning on a data set with similar content to the COCO data set, freeze the backbone and neck of the network to help the network training converge faster.

Version History

Introduced in R2023b

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainSOLOV2` function uses a GPU if one is available. You can specify the hardware that the `trainSOLOV2` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

`solov2` | `segmentObjects` | `evaluateInstanceSegmentation` | Experiment Manager

Topics

“Perform Instance Segmentation Using SOLOv2”
“Get Started with SOLOv2 for Instance Segmentation”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainYOLOv2ObjectDetector

Train YOLO v2 object detector

Syntax

```
trainedDetector = trainYOLOv2ObjectDetector(trainingData,detector,options)
trainedDetector = trainYOLOv2ObjectDetector(trainingData,checkpoint,options)
[trainedDetector,info] = trainYOLOv2ObjectDetector( ___ )
 ___  = trainYOLOv2ObjectDetector( ___ ,Name=Value)
```

Description

`trainedDetector = trainYOLOv2ObjectDetector(trainingData,detector,options)` returns an object detector trained using the you only look once version 2 (YOLO v2) network specified by `detector`. The `options` argument specifies training parameters for the detection network.

You can use this syntax for training an untrained detector or for fine-tuning a pretrained detector.

`trainedDetector = trainYOLOv2ObjectDetector(trainingData,checkpoint,options)` resumes training from the saved detector checkpoint.

You can use this syntax to:

- Add more training data and continue the training.
- Improve training accuracy by increasing the maximum number of iterations.

`[trainedDetector,info] = trainYOLOv2ObjectDetector( ___ )` also returns information on the training progress, such as the training accuracy and learning rate for each iteration.

`___ = trainYOLOv2ObjectDetector( ___ ,Name=Value)` uses additional options specified by one or more name-value arguments and any of the previous inputs. For example, `ExperimentMonitor=[]` specifies not to track metrics with the Experiment Manager app.

Examples

Train YOLO v2 Network for Vehicle Detection

Load the training data for vehicle detection into the workspace.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Specify the directory in which training samples are stored. Add full path to the file names in training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Randomly shuffle data for training.

```
rng(0)
shuffledIdx = randperm(height(trainingData));
trainingData = trainingData(shuffledIdx,:);
```

Create an `imageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the datastores.

```
ds = combine(imds,blds);
```

Specify the class names using the label columns from the table.

```
classes = trainingData.Properties.VariableNames(2:end);
```

Specify anchor boxes.

```
anchorBoxes = [8 8; 32 48; 40 24; 72 48];
```

Load a preinitialized YOLO v2 object detection network.

```
load("yolov2VehicleDetectorNet.mat","net");
```

Create the YOLO v2 object detection network.

```
detector = yolov2ObjectDetector(net,classes,anchorBoxes)
```

```
detector =
  yolov2ObjectDetector with properties:

                 Network: [1×1 dlnetwork]
               InputSize: [128 128 3]
       TrainingImageSize: [128 128]
             AnchorBoxes: [4×2 double]
              ClassNames: vehicle
   ReorganizeLayerSource: ''
             LossFactors: [5 1 1 1]
               ModelName: ''
```

Configure the network training options.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.001, ...
    Verbose=true, ...
    MiniBatchSize=16, ...
    MaxEpochs=30, ...
    Shuffle="never", ...
    VerboseFrequency=30, ...
    CheckpointPath=tempdir);
```

Train the YOLO v2 network.

```
[trainedDetector,info] = trainYOLOv2ObjectDetector(ds,detector,options);
```

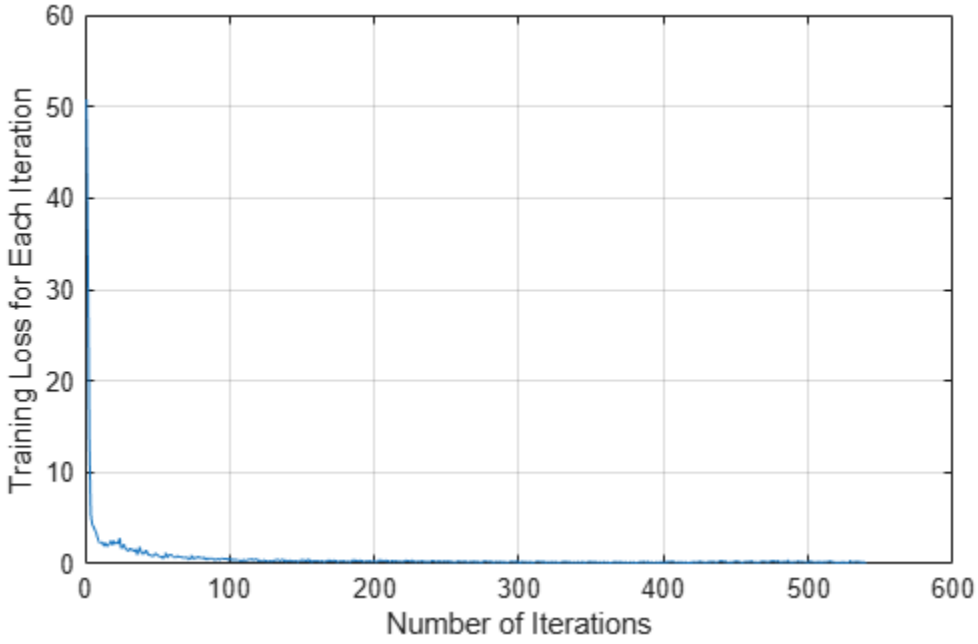
```
*************************************************************************
Training a YOLO v2 Object Detector for the following object classes:

* vehicle

Training on single CPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |         7.13 |         50.8 |          0.0010 |
|       2 |          30 |       00:00:16 |         1.18 |          1.4 |          0.0010 |
|       4 |          60 |       00:00:32 |         0.98 |          1.0 |          0.0010 |
|       5 |          90 |       00:00:48 |         0.59 |          0.3 |          0.0010 |
|       7 |         120 |       00:01:04 |         0.53 |          0.3 |          0.0010 |
|       9 |         150 |       00:01:19 |         0.63 |          0.4 |          0.0010 |
|      10 |         180 |       00:01:34 |         0.45 |          0.2 |          0.0010 |
|      12 |         210 |       00:01:49 |         0.39 |          0.2 |          0.0010 |
|      14 |         240 |       00:02:04 |         0.60 |          0.4 |          0.0010 |
|      15 |         270 |       00:02:18 |         0.42 |          0.2 |          0.0010 |
|      17 |         300 |       00:02:33 |         0.35 |          0.1 |          0.0010 |
|      19 |         330 |       00:02:47 |         0.47 |          0.2 |          0.0010 |
|      20 |         360 |       00:03:02 |         0.36 |          0.1 |          0.0010 |
|      22 |         390 |       00:03:17 |         0.34 |          0.1 |          0.0010 |
|      24 |         420 |       00:03:32 |         0.44 |          0.2 |          0.0010 |
|      25 |         450 |       00:03:47 |         0.54 |          0.3 |          0.0010 |
|      27 |         480 |       00:04:02 |         0.39 |          0.2 |          0.0010 |
|      29 |         510 |       00:04:17 |         0.44 |          0.2 |          0.0010 |
|      30 |         540 |       00:04:32 |         0.37 |          0.1 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.
Detector training complete.
*************************************************************************
```

Verify the training accuracy by inspecting the training loss for each iteration.

```
figure
plot(info.TrainingLoss)
grid on
xlabel("Number of Iterations")
ylabel("Training Loss for Each Iteration")
```



Read a test image into the workspace.

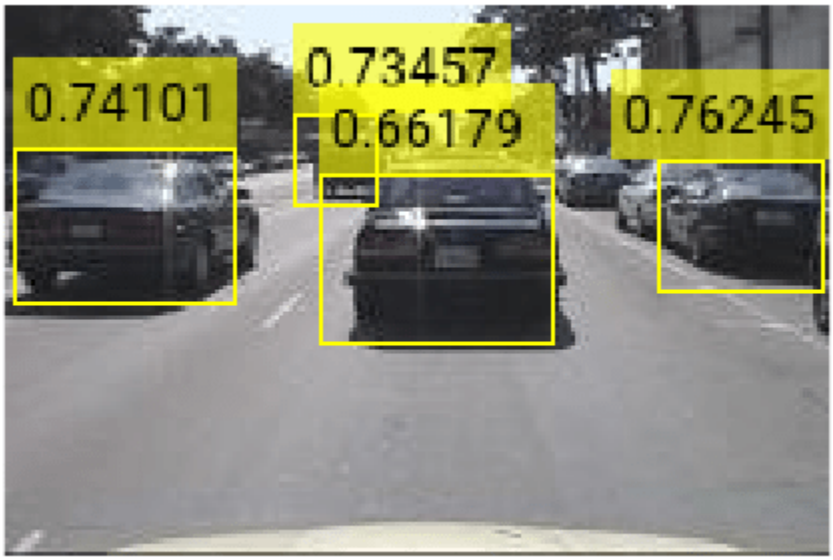
```
img = imread("detectcars.png");
```

Run the trained YOLO v2 object detector on the test image for vehicle detection.

```
[bboxes,scores] = detect(trainedDetector,img);
```

Display the detection results.

```
if(~isempty(bboxes))
    img = insertObjectAnnotation(img,"rectangle",bboxes,scores);
end
figure
imshow(img)
```



Input Arguments

`trainingData` — Labeled ground truth images

datastore | table

Labeled ground truth images, specified as a datastore or a table.

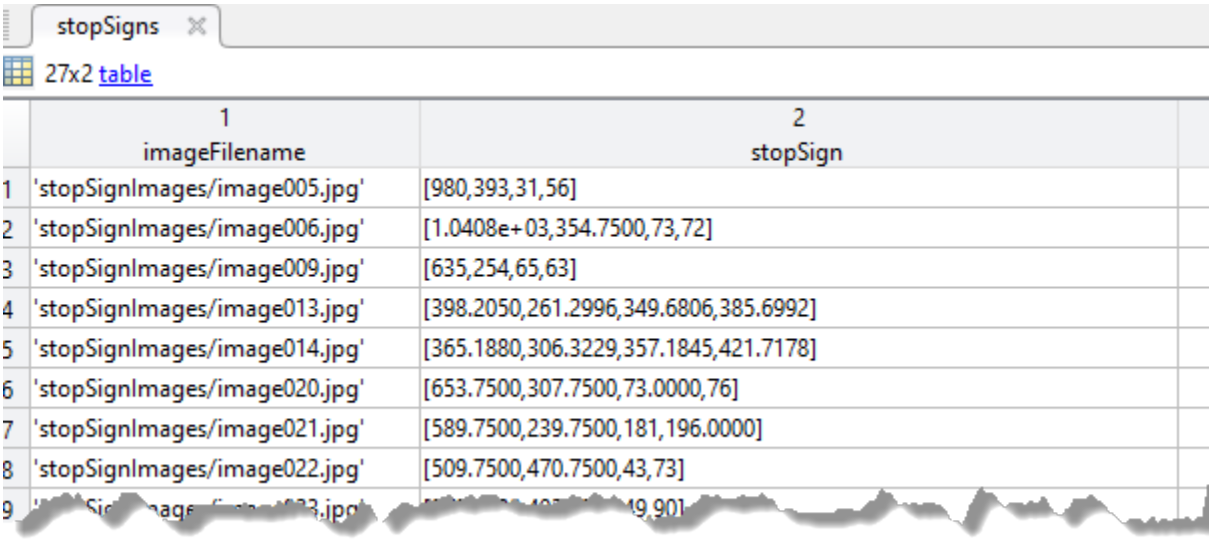
- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns. The second column must contain the bounding boxes, and the third column must contain the labels. The first column can contain image or point cloud data.

data	***boxes***	***labels***
The first column must be images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	The third column must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

  For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box

in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.


stopSigns

27x2 table

	1 imageFilename	2 stopSign
1	'stopSignImages/image005.jpg'	[980,393,31,56]
2	'stopSignImages/image006.jpg'	[1.0408e+03,354.7500,73,72]
3	'stopSignImages/image009.jpg'	[635,254,65,63]
4	'stopSignImages/image013.jpg'	[398.2050,261.2996,349.6806,385.6992]
5	'stopSignImages/image014.jpg'	[365.1880,306.3229,357.1845,421.7178]
6	'stopSignImages/image020.jpg'	[653.7500,307.7500,73.0000,76]
7	'stopSignImages/image021.jpg'	[589.7500,239.7500,181,196.0000]
8	'stopSignImages/image022.jpg'	[509.7500,470.7500,43,73]
9		

Note When the training data is specified using a table, the `trainYOLOv2ObjectDetector` function checks these conditions

- The bounding box values must be integers. Otherwise, the function automatically rounds each noninteger values to its nearest integer.
- The bounding box must not be empty and must be within the image region. While training the network, the function ignores empty bounding boxes and bounding boxes that lie partially or fully outside the image region.

`detector` — Pretrained or untrained YOLO v2 object detector
`yolov2ObjectDetector` object

Pretrained or untrained YOLO v2 object detector, specified as a `yolov2ObjectDetector` object. If `detector` is a pretrained detector, then you can continue training the detector with additional training data or perform more training iterations to improve detector accuracy.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Note The `trainYOLOv2ObjectDetector` function does not support these training options:

- The `trainingOptions` `Shuffle` values, `"once"` and `"every-epoch"` are not supported when you use a datastore input.

- Datastore inputs are not supported when you set the `DispatchInBackground` training option to `true`.

`checkpoint` — Saved detector checkpoint
`yolov2ObjectDetector` object

Saved detector checkpoint, specified as a `yolov2ObjectDetector` object. To periodically save a detector checkpoint during training, specify `CheckpointPath`. To control how frequently check points are saved see the `CheckPointFrequency` and `CheckPointFrequencyUnit` training options.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, if the `CheckpointPath` property of the object specified by `options` is `"/checkpath"`, you can load a checkpoint MAT file by using this code.

```
data = load("/checkpath/yolov2_checkpoint__216__2018_11_16__13_34_30.mat");
checkpoint = data.detector;
```

The name of the MAT file includes the iteration number and timestamp of when the detector checkpoint was saved. The detector is saved in the `detector` variable of the file. Pass this file back into the `trainYOLOv2ObjectDetector` function:

```
yoloDetector = trainYOLOv2ObjectDetector(trainingData,checkpoint,options);
```

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ExperimentManager="none"` specifies not to monitor the detector training.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"ExperimentManager","none"`

`ExperimentManager` — Detector training experiment monitoring
`"none"` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see “Train Object Detectors in Experiment Manager”.

Information monitored during training:

- Training loss at each iteration.
- Training accuracy at each iteration.
- Training root mean square error (RMSE) for the box regression layer.
- Learning rate at each iteration.

Validation information when the training `options` input contains validation data:

- Validation loss at each iteration.
- Validation accuracy at each iteration.
- Validation RMSE at each iteration.

Output Arguments

`trainedDetector` — Trained YOLO v2 object detector

`yolov2ObjectDetector` object

Trained YOLO v2 object detector, returned as `yolov2ObjectDetector` object.

`info` — Training progress information

structure array

Training progress information, returned as a structure array with seven fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration is the mean squared error (MSE) calculated as the sum of localization error, confidence loss, and classification loss. For more information about the training loss function, see YOLO v2 Training Loss on page 3-1402.
- `TrainingRMSE` — Training root mean squared error (RMSE) is the RMSE calculated from the training loss at each iteration.
- `BaseLearnRate` — Learning rate at each iteration.
- `ValidationLoss` — Validation loss at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `FinalValidationLoss` — Final validation loss at end of the training.
- `FinalValidationRMSE` — Final validation RMSE at end of the training.

Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are assigned as `NaN`. The struct contains `ValidationLoss`, `ValidationAccuracy`, `ValidationRMSE`, `FinalValidationLoss`, and `FinalValidationRMSE` fields only when `options` specifies validation data.

More About

Data Preprocessing

By default, the `trainYOLOv2ObjectDetector` function preprocesses the training images by:

- Resizing the input images to match the input size of the network.
- Normalizing the pixel values of the input images to lie in the range [0, 1].

When you specify the training data by using a table, the `trainYOLOv2ObjectDetector` function also augments the input dataset by:

- Reflecting the training data horizontally. The probability for horizontally flipping each image in the training data is 0.5.
- Uniformly scaling (resizing) the training data by a scale factor that is randomly picked from a continuous uniform distribution in the range [1, 1.1].

- Random color jittering for brightness, hue, saturation, and contrast.

When you specify the training data by using a datastore, the `trainYOLOv2ObjectDetector` function does not perform data augmentation. Instead you can augment the training data in datastore by using the `transform` function and then, train the network with the augmented training data. For more information on how to apply augmentation while using datastores, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).

YOLO v2 Training Loss

During training, the `trainYOLOv2ObjectDetector` function predicts refined bounding box locations by optimizing the mean squared error (MSE) loss between predicted bounding boxes and the ground truth. The loss function is defined as

$$
\begin{aligned}
&K_1 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&\quad + K_1 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \\
&\qquad + K_2 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \\
&\qquad\quad + K_3 \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 \\
&\qquad\qquad + K_4 \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2
\end{aligned}
$$

where:

- S is the number of grid cells.
- B is the number of bounding boxes in each grid cell.
- 1_{ij}^{obj} is 1 if the *jth* bounding box in grid cell i is responsible for detecting the object. Otherwise it is set to 0. A grid cell i is responsible for detecting the object, if the overlap between the ground truth and a bounding box in that grid cell is greater than or equal to 0.6.
- 1_{ij}^{noobj} is 1 if the *jth* bounding box in grid cell i does not contain any object. Otherwise it is set to 0.
- 1_i^{obj} is 1 if an object is detected in grid cell i. Otherwise it is set to 0.
- K_1, K_2, K_3, and K_4 are the weights. To adjust the weights, modify the `LossFactors on page 2-0` property of the `detector`.

The loss function can be split into three parts:

- **Localization loss**

  The first and second terms in the loss function comprise the localization loss. It measures error between the predicted bounding box and the ground truth. The parameters for computing the localization loss include the position, size of the predicted bounding box, and the ground truth. The parameters are defined as follows.

- (x_i, y_i), is the center of the *jth* bounding box relative to grid cell i.
- $(\widehat{x}_i, \widehat{y}_i)$, is the center of the ground truth relative to grid cell i.
- w_i and h_i is the width and the height of the *jth* bounding box in grid cell i, respectively. The size of the predicted bounding box is specified relative to the input image size.
- $\widehat{w}_i$ and $\widehat{h}_i$ is the width and the height of the ground truth in grid cell i, respectively.
- K_1 is the weight for localization loss. Increase this value to increase the weightage for bounding box prediction errors.

- **Confidence loss**

  The third and fourth terms in the loss function comprise the confidence loss. The third term measures the objectness (confidence score) error when an object is detected in the *jth* bounding box of grid cell i. The fourth term measures the objectness error when no object is detected in the *jth* bounding box of grid cell i. The parameters for computing the confidence loss are defined as follows.

  - C_i is the confidence score of the *jth* bounding box in grid cell i.
  - $\hat{C}_i$ is the confidence score of the ground truth in grid cell i.
  - K_2 is the weight for objectness error, when an object is detected in the predicted bounding box. You can adjust the value of K_2 to weigh confidence scores from grid cells that contain objects.
  - K_3 is the weight for objectness error, when an object is not detected in the predicted bounding box. You can adjust the value of K_3 to weigh confidence scores from grid cells that do not contain objects.

  The confidence loss can cause the training to diverge when the number of grid cells that do not contain objects is more than the number of grid cells that contain objects. To remedy this, increase the value for K_2 and decrease the value for K_3.

- **Classification loss**

  The fifth term in the loss function comprises the classification loss. For example, suppose that an object is detected in the predicted bounding box contained in grid cell i. Then, the classification loss measures the squared error between the class conditional probabilities for each class in grid cell i. The parameters for computing the classification loss are defined as follows.

  - $p_i(c)$ is the estimated conditional class probability for object class c in grid cell i.
  - $\widehat{p}_i(c)$ is the actual conditional class probability for object class c in grid cell i.
  - K_4 is the weight for classification error when an object is detected in the grid cell. Increase this value to increase the weightage for classification loss.

Tips

- To generate the ground truth, use the Image Labeler or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.
- To improve prediction accuracy:
  - Increase the number of images you can use to train the network. You can expand the training dataset through data augmentation. For information on how to apply data augmentation for preprocessing, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).

- Perform multiscale training by specifying an input detector whose `TrainingImageSize on page 2-0` property is a matrix with two or more rows. For each training epoch, the `trainYOLOv2ObjectDetector` function randomly resizes the input training images to one of the specified training image sizes.
- Choose anchor boxes appropriate to the dataset for training the network. You can use the `estimateAnchorBoxes` function to compute anchor boxes directly from the training data.

Version History

Introduced in R2019a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024b: Modified training process
Behavior change in future release

The `trainYOLOv2ObjectDetector` supports a new process to train a `yolov2ObjectDetector` object. The function now uses:

- Class names specified by the `ClassNames on page 2-0` property of the `yolov2ObjectDetector` object. The function no longer uses the class names from the `trainingData` table.
- Anchor boxes specified by the `AnchorBoxes on page 2-0` property of the `yolov2ObjectDetector` object.
- MSE loss weights specified by the `LossFactors on page 2-0` property of the `yolov2ObjectDetector` object.

The `TrainingImageSize` name-value argument is no longer recommended. Instead, specify training image sizes for multiscale training by using the `TrainingImageSize on page 2-0` name-value argument of the `yolov2ObjectDetector` object.

R2024b: `DAGNetwork` objects are not recommended
Not recommended starting in R2024b

Starting in R2024b, `DAGNetwork` objects are not recommended. Instead, specify the network architecture using a `yolov2ObjectDetector` object.

There are no plans to remove support for `DAGNetwork` objects.

References

[1] Joseph. R, S. K. Divvala, R. B. Girshick, and F. Ali. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788. Las Vegas, NV: CVPR, 2016.

[2] Joseph. R and F. Ali. "YOLO 9000: Better, Faster, Stronger." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525. Honolulu, HI: CVPR, 2017.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainingOptions` | `objectDetectorTrainingData` | `trainYOLOv4ObjectDetector`

Objects
`yolov2ObjectDetector` | `yolov4ObjectDetector` | `boxLabelDatastore`

Topics
“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using YOLO v2”
“Train Object Detectors in Experiment Manager”
“Getting Started with YOLO v2”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainYOLOv3ObjectDetector

Train YOLO v3 object detector

Syntax

```
detector = trainYOLOv3ObjectDetector(trainingData,detector,options)
detector = trainYOLOv3ObjectDetector(trainingData,checkpoint,options)
[detector,info] = trainYOLOv3ObjectDetector( ___ )
[ ___ ] = trainYOLOv3ObjectDetector( ___ ,Name=Value)
```

Description

`detector = trainYOLOv3ObjectDetector(trainingData,detector,options)` returns an object detector trained using the you only look once version 3 (YOLO v3) network specified by `detector`. The input `detector` can be an untrained or pretrained YOLO v3 object detector. The `options` input specifies training parameters for the detection network.

You can also use this syntax to fine-tune a pretrained YOLO v3 object detector.

`detector = trainYOLOv3ObjectDetector(trainingData,checkpoint,options)` resumes training from the saved detector checkpoint.

You can use this syntax to:

- Add more training data and continue training.
- Improve training accuracy by increasing the maximum number of iterations.

`[detector,info] = trainYOLOv3ObjectDetector( ___ )` also returns information on training progress, using any combination of input arguments from previous syntaxes, such as the training loss and learning rate for each iteration.

`[ ___ ] = trainYOLOv3ObjectDetector( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ExperimentMonitor=[]` specifies not to track metrics with Experiment Manager.

Note This functionality requires Deep Learning Toolbox. To use the pretrained YOLO v3 deep learning networks trained on COCO dataset, you must install the Computer Vision Toolbox Model for YOLO v3 Object Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Train YOLO v3 Network for Vehicle Detection

Train a pretrained YOLO v3 object detector to detect vehicles in an image. The object detector uses a tiny YOLO v3 network, trained on the COCO data set as the base network.

Load a `.mat` file containing training data, and extract the training data into the workspace. The training data is a table in which the first column contains the training images and the remaining columns contain the corresponding labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Specify the directory that contains the training samples. Add the full paths to the filenames in the training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an image datastore using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a box label datastore using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the image and box label datastores.

```
ds = combine(imds,blds);
```

Specify the anchor boxes to use for training the network. You must assign at least one anchor box to each output layer of the network. Because the tiny YOLO v3 network contains two output layers, specify the anchor boxes as a two-element cell array.

```
anchorBoxes = {[91 115; 82 106; 58 83];[41 58; 24 32;19 23]};
```

Specify the names of the classes to detect.

```
classes = {'vehicle'};
```

Create a pretrained YOLO v3 deep learning network configured to retrain on the new dataset by using the `yolov3ObjectDetector` function.

```
detector = yolov3ObjectDetector("tiny-yolov3-coco",classes,anchorBoxes);
detector.Network
```

```
ans =
  dlnetwork with properties:

         Layers: [44×1 nnet.cnn.layer.Layer]
    Connections: [45×2 table]
     Learnables: [48×3 table]
          State: [22×3 table]
     InputNames: {'input'}
    OutputNames: {'convOut1'  'convOut2'}
    Initialized: 1

  View summary with summary.
```

Specify the training options.

```
options = trainingOptions("adam", ...
    InitialLearnRate=0.001, ...
```

```
    MiniBatchSize=8, ...
    MaxEpochs=30, ...
    BatchNormalizationStatistics="moving", ...
    ResetInputNormalization=false, ...
    VerboseFrequency=30);
```

Retrain the pretrained YOLO v3 network on the new data set by using the `trainYOLOv3ObjectDetector` function.

```
trainedDetector = trainYOLOv3ObjectDetector(ds,detector,options);
```

```
*************************************************************************
Training a YOLO v3 Object Detector for the following object classes:

* vehicle


    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss
    _____    _________    ___________    _________    ____________

      1          30         00:00:39        0.001         27.477
      2          60         00:01:21        0.001         12.281
      3          90         00:01:59        0.001         7.3423
      4         120         00:02:36        0.001         5.5316
      5         150         00:03:13        0.001         4.6442
      5         180         00:03:50        0.001         3.532
      6         210         00:04:27        0.001          5.46
      7         240         00:05:04        0.001         3.7652
      8         270         00:05:41        0.001         2.0736
      9         300         00:06:18        0.001         2.8613
      9         330         00:06:54        0.001         2.1912
     10         360         00:07:31        0.001         1.7878
     11         390         00:08:08        0.001         1.6713
     12         420         00:08:44        0.001         1.0196
     13         450         00:09:20        0.001         1.4047
     13         480         00:09:57        0.001         1.2047
     14         510         00:10:33        0.001         1.0924
     15         540         00:11:10        0.001         1.0899
     16         570         00:11:47        0.001        0.80751
     17         600         00:12:23        0.001        0.77776
     18         630         00:12:59        0.001        0.92279
     18         660         00:13:36        0.001         0.6374
     19         690         00:14:12        0.001        0.64517
     20         720         00:14:49        0.001        0.52645
     21         750         00:15:26        0.001        0.61458
     22         780         00:16:03        0.001        0.65221
     22         810         00:16:39        0.001        0.69775
     23         840         00:17:16        0.001        0.51352
     24         870         00:17:52        0.001        0.82176
     25         900         00:18:29        0.001        0.86924
     26         930         00:19:05        0.001         0.887
     26         960         00:19:42        0.001        0.57564
     27         990         00:20:18        0.001        0.40471
     28        1020         00:20:55        0.001        0.37672
     29        1050         00:21:31        0.001        0.28895
     30        1080         00:22:07        0.001        0.54886
     30        1110         00:22:44        0.001         0.4914

*************************************************************************
```

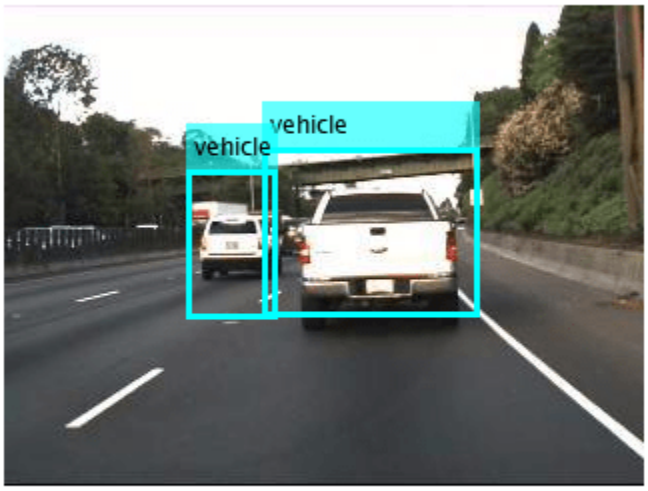
```
Detector training complete.
*********************************************************************
```

Read a test image. Use the trained YOLO v3 object detector to detect vehicles and display the detection results.

```
I = imread('highway.png');
[bboxes, scores, labels] = detect(trainedDetector,I,Threshold=0.05);
results = table(bboxes,labels,scores)
```

```
results=2×3 table
    131    73    107    83    vehicle    0.1546
     93    84     44    73    vehicle    0.1259
```

```
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels,LineWidth=3,AnnotationColor="cya
figure
imshow(detectedImg)
```



Fine-Tune YOLO v3 Network for Multiclass Object Detection

Perform transfer learning to fine-tune a pretrained YOLO v3 object detector by freezing its backbone. The detector has been trained for detecting stop signs and the front views and rear views of cars in an image. The detector uses a DarkNet-53 network, trained on the COCO data set, as the base network.

Load a `.mat` file containing training data, and extract the training data into the workspace. The training data is a table in which the first column contains the training images and the remaining columns contain the corresponding labeled bounding boxes.

```
data = load("stopSignsAndCars.mat");
trainingData = data.stopSignsAndCars;
```

Specify the directory that contains the training samples. Add the full paths to the filenames in the training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an image datastore using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a box label datastore using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the image and box label datastores.

```
ds = combine(imds,blds);
```

Specify the input size to use for resizing the training images. You must also resize the bounding boxes based on the specified input size.

```
inputSize = [224 224 3];
```

Resize and rescale the training images and bounding boxes by using the `preprocessData` helper function. Convert the preprocessed data to a `datastore` object by using the `transform` function.

```
trainingDataForEstimation = transform(ds,@(data)preprocessData(data,inputSize));
```

Specify the anchor boxes to use for training the network. You must assign at least one anchor box to each output layer of the network. Because the `darknet53-coco` YOLO v3 network contains three output layers, specify the anchor boxes as a three-element cell array.

```
numAnchors = 9;
[anchors,meanIoU] = estimateAnchorBoxes(trainingDataForEstimation,numAnchors);
```

Use larger anchor boxes at lower scale and smaller anchor boxes at higher scale output layers. To do so, sort `anchors` by area, in descending order, and assign the first three to the first output layer, the next three to the second output layer, and the last three to the third output layer of the network.

```
area = anchors(:,1).*anchors(:,2);
[~,idx] = sort(area,"descend");
anchors = anchors(idx,:);
anchorBoxes = {anchors(1:3,:); anchors(4:6,:); anchors(7:9,:)};
```

Specify the names of the classes to detect.

```
classes = {'stopSign','carRear','carFront'};
```

Create a pretrained YOLO v3 deep learning network configured to retrain on the new dataset by using the `yolov3ObjectDetector` function.

```
detector = yolov3ObjectDetector("darknet53-coco",classes,anchorBoxes,InputSize=inputSize);
detector.Network
```

```
ans =
  dlnetwork with properties:

         Layers: [247×1 nnet.cnn.layer.Layer]
    Connections: [273×2 table]
```

```
     Learnables: [294×3 table]
          State: [144×3 table]
     InputNames: {'input'}
    OutputNames: {'convOut1'  'convOut2'  'convOut3'}
    Initialized: 1

  View summary with summary.
```

Specify the training options.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.0001, ...
    MiniBatchSize=4, ...
    MaxEpochs=30, ...
    BatchNormalizationStatistics="moving", ...
    ResetInputNormalization=false, ...
    VerboseFrequency=30);
```

Retrain the pretrained YOLO v3 network on the new data set by using the `trainYOLOv3ObjectDetector` function. To freeze the backbone of the pretrained YOLO v3 network during training, specify the `FreezeSubNetwork` name-value argument to `"backbone"`. The weights of the frozen backbone layers do not change during training, which increases training speed and reduces GPU memory consumption.

```
trainedDetector = trainYOLOv3ObjectDetector(ds,detector,options,FreezeSubNetwork="backbone");
```

```
*************************************************************************
Training a YOLO v3 Object Detector for the following object classes:

* stopSign
* carRear
* carFront


    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss
    _____    _________    ___________    _________    ____________
      3         30         00:00:30       0.0001         35.9
      6         60         00:00:58       0.0001        4.8118
      9         90         00:01:26       0.0001        2.5195
     11         120        00:01:54       0.0001        2.5046
     14         150        00:02:22       0.0001        2.4547
     17         180        00:02:50       0.0001        2.8295
     20         210        00:03:19       0.0001         4.031
     22         240        00:03:47       0.0001       0.76425
     25         270        00:04:16       0.0001       0.86645
     28         300        00:04:44       0.0001        1.7476
     30         330        00:05:13       0.0001        0.7068

*************************************************************************
Detector training complete.
*************************************************************************
```

Read a test image. Use the fine-tuned YOLO v3 object detector to detect the stop signs and the front views and rear views of cars, and display the detection results.

```
I = imread("Test_Image.jpg");
[bboxes,scores,labels] = detect(trainedDetector,I,Threshold=0.2);
```

```
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels,LineWidth=3,AnnotationColor="cya
results = table(bboxes,labels,scores)
```

```
results=3×3 table
     88    388    124    65    carFront    0.3984
    430    366    102    87     carRear    0.7215
    795    293     65    58    stopSign    0.9798
```

```
figure
imshow(detectedImg)
```



Supporting Function

```
function data = preprocessData(data,targetSize)
for num = 1:size(data,1)
    I = data{num,1};
    imgSize = size(I);
    bboxes = data{num,2};
    I = im2single(imresize(I,targetSize(1:2)));
    scale = targetSize(1:2)./imgSize(1:2);
    bboxes = bboxresize(bboxes,scale);
    data(num,1:2) = {I bboxes};
end
end
```

Input Arguments

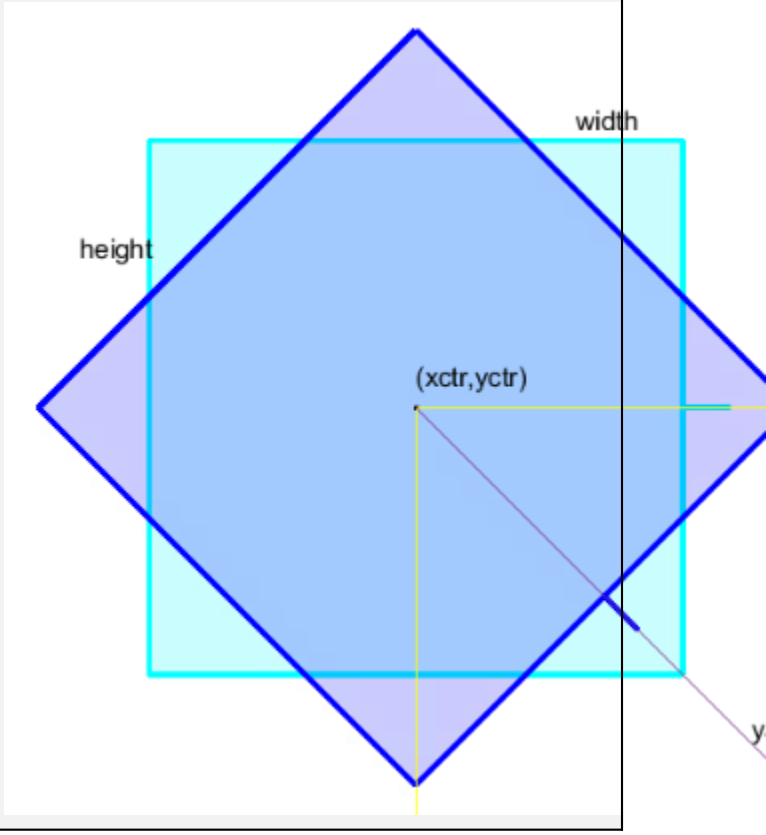
`trainingData` — Labeled ground truth images
datastore

Labeled ground truth images, specified as a datastore. Data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns in the format {*data boxes labels*}.

The first column, *data*, must contain the image data, stored as a cell array. The second column, *boxes*, must contain the bounding boxes, stored as a matrix with dimensions determined by the type of bounding box. The third column, *labels*, must be a cell array in which each cell contains an *M*-by-1 categorical vector of object class names, where *M* is the number of bounding boxes. All the categorical data returned by the datastore must use the same categories.

This table describes the format of each element of the bounding boxes column.

Bounding Box	***Description***
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	*Description*
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

Note You can convert a pretrained, axis-aligned network to a rotated rectangle network by providing rotated rectangle training data. When you provide the rotated rectangle training data, the `trainYOLOv3ObjectDetector` function fine-tunes the network, enabling the rotated rectangle detections.

`detector` — Pretrained or untrained YOLO v3 object detector
`yolov3ObjectDetector` object

Pretrained or untrained YOLO v3 object detector, specified as a `yolov3ObjectDetector` object.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function. You must set the `BatchNormalizationStatistics` property of the object to `"moving"`.

`checkpoint` — Saved detector checkpoint
`yolov3ObjectDetector` object

Saved detector checkpoint, specified as a `yolov3ObjectDetector` object. To periodically save a detector checkpoint during training as a MAT file, specify a location for the file using the `CheckpointPath` property of the training options object `options`. To control how frequently the detector saves checkpoints, use the `CheckPointFrequency` and `CheckPointFrequencyUnit` properties of the training options object.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, this code loads the checkpoint MAT file of a detector from the `"checkpath"` folder in the current working directory to which the detector saves checkpoints during training.

```
data = load("checkpath/net_checkpoint__6__2023_11_17__16_03_08.mat");
checkpoint = data.net;
```

The name of each MAT file includes the iteration number and timestamp at which the detector saves the checkpoint. The file stores the detector in the `net` variable. To continue training the network, specify the detector extracted from the file to the `trainYOLOv3ObjectDetector` function:

```
yoloDetector = trainYOLOv3ObjectDetector(trainingData,checkpoint,options);
```

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ExperimentMonitor=[]` specifies not to track metrics with Experiment Manager.

`FreezeSubNetwork` — Subnetworks to freeze
`"none"` (default) | `"backbone"` | `"backboneAndNeck"`

Subnetworks to freeze during training, specified as one of these values:

- `"none"` — Do not freeze subnetworks.
- `"backbone"` — Freeze the feature extraction subnetwork.
- `"backboneAndNeck"` — Freeze both the feature extraction and the path aggregation subnetworks.

The weight of layers in frozen subnetworks does not change during training.

Note You cannot use the `FreezeSubNetwork` argument values `"backbone"` and `"backboneAndNeck"` with a custom YOLO v3 object detector created using the syntax `yolov3ObjectDetector(net,classes,aboxes)`.

`ExperimentMonitor` — Detector training experiment monitoring

`[]` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see "Train Object Detectors in Experiment Manager".

The app monitors this Information during training:

- Training loss at each iteration
- Learning rate at each iteration
- Validation loss at each iteration, if the `options` input contains validation data

Output Arguments

`detector` — Trained YOLO v3 object detector

`yolov3ObjectDetector` object

Trained YOLO v3 object detector, returned as a `yolov3ObjectDetector` object. You can train a YOLO v3 object detector to detect multiple object classes.

`info` — Training progress information

structure array

Training progress information, returned as a structure array with these fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The `trainYOLOv3ObjectDetector` function uses mean square error for computing bounding box regression loss and cross-entropy for computing classification loss.
- `BaseLearnRate` — Learning rate at each iteration.
- `OutputNetworkIteration` — Iteration number of the returned network.
- `ValidationLoss` — Validation loss at each iteration.
- `FinalValidationLoss` — Final validation loss at the end of the training.

Each field is a numeric vector with one element per training iteration. If the function does not calculate a metric at a specific iteration, the corresponding element of that vector has a value of `NaN`. The structure contains the `ValidationLoss` and `FinalValidationLoss` fields only when `options` specifies validation data.

Tips

- To generate the ground truth, use the Image Labeler or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.

- Use these processes to improve prediction accuracy:
  - Increase the number of images you use to train the network. You can expand the training data set through data augmentation. For information on how to apply data augmentation for preprocessing, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).
  - Choose anchor boxes appropriate to the training dataset. You can use the `estimateAnchorBoxes` function to compute anchor boxes directly from the training data.
- When you train a rotated rectangle bounding box detector, use a learning rate approximately one order of magnitude below that of its axis-aligned counterpart.
- When you perform transfer learning using a YOLO v3 object detector, consider freezing the subnetworks using the `FreezeSubNetwork` name-value argument to increase training speed and reduce GPU memory consumption.

Version History

Introduced in R2024a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024b: Monitor mean average precision metric during training

Starting in R2024b, you can use an `mAPObjectDetectionMetric` object to track the mean average precision (mAP) metric while training the YOLO v3 object detector. To use the metric, specify it to the `Metrics` name-value argument of the `trainingOptions` function.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainYOLOv3ObjectDetector` function uses a GPU if one is available. You can specify the hardware that the `trainYOLOv3ObjectDetector` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainingOptions` | `objectDetectorTrainingData` | `trainYOLOv2ObjectDetector` | `trainYOLOv4ObjectDetector`

Objects
`yolov3ObjectDetector`

Topics
“Object Detection Using YOLO v3 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Train Object Detectors in Experiment Manager”
“Getting Started with YOLO v3”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainYOLOv4ObjectDetector

Train YOLO v4 object detector

Syntax

```
detector = trainYOLOv4ObjectDetector(trainingData,detector,options)
detector = trainYOLOv4ObjectDetector(trainingData,checkpoint,options)
[detector,info] = trainYOLOv4ObjectDetector( ___ )
 ___  = trainYOLOv4ObjectDetector( ___ ,Name=Value)
```

Description

`detector = trainYOLOv4ObjectDetector(trainingData,detector,options)` returns an object detector trained using you only look once version 4 (YOLO v4) network specified by the input `detector`. The input `detector` can be an untrained or pretrained YOLO v4 object detector. The `options` input specifies training parameters for the detection network.

You can also use this syntax for fine-tuning a pretrained YOLO v4 object detector.

`detector = trainYOLOv4ObjectDetector(trainingData,checkpoint,options)` resumes training from the saved detector checkpoint.

You can use this syntax to:

- Add more training data and continue the training.
- Improve training accuracy by increasing the maximum number of iterations.

`[detector,info] = trainYOLOv4ObjectDetector( ___ )` also returns information on the training progress, such as the training accuracy and learning rate for each iteration.

` ___  = trainYOLOv4ObjectDetector( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `trainYOLOv4ObjectDetector(trainingData,ExperimentManager="none")` sets the metrics to track with Experiment Manager to `"none"`.

Note To run this function, you will require the Deep Learning Toolbox.

Examples

Train YOLO v4 Network for Vehicle Detection

This example shows how to fine-tune a pretrained YOLO v4 object detector for detecting vehicles in an image.

Load a tiny YOLO v4 object detector, pretrained on the COCO dataset, and inspect its properties.

```
detector = yolov4ObjectDetector("tiny-yolov4-coco")
```

```
detector =
  yolov4ObjectDetector with properties:

       Network: [1×1 dlnetwork]
    AnchorBoxes: {2×1 cell}
     ClassNames: {80×1 cell}
      InputSize: [416 416 3]
      ModelName: 'tiny-yolov4-coco'
```

The number of anchor boxes must be same the number of output layers in the YOLO v4 network. The tiny YOLO v4 network contains two output layers.

```
detector.Network
```

```
ans =
  dlnetwork with properties:

         Layers: [74×1 nnet.cnn.layer.Layer]
    Connections: [80×2 table]
     Learnables: [80×3 table]
          State: [38×3 table]
     InputNames: {'input_1'}
    OutputNames: {'conv_31'  'conv_38'}
    Initialized: 1
```

Prepare Training Data

Load a .mat file containing the vehicle data set to use for training, in which data is stored as a table. The first column contains the training images and the remaining columns contain the labeled bounding boxes.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Specify the directory in which to store the training samples. Add the full path to the file names in training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `ImageDatastore` using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the datastores.

```
ds = combine(imds,blds);
```

Specify the input size to use for resizing the training images. The size of the training images must be a multiple of 32 for when you use the `tiny-yolov4-coco` and `csp-darknet53-coco` pretrained YOLO v4 deep learning networks. You must also resize the bounding boxes based on the specified input size.

```
inputSize = [224 224 3];
```

Resize and rescale the training images and the bounding boxes using the `preprocessData` helper function. Convert the preprocessed data to a `datastore` object using the `transform` function.

```
trainingDataForEstimation = transform(ds,@(data)preprocessData(data,inputSize));
```

Estimate Anchor Boxes

Estimate the anchor boxes from the training data. You must assign the same number of anchor boxes to each output layer in the YOLO v4 network.

```
numAnchors = 6;
[anchors,meanIoU] = estimateAnchorBoxes(trainingDataForEstimation,numAnchors);
area = anchors(:,1).*anchors(:,2);
[~,idx] = sort(area,"descend");
anchors = anchors(idx,:);
anchorBoxes = {anchors(1:3,:);anchors(4:6,:)};
```

Configure and Train YOLO v4 Network

Specify the class names and configure the pretrained YOLO v4 deep learning network to be retrained for the new data set using the `yolov4ObjectDetector` function.

```
classes = ["vehicle"];
detector = yolov4ObjectDetector("tiny-yolov4-coco",classes,anchorBoxes,InputSize=inputSize);
```

Specify the training options and retrain the pretrained YOLO v4 network on the new data set using the `trainYOLOv4ObjectDetector` function.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.001, ...
    MiniBatchSize=16, ...
    MaxEpochs=40, ...
    ResetInputNormalization=false, ...
    VerboseFrequency=30);
trainedDetector = trainYOLOv4ObjectDetector(ds,detector,options);
```

```
*************************************************************************
Training a YOLO v4 Object Detector for the following object classes:

* vehicle


    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss
    _____    _________    ___________    _________    ____________
      2         30         00:01:07       0.001         7.215
      4         60         00:01:44       0.001         1.7371
      5         90         00:02:21       0.001        0.97954
      7         120        00:02:57       0.001        0.59412
      8         150        00:03:34       0.001        0.65631
     10         180        00:04:10       0.001         1.0774
     12         210        00:04:46       0.001         0.4807
     13         240        00:05:22       0.001        0.40389
     15         270        00:05:59       0.001        0.57931
     16         300        00:06:35       0.001        0.90734
     18         330        00:07:11       0.001        0.24902
     19         360        00:07:48       0.001        0.32441
```

```
    21          390         00:08:24         0.001         0.23054
    23          420         00:09:00         0.001         0.70897
    24          450         00:09:36         0.001         0.31744
    26          480         00:10:12         0.001         0.36323
    27          510         00:10:49         0.001         0.13696
    29          540         00:11:25         0.001         0.14913
    30          570         00:12:01         0.001         0.37757
    32          600         00:12:37         0.001         0.36985
    34          630         00:13:14         0.001         0.14034
    35          660         00:13:50         0.001         0.14731
    37          690         00:14:26         0.001         0.15907
    38          720         00:15:03         0.001         0.11737
    40          750         00:15:40         0.001          0.1855

**************************************************************************
Detector training complete.
**************************************************************************
```

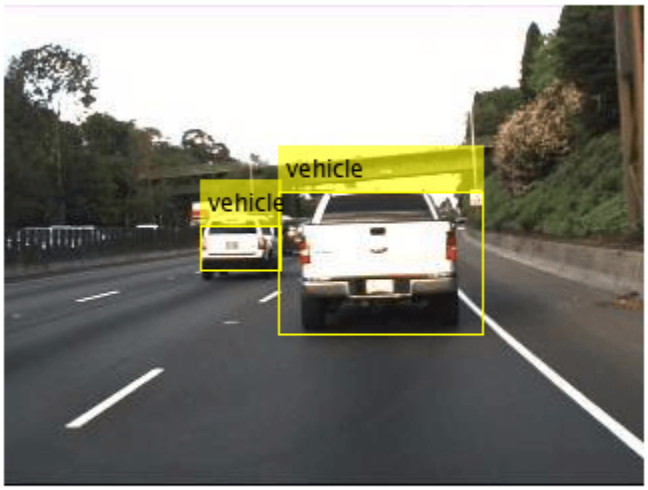
Detect Vehicles in Test Image

Load a test image from the workspace.

```
I = imread("highway.png");
```

Use the fine-tuned YOLO v4 object detector to detect vehicles in the test image and display the detection results.

```
[bboxes,scores,labels] = detect(trainedDetector,I,Threshold=0.05);
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Supporting Functions

```
function data = preprocessData(data,targetSize)
for num = 1:size(data,1)
    I = data{num,1};
    imgSize = size(I);
    bboxes = data{num,2};
    I = im2single(imresize(I,targetSize(1:2)));
    scale = targetSize(1:2)./imgSize(1:2);
    bboxes = bboxresize(bboxes,scale);
    data(num,1:2) = {I,bboxes};
end
end
```

Input Arguments

`trainingData` — Labeled ground truth images

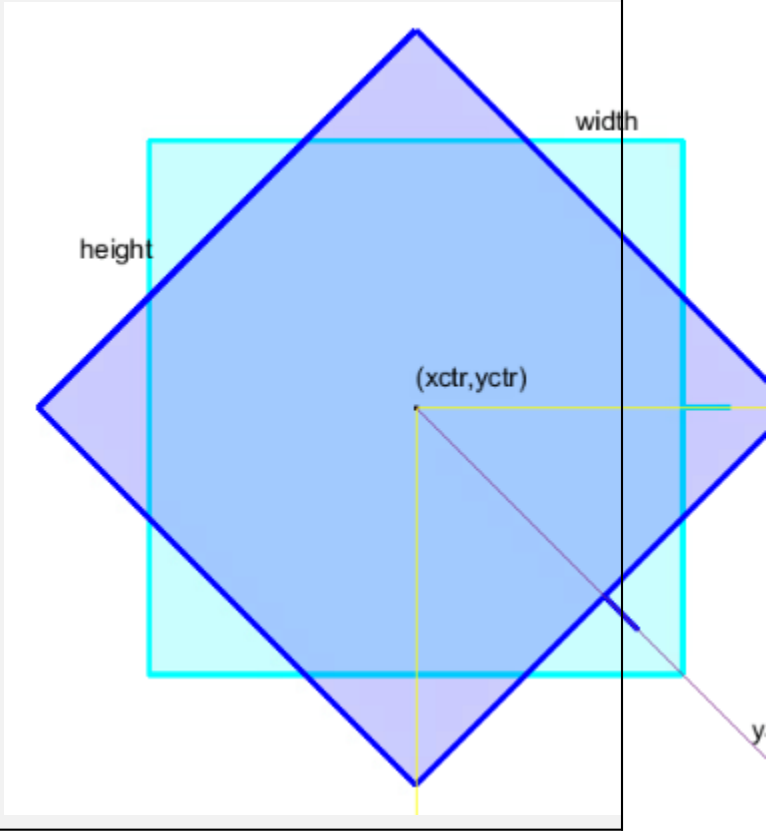
datastore

Labeled ground truth images, specified as a datastore. Data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns in the format {*data*,*boxes*,*labels*}.

The first column, *data*, must contain the image data, stored as a cell array. The second column, *boxes*, must contain the bounding boxes. The third column, *labels*, must be a cell array that contains *M*-by-1 categorical vectors containing object class names, where *M* is the number of bounding boxes. All the categorical data returned by the datastore must use the same categories.

The table describes the format of the bounding boxes column.

Bounding Box	***Description***
Axis-aligned rectangle	Defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and y specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Bounding Box	*Description*
Rotated rectangle	Defined in spatial coordinates as an *M*-by-5 numeric matrix with rows of the form [*xctr yctr w h yaw*], where: • *M* is the number of rotated rectangles. • *xctr* and *yctr* specify the center of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis before rotation. • *h* specifies the height of the rectangle, which is its length along the *y*-axis before rotation. • *yaw* specifies the rotation angle in degrees. The rotation is clockwise-positive around the center of the bounding box. 

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

Note A pretrained axis-aligned network can be converted to a rotated rectangle network by providing rotated rectangle training data. When you provide the rotated rectangle training data, the `trainYOLOv4ObjectDetector` function fine tunes the network heads allowing the rotated rectangle detections to occur.

`detector` — Pretrained or untrained YOLO v4 object detector
`yolov4ObjectDetector` object

Pretrained or untrained YOLO v4 object detector, specified as a `yolov4ObjectDetector` object.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

`checkpoint` — Saved detector checkpoint
`yolov4ObjectDetector` object

Saved detector checkpoint, specified as a `yolov4ObjectDetector` object. To periodically save a detector checkpoint during training, specify `CheckpointPath`. To control how frequently check points are saved see the `CheckPointFrequency` and `CheckPointFrequencyUnit` training options.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, if the `CheckpointPath` property of the object specified by `options` is `'checkpath'`, you can load a checkpoint MAT file by using this code. `'checkpath'` is the name of a folder in the current working directory to which the detector checkpoint has to be saved during training.

```
data = load('checkpath/net_checkpoint__19__2021_12_29__01_04_15.mat');
checkpoint = data.net;
```

The name of the MAT file includes the iteration number and timestamp of when the detector checkpoint was saved. The detector is saved in the `net` variable of the file. Pass this file back into the `trainYOLOv4ObjectDetector` function:

```
yoloDetector = trainYOLOv4ObjectDetector(trainingData,checkpoint,options);
```

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ExperimentManager="none"` specifies the metrics to track with Experiment Manager to `"none"`.

`FreezeSubNetwork` — Subnetworks to freeze
`"none"` (default) | `"backbone"` | `"backboneAndNeck"`

Subnetworks to freeze during training, specified as one of these values:

- `"none"` — Do not freeze subnetworks
- `"backbone"` — Freeze the feature extraction subnetwork
- `"backboneAndNeck"` — Freeze both the feature extraction and the path aggregation subnetworks

The weight of layers in frozen subnetworks does not change during training.

Note You cannot use the `FreezeSubNetwork` argument values `"backbone"` and `"backboneAndNeck"` with a custom YOLO v4 object detector created using the syntax `yolov4ObjectDetector(net,classes,aboxes)`.

`ExperimentManager` — Detector training experiment monitoring
`"none"` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see “Train Object Detectors in Experiment Manager”.

Information monitored during training:

- Training loss at each iteration.
- Learning rate at each iteration.

Validation information when the training `options` input contains validation data:

- Validation loss at each iteration.

Output Arguments

`detector` — Trained YOLO v4 object detector
`yolov4ObjectDetector` object

Trained YOLO v4 object detector, returned as `yolov4ObjectDetector` object. You can train a YOLO v4 object detector to detect multiple object classes.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with seven fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The `trainYOLOv4ObjectDetector` function uses mean square error for computing bounding box regression loss and cross-entropy for computing classification loss.
- `BaseLearnRate` — Learning rate at each iteration.
- `OutputNetworkIteration` — Iteration number of returned network.
- `ValidationLoss` — Validation loss at each iteration.
- `FinalValidationLoss` — Final validation loss at end of the training.

Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are assigned as `NaN`. The struct contains `ValidationLoss` and `FinalValidationLoss` fields only when `options` specifies validation data.

Tips

- To generate the ground truth, use the Image Labeler or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.
- To improve prediction accuracy,
  - Increase the number of images you can use to train the network. You can expand the training dataset through data augmentation. For information on how to apply data augmentation for preprocessing, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).
  - Choose anchor boxes appropriate to the dataset for training the network. You can use the `estimateAnchorBoxes` function to compute anchor boxes directly from the training data.
- A pretrained axis-aligned network can be converted to a rotated rectangle network by providing rotated rectangle training data. When you provide the rotated rectangle training data, the `trainYOLOv4ObjectDetector` function fine tunes the network heads allowing the rotated rectangle detections to occur.
- When you train a rotated rectangle bounding box detector, use a learning rate approximately one order of magnitude below that of its axis-aligned counterpart training rate.
- When you perform transfer learning using a YOLO v4 object detector, consider freezing the subnetworks using the name-value argument `FreezeSubNetwork` to increase training speed and reduce GPU memory consumption.

Version History

Introduced in R2022a

R2025a: MATLAB Compiler support will be removed

Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024a: Subnetwork freezing support

The `trainYOLOv4ObjectDetector` function now supports freezing subnetworks during training using the new `FreezeSubNetwork` name-value argument.

R2024b: Monitor mean average precision metric during training

Starting in R2024b, you can use an `mAPObjectDetectionMetric` object to track the mean average precision (mAP) metric while training the YOLO v4 object detector. To use the metric, specify it to the `Metrics` name-value argument of the `trainingOptions` function.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainYOLOv4ObjectDetector` function uses a GPU if one is available. You can specify the hardware that the `trainYOLOv4ObjectDetector` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Apps

Image Labeler | Video Labeler

Functions

`trainingOptions` | `yolov4ObjectDetector` | `objectDetectorTrainingData` | `trainYOLOXObjectDetector` | `trainYOLOv2ObjectDetector`

Objects

Topics

“Object Detection Using YOLO v4 Deep Learning”
“Estimate Anchor Boxes from Training Data”
“Object Detection in Large Satellite Imagery Using Deep Learning”
“Train Object Detectors in Experiment Manager”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Choose an Object Detector”

trainYOLOXObjectDetector

Train YOLOX object detector

Syntax

```
detector = trainYOLOXObjectDetector(trainingData,detectorIn,options)
[detector,info] = trainYOLOXObjectDetector( ___ )
[ ___ ] = trainYOLOXObjectDetector( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`detector = trainYOLOXObjectDetector(trainingData,detectorIn,options)` trains a YOLOX object detector using the untrained or pretrained YOLOX network, specified by `detectorIn`. The `options` input specifies the network training parameters.

You can also use this syntax for fine-tuning a pretrained YOLOX object detector.

Note This functionality requires Deep Learning Toolbox.

`[detector,info] = trainYOLOXObjectDetector( ___ )` returns information on training progress, such as training loss, for each iteration using the input arguments from the previous syntax.

`[ ___ ] = trainYOLOXObjectDetector( ___ ,Name=Value)` specifies options using one or more name-value arguments. For example, `trainYOLOXObjectDetector(trainingData,ExperimentMonitor=[])` specifies not to track metrics with Experiment Manager.

Examples

Train YOLOX Network for Vehicle Detection

Prepare Training Data

Load a MAT file containing information about a vehicle data set to use for training into the workspace. The MAT file stores the information stored as a table. The first column contains the training images and the remaining columns contain the labeled bounding boxes for those images.

```
data = load("vehicleTrainingData.mat");
trainingData = data.vehicleTrainingData;
```

Specify the directory that contains the training sample image data files and box labels. Add the full path to the filenames in the training data.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata");
trainingData.imageFilename = fullfile(dataDir,trainingData.imageFilename);
```

Create an `imageDatastore` object using the files from the table.

```
imds = imageDatastore(trainingData.imageFilename);
```

Create a `boxLabelDatastore` object using the label columns from the table.

```
blds = boxLabelDatastore(trainingData(:,2:end));
```

Combine the image datastore and box label datastore.

```
ds = combine(imds,blds);
```

Specify the input size to use for resizing the training images.

```
inputSize = [128 228 3];
```

Partition Data

Split the data set into training and validation sets. Allocate approximately 80% of the data for training, and the rest for validation.

```
numImages = numpartitions(ds);
shuffledIndices = randperm(numImages);
dsTrain = subset(ds,shuffledIndices(1:round(0.8*numImages)));
dsVal = subset(ds,shuffledIndices(numpartitions(dsTrain)+1:end));
```

Define YOLOX Object Detector Network Architecture

Create a YOLOX object detector by using the `yoloxObjectDetector` function. Specify `"tiny-coco"` as the pretrained network to use as the base network. Specify the class names and the network input size.

```
classes = {'vehicle'};
detector = yoloxObjectDetector("tiny-coco",classes,InputSize=inputSize);
```

Train YOLOX Object Detection Network

Specify network training options using the `trainingOptions` (Deep Learning Toolbox) function. Use the `mAPObjectDetectionMetric` object to track the mean average precision (mAP) metric when you train the detector.

```
options = trainingOptions("sgdm", ...
    InitialLearnRate=0.001, ...
    MiniBatchSize=16,...
    MaxEpochs=4, ...
    ResetInputNormalization=false, ...
    Metrics=mAPObjectDetectionMetric(Name="mAP50"), ...
    ObjectiveMetricName="mAP50", ...
    ValidationData=dsVal, ...
    ValidationFrequency=20, ...
    VerboseFrequency=2);
```

Train the pretrained YOLOX network on the new data set by using the `trainYOLOXObjectDetector` function.

```
trainedDetector = trainYOLOXObjectDetector(dsTrain,detector,options);
```

```
    Epoch    Iteration    TimeElapsed    LearnRate    TrainingYoloXLoss    ValidationYoloXLoss
```

```
_____    _________    ___________    ________    _________________    _________________

  1          2        00:00:20       0.001          5.1181
  1          4        00:00:25       0.001          5.0637
  1          6        00:00:29       0.001          2.9549
  1          8        00:00:33       0.001          4.5492
  1         10        00:00:37       0.001          3.3555
  1         12        00:00:41       0.001          3.1534
  1         14        00:00:46       0.001           2.812
  2         16        00:00:50       0.001          2.4134
  2         18        00:00:55       0.001           2.203
  2         20        00:00:59       0.001          2.1435                2.4607
  2         22        00:01:20       0.001           2.276
  2         24        00:01:25       0.001          2.4464
  2         26        00:01:31       0.001          2.3203
  2         28        00:01:36       0.001          2.0818
  3         30        00:01:42       0.001          1.9096
  3         32        00:01:48       0.001          2.1849
  3         34        00:01:54       0.001           1.888
  3         36        00:01:59       0.001          1.9287
  3         38        00:02:04       0.001           1.944
  3         40        00:02:09       0.001          2.3037                2.1339
  3         42        00:02:32       0.001          1.7938
  4         44        00:02:37       0.001          2.3005
  4         46        00:02:42       0.001          1.9379
  4         48        00:02:46       0.001          2.0413
  4         50        00:02:50       0.001          1.9138
  4         52        00:02:54       0.001          1.9997
  4         54        00:02:58       0.001          1.9646
  4         56        00:03:03       0.001          2.0116                2.1982
```

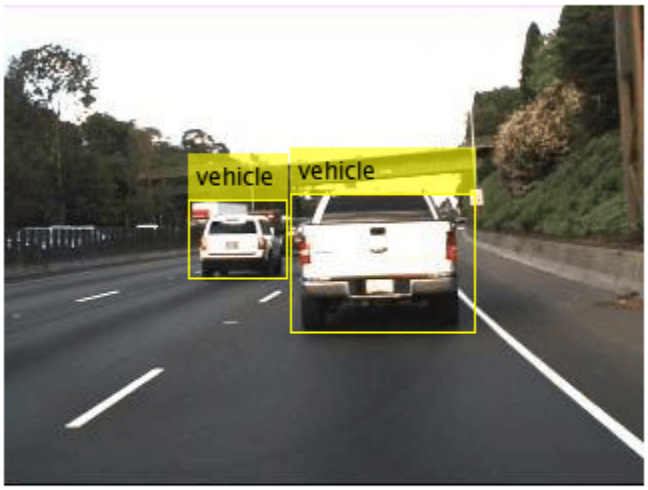
Detect Vehicles in Test Image

Read a test image.

```
img = imread("highway.png");
```

Use the fine-tuned YOLOX object detector to detect vehicles in the test image and display the detection results.

```
[boxes,scores,labels] = detect(trainedDetector,img,Threshold=0.5);
detectedImage = insertObjectAnnotation(img,"rectangle",boxes,labels);
figure
imshow(detectedImage)
```



Input Arguments

`trainingData` — Labeled ground truth images

datastore

Labeled ground truth images, specified as a datastore. You must configure the datastore so that calling it with the `read` and `readall` functions returns a cell array or table with these three columns:

Image Data	Bounding Boxes	Bounding Box Labels
Single-cell cell array, which contains the input image.	Single-cell array, which contains the bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.	Single-cell cell array, which contains an *M*-by-1 categorical vector of object class names. All the categorical data returned by the datastore must use the same categories.

Use the `combine` function on two datastores to create a datastore that returns these three data columns using `read`. The datastore that returns the first column of data should be an `ImageDatastore`, while the datastore that returns the second and third columns of data should be a `boxLabelDatastore`.

`detectorIn` — Pretrained or untrained YOLOX object detector
`yoloxObjectDetector` object

Pretrained or untrained YOLOX object detector, specified as a `yoloxObjectDetector` object.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ExperimentMonitor=[]` specifies not to track metrics with Experiment Manager.

`ExperimentMonitor` — Detector training experiment monitoring
`[]` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, or produce training plots. For more information on using this app, see the "Train Object Detectors in Experiment Manager" example.

The app monitors this information during training:

- Training loss at each iteration
- Learning rate at each iteration
- Validation loss at each iteration, if the `options` input contains validation data

`FreezeSubNetwork` — Subnetwork to freeze
`"none"` (default) | `"backbone"`

Subnetwork to freeze during training, specified as one of these values:

- `"none"` — Do not freeze a subnetwork.
- `"backbone"` — Freeze the feature extraction subnetwork, including the layers following the region of interest (ROI) align layer.

The weight of layers in frozen subnetworks does not change during training.

`ClassWeights` — Class weights for loss function
`1` (default) | positive scalar | *C*-element numeric vector

Class weights for the loss function, specified as a positive scalar or as a *C*-element numeric vector, where *C* is the number of classes in the `ClassNames` property of the `yoloxObjectDetector` object `detectorIn`, and all elements in the vector are positive.

By default, all classes have a weight of `1`, indicating equal weight for all classes. If you specify `ClassWeights` as any positive scalar, the function assigns equal weight to all classes. To assign different weights to each class, specify `ClassWeights` as a *C*-element numeric vector with positive values.

Class imbalance in the training data can affect the accuracy of the detector. You can reduce the effect of this class imbalance by assigning class weights inversely related to the class distribution in the training data. Assign higher weights to rare classes that have lower frequency in the training data and lower weights to common classes that have higher frequency in the training data. Ensure that the weights are approximately in inverse proportion to the class distribution in the training data. If you specify the weights of rare classes extremely higher than the required proportion, the model can become unstable or overfit to rare classes. If you specify the weights of rare classes extremely lower than the required proportion, the model may not reduce the effect of the class imbalance sufficiently.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`detector` — Trained YOLOX object detector

`yoloxObjectDetector` object

Trained YOLOX object detector, returned as a `yoloxObjectDetector` object. You can train a YOLOX object detector to detect multiple object classes.

`info` — Training progress information

structure array

Training progress information, returned as a structure array with these fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The `trainYOLOXObjectDetector` function uses mean square error for computing bounding box regression loss and cross-entropy for computing classification loss.
- `BaseLearnRate` — Learning rate at each iteration.
- `OutputNetworkIteration` — Iteration number of the returned network.
- `ValidationLoss` — Validation loss at each iteration.
- `FinalValidationLoss` — Final validation loss at the end of the training.

Each field is a numeric vector with one element per training iteration. The function returns a value of `NaN` for iterations at which it does not calculate that value. The structure contains the `ValidationLoss` and `FinalValidationLoss` fields only when `options` specifies validation data.

Tips

- To generate the labeled ground truth image data, use the Image Labeler or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.
- To help improve prediction accuracy, increase the number of images you use to train the network. You can expand the training data set using data augmentation. For information on how to apply data augmentation for preprocessing, see “Preprocess Images for Deep Learning” (Deep Learning Toolbox).

Version History

Introduced in R2023b

R2026a: Specify class weights for loss function

Specify class weights for the loss function by using the new `ClassWeights` name-value argument.

R2025a: MATLAB Compiler support will be removed

Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU acceleration.

By default, the `trainYOLOXObjectDetector` function uses a GPU if one is available. You can specify the hardware that the `trainYOLOXObjectDetector` function uses by setting the `ExecutionEnvironment` training option using the `trainingOptions` function.

For more information, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

See Also

Apps

Image Labeler | Video Labeler

Functions

`trainingOptions` | `mAPObjectDetectionMetric` | `objectDetectorTrainingData` | `detect` | `evaluateObjectDetection`

Objects

`yoloxObjectDetector`

Topics

“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Getting Started with YOLOX for Object Detection”
“Train Object Detectors in Experiment Manager”
“Datastores for Deep Learning” (Deep Learning Toolbox)

detect

Detect objects using ACF object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
detectionResults = detect(detector,ds)
[___]= detect(detector,I,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within image `I` using the input aggregate channel features (ACF) object detector. The locations of objects detected are returned as a set of bounding boxes.

`[bboxes,scores] = detect(detector,I)` also returns the detection scores for each bounding box.

`detectionResults = detect(detector,ds)` detects objects within all the images returned by the `read` function of the input datastore.

`[___]= detect(detector,I,roi)` detects objects within the rectangular search region specified by `roi`, using either of the preceding syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `WindowStride=2` sets the stride of the sliding window used to detects objects to `2`.

Examples

Train Stop Sign Detector Using ACF Object Detector

Use the `trainACFObjectDetector` with training images to create an ACF object detector that can detect stop signs. Test the detector with a separate image.

Load the training data.

```
load('stopSignsAndCars.mat')
```

Prefix the full path to the stop sign images.

```
stopSigns = fullfile(toolboxdir('vision'),'visiondata',stopSignsAndCars{:,1});
```

Create datastores to load the ground truth data for stop signs.

```
imds = imageDatastore(stopSigns);
blds = boxLabelDatastore(stopSignsAndCars(:,2));
```

Combine the image and box label datastores.

```
ds = combine(imds,blds);
```

Train the ACF detector. Set the number of negative samples to use at each stage to 2. You can turn off the training progress output by specifying `Verbose=false`, as a `Name-Value` argument.

```
acfDetector = trainACFObjectDetector(ds,NegativeSamplesFactor=2);
```

```
ACF Object Detector Training
The training will take 4 stages. The model size is 34x31.
Sample positive examples(~100% Completed)
Compute approximation coefficients...Completed.
Compute aggregated channel features...Completed.
--------------------------------------------
Stage 1:
Sample negative examples(~100% Completed)
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 19 weak learners.
--------------------------------------------
Stage 2:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 20 weak learners.
--------------------------------------------
Stage 3:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 54 weak learners.
--------------------------------------------
Stage 4:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 61 weak learners.
--------------------------------------------
ACF object detector training is completed. Elapsed time is 17.9982 seconds.
```

Test the ACF detector on a test image.

```
img = imread('stopSignTest.jpg');
[bboxes,scores] = detect(acfDetector,img);
```

Display the detection results and insert the bounding boxes for objects into the image.

```
for i = 1:length(scores)
   annotation = sprintf('Confidence = %.1f',scores(i));
   img = insertObjectAnnotation(img,'rectangle',bboxes(i,:),annotation);
end

figure
imshow(img)
```



Input Arguments

`detector` — ACF object detector

`acfObjectDetector` object

ACF object detector, specified as an `acfObjectDetector` object. To create this object, call the `trainACFObjectDetector` function with training data as input.

`I` — Input image

grayscale image | RGB image

Input image, specified as a real, nonsparse, grayscale or RGB image.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore

`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be a grayscale or RGB. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns. Therefore, datastore `read` function must return image data in the first column.

`roi` — Search region of interest

[*x y width height*] vector

Search region of interest, specified as an [*x y width height*] vector. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `WindowStride=2` sets the stride of the sliding window used to detects objects to 2.

`NumScaleLevels` — Number of scale levels per octave
8 (default) | positive integer

Number of scale levels per octave, specified a positive integer. Each octave is a power-of-two downscaling of the image. To detect people at finer scale increments, increase this number. Recommended values are in the range [4, 8].

`WindowStride` — Stride for sliding window
4 (default) | positive integer

Stride for the sliding window, specified as a positive integer. This value indicates the distance for the function to move the window in both the *x* and *y* directions. The sliding window scans the images for object detection.

`SelectStrongest` — Select strongest bounding box for each object
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Return the strongest bounding box per object. To select these boxes, `detect` calls the `selectStrongestBbox` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.
- `false` — Return all detected bounding boxes. You can then create your own custom operation to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
[*height width*] vector

Minimum region size that contains a detected object, specified as a vector of the form [*height width*]. Units are in pixels.

By default, `MinSize` is the smallest object that the trained `detector` can detect.

`MaxSize` — Maximum region size
`size(I)` (default) | [*height width*] vector

Maximum region size that contains a detected object, specified as a vector of the form [*height width*]. Units are in pixels.

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `'MaxSize'` is set to the height and width of the input image, `I`.

`Threshold` — Classification accuracy threshold
–1 (default) | numeric scalar

Classification accuracy threshold, specified as a numeric scalar. Recommended values are in the range [–1, 1]. During multiscale object detection, the threshold value controls the accuracy and speed for classifying image subregions as either objects or nonobjects. To speed up the performance at the risk of missing true detections, increase this threshold.

Output Arguments

`bboxes` — Location of objects detected within image

M-by-4 matrix

Location of objects detected within the input image, returned as an *M*-by-4 matrix, where *M* is the number of bounding boxes. Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

`scores` — Detection confidence scores

M-by-1 vector

Detection confidence scores, returned as an *M*-by-1 vector, where *M* is the number of bounding boxes. Scores are returned in the range [`-inf inf`]. A higher score indicates higher confidence in the detection.

`detectionResults` — Detection results

2-column table

Detection results, returned as a 2-column table with variable names, *Boxes*, and *Scores*.

The *Boxes* column contains *M*-by-4 matrices, representing *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

The *Scores* column contains *M*-by-1 vectors representing the detection confidence scores for each of the *M* bounding boxes.

Version History

Introduced in R2017a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To generate code, use `toStruct` function to pass the `acfObjectDetector` object to the `detect` function. For more information, see “Generate Code for Detecting Objects in Images by Using ACF Object Detector”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainACFObjectDetector` | `detectPeopleACF` | `trainCascadeObjectDetector` | `selectStrongestBbox`

Objects
`acfObjectDetector`

detect

(Not recommended) Detect objects using R-CNN deep learning detector

Note The `detect` function and R-CNN object detectors are not recommended. Use a different type of object detector instead. For more information, see “Version History”.

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[___,labels] = detect(detector,I)
[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within image `I` using an R-CNN (regions with convolutional neural networks) object detector. The locations of objects detected are returned as a set of bounding boxes.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[bboxes,scores] = detect(detector,I)` also returns the detection scores for each bounding box.

`[___,labels] = detect(detector,I)` also returns a categorical array of labels assigned to the bounding boxes, using either of the preceding syntaxes. The labels used for object classes are defined during training using the `trainRCNNObjectDetector` function.

`[___] = detect(___,roi)` detects objects within the rectangular search region specified by `roi`.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes.

Examples

Train R-CNN Stop Sign Detector

Load training data and network layers.

```
load('rcnnStopSigns.mat', 'stopSigns', 'layers')
```

Add the image directory to the MATLAB path.

```
imDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondata',...
  'stopSignImages');
addpath(imDir);
```

Set network training options to use mini-batch size of 32 to reduce GPU memory usage. Lower the InitialLearnRate to reduce the rate at which network parameters are changed. This is beneficial when fine-tuning a pre-trained network and prevents the network from changing too rapidly.

```
options = trainingOptions('sgdm', ...
  'MiniBatchSize', 32, ...
  'InitialLearnRate', 1e-6, ...
  'MaxEpochs', 10);
```

Train the R-CNN detector. Training can take a few minutes to complete.

```
rcnn = trainRCNNObjectDetector(stopSigns, layers, options, 'NegativeOverlapRange', [0 0.3]);
```

```
*********************************************************************
Training an R-CNN Object Detector for the following object classes:

* stopSign

--> Extracting region proposals from 27 training images...done.

--> Training a neural network to classify objects in training data...

Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:01 |       96.88% |       0.1651 |      1.0000e-06 |
|       2 |          50 |       00:00:15 |       96.88% |       0.0807 |      1.0000e-06 |
|       3 |         100 |       00:00:27 |       96.88% |       0.1340 |      1.0000e-06 |
|       5 |         150 |       00:00:38 |       96.88% |       0.0225 |      1.0000e-06 |
|       6 |         200 |       00:00:49 |       93.75% |       0.6584 |      1.0000e-06 |
|       8 |         250 |       00:01:00 |       93.75% |       0.5233 |      1.0000e-06 |
|       9 |         300 |       00:01:11 |      100.00% |   2.9456e-05 |      1.0000e-06 |
|      10 |         350 |       00:01:23 |      100.00% |       0.0009 |      1.0000e-06 |
|=========================================================================================|
Training finished: Max epochs completed.

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*********************************************************************
```

Test the R-CNN detector on a test image.

```
img = imread('stopSignTest.jpg');
```

```
[bbox, score, label] = detect(rcnn, img, MiniBatchSize=32);
```

Display strongest detection result.

```
[score, idx] = max(score);

bbox = bbox(idx, :);
annotation = sprintf('%s: (Confidence = %f)', label(idx), score);

detectedImg = insertObjectAnnotation(img, 'rectangle', bbox, annotation);

figure
imshow(detectedImg)
```



Remove the image directory from the path.

```
rmpath(imDir);
```

Input Arguments

`detector` — R-CNN object detector

`rcnnObjectDetector` object

R-CNN object detector, specified as an `rcnnObjectDetector` object. To create this object, call the `trainRCNNObjectDetector` function with training data as input.

`I` — Input image

H-by-*W*-by-*C* numeric array of images

Input image, specified as an *H*-by-*W*-by-*C* numeric array of images. Images must be real, nonsparse, grayscale or RGB image.

- *H*: Height
- *W*: Width
- *C*: The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to `1`. For RGB color images, it must be equal to `3`.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as an [*x y width height*] vector. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `detect(detector,I,NumStrongestRegions=1000)` limits the number of strongest region proposals to 1000.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | integer

Maximum number of strongest region proposals, specified as an integer. Reduce this value to speed up processing time at the cost of detection accuracy. To use all region proposals, specify this value as `Inf`.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select strongest bounding box for each detected object, specified as a logical `true` or `false`, where:

- `true` — Return the strongest bounding box per object. To select these boxes, `detect` calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their scores.

  For example:

```
selectStrongestBboxMulticlass(bbox,scores, ...
            'RatioType','Min', ...
            'OverlapThreshold',0.5);
```

- `false` — Return all detected bounding boxes. You can then use a custom operation to eliminate overlapping bounding boxes.

`MiniBatchSize` — Size of smaller batches for R-CNN data processing
128 (default) | integer

Size of smaller batches for R-CNN data processing, specified as an integer. Larger batch sizes lead to faster processing but take up more memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`bboxes` — Location of objects detected within image
M-by-4 matrix

Location of objects detected within the image, returned as an *M*-by-4 matrix defining *M* bounding boxes. Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of a bounding box in pixels.

`scores` — Detection scores
M-by-1 vector

Detection scores, returned as an *M*-by-1 vector. A higher score indicates higher confidence in the detection.

`labels` — Labels for bounding boxes
M-by-1 categorical array

Labels for bounding boxes, returned as an *M*-by-1 categorical array of *M* labels. You define the class names used to label the objects when you train the input `detector`.

Version History

Introduced in R2016b

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainRCNNObjectDetector` | `selectStrongestBboxMulticlass` | `evaluateObjectDetection`

Objects
`rcnnObjectDetector`

detect

(Not recommended) Detect objects using Fast R-CNN object detector

Note The `detect` function and Fast R-CNN object detectors are not recommended. Use a different type of object detector instead. For more information, see “Version History”.

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[___,labels] = detect(detector,I)
[___] = detect(___,roi)

detectionResults = detect(detector,ds)

[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using a Fast R-CNN (regions with convolutional neural networks) object detector. The locations of objects detected are returned as a set of bounding boxes.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[bboxes,scores] = detect(detector,I)` also returns the detection scores for each bounding box.

`[___,labels] = detect(detector,I)` also returns a categorical array of labels assigned to the bounding boxes, using either of the preceding syntaxes. The labels used for object classes are defined during training using the `trainFastRCNNObjectDetector` function.

`[___] = detect(___,roi)` detects objects within the rectangular search region specified by `roi`.

`detectionResults = detect(detector,ds)` detects objects within the series of images returned by the `read` function of the input datastore.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes.

Examples

Detect Vehicles Using Faster R-CNN

Detect vehicles within an image by using a Faster R-CNN object detector.

Load a Faster R-CNN object detector pretrained to detect vehicles.

```
data = load('fasterRCNNVehicleTrainingData.mat', 'detector');
detector = data.detector;
```

Read in a test image.

```
I = imread('highway.png');
imshow(I)
```



Run the detector on the image and inspect the results. The labels come from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 2×4

   150    86    80    72
    91    89    67    48


scores = 2×1 single column vector

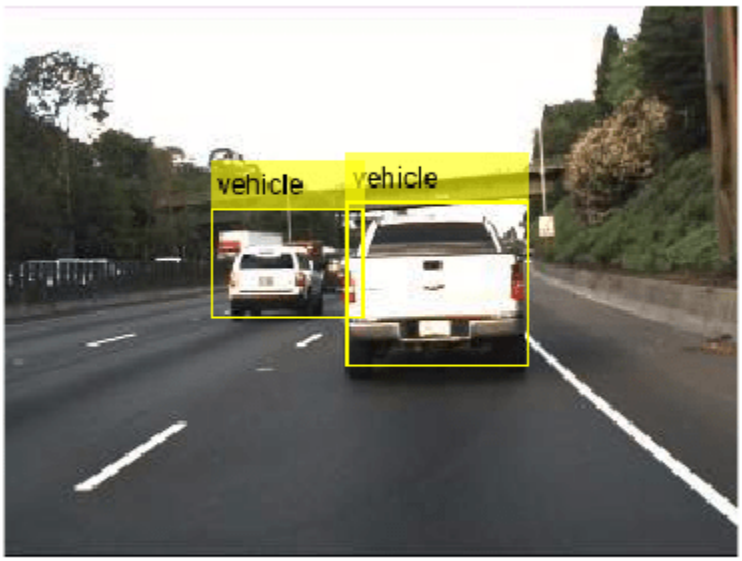
    1.0000
    0.9001


labels = 2×1 categorical
     vehicle
```

```
vehicle
```

The detector has high confidence in the detections. Annotate the image with the bounding boxes for the detections and the corresponding detection scores.

```
detectedI = insertObjectAnnotation(I,'Rectangle',bboxes,cellstr(labels));
figure
imshow(detectedI)
```



Input Arguments

`detector` — Fast R-CNN object detector

`fastRCNNObjectDetector` object

Fast R-CNN object detector, specified as a `fastRCNNObjectDetector` object. To create this object, call the `trainFastRCNNObjectDetector` function with training data as input.

`I` — Input image

H-by-*W*-by-*C*-by-*B* numeric array of images

Input image, specified as an *H*-by-*W*-by-*C*-by-*B* numeric array of images. Images must be real, nonsparse, grayscale or RGB image.

- *H* — Height in pixels.
- *W* — Width in pixels
- *C* — The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to 1. For RGB color images, it must be equal to 3.
- *B* — Number of images in the array.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`ds` — Datastore
`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be a grayscale, RGB, or multichannel image. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as a four-element vector of the form [*x y width height*]. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `detect(detector,I,NumStrongestRegions=1000)` limits the number of strongest region proposals to 1000.

`Threshold` — Detection threshold
`0.5` (default) | scalar in the range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals, specified as a positive integer. Reduce this value to speed up processing time at the cost of detection accuracy. To use all region proposals, specify this value as `Inf`.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as a logical `true` or `false`, where:

- `true` — Return the strongest bounding box per object. To select these boxes, `detect` calls the `selectStrongestBboxMulticlass` function, which uses non-maximum suppression to eliminate overlapping bounding boxes based on their confidence scores.

  For example:

  ```
  selectStrongestBboxMulticlass(bbox,scores, ...
              'RatioType','Min', ...
              'OverlapThreshold',0.5);
  ```

- `false` — Return all detected bounding boxes. You can then create your own custom operation to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
[*height width*] vector

Minimum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

By default, `MinSize` is the smallest object that the trained `detector` can detect.

`MaxSize` — Maximum region size
`size(I)` (default) | [*height width*] vector

Maximum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `'MaxSize'` is set to the height and width of the input image `I`.

`MiniBatchSize` — Minimum batch size
128 (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of images. Images are grouped into minibatches and processed as a batch to improve computation efficiency. Increase the minibatch size to decrease processing time. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

Output Arguments

`bboxes` — Location of objects detected
M-by-4 matrix | *B*-by-1 cell array

Location of objects detected within the input image or images, returned as an *M*-by-4 matrix or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-4 matrices when the input contains an array of images.

Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

`scores` — Detection scores
M-by-1 vector | *B*-by-1 cell array

Detection confidence scores, returned as an *M*-by-1 vector or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-1 vectors when the input contains an array of images. A higher score indicates higher confidence in the detection.

`labels` — Labels for bounding boxes
M-by-1 categorical array | *B*-by-1 cell array

Labels for bounding boxes, returned as an *M*-by-1 categorical array or a *B*-by-1 cell array. *M* is the number of labels in an image, and *B* is the number of *M*-by-1 categorical arrays when the input contains an array of images. You define the class names used to label the objects when you train the input `detector`.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column contains *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

Version History

Introduced in R2017a

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainFastRCNNObjectDetector` | `selectStrongestBboxMulticlass` | `evaluateObjectDetection`

Objects
`fastRCNNObjectDetector`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)

detect

(Not recommended) Detect objects using Faster R-CNN object detector

Note The `detect` function and Faster R-CNN object detectors are not recommended. Use a different type of object detector instead. For more information, see “Version History”.

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[___,labels] = detect(detector,I)

detectionResults = detect(detector,ds)

[___] = detect(___,roi)
[___] = detect(___,Name,Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using a Faster R-CNN (regions with convolutional neural networks) object detector. The locations of objects detected are returned as a set of bounding boxes.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[bboxes,scores] = detect(detector,I)` also returns the detection scores for each bounding box.

`[___,labels] = detect(detector,I)` also returns a categorical array of labels assigned to the bounding boxes, using either of the preceding syntaxes. The labels used for object classes are defined during training using the `trainFasterRCNNObjectDetector` function.

`detectionResults = detect(detector,ds)` detects objects within the series of images returned by the `read` function of the input datastore.

`[___] = detect(___,roi)` detects objects within the rectangular search region specified by `roi`.

`[___] = detect(___,Name,Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `NumStrongestRegions=1000` limits the number of strongest region proposals to 1000.

Examples

Detect Vehicles Using Faster R-CNN

Detect vehicles within an image by using a Faster R-CNN object detector.

Load a Faster R-CNN object detector pretrained to detect vehicles.

```
data = load('fasterRCNNVehicleTrainingData.mat', 'detector');
detector = data.detector;
```

Read in a test image.

```
I = imread('highway.png');
imshow(I)
```



Run the detector on the image and inspect the results. The labels come from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 2×4

   150    86    80    72
    91    89    67    48


scores = 2×1 single column vector

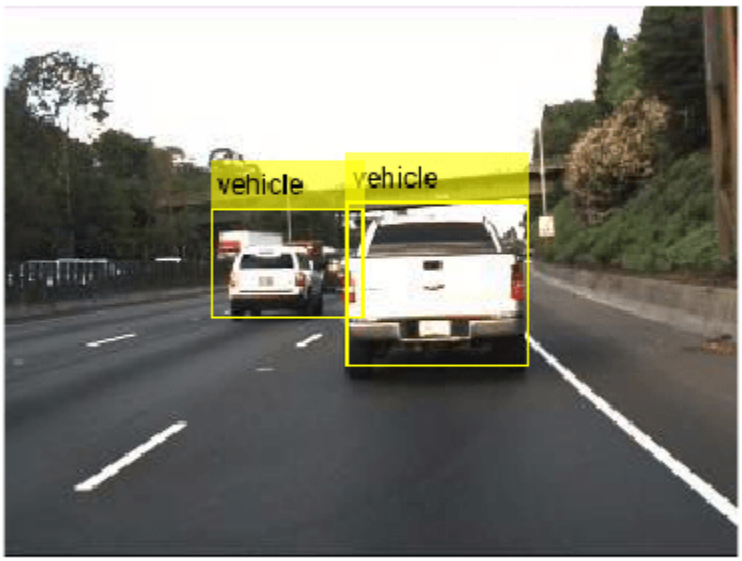
    1.0000
    0.9001


labels = 2×1 categorical
     vehicle
```

```
vehicle
```

The detector has high confidence in the detections. Annotate the image with the bounding boxes for the detections and the corresponding detection scores.

```
detectedI = insertObjectAnnotation(I,'Rectangle',bboxes,cellstr(labels));
figure
imshow(detectedI)
```



Input Arguments

`detector` — Faster R-CNN object detector

`fasterRCNNObjectDetector` object

Faster R-CNN object detector, specified as a `fasterRCNNObjectDetector` object. To create this object, call the `trainFasterRCNNObjectDetector` function with training data as input.

`I` — Input image

H-by-*W*-by-*C*-by-*B* numeric array of images

Input image, specified as an *H*-by-*W*-by-*C*-by-*B* numeric array of images. Images must be real, nonsparse, grayscale or RGB image.

- *H* — Height in pixels.
- *W* — Width in pixels
- *C* — The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to 1. For RGB color images, it must be equal to 3.
- *B* — Number of images in the array.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`ds` — Datastore
`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be a grayscale, RGB, or multichannel image. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as a four-element vector of the form [*x y width height*]. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `detect(detector,I,NumStrongestRegions=1000)` limits the number of strongest region proposals to 1000.

`Threshold` — Detection threshold
`0.5` (default) | scalar in the range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | positive integer | `Inf`

Maximum number of strongest region proposals, specified as a positive integer. Reduce this value to speed up processing time at the cost of detection accuracy. To use all region proposals, specify this value as `Inf`.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as a logical `true` or `false`, where:

- `true` — Return the strongest bounding box per object. To select these boxes, `detect` calls the `selectStrongestBboxMulticlass` function, which uses non-maximum suppression to eliminate overlapping bounding boxes based on their confidence scores.

  For example:

  ```
  selectStrongestBboxMulticlass(bbox,scores, ...
              'RatioType','Min', ...
              'OverlapThreshold',0.5);
  ```

- `false` — Return all detected bounding boxes. You can then create your own custom operation to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
[*height width*] vector

Minimum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

By default, `MinSize` is the smallest object that the trained `detector` can detect.

`MaxSize` — Maximum region size
`size(I)` (default) | [*height width*] vector

Maximum region size that contains a detected object, specified as a [*height width*] vector. Units are in pixels.

To reduce computation time, set this value to the known maximum region size for the objects being detected in the image. By default, `'MaxSize'` is set to the height and width of the input image `I`.

`MiniBatchSize` — Minimum batch size
128 (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of images. Images are grouped into minibatches and processed as a batch to improve computation efficiency. Increase the minibatch size to decrease processing time. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

Output Arguments

`bboxes` — Location of objects detected
M-by-4 matrix | *B*-by-1 cell array

Location of objects detected within the input image or images, returned as an *M*-by-4 matrix or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-4 matrices when the input contains an array of images.

Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

`scores` — Detection scores
M-by-1 vector | *B*-by-1 cell array

Detection confidence scores, returned as an *M*-by-1 vector or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-1 vectors when the input contains an array of images. A higher score indicates higher confidence in the detection.

`labels` — Labels for bounding boxes
M-by-1 categorical array | *B*-by-1 cell array

Labels for bounding boxes, returned as an *M*-by-1 categorical array or a *B*-by-1 cell array. *M* is the number of labels in an image, and *B* is the number of *M*-by-1 categorical arrays when the input contains an array of images. You define the class names used to label the objects when you train the input `detector`.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column contains *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

Version History

Introduced in R2017a

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainYOLOv2ObjectDetector` | `trainFasterRCNNObjectDetector` | `selectStrongestBboxMulticlass` | `evaluateObjectDetection`

Objects
`fasterRCNNObjectDetector` | `boxLabelDatastore`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)

detect

Detect objects using YOLO v2 object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[___,labels] = detect(detector,I)
[___,labels,info] = detect(detector,I)

detectionResults = detect(detector,ds)

[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

`bboxes = detect(detector,I)` detects objects within a single image or an array of images, `I`, using you only look once version 2 (YOLO v2) object detector. The input size of the image must be greater than or equal to the network input size of the pretrained detector. The locations of objects detected are returned as a set of bounding boxes.

Note The use of a CUDA-enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[bboxes,scores] = detect(detector,I)` also returns the class-specific confidence scores for each bounding box.

`[___,labels] = detect(detector,I)` returns a categorical array of labels assigned to the bounding boxes in addition to the output arguments from the previous syntax. The labels used for object classes are defined during training using the `trainYOLOv2ObjectDetector` function.

`[___,labels,info] = detect(detector,I)` also returns information about the class probabilities and objectness scores for each detection.

`detectionResults = detect(detector,ds)` detects objects within all the images returned by the `read` function of the input datastore.

`[___] = detect(___,roi)` detects objects within the rectangular search region specified by `roi`. Use output arguments from any of the previous syntaxes. Specify input arguments from any of the previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Threshold=0.75` sets the threshold to remove detections to `0.75`.

Examples

Detect Vehicles Using YOLO v2 Object Detection Network

Load a YOLO v2 object detector pretrained to detect vehicles.

```
vehicleDetector = load('yolov2VehicleDetector.mat','detector');
detector = vehicleDetector.detector;
```

Read a test image into the workspace.

```
I = imread('highway.png');
```

Display the input test image.

```
imshow(I);
```



Run the pretrained YOLO v2 object detector on the test image. Inspect the results for vehicle detection. The labels are derived from the `ClassNames` property of the detector.

```
[bboxes,scores,labels] = detect(detector,I)
```

```
bboxes = 1×4

    78    81    64    63

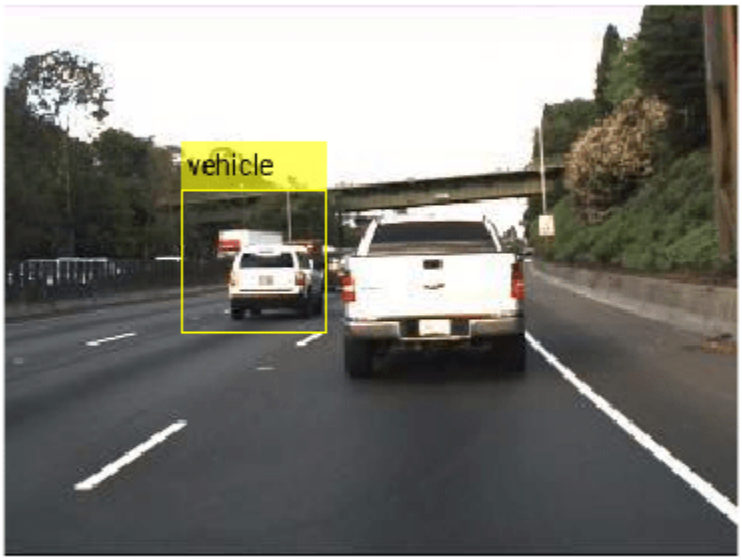

scores = single

0.6224
```

```
labels = categorical
     vehicle
```

Annotate the image with the bounding boxes for the detections.

```
if ~isempty(bboxes)
    detectedI = insertObjectAnnotation(I,'rectangle',bboxes,cellstr(labels));
end
figure
imshow(detectedI)
```



Input Arguments

`detector` — YOLO v2 object detector

`yolov2ObjectDetector` object

YOLO v2 object detector, specified as a `yolov2ObjectDetector` object. To create this object, call the `trainYOLOv2ObjectDetector` function with the training data as input.

`I` — Input image

H-by-*W*-by-*C*-by-*B* numeric array of images

Input image, specified as an *H*-by-*W*-by-*C*-by-*B* numeric array of images. Images must be real, nonsparse, grayscale or RGB image.

- *H* — Height in pixels.
- *W* — Width in pixels
- *C* — The channel size in each image must be equal to the network's input channel size. For example, for grayscale images, *C* must be equal to 1. For RGB color images, it must be equal to 3.

- *B* — Number of images in the array.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`ds` — Datastore
`datastore` object

Datastore, specified as a `datastore` object containing a collection of images. Each image must be a grayscale, RGB, or multichannel image. The function processes only the first column of the datastore, which must contain images and must be cell arrays or tables with multiple columns.

`roi` — Search region of interest
[*x y width height*] vector

Search region of interest, specified as a four-element vector of the form [*x y width height*]. The vector specifies the upper left corner and size of a region in pixels.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Threshold=0.25` sets the threshold to remove detections to `0.25`.

`Threshold` — Detection threshold
`0.5` (default) | scalar in the range [0, 1]

Detection threshold, specified as a scalar in the range [0, 1]. Detections that have scores less than this threshold value are removed. To reduce false positives, increase this value.

`SelectStrongest` — Select strongest bounding box
`true` (default) | `false`

Select the strongest bounding box for each detected object, specified as `true` or `false`.

- `true` — Returns the strongest bounding box per object. The method calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `selectStrongestBboxMulticlass` function is called as follows

```
selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                              RatioType="Union", ...
                              OverlapThreshold=0.5);
```

- `false` — Return all the detected bounding boxes. You can then write your own custom method to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of the form [*height width*]

Minimum region size, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest region containing the object.

By default, `MinSize` is 1-by-1.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of the form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest region containing the object.

By default, `MaxSize` is set to the height and width of the input image, `I`. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test image.

`MiniBatchSize` — Minimum batch size
`128` (default) | scalar

Minimum batch size, specified as a scalar value. Use the `MiniBatchSize` to process a large collection of image. Images are grouped into minibatches and processed as a batch, which can improve computational efficiency at the cost of increased memory demand. Decrease the size to use less memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified one of the following:

- `"auto"` — Automatically apply a number of optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available when using a GPU only. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel Computing Toolbox or a suitable GPU is not available, then the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

- `"none"` — Disable all acceleration.

The default option is `"auto"`. If `"auto"` is specified, MATLAB applies a number of compatible optimizations. If you use the `"auto"` option, MATLAB does not ever generate a MEX function.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits, but at the expense of an increased initial run time. Subsequent calls with compatible parameters are faster. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is only available for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is only available when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

`bboxes` — Location of objects detected

M-by-4 matrix | *B*-by-1 cell array

Location of objects detected within the input image or images, returned as an *M*-by-4 matrix or a *B*-by-1 cell array. *M* is the number of bounding boxes in an image, and *B* is the number of *M*-by-4 matrices when the input contains an array of images.

Each row of `bboxes` contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of that corresponding bounding box in pixels.

`scores` — Detection scores

M-by-1 numeric vector | *T*-by-1 cell array

Detection confidence scores for each bounding box, returned as one of these options:

- *M*-by-1 numeric vector — The input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — The input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

A higher score indicates higher confidence in the detection. The confidence score for each detection is a product of the corresponding objectness score and maximum class probability. The objectness score is the probability that the object in the bounding box belongs to a class in the image. The maximum class probability is the largest probability that a detected object in the bounding box belongs to a particular class.

`labels` — Labels for bounding boxes

M-by-1 categorical vector | *T*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector if the input is a single test image.
- *T*-by-1 cell array if the input is an array of test images. *T* is the number of test images in the array. Each cell in the array contains a *M*-by-1 categorical vector containing the names of the object classes.

M is the number of bounding boxes detected in an image.

`detectionResults` — Detection results
3-column table

Detection results, returned as a 3-column table with variable names, *Boxes*, *Scores*, and *Labels*. The *Boxes* column contains *M*-by-4 matrices, of *M* bounding boxes for the objects found in the image. Each row contains a bounding box as a 4-element vector in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size in pixels of the bounding box in the corresponding image.

`info` — Class probabilities and objectness scores
structure array

Class probabilities and objectness scores of the detections, returned as a structure array with these fields.

- `ClassProbabilities` — Class probabilities for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the class probabilities as an *M*-by-*N* numeric matrix. *M* is the number of bounding boxes and *N* is the number of classes. Each class probability is a numeric scalar, indicating the probability that the detected object in the bounding box belongs to a class in the image.
- `ObjectnessScores` — Objectness scores for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the objectness score for each bounding box as an *M*-by-1 numeric vector. *M* is the number of bounding boxes. Each objectness score is a numeric scalar, indicating the probability that the bounding box contains an object belonging to one of the classes in the image.

More About

Data Preprocessing

By default, the `detect` function preprocesses the test image for object detection by:

- Resizing it to a nearest possible image size used for training the YOLO v2 network. The function determines the nearest possible image size from the `TrainingImageSize` property of the `yolov2ObjectDetector` object.
- Normalizing its pixel values to lie in same range as that of the images used to train the YOLO v2 object detector. For example, if the detector was trained on `uint8` images, the test image must also have pixel values in the range [0, 255]. Otherwise, use the `im2uint8` or `rescale` function to rescale the pixel values in the test image.

Version History

Introduced in R2019a

R2024a: Option to return class probabilities and objectness scores

Specify the `info` output argument to return information about the class probability and objectness score for each detection.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MiniBatchSize`, and `MaxSize` name-value arguments are supported.
- The `MiniBatchSize` name-value argument is not supported for code generation when the image input `I` is an *H*-by-*W*-by-*C*-by-*B* numeric array of images. All images will processed in one batch.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `roi` argument to the `detect` method must be a codegen constant (`coder.const()`) and a 1-by-4 vector.
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize` name-value arguments are supported.
- The height, width, channel, and batch size of the input image must be fixed size.
- The minimum batch size value passed to detect method must be fixed size.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

Apps

Image Labeler | Video Labeler

Functions

`selectStrongestBboxMulticlass` | `trainYOLOv2ObjectDetector` | `evaluateObjectDetection`

Objects

`yolov2ObjectDetector` | `boxLabelDatastore`

Topics

“Create Custom YOLO v2 Object Detection Network”
“Object Detection Using YOLO v2 Deep Learning”
“Multiclass Object Detection Using YOLO v2 Deep Learning”

“Estimate Anchor Boxes from Training Data”
“Code Generation for Object Detection by Using YOLO v2”
“Getting Started with YOLO v2”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Datastores for Deep Learning” (Deep Learning Toolbox)

detect

Detect objects using YOLOX object detector

Syntax

```
bboxes = detect(detector,I)
[bboxes,scores] = detect(detector,I)
[bboxes,scores,labels] = detect(detector,I)
[bboxes,scores,labels,info] = detect(detector,I)

detectionResults = detect(detector,ds)

[___] = detect(___,roi)
[___] = detect(___,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`bboxes = detect(detector,I)` detects objects within a single image or a batch of images, `I`, using a YOLOX object detector, `detector`. The `detect` function returns the locations of objects detected in the input image as a set of bounding boxes.

Note This functionality requires Deep Learning Toolbox.

`[bboxes,scores] = detect(detector,I)` also returns the class-specific confidence scores for each bounding box.

`[bboxes,scores,labels] = detect(detector,I)` returns a categorical array of labels assigned to the bounding boxes. You must define the labels for object classes during training.

`[bboxes,scores,labels,info] = detect(detector,I)` also returns information about the class probabilities and objectness scores for each detection.

`detectionResults = detect(detector,ds)` returns object detection predictions within all the images returned by the `read` function of the input datastore `ds`, as a table.

`[___] = detect(___,roi)` detects objects within the rectangular search region `roi`, in addition to any combination of arguments from previous syntaxes.

`[___] = detect(___,Name=Value)` specifies options using one or more name-value arguments, in addition to any combination of arguments from previous syntaxes.. For example, `Threshold=0.25` specifies a detection threshold of 0.25.

Examples

Detect Objects Using Pretrained YOLOX Object Detector

Specify the name of a pretrained YOLOX deep learning network.

```
name = "tiny-coco";
```

Create YOLOX object detector by using the pretrained YOLOX network.

```
detector = yoloxObjectDetector(name);
```

Detect objects in a test image by using the pretrained YOLOX object detector.

```
img = imread("tima.png");
[bboxes,scores,labels] = detect(detector,img,Threshold=0.6)
```

```
bboxes = 1×4 single row vector

  185.1392  255.8597  119.6875  217.3186


scores = single

0.7775

labels = categorical
     cat
```

Display the detection results.

```
detectedImg = insertObjectAnnotation(img,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects in Image Datastore Using YOLOX Detector

Load a pretrained YOLOX object detector.

```
detector = yoloxObjectDetector("small-coco");
```

Specify the location of test data, then create an image datastore that reads the test data.

```
location = fullfile(matlabroot,"toolbox","vision","visiondata","vehicles");
imds = imageDatastore(location);
```

Detect objects in the datastore. Set the `Threshold` name-value argument to `0.4` and `MiniBatchSize` name-value argument to `32`.

```
detectionResults = detect(detector,imds,Threshold=0.4,MiniBatchSize=32);
```

Read an image from the test data set and extract the corresponding detection results.

```
num = 20;
I = readimage(imds,num);
bboxes = detectionResults.Boxes{num};
labels = detectionResults.Labels{num};
scores = detectionResults.Scores{num}
```

```
scores = 5×1 single column vector

    0.6125
    0.7789
    0.7595
    0.6787
    0.4322
```

Perform non-maximal suppression to select strongest bounding boxes from the overlapping clusters. Set the `OverlapThreshold` name-value argument to `0.5`.

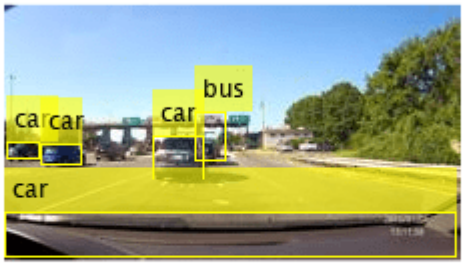
```
[bboxes,scores,labels] = selectStrongestBboxMulticlass(bboxes, ...
    scores,labels,OverlapThreshold=0.5);
```

Display the detection results.

```
results = table(bboxes,labels,scores)
```

```
results=5×3 table
     2.0755     69.2511     16.8517     9.0757    car    0.6125
    19.2191     70.2047     21.2572    10.8475    car    0.7789
    75.1651     65.7733     25.7689    23.2266    car    0.7595
    96.4793     54.2146     16.1751    24.6539    bus    0.6787
          1    104.9146    225.5654    22.6630    car    0.4322
```

```
detectedImg = insertObjectAnnotation(I,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Detect Objects Within ROI by Using YOLOX Detector

Load a pretrained YOLOX object detector.

```
detector = yoloxObjectDetector("small-coco");
```

Read a test image.

```
img = imread("aruba.png");
```

Specify a region of interest (ROI) within the test image.

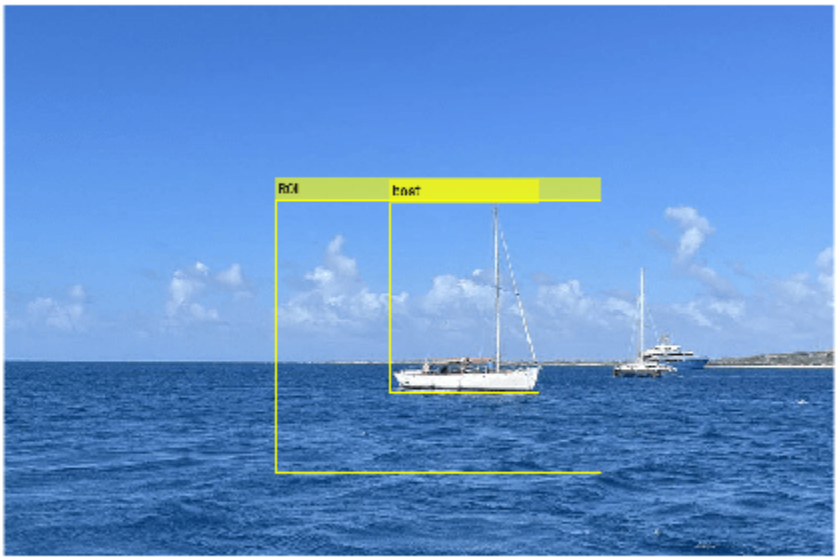
```
roiBox = [250 180 300 250];
```

Detect objects within the specified ROI.

```
[bboxes,scores,labels] = detect(detector,img,roiBox,Threshold=0.55);
```

Display the ROI and the detection results.

```
img = insertObjectAnnotation(img,"Rectangle",roiBox,"ROI",AnnotationColor="yellow");
detectedImg = insertObjectAnnotation(img,"Rectangle",bboxes,labels);
figure
imshow(detectedImg)
```



Input Arguments

`detector` — YOLOX object detector

`yoloxObjectDetector` object

YOLOX object detector, specified as a `yoloxObjectDetector` object.

`I` — Test images

H-by-*W*-by-3 numeric array | *H*-by-*W*-by-3-by-*B* numeric array

Test images, specified as a numeric array of size *H*-by-*W*-by-3 or *H*-by-*W*-by-3-by-*B*. You must specify real and nonsparse RGB or grayscale images with 3 channels.

- *H* — Height of the input images.
- *W* — Width of the input images.
- *B* — Number of test images in the batch. The `detect` function computes the object detection results for each test image in the batch.

Note The channel size of each image must be equal to the input channel size of the network. If you have grayscale test images, `detector` must be a YOLOX network that was trained on 3-channel grayscale images. Convert grayscale images to 3-channel images before training.

When the test image size does not match the network input size, the detector resizes the input image to the value of the `InputSize` property of `detector`, unless you specify `AutoResize` as `false`.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single`

`ds` — Datastore of test images
`ImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Datastore of test images, specified as an `imageDatastore` object, `CombinedDatastore` object, or `TransformedDatastore` object containing full filenames of the test images. The images in the datastore must be grayscale or RGB images.

`roi` — Region of interest to search
vector of form [*x y width height*]

Region of interest (ROI) to search, specified as a vector of form [*x y width height*]. The vector specifies the upper-left corner and size of a region, in pixels. If the input data is a datastore, the `detect` function applies the same ROI to every image.

Note You can specify the ROI to search only when the `detect` function automatically resizes the input test images to the network input size. To use `roi`, reset `AutoResize` to its default value.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detect(detector,I,Threshold=0.25)` specifies a detection threshold of 0.25.

`Threshold` — Detection threshold
`0.25` (default) | scalar in the range `[0, 1]`

Detection threshold, specified as a scalar in the range `[0, 1]`. The function removes detections that have scores less than this threshold value. To reduce false positives, increase this value at the possible expense of missing some objects.

`SelectStrongest` — Strongest bounding box selection
`true` or `1` (default) | `false` or `0`

Strongest bounding box selection for each detected object, specified as a numeric or logical `1` (`true`) or `0` (`false`).

- `true` — Return the strongest bounding box for each object. The `detect` function calls the `selectStrongestBboxMulticlass` function, which uses nonmaximal suppression to eliminate overlapping bounding boxes based on their confidence scores.

  By default, the `detect` function uses this call to the `selectStrongestBboxMulticlass`.

  ```
  selectStrongestBboxMulticlass(bboxes,scores,labels, ...
                                RatioType="Union", ...
                                OverlapThreshold=0.45);
  ```

- `false` — Return all the detected bounding boxes. You can write a custom function to eliminate overlapping bounding boxes.

`MinSize` — Minimum region size
`[1 1]` (default) | vector of form [*height width*]

Minimum region size containing an object, specified as a vector of the form [*height width*]. Units are in pixels. The minimum region size defines the size of the smallest object that can be detected by the trained network. When the minimum size is known, you can reduce computation time by setting `MinSize` to that value.

`MaxSize` — Maximum region size
`size(I)` (default) | vector of form [*height width*]

Maximum region size, specified as a vector of the form [*height width*]. Units are in pixels. The maximum region size defines the size of the largest object that can be detected by the trained network.

By default, `MaxSize` is set to the height and width of the input image `I`. To reduce computation time, set this value to the known maximum region size for the objects that can be detected in the input test image.

`MiniBatchSize` — Minimum batch size
`128` (default) | positive integer

Minimum batch size, specified as a positive integer. Adjust the `MiniBatchSize` value to help process a large collection of images. The `detect` function groups images into minibatches of the specified size and processes them as a batch, which can improve computation efficiency at the cost of increased memory demand. Decrease the minibatch size to use less memory.

`AutoResize` — Automatic resizing of input images
`true` or `1` (default) | `false` or `0`

Automatic resizing of input images to preserve the aspect ratio, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `AutoResize` is set to `1` (or `true`), the `detect` function resizes images to the nearest `InputSize` and the aspect ratio is preserved. Set `AutoResize` to logical `false` or `0` when performing image tiling-based training or inference at full test image size.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as one of these values:

- `"auto"` — Use a GPU if Parallel Computing Toolbox is installed and a supported GPU device is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

`Acceleration` — Performance optimization
`"auto"` (default) | `"mex"` | `"none"`

Performance optimization, specified as one of these options:

- `"auto"` — Automatically apply a number of compatible optimizations suitable for the input network and hardware resource.
- `"mex"` — Compile and execute a MEX function. This option is available only when using a GPU. Using a GPU requires Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If Parallel

Computing Toolbox or a suitable GPU is not available, then the `detect` function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

- `"none"` — Disable all acceleration.

Using the `Acceleration` options `"auto"` and `"mex"` can offer performance benefits on subsequent calls with compatible parameters, at the expense of an increased initial run time. Use performance optimization when you plan to call the function multiple times using new input data.

The `"mex"` option generates and executes a MEX function based on the network and parameters used in the function call. You can have several MEX functions associated with a single network at one time. Clearing the network variable also clears any MEX functions associated with that network.

The `"mex"` option is available only for input data specified as a numeric array, cell array of numeric arrays, table, or image datastore. No other types of datastore support the `"mex"` option.

The `"mex"` option is available only when you are using a GPU. You must also have a C/C++ compiler installed. For setup instructions, see “Set Up Compiler” (GPU Coder).

`"mex"` acceleration does not support all layers. For a list of supported layers, see “Supported Layers” (GPU Coder).

Output Arguments

`bboxes` — Locations of objects detected

M-by-4 matrix | *B*-by-1 cell array

Locations of objects detected within the input image or images, returned as one of these options:

- *M*-by-4 matrix — The input is a single test image. *M* is the number of bounding boxes detected in an image.
- *B*-by-1 cell array — The input is a batch of images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-4 matrix specifying the detected bounding boxes.

`scores` — Detection confidence scores

M-by-1 numeric vector | *B*-by-1 cell array

Detection confidence scores for each bounding box, returned as one of these options:

- *M*-by-1 numeric vector — The input is a single test image. *M* is the number of bounding boxes detected in the image.
- *B*-by-1 cell array — The input is a batch of test images, where *B* is the number of test images in the batch. Each cell in the array contains an *M*-element row vector, where each element indicates the detection score for a bounding box in the corresponding image.

A higher score indicates higher confidence in the detection. The confidence score for each detection is a product of the corresponding objectness score and maximum class probability. The objectness score is the probability that the object in the bounding box belongs to a class in the image. The maximum class probability is the largest probability that a detected object in the bounding box belongs to a particular class.

`labels` — Labels for bounding boxes

M-by-1 categorical vector | *B*-by-1 cell array

Labels for bounding boxes, returned as one of these options:

- *M*-by-1 categorical vector — The input is a single test image. *M* is the number of bounding boxes detected in an image.
- *B*-by-1 cell array — The input is an array of test images. *B* is the number of test images in the batch. Each cell in the array contains an *M*-by-1 categorical vector containing the names of the object classes.

`detectionResults` — Detection results

three-column table

Detection results when the input is a datastore of test images, `ds`, returned as a table with these columns:

`bboxes`	`scores`	`labels`
Predicted bounding boxes, defined in spatial coordinates as an *M*-by-4 numeric matrix with rows of the form [*x y w h*], where: • *M* is the number of axis-aligned rectangles. • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.	Class-specific confidence scores in the range `[0, 1]` for each bounding box, returned as an *M*-by-1 numeric vector.	Predicted object labels assigned to bounding boxes, returned as an *M*-by-1 categorical vector. All categorical data returned by the datastore must contain the same categories.

`info` — Class probabilities and objectness scores

structure array

Class probabilities and objectness scores of the detections, returned as a structure array with these fields.

- `ClassProbabilities` — Class probabilities for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the class probabilities as an *M*-by-*N* numeric matrix. *M* is the number of bounding boxes and *N* is the number of classes. Each class probability is a numeric scalar, indicating the probability that the detected object in the bounding box belongs to a class in the image.
- `ObjectnessScores` — Objectness scores for each of the detections, returned as a *B*-by-1 cell array. *B* is the number of images in the input batch of images, `I`. Each cell in the array contains the objectness score for each bounding box as an *M*-by-1 numeric vector. *M* is the number of bounding boxes. Each objectness score is a numeric scalar, indicating the probability that the bounding box contains an object belonging to one of the classes in the image.

Version History

Introduced in R2023b

R2024a: Option to return class probabilities and objectness scores

Specify the `info` output argument to return information about the class probability and objectness score for each detection.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- To prepare a `yoloxObjectDetector` object for CPU code generation, use `vision.loadYOLOXObjectDetector`.
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- The `AutoResize` argument to the `detect` method must be a code generation constant (`coder.const()`).
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, `MiniBatchSize`, and `AutoResize` name-value pairs for `detect` are supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- To prepare a `yoloxObjectDetector` object for GPU code generation, use `vision.loadYOLOXObjectDetector`.
- The `roi` argument to the `detect` method must be a code generation constant (`coder.const()`) and a 1x4 vector.
- The `AutoResize` argument to the `detect` method must be a code generation constant (`coder.const()`).
- Only the `Threshold`, `SelectStrongest`, `MinSize`, `MaxSize`, and `MiniBatchSize`, and `AutoResize` name-value pairs for `detect` are supported.

See Also

`yoloxObjectDetector` | `trainYOLOXObjectDetector` | `forward` | `imageDatastore` | `CombinedDatastore` | `TransformedDatastore` | `evaluateObjectDetection`

Topics

“Detect Defects on Printed Circuit Boards Using YOLOX Network”
“Getting Started with YOLOX for Object Detection”
“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Datastores for Deep Learning” (Deep Learning Toolbox)

classifyRegions

(Not recommended) Classify objects in image regions using R-CNN object detector

Note The `classifyRegions` function and R-CNN object detectors are not recommended. Use a different type of object detector instead. For more information, see “Version History”.

Syntax

```
[labels,scores] = classifyRegions(detector,I,rois)
[labels,scores,allScores] = classifyRegions(detector,I,rois)
[___] = classifyRegions(___,Name=Value)
```

Description

`[labels,scores] = classifyRegions(detector,I,rois)` classifies objects within the regions of interest of image `I`, using an R-CNN (regions with convolutional neural networks) object detector. For each region, `classifyRegions` returns the class label with the corresponding highest classification score.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[labels,scores,allScores] = classifyRegions(detector,I,rois)` also returns all the classification scores of each region. The scores are returned in an *M*-by-*N* matrix of *M* regions and *N* class labels.

`[___] = classifyRegions(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `ExecutionEnvironment="cpu"` classifies objects within image regions using only the CPU hardware.

Examples

Classify Multiple Image Regions

Load a pretrained detector.

```
load('rcnnStopSigns.mat','rcnn')
```

Read the test image.

```
img = imread('stopSignTest.jpg');
```

Specify multiple regions to classify within the test image.

```
rois = [416   143    33    27
        347   168    36    54];
```

Classify regions.

```
[labels,scores] = classifyRegions(rcnn,img,rois);
detectedImg = insertObjectAnnotation(img,'rectangle',rois,cellstr(labels));
figure
imshow(detectedImg)
```



Input Arguments

`detector` — R-CNN object detector

`rcnnObjectDetector` object

R-CNN object detector, specified as an `rcnnObjectDetector` object. To create this object, call the `trainRCNNObjectDetector` function with training data as input.

`I` — Input image

grayscale image | RGB image

Input image, specified as a real, nonsparse, grayscale or RGB image.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`rois` — Regions of interest
M-by-4 matrix

Regions of interest within the image, specified as an *M*-by-4 matrix defining *M* rectangular regions. Each row contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of a region in pixels.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ExecutionEnvironment="cpu"` classifies objects within image regions using only the CPU hardware.

`MiniBatchSize` — Size of smaller batches for R-CNN data processing
`128` (default) | integer

Size of smaller batches for R-CNN data processing, specified as an integer. Larger batch sizes lead to faster processing but take up more memory.

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource used to classify image regions, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

Output Arguments

`labels` — Classification labels of regions
M-by-1 categorical array

Classification labels of regions, returned as an *M*-by-1 categorical array. *M* is the number of regions of interest in `rois`. Each class name in `labels` corresponds to a classification score in `scores` and a region of interest in `rois`. `classifyRegions` obtains the class names from the input `detector`.

`scores` — Highest classification score per region
M-by-1 vector of values in the range [0, 1]

Highest classification score per region, returned as an *M*-by-1 vector of values in the range [0, 1]. *M* is the number of regions of interest in `rois`. Each classification score in `scores` corresponds to a class name in `labels` and a region of interest in `rois`. A higher score indicates higher confidence in the classification.

`allScores` — All classification scores per region
M-by-*N* matrix of values in the range [0, 1]

All classification scores per region, returned as an *M*-by-*N* matrix of values in the range [0, 1]. *M* is the number of regions in `rois`. *N* is the number of class names stored in the input `detector`. Each row of classification scores in `allscores` corresponds to a region of interest in `rois`. A higher score indicates higher confidence in the classification.

Version History

Introduced in R2016b

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainRCNNObjectDetector`

Objects
`rcnnObjectDetector`

classifyRegions

(Not recommended) Classify objects in image regions using Fast R-CNN object detector

Note The `classifyRegions` function and Fast R-CNN object detectors are not recommended. Use a different type of object detector instead. For more information, see "Version History".

Syntax

```
[labels,scores] = classifyRegions(detector,I,rois)
[labels,scores,allScores] = classifyRegions(detector,I,rois)
[___] = classifyRegions(___,'ExecutionEnvironment',resource)
```

Description

`[labels,scores] = classifyRegions(detector,I,rois)` classifies objects within the regions of interest of image `I`, using a Fast R-CNN (regions with convolutional neural networks) object detector. For each region, `classifyRegions` returns the class label with the corresponding highest classification score.

When using this function, use of a CUDA enabled NVIDIA GPU is highly recommended. The GPU reduces computation time significantly. Usage of the GPU requires Parallel Computing Toolbox. For information about the supported compute capabilities, see "GPU Computing Requirements" (Parallel Computing Toolbox).

`[labels,scores,allScores] = classifyRegions(detector,I,rois)` also returns all the classification scores of each region. The scores are returned in an *M*-by-*N* matrix of *M* regions and *N* class labels.

`[___] = classifyRegions(___,'ExecutionEnvironment',resource)` specifies the hardware resource used to classify object within image regions: `"auto"`, `"cpu"`, or `"gpu"`. You can use this syntax with either of the preceding syntaxes.

Examples

Classify Image Regions Using Fast R-CNN

Configure a Fast R-CNN object detector and use it to classify objects within multiple regions of an image.

Load a `fastRCNNObjectDetector` object that is pretrained to detect stop signs.

```
data = load('rcnnStopSigns.mat','fastRCNN');
fastRCNN = data.fastRCNN;
```

Read in a test image containing a stop sign.

```
I = imread('stopSignTest.jpg');
figure
imshow(I)
```



Specify regions of interest to classify within the test image.

```
rois = [416   143    33    27
        347   168    36    54];
```

Classify the image regions and inspect the output labels and classification scores. The labels come from the `ClassNames` property of the detector.

```
[labels,scores] = classifyRegions(fastRCNN,I,rois)
```

```
labels = 2×1 categorical
     stopSign
     Background


scores = 2×1 single column vector

    0.9969
    1.0000
```

The detector has high confidence in the classifications. Display the classified regions on the test image.

```
detectedI = insertObjectAnnotation(I,'rectangle',rois,cellstr(labels));

figure
imshow(detectedI)
```



Input Arguments

`detector` — Fast R-CNN object detector

`fastRCNNObjectDetector` object

Fast R-CNN object detector, specified as a `fastRCNNObjectDetector` object. To create this object, call the `trainFastRCNNObjectDetector` function with training data as input.

`I` — Input image

grayscale image | RGB image

Input image, specified as a real, nonsparse, grayscale or RGB image.

The detector is sensitive to the range of the input image. Therefore, ensure that the input image range is similar to the range of the images used to train the detector. For example, if the detector was trained on `uint8` images, rescale this input image to the range [0, 255] by using the `im2uint8` or `rescale` function. The size of this input image should be comparable to the sizes of the images used in training. If these sizes are very different, the detector has difficulty detecting objects because the scale of the objects in the input image differs from the scale of the objects the detector was trained to identify. Consider whether you used the `SmallestImageDimension` property during training to modify the size of training images.

Data Types: `uint8` | `uint16` | `int16` | `double` | `single` | `logical`

`rois` — Regions of interest

M-by-4 matrix

Regions of interest within the image, specified as an *M*-by-4 matrix defining *M* rectangular regions. Each row contains a four-element vector of the form [*x y width height*]. This vector specifies the upper left corner and size of a region in pixels.

`resource` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource used to classify image regions, specified as `"auto"`, `"gpu"`, or `"cpu"`.

- `"auto"` — Use a GPU if it is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see "GPU Computing Requirements" (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

Example: `classifyRegions(___,'ExecutionEnvironment',"cpu")`

Output Arguments

`labels` — Classification labels of regions
M-by-1 categorical array

Classification labels of regions, returned as an *M*-by-1 categorical array. *M* is the number of regions of interest in `rois`. Each class name in `labels` corresponds to a classification score in `scores` and a region of interest in `rois`. `classifyRegions` obtains the class names from the input `detector`.

`scores` — Highest classification score per region
M-by-1 vector of values in the range [0, 1]

Highest classification score per region, returned as an *M*-by-1 vector of values in the range [0, 1]. *M* is the number of regions of interest in `rois`. Each classification score in `scores` corresponds to a class name in `labels` and a region of interest in `rois`. A higher score indicates higher confidence in the classification.

`allScores` — All classification scores per region
M-by-*N* matrix of values in the range [0, 1]

All classification scores per region, returned as an *M*-by-*N* matrix of values in the range [0, 1]. *M* is the number of regions in `rois`. *N* is the number of class names stored in the input `detector`. Each row of classification scores in `allscores` corresponds to a region of interest in `rois`. A higher score indicates higher confidence in the classification.

Version History

Introduced in R2017a

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see "Choose an Object Detector".

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainFastRCNNObjectDetector`

Objects
`fastRCNNObjectDetector`

trainFastRCNNObjectDetector

(Not recommended) Train Fast R-CNN deep learning object detector

Note `trainFastRCNNObjectDetector` is not recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` detector. For more information, see “Version History”.

Syntax

```
trainedDetector = trainFastRCNNObjectDetector(trainingData,network,options)
[trainedDetector,info] = trainFastRCNNObjectDetector( ___ )

trainedDetector = trainFastRCNNObjectDetector(trainingData,checkpoint,
options)

trainedDetector = trainFastRCNNObjectDetector(trainingData,detector,options)

trainedDetector = trainFastRCNNObjectDetector( ___ ,'RegionProposalFcn',
proposalFcn)

trainedDetector = trainFastRCNNObjectDetector( ___ ,Name,Value)
```

Description

Train a Detector

`trainedDetector = trainFastRCNNObjectDetector(trainingData,network,options)` trains a Fast R-CNN (regions with convolution neural networks) object detector using deep learning. You can train a Fast R-CNN detector to detect multiple object classes.

This function requires that you have Deep Learning Toolbox. It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[trainedDetector,info] = trainFastRCNNObjectDetector( ___ )` also returns information on the training progress, such as training loss and accuracy, for each iteration.

Resume Training a Detector

`trainedDetector = trainFastRCNNObjectDetector(trainingData,checkpoint, options)` resumes training from a detector checkpoint.

Fine Tune a Detector

`trainedDetector = trainFastRCNNObjectDetector(trainingData,detector,options)` continues training a detector with additional training data or performs more training iterations to improve detector accuracy.

Custom Region Proposal

`trainedDetector = trainFastRCNNObjectDetector( ___ ,'RegionProposalFcn', proposalFcn)` optionally trains a custom region proposal function, `proposalFcn`, using any of the

previous inputs. If you do not specify a proposal function, then the function uses a variation of the Edge Boxes [2] algorithm.

Additional Properties

`trainedDetector = trainFastRCNNObjectDetector( ___ ,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments.

Examples

Train Fast R-CNN Stop Sign Detector

Load training data.

```
data = load('rcnnStopSigns.mat', 'stopSigns', 'fastRCNNLayers');
stopSigns = data.stopSigns;
fastRCNNLayers = data.fastRCNNLayers;
```

Add fullpath to image files.

```
stopSigns.imageFilename = fullfile(toolboxdir('vision'),'visiondata', ...
    stopSigns.imageFilename);
```

Randomly shuffle data for training.

```
rng(0);
shuffledIdx = randperm(height(stopSigns));
stopSigns = stopSigns(shuffledIdx,:);
```

Create an imageDatastore using the files from the table.

```
imds = imageDatastore(stopSigns.imageFilename);
```

Create a boxLabelDatastore using the label columns from the table.

```
blds = boxLabelDatastore(stopSigns(:,2:end));
```

Combine the datastores.

```
ds = combine(imds, blds);
```

The stop sign training images have different sizes. Preprocess the data to resize the image and boxes to a predefined size.

```
ds = transform(ds,@(data)preprocessData(data,[920 968 3]));
```

Set the network training options.

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize', 10, ...
    'InitialLearnRate', 1e-3, ...
    'MaxEpochs', 10, ...
    'CheckpointPath', tempdir);
```

Train the Fast R-CNN detector. Training can take a few minutes to complete.

```
frcnn = trainFastRCNNObjectDetector(ds, fastRCNNLayers , options, ...
    'NegativeOverlapRange', [0 0.1], ...
    'PositiveOverlapRange', [0.7 1]);

********************************************************************
Training a Fast R-CNN Object Detector for the following object classes:

* stopSign

--> Extracting region proposals from training datastore...done.

Training on single GPU.
|=========================================================================================================
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  Base Lea
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |      Rate
|=========================================================================================================
|       1 |           1 |       00:00:29 |       0.3787 |       93.59% |         0.96 |
|      10 |          10 |       00:05:14 |       0.3032 |       98.52% |         0.95 |
|=========================================================================================================

Detector training complete.
********************************************************************
```

Test the Fast R-CNN detector on a test image.

```
img = imread('stopSignTest.jpg');
```

Run the detector.

```
[bbox, score, label] = detect(frcnn, img);
```

Display detection results.

```
detectedImg = insertObjectAnnotation(img,'rectangle',bbox,score);
figure
imshow(detectedImg)
```



Supporting Functions

```
function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to the targetSize.
scale = targetSize(1:2)./size(data{1},[1 2]);
data{1} = imresize(data{1},targetSize(1:2));
bboxes = round(data{2});
data{2} = bboxresize(bboxes,scale);
end
```

Input Arguments

`trainingData` — Labeled ground truth

datastore | table

Labeled ground truth, specified as a datastore or a table.

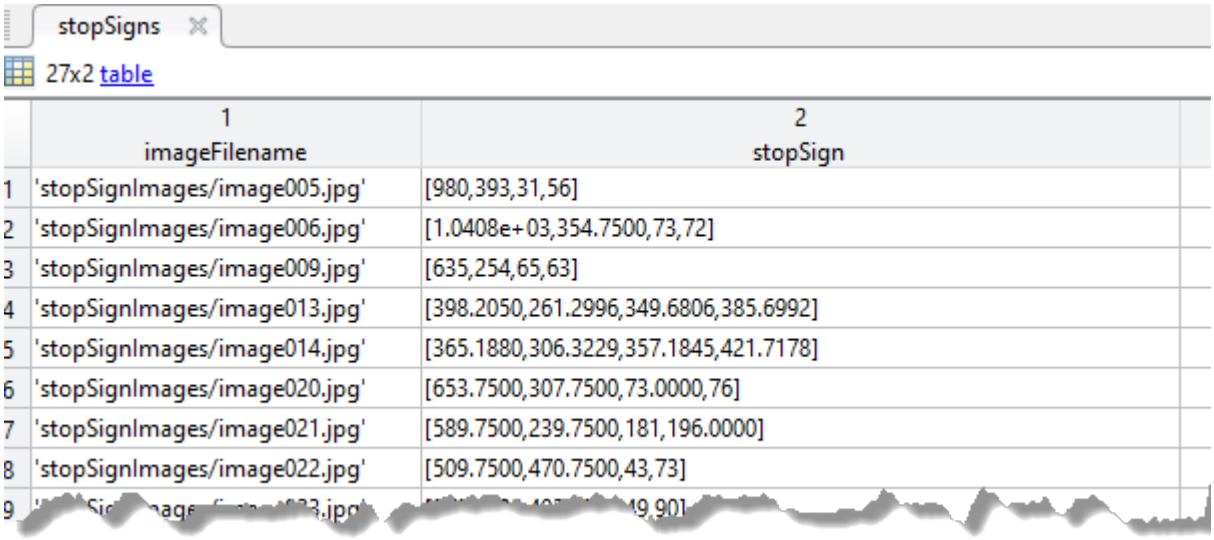
Each bounding box must be in the format [*x y width height*].

- If you use a datastore, your data must be set up so that calling the datastore with the `read` and `readall` functions returns a cell array or table with three columns. The second column must contain the bounding boxes, and the third column must contain the labels. The first column can contain image or point cloud data.

data	*boxes*	*labels*
The first column must be images.	*M*-by-4 matrices of bounding boxes of the form [*x*, *y*, *width*, *height*], where [x,y] represent the top-left coordinates of the bounding box.	The third column must be a cell array that contains *M*-by-1 categorical vectors containing object class names. All categorical data returned by the datastore must contain the same categories.

For more information, see “Datastores for Deep Learning” (Deep Learning Toolbox).

- If you use a table, the table must have two or more columns. The first column of the table must contain image file names with paths. The images must be grayscale or truecolor (RGB) and they can be in any format supported by `imread`. Each of the remaining columns must be a cell vector that contains *M*-by-4 matrices that represent a single object class, such as *vehicle*, *flower*, or *stop sign*. The columns contain 4-element double arrays of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. The format specifies the upper-left corner location and size of the bounding box in the corresponding image. To create a ground truth table, you can use the Image Labeler app or Video Labeler app. To create a table of training data from the generated ground truth, use the `objectDetectorTrainingData` function.



`network` — Network

`SeriesNetwork` object | array of `Layer` objects | `LayerGraph` object | network name

Network, specified as a `SeriesNetwork`, an array of `Layer` objects, a `layerGraph` object, or by the network name. The network is trained to classify the object classes defined in the `trainingData` table. The `SeriesNetwork`, `Layer`, and `layerGraph` objects are available in the Deep Learning Toolbox.

- When you specify the network as a `SeriesNetwork`, an array of `Layer` objects, or by the network name, the network is automatically transformed into a Fast R-CNN network by adding an ROI max pooling layer, and new classification and regression layers to support object detection. Additionally, the `GridSize` property of the ROI max pooling layer is set to the output size of the last max pooling layer in the network.

- The array of `Layer` objects must contain a classification layer that supports the number of object classes, plus a background class. Use this input type to customize the learning rates of each layer. An example of an array of `Layer` objects:

```
layers = [imageInputLayer([28 28 3])
        convolution2dLayer([5 5],10)
        reluLayer()
        fullyConnectedLayer(10)
        softmaxLayer()
        classificationLayer()];
```

- When you specify the network as `SeriesNetwork`, `Layer` array, or network by name, the weights for additional convolution and fully-connected layers that you add to create the network, are initialized to `'narrow-normal'`.
- The network name must be one of the following valid network names. You must also install the corresponding Add-on.

Network Name	Feature Extraction Layer Name	ROI Pooling Layer OutputSize	Description
`alexnet`	`'relu5'`	[6 6]	Last max pooling layer is replaced by ROI max pooling layer
`vgg16`	`'relu5_3'`	[7 7]	
`vgg19`	`'relu5_4'`		
`squeezenet`	`'fire5-concat'`	[14 14]	
`resnet18`	`'res4b_relu'`		ROI pooling layer is inserted after the feature extraction layer.
`resnet50`	`'activation_40_relu'`		
`resnet101`	`'res4b22_relu'`		
`googlenet`	`'inception_4d-output'`		
`mobilenetv2`	`'block_13_expand_relu'`		
`inceptionv3`	`'mixed7'`	[17 17]	
`inceptionresnetv2`	`'block17_20_ac'`		

- The `LayerGraph` object must be a valid Fast R-CNN object detection network. You can also use a `LayerGraph` object to train a custom Fast R-CNN network.

Tip If your network is a `DAGNetwork`, use the `layerGraph` function to convert the network to a `LayerGraph` object. Then, create a custom Fast R-CNN network as described by the “Create Fast R-CNN Object Detection Network” example.

See “Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN” to learn more about how to create a Fast R-CNN network.

`options` — Training options
`trainingOptions` output

Training options, returned by the `trainingOptions` function from the Deep Learning Toolbox. To specify solver and other options for network training, use `trainingOptions`.

Note `trainFastRCNNObjectDetector` does not support these training options:

- The `trainingOptions` `'once'` and `'every-epoch'` `Shuffle` options are not supported for combined datastore inputs.
- The `trainingOptions` `'parallel'` and `'multi-gpu'` `ExecutionEnvironment` options are not supported when you use a combined datastore input.
- Datastore inputs are not supported when you set the `DispatchInBackground` training option to `true`.

`checkpoint` — Saved detector checkpoint

`fastRCNNObjectDetector` object

Saved detector checkpoint, specified as a `fastRCNNObjectDetector` object. To periodically save a detector checkpoint during training, specify `CheckpointPath`. To control how frequently check points are saved see the `CheckPointFrequency` and `CheckPointFrequencyUnit` training options.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, if the `'CheckpointPath'` property of `options` is `'/tmp'`, load a checkpoint MAT file using:

```
data = load('/tmp/faster_rcnn_checkpoint__105__2016_11_18__14_25_08.mat');
```

The name of the MAT file includes the iteration number and timestamp of when the detector checkpoint was saved. The detector is saved in the `detector` variable of the file. Pass this file back into the `trainFastRCNNObjectDetector` function:

```
frcnn = trainFastRCNNObjectDetector(stopSigns,...
                            data.detector,options);
```

`detector` — Previously trained Fast R-CNN object detector

`fastRCNNObjectDetector` object

Previously trained Fast R-CNN object detector, specified as a `fastRCNNObjectDetector` object.

`proposalFcn` — Region proposal method

function handle

Region proposal method, specified as a function handle. If you do not specify a region proposal function, the function implements a variant of the Edge Boxes [2] algorithm. The function must have the form:

```
[bboxes,scores] = proposalFcn(I)
```

The input, `I`, is an image defined in the `trainingData` table. The function must return rectangular bound boxes, `bboxes`, in an *m*-by-4 array. Each row of `bboxes` contains a four-element vector, `[x,y,width,height]`. This vector specifies the upper-left corner and size of a bounding box in pixels. The function must also return a score for each bounding box in an *m*-by-1 vector. Higher score values indicate that the bounding box is more likely to contain an object. The scores are used to select the strongest *n* regions, where *n* is defined by the value of `NumStrongestRegions`.

Dependencies

If you do not specify a custom proposal function and you use a table for the input training data, the function uses a variation of the Edge Boxes algorithm. If you use a datastore for input training data for multichannel images, you must specify a custom region proposal function.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PositiveOverlapRange',[0.75 1]`

`PositiveOverlapRange` — Bounding box overlap ratios for positive training samples
`[0.5 1]` (default) | two-element vector

Bounding box overlap ratios for positive training samples, specified as a two-element vector. The vector contains values in the range [0,1]. Region proposals that overlap with ground truth bounding boxes within the specified range are used as positive training samples.

The overlap ratio used for both the `PositiveOverlapRange` and `NegativeOverlapRange` is defined as:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

A and B are bounding boxes.

`NegativeOverlapRange` — Bounding box overlap ratios for negative training samples
`[0.1 0.5]` (default) | two-element vector

Bounding box overlap ratios for negative training samples, specified as a two-element vector. The vector contains values in the range [0,1]. Region proposals that overlap with the ground truth bounding boxes within the specified range are used as negative training samples.

The overlap ratio used for both the `PositiveOverlapRange` and `NegativeOverlapRange` is defined as:

$$\frac{area(A \cap B)}{area(A \cup B)}$$

A and B are bounding boxes.

`NumStrongestRegions` — Maximum number of strongest region proposals
`2000` (default) | positive integer

Maximum number of strongest region proposals to use for generating training samples, specified as a positive integer. Reduce this value to speed up processing time at the cost of training accuracy. To use all region proposals, set this value to `Inf`.

`NumRegionsToSample` — Number of region proposals
128 (default) | integer

Number of region proposals to randomly sample from each training image, specified as an integer. Reduce the number of regions to sample to reduce memory usage and speed-up training. Reducing the value can also decrease training accuracy.

`SmallestImageDimension` — Length of smallest image dimension
`[]` (default) | positive integer

Length of smallest image dimension, either width or height, specified as a positive integer. Training images are resized such that the length of the shortest dimension is equal to the specified integer. By default, training images are not resized. Resizing training images helps reduce computational costs and memory used when training images are large. Typical values range from 400–600 pixels.

Dependencies

- The `SmallestImageDimension` property only supports table input training data. To resize the input data of a datastore input, use the `transform` function.

`FreezeBatchNormalization` — Frozen batch normalization
`true` (default) | `false`

Frozen batch normalization during training, specified as `true` or `false`. The value indicates whether the input layers to the network are frozen during training. Set this value to `true` if you are training with a small mini-batch size. Small batch sizes result in poor estimates of the batch mean and variance that is required for effective batch normalization.

If you do not specify a value for `FreezeBatchNormalization`, the function sets the property to

- `true` if the `MiniBatchSize` name-value argument for the `trainingOptions` function is less than 8.
- `false` if the `MiniBatchSize` name-value argument for the `trainingOptions` function is greater than or equal to 8.

You must specify a value for `FreezeBatchNormalization` to override this default behavior.

`ExperimentManager` — Detector training experiment monitoring
`'none'` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, and to produce training plots. For an example using this app, see “Train Object Detectors in Experiment Manager”.

Information monitored during training:

- Training loss at each iteration.
- Training accuracy at each iteration.
- Training root mean square error (RMSE) for the box regression layer.
- Learning rate at each iteration.

Output Arguments

`trainedDetector` — Trained Fast R-CNN object detector
`fastRCNNObjectDetector` object

Trained Fast R-CNN object detector, returned as a `fastRCNNObjectDetector` object.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with eight fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration is the mean squared error (MSE) calculated as the sum of localization error, confidence loss, and classification loss. For more information about the training loss function, see “YOLO v2 Training Loss” on page 3-1402.
- `TrainingAccuracy` — Training set accuracy at each iteration.
- `TrainingRMSE` — Training root mean squared error (RMSE) is the RMSE calculated from the training loss at each iteration.
- `BaseLearnRate` — Learning rate at each iteration.
- `ValidationLoss` — Validation loss at each iteration.
- `ValidationAccuracy` — Validation accuracy at each iteration.
- `ValidationRMSE` — Validation RMSE at each iteration.
- `FinalValidationLoss` — Final validation loss at end of the training.
- `FinalValidationRMSE` — Final validation RMSE at end of the training.

Each field is a numeric vector with one element per training iteration. Values that have not been calculated at a specific iteration are assigned as `NaN`. The struct contains `ValidationLoss`, `ValidationAccuracy`, `ValidationRMSE`, `FinalValidationLoss`, and `FinalValidationRMSE` fields only when `options` specifies validation data.

Tips

- To accelerate data preprocessing for training, `trainFastRCNNObjectDetector` automatically creates and uses a parallel pool based on your parallel preference settings. For more details about setting these preferences, see parallel preference settings. Using parallel computing preferences requires Parallel Computing Toolbox.
- VGG-16, VGG-19, ResNet-101, and Inception-ResNet-v2 are large models. Training with large images can produce "Out of Memory" errors. To mitigate these errors, try one or more of these options:
  - Reduce the size of your images by using the '`SmallestImageDimension`' argument.
  - Decrease the value of the '`NumRegionsToSample`' name-value argument value.
- This function supports transfer learning. When you input a `network` by name, such as `'resnet50'`, then the function automatically transforms the network into a valid Fast R-CNN network model based on the pretrained `resnet50` model. Alternatively, manually specify a custom Fast R-CNN network by using the `LayerGraph` extracted from a pretrained DAG network. For more details, see “Create Fast R-CNN Object Detection Network”.

- This table describes how to transform each named network into a Fast R-CNN network. The feature extraction layer name specifies which layer is processed by the ROI pooling layer. The ROI output size specifies the size of the feature maps output by the ROI pooling layer.

Network Name	Feature Extraction Layer Name	ROI Pooling Layer OutputSize	Description
`alexnet`	`'relu5'`	[6 6]	Last max pooling layer is replaced by ROI max pooling layer
`vgg16`	`'relu5_3'`	[7 7]	
`vgg19`	`'relu5_4'`		
`squeezenet`	`'fire5-concat'`	[14 14]	
`resnet18`	`'res4b_relu'`		ROI pooling layer is inserted after the feature extraction layer.
`resnet50`	`'activation_40_relu'`		
`resnet101`	`'res4b22_relu'`		
`googlenet`	`'inception_4d-output'`		
`mobilenetv2`	`'block_13_expand_relu'`		
`inceptionv3`	`'mixed7'`	[17 17]	
`inceptionresnetv2`	`'block17_20_ac'`		

  To modify and transform a network into a Fast R-CNN network, see “Design an R-CNN, Fast R-CNN, and a Faster R-CNN Model”.

- Use the `trainingOptions` function to enable or disable verbose printing.

Version History

Introduced in R2017a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024b: Not Recommended
Not recommended starting in R2024b

Starting in R2024b, R-CNN object detectors are no longer recommended. Instead, use a different type of object detector, such as a `yoloxObjectDetector` or `yolov4ObjectDetector` object. These object detectors are faster than R-CNN object detectors. For more information, see “Choose an Object Detector”.

References

[1] Girshick, Ross. "Fast R-CNN." *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[2] Zitnick, C. Lawrence, and Piotr Dollar. "Edge Boxes: Locating Object Proposals From Edges." *Computer Vision-ECCV 2014*. Springer International Publishing, 2014, pp. 391–405.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

To run in parallel, set `'UseParallel'` to `true` or enable this by default using the Computer Vision Toolbox preferences.

For more information, see “Parallel Computing Toolbox Support”.

See Also

Apps

Image Labeler | Video Labeler

Functions

`objectDetectorTrainingData` | `trainFasterRCNNObjectDetector` | `trainingOptions` | `estimateAnchorBoxes`

Objects

`Layer` | `boxLabelDatastore`

Topics

“Get Started with Object Detection Using Deep Learning”
“Choose an Object Detector”
“Anchor Boxes for Object Detection”
“Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Train Object Detectors in Experiment Manager”

peopleDetectorACF

Detect people using aggregate channel features

Syntax

```
detector = peopleDetectorACF
detector = peopleDetectorACF(name)
```

Description

`detector = peopleDetectorACF` returns a pretrained upright people detector using aggregate channel features (ACF). The detector is an `acfObjectDetector` object, and is trained using the INRIA person data set.

`detector = peopleDetectorACF(name)` returns a pretrained upright people detector based on the specified model `name`.

Examples

Detect People Using Aggregated Channel Features

Load the upright people detector.

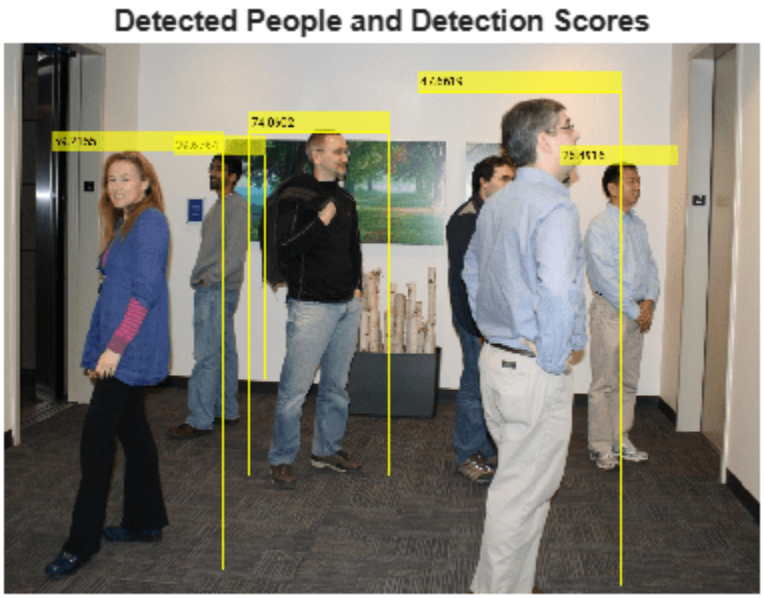
```
detector = peopleDetectorACF;
```

Read an image. Detect people in the image.

```
I = imread('visionteam1.jpg');
[bboxes,scores] = detect(detector,I);
```

Annotate detected people with bounding boxes and their detection scores.

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
title('Detected People and Detection Scores')
```



Input Arguments

`name` — ACF classification model

`'inria-100x41'` (default) | `'caltech-50x21'`

ACF classification model, specified as `'inria-100x41'` or `'caltech-50x21'`. The `'inria-100x41'` model was trained using the INRIA Person data set. The `'caltech-50x21'` model was trained using the Caltech Pedestrian data set.

Output Arguments

`detector` — Trained ACF-based object detector

`acfObjectDetector` object

Trained ACF-based object detector, returned as an `acfObjectDetector` object. The detector is trained to detect upright people in an image.

Version History

Introduced in R2017a

References

[1] Dollar, P., R. Appel, S. Belongie, and P. Perona. "Fast Feature Pyramids for Object Detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 36, Issue 8, 2014, pp. 1532–1545.

[2] Dollar P., C. Wojek, B. Shiele, and P. Perona. "Pedestrian Detection: An Evaluation of the State of the Art." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 34, Issue 4, 2012, pp. 743–761.

[3] Dollar, P., C., Wojek, B. Shiele, and P. Perona. "Pedestrian Detection: A Benchmark." *IEEE Conference on Computer Vision and Pattern Recognition*. 2009.

See Also

`acfObjectDetector` | `detect` | `trainACFObjectDetector` | `vision.PeopleDetector` | `vision.CascadeObjectDetector` | `selectStrongestBbox`

Topics

“Tracking Pedestrians from a Moving Car”
“Point Feature Types”

objectDetectorTrainingData

Create training data for an object detector

Syntax

```
[imds,blds] = objectDetectorTrainingData(gTruth)
[___,arrds] = objectDetectorTrainingData(gTruth)

trainingDataTable = objectDetectorTrainingData(gTruth)

[___] = objectDetectorTrainingData(gTruth,Name=Value)
```

Description

`[imds,blds] = objectDetectorTrainingData(gTruth)` creates an image datastore and a box label datastore training data from the specified ground truth. The function only selects images from the ground truth input that contain one or more annotated objects.

You can combine the image and box label datastores using `combine(imds,blds)` to create a datastore needed for training. Use the combined datastore with the training functions, such as `trainACFObjectDetector`, `trainYOLOv2ObjectDetector`, `trainYOLOv3ObjectDetector`, and `trainYOLOv4ObjectDetector`.

This function supports parallel computing using multiple MATLAB workers. Enable parallel computing using the “Computer Vision Toolbox Preferences” dialog.

`[___,arrds] = objectDetectorTrainingData(gTruth)` also returns a datastore `arrds`, that contains the attributes and sublabels associated with the labels.

`trainingDataTable = objectDetectorTrainingData(gTruth)` returns a table of training data from the specified ground truth. `gTruth` is an array of `groundTruth` objects. You can use the table to train an object detector using the Computer Vision Toolbox training functions.

`[___] = objectDetectorTrainingData(gTruth,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `Verbose=True` enables display to the workspace environment.

If you create the `groundTruth` objects in `gTruth` using a video file, a custom data source, or an `imageDatastore` object with different custom read functions, then you can specify any combination of name-value arguments. If you create the `groundTruth` objects from an image collection or image sequence data source, then you can specify only the `SamplingFactor` and the `LabelType` name-value arguments.

Examples

Train YOLO v2 Vehicle Detector

Train a vehicle detector based on a YOLO v2 network.

Add the folder containing training images to the workspace.

```
imageDir = fullfile(matlabroot,"toolbox","vision","visiondata","vehicles");
addpath(imageDir);
```

Load the vehicle ground truth data.

```
data = load("vehicleTrainingGroundTruth.mat");
gTruth = data.vehicleTrainingGroundTruth;
```

Create an image datastore and box label datastore using the ground truth object.

```
[imds,bxds] = objectDetectorTrainingData(gTruth);
```

Combine the datastores.

```
cds = combine(imds,bxds);
```

Load the detector containing the `dlnetwork` object for training.

```
load("yolov2VehicleDetectorNet.mat","net");
classes = "vehicle";
aboxes = [8 8; 32 48; 40 24; 72 48];
detector = yolov2ObjectDetector(net,classes,aboxes);
```

Configure training options.

```
options = trainingOptions("sgdm", ...
       InitialLearnRate=0.001, ...
       Verbose=true, ...
       MiniBatchSize=16, ...
       MaxEpochs=30, ...
       Shuffle="every-epoch", ...
       VerboseFrequency=10);
```

Train the detector.

```
[detector,info] = trainYOLOv2ObjectDetector(cds,detector,options);
```

```
*************************************************************************
Training a YOLO v2 Object Detector for the following object classes:

* vehicle

Training on single CPU.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:03 |         7.17 |         51.4 |          0.0010 |
|       1 |          10 |       00:00:09 |         1.61 |          2.6 |          0.0010 |
|       2 |          20 |       00:00:15 |         1.36 |          1.9 |          0.0010 |
|       2 |          30 |       00:00:20 |         1.23 |          1.5 |          0.0010 |
|       3 |          40 |       00:00:26 |         1.10 |          1.2 |          0.0010 |
|       3 |          50 |       00:00:32 |         1.09 |          1.2 |          0.0010 |
|       4 |          60 |       00:00:38 |         0.84 |          0.7 |          0.0010 |
|       4 |          70 |       00:00:44 |         0.87 |          0.8 |          0.0010 |
|       5 |          80 |       00:00:50 |         0.76 |          0.6 |          0.0010 |
|       5 |          90 |       00:00:56 |         0.80 |          0.6 |          0.0010 |
```

```
|       6 |         100 |       00:01:01 |         0.79 |          0.6 |          0.0010 |
|       7 |         110 |       00:01:07 |         0.58 |          0.3 |          0.0010 |
|       7 |         120 |       00:01:12 |         0.62 |          0.4 |          0.0010 |
|       8 |         130 |       00:01:18 |         0.64 |          0.4 |          0.0010 |
|       8 |         140 |       00:01:23 |         0.65 |          0.4 |          0.0010 |
|       9 |         150 |       00:01:29 |         0.57 |          0.3 |          0.0010 |
|       9 |         160 |       00:01:35 |         0.58 |          0.3 |          0.0010 |
|      10 |         170 |       00:01:40 |         0.62 |          0.4 |          0.0010 |
|      10 |         180 |       00:01:45 |         0.55 |          0.3 |          0.0010 |
|      11 |         190 |       00:01:50 |         0.60 |          0.4 |          0.0010 |
|      12 |         200 |       00:01:55 |         0.49 |          0.2 |          0.0010 |
|      12 |         210 |       00:02:00 |         0.56 |          0.3 |          0.0010 |
|      13 |         220 |       00:02:05 |         0.55 |          0.3 |          0.0010 |
|      13 |         230 |       00:02:10 |         0.48 |          0.2 |          0.0010 |
|      14 |         240 |       00:02:15 |         0.51 |          0.3 |          0.0010 |
|      14 |         250 |       00:02:20 |         0.48 |          0.2 |          0.0010 |
|      15 |         260 |       00:02:26 |         0.50 |          0.2 |          0.0010 |
|      15 |         270 |       00:02:31 |         0.59 |          0.4 |          0.0010 |
|      16 |         280 |       00:02:36 |         0.54 |          0.3 |          0.0010 |
|      17 |         290 |       00:02:41 |         0.48 |          0.2 |          0.0010 |
|      17 |         300 |       00:02:46 |         0.36 |          0.1 |          0.0010 |
|      18 |         310 |       00:02:52 |         0.50 |          0.2 |          0.0010 |
|      18 |         320 |       00:02:57 |         0.49 |          0.2 |          0.0010 |
|      19 |         330 |       00:03:02 |         0.44 |          0.2 |          0.0010 |
|      19 |         340 |       00:03:07 |         0.44 |          0.2 |          0.0010 |
|      20 |         350 |       00:03:12 |         0.44 |          0.2 |          0.0010 |
|      20 |         360 |       00:03:17 |         0.51 |          0.3 |          0.0010 |
|      21 |         370 |       00:03:22 |         0.48 |          0.2 |          0.0010 |
|      22 |         380 |       00:03:26 |         0.50 |          0.3 |          0.0010 |
|      22 |         390 |       00:03:31 |         0.52 |          0.3 |          0.0010 |
|      23 |         400 |       00:03:36 |         0.46 |          0.2 |          0.0010 |
|      23 |         410 |       00:03:41 |         0.37 |          0.1 |          0.0010 |
|      24 |         420 |       00:03:46 |         0.45 |          0.2 |          0.0010 |
|      24 |         430 |       00:03:51 |         0.39 |          0.2 |          0.0010 |
|      25 |         440 |       00:03:56 |         0.41 |          0.2 |          0.0010 |
|      25 |         450 |       00:04:01 |         0.36 |          0.1 |          0.0010 |
|      26 |         460 |       00:04:06 |         0.41 |          0.2 |          0.0010 |
|      27 |         470 |       00:04:11 |         0.48 |          0.2 |          0.0010 |
|      27 |         480 |       00:04:16 |         0.40 |          0.2 |          0.0010 |
|      28 |         490 |       00:04:21 |         0.44 |          0.2 |          0.0010 |
|      28 |         500 |       00:04:26 |         0.39 |          0.2 |          0.0010 |
|      29 |         510 |       00:04:31 |         0.25 |      6.1e-02 |          0.0010 |
|      29 |         520 |       00:04:37 |         0.33 |          0.1 |          0.0010 |
|      30 |         530 |       00:04:42 |         0.36 |          0.1 |          0.0010 |
|      30 |         540 |       00:04:47 |         0.33 |          0.1 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.
Detector training complete.
*********************************************************************
```

Read a test image.

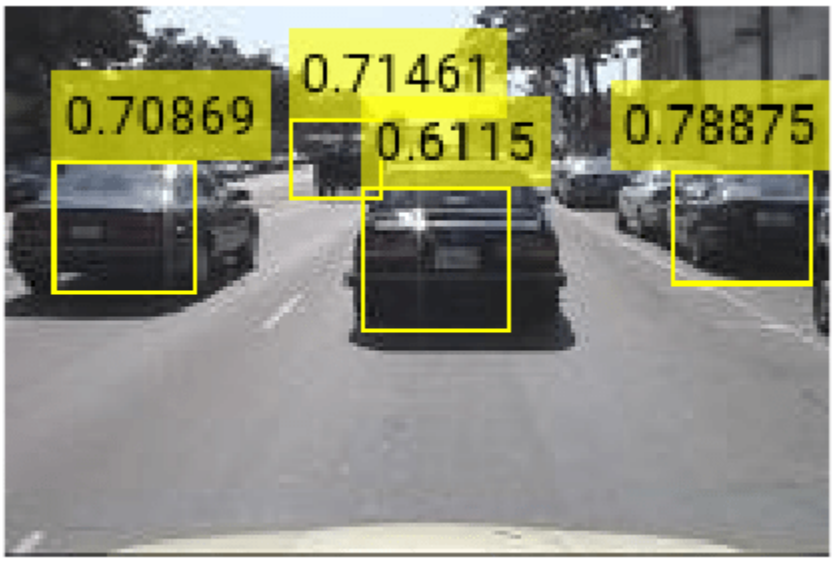
```
I = imread("detectcars.png");
```

Run the detector.

```
[bboxes,scores] = detect(detector,I);
```

Display the results.

```
if(~isempty(bboxes))
  I = insertObjectAnnotation(I,"rectangle",bboxes,scores);
end
figure
imshow(I)
```



Train ACF-Based Stop Sign Detector

Use training data to train an ACF-based object detector for stop signs

Add the folder containing images to the MATLAB® path.

```
imageDir = fullfile(matlabroot, 'toolbox', 'vision', 'visiondata', 'stopSignImages');
addpath(imageDir);
```

Load ground truth data, which contains data for stops signs and cars.

```
load('stopSignsAndCarsGroundTruth.mat','stopSignsAndCarsGroundTruth')
```

View the label definitions to see the label types in the ground truth.

```
stopSignsAndCarsGroundTruth.LabelDefinitions
```

```
ans=3×3 table
       Name          Type        Group
    ____________    _________    ________

    {'stopSign'}    Rectangle    {'None'}
    {'carRear' }    Rectangle    {'None'}
    {'carFront'}    Rectangle    {'None'}
```

Select the stop sign data for training.

```
stopSignGroundTruth = selectLabelsByName(stopSignsAndCarsGroundTruth,'stopSign');
```

Create the training data for a stop sign object detector.

```
trainingData = objectDetectorTrainingData(stopSignGroundTruth);
summary(trainingData)
```

```
trainingData: 41×2 table

Variables:

    imageFilename: cell array of character vectors
    stopSign: cell

Statistics for applicable variables:

                    NumMissing

    imageFilename        0
    stopSign             0
```

Train an ACF-based object detector.

```
acfDetector = trainACFObjectDetector(trainingData,'NegativeSamplesFactor',2);
```

```
ACF Object Detector Training
The training will take 4 stages. The model size is 34x31.
Sample positive examples(~100% Completed)
Compute approximation coefficients...Completed.
Compute aggregated channel features...Completed.
-------------------------------------------
Stage 1:
Sample negative examples(~100% Completed)
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 19 weak learners.
-------------------------------------------
Stage 2:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 20 weak learners.
-------------------------------------------
Stage 3:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 54 weak learners.
-------------------------------------------
Stage 4:
Sample negative examples(~100% Completed)
Found 84 new negative examples for training.
Compute aggregated channel features...Completed.
Train classifier with 42 positive examples and 84 negative examples...Completed.
The trained classifier has 61 weak learners.
```

```
-------------------------------------------
ACF object detector training is completed. Elapsed time is 20.3318 seconds.
```

Test the ACF-based detector on a sample image.

```
I = imread('stopSignTest.jpg');
bboxes = detect(acfDetector,I);
```

Display the detected object.

```
annotation = acfDetector.ModelName;
I = insertObjectAnnotation(I,'rectangle',bboxes,annotation);

figure
imshow(I)
```



Remove the image folder from the path.

```
rmpath(imageDir);
```

Read All Label Attributes from groundTruth

Load image locations, label definitions and label data.

```
data = load('labelsWithAttributes.mat');
images = fullfile(matlabroot,'toolbox','vision','visiondata','stopSignImages', data.imageFilename
```

Create a ground truth object.

```
dataSource = groundTruthDataSource(images);
gTruth = groundTruth(groundTruthDataSource(images), data.labeldefs, data.labelData);
```

Create an image datastore, box label datastore, and array datastore using the ground truth object.

```
[imds, blds, arrds] = objectDetectorTrainingData(gTruth);
```

Read all attributes.

```
readall(arrds)
```

```
ans=2×1 cell array
    {1×1 struct}
    {1×1 struct}
```

Input Arguments

`gTruth` — Ground truth data

scalar | array of `groundTruth` objects

Ground truth data, specified as a scalar or an array of `groundTruth` objects. You can create ground truth objects from existing ground truth data by using the `groundTruth` object.

If you use custom data sources in `groundTruth` with parallel computing enabled, then the reader function is expected to work with a pool of MATLAB workers to read images from the data source in parallel.

Note If you specify `gTruth` as an array of `groundTruth` objects, all label definitions must have the same label names.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: (`SamplingFactor=5`) sets the subsampling factor to `5`.

`SamplingFactor` — Factor for subsampling images

`"auto"` (default) | integer | vector of integers

Factor for subsampling images in the ground truth data source, specified as `"auto"`, an integer, or a vector of integers. For a sampling factor of *N*, the returned training data includes every *N*th image in the ground truth data source. The function ignores ground truth images with empty label data. To set the `SamplingFactor` with projected cuboid data, you must specify the `LabelType` name-value argument to `labelType.ProjectedCuboid`.

Use sampled data to reduce repeated data, such as a sequence of images with the same scene and labels. It can also help in reducing training time.

Value	Sampling Factor
`"auto"`	The function samples data sources with timestamps, such as a video, with a factor of `5`, and `1` for a collection of images.
Integer	Manually set the sampling factor to apply to all data.
Vector of integers	When you input an array of ground truth objects, the function uses the sampling factor specified by the corresponding vector element.

`LabelType` — Type of label to extract from ground truth data
`"labelType.Rectangle"` (default) | `"labelType.RotatedRectangle"` | `"labelType.ProjectedCuboid"` | character vector

Type of label to extract from ground truth data, specified as `"labelType.Rectangle"`, `"labelType.RotatedRectangle"`, or `"labelType.ProjectedCuboid"`. Use the type of label consistent with the type of object detector you want to train.

Note The `trainYOLOv2ObjectDetector` function does not support `"labelType.RotatedRectangle"`.

`WriteLocation` — Name of folder
`pwd` (current working folder) (default) | string scalar | character vector

Folder name to write extracted images to, specified as a string scalar or character vector. The specified folder must exist and have write permissions.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` , with different custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`ImageFormat` — Image file format
`"PNG"` (default) | string scalar | character vector

Image file format, specified as a string scalar or character vector. File formats must be supported by `imwrite`.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` with different custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`NamePrefix` — Prefix for output image file names
string scalar | character vector

Prefix for output image file names, specified as a string scalar or character vector. The image files are named as:

```
<name_prefix><source_number>_<image_number>.<image_format>
```

The default value uses the name of the data source that the images were extracted from, `strcat(sourceName,"_")`, for video and a custom data source, or `"datastore"`, for an image datastore.

This argument applies only for:

- `groundTruth` objects created using a video file or a custom data source.
- An array of `groundTruth` objects created using `imageDatastore` with different custom `read` functions.

The function ignores this argument when:

- The input `groundTruth` object was created from an image sequence data source.
- The array of input `groundTruth` objects all contain image datastores using the same custom `read` function.
- Any of the input `groundTruth` objects containing datastores, use the default `read` functions.

`Verbose` — Flag to display training progress
`true` (`1`) (default) | `false` (`0`)

Flag to display training progress at the MATLAB command line, specified as either `true` (`1`) or `false` (`0`). This property applies only for `groundTruth` objects created using a video file or a custom data source.

Output Arguments

`imds` — Image datastore
`imageDatastore` object

Image datastore, returned as an `imageDatastore` object containing images extracted from the `gTruth` objects. The images in `imds` contain at least one class of annotated labels. The function ignores images that are not annotated.

`blds` — Box label datastore
`boxLabelDatastore` object

Box label datastore, returned as a `boxLabelDatastore` object. The datastore contains categorical vectors for ROI label names and *M*-by-4 matrices of *M* bounding boxes. The locations and sizes of the bounding boxes are represented as double *M*-by-4 element vectors in the format [*x*,*y*,*width*,*height*].

`arrds` — Array datastore

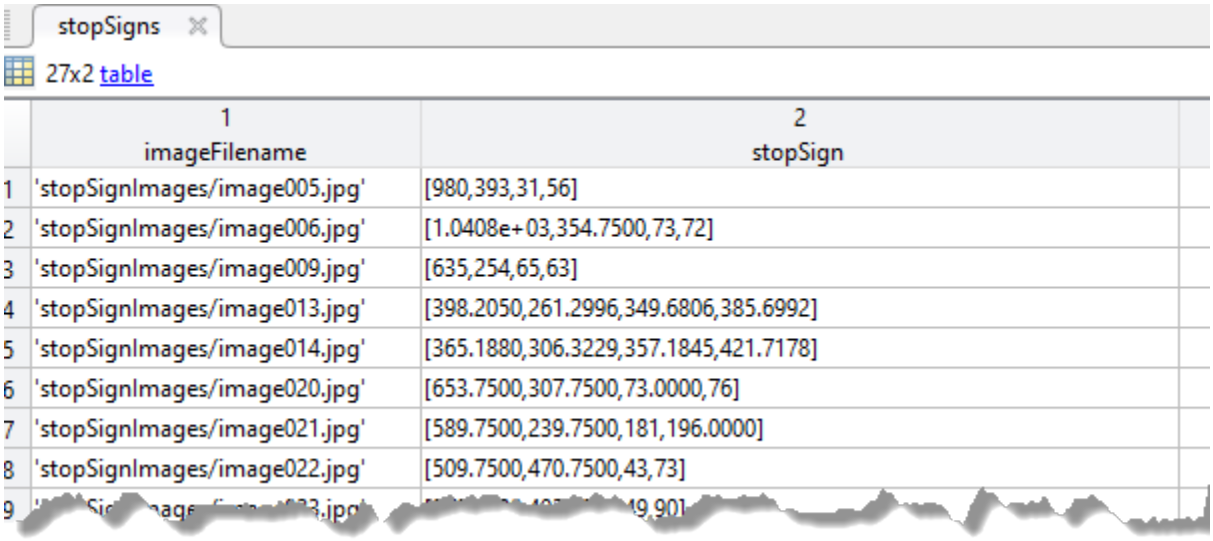
`struct` array

Array datastore, returned as a `struct` array. The fields of the `struct` contain the attributes and sublabel names for the corresponding labels in the box label datastore `blds`. The sublabel data is packaged into the `struct` with a `Position` field along with the fields that correspond to the sublabel attributes.

`trainingDataTable` — Training data table

table

Training data table, returned as a table with two or more columns. The first column of the table contains image file names with paths. The images can be grayscale or truecolor (RGB) and in any format supported by `imread`. Each of the remaining columns correspond to an ROI label and contains the locations of bounding boxes in the image (specified in the first column), for that label. The bounding boxes are specified as *M*-by-4 matrices of *M* bounding boxes in the format [*x*,*y*,*width*,*height*]. [*x*,*y*] specifies the upper-left corner location. To create a ground truth table, you can use the Image Labeler app or Video Labeler app.


stopSigns

27x2 table

	1 imageFilename	2 stopSign
1	'stopSignImages/image005.jpg'	[980,393,31,56]
2	'stopSignImages/image006.jpg'	[1.0408e+03,354.7500,73,72]
3	'stopSignImages/image009.jpg'	[635,254,65,63]
4	'stopSignImages/image013.jpg'	[398.2050,261.2996,349.6806,385.6992]
5	'stopSignImages/image014.jpg'	[365.1880,306.3229,357.1845,421.7178]
6	'stopSignImages/image020.jpg'	[653.7500,307.7500,73.0000,76]
7	'stopSignImages/image021.jpg'	[589.7500,239.7500,181,196.0000]
8	'stopSignImages/image022.jpg'	[509.7500,470.7500,43,73]
9		

The output table ignores any sublabel or attribute data present in the input `gTruth` object.

Version History

Introduced in R2017a

R2022b: Project cuboids from 3-D world coordinates to 2-D image coordinates

- Updated to support 3-D projected cuboid labels.
- Returns extracted attributes and sublabels as a third output. The attributes and sublabels are packaged as an array datastore.

See Also

Apps
Image Labeler | Video Labeler

Functions
`trainACFObjectDetector` | `trainYOLOv2ObjectDetector` | `trainYOLOv3ObjectDetector` | `trainYOLOv4ObjectDetector` | `estimateAnchorBoxes`

Objects
`groundTruth` | `acfObjectDetector` | `imageDatastore` | `boxLabelDatastore`

Topics
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Training Data for Object Detection and Semantic Segmentation”

estimateFisheyeParameters

Calibrate fisheye camera

Syntax

```
[fisheyeParams,imagesUsed,estimationErrors] = estimateFisheyeParameters(
imagePoints,worldPoints,imageSize)
[___] = estimateFisheyeParameters(___,Name=Value)
```

Description

`[fisheyeParams,imagesUsed,estimationErrors] = estimateFisheyeParameters(imagePoints,worldPoints,imageSize)` returns a `fisheyeParameters` object containing estimates for the intrinsic and extrinsic parameters of a fisheye camera. The function also returns the images you used to estimate the fisheye parameters and the standard estimation errors for the fisheye camera calibration.

`[___] = estimateFisheyeParameters(___,Name=Value)` specifies options using one or more name-value arguments in addition to the previous syntax. For example, `WorldUnits="cm"` set the world units to centimeters.

Examples

Fisheye Camera Calibration

Use calibration images to detect a checkerboard calibration pattern. Calibrate the camera using corners extracted from the pattern and visualize the results of the calibration.

Load calibration images using an image datastore.

```
calibrationImages = imageDatastore("calibImages");
imageFileNames = calibrationImages.Files;
```

Detect the calibration pattern from the images. The `PartialDetections` name-value argument is set to true by default to allow detection of partial checkerboards.

```
[imagePoints,patternDims,imagesUsed] = detectCheckerboardPoints(imageFileNames,HighDistortion=tru
```

Collect the calibration images in which checkerboards were detected.

```
usedImageFileNames = imageFileNames(imagesUsed);
images = imageDatastore(usedImageFileNames);
```

Generate world coordinates for the corners of the checkerboard squares.

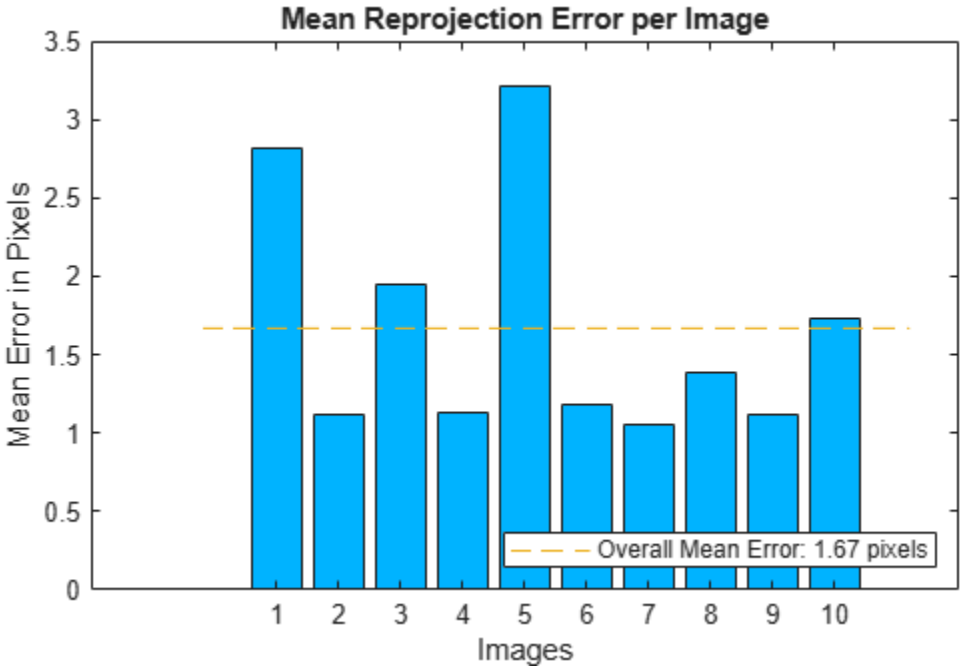
```
squareSize = 20; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Estimate the fisheye camera calibration parameters based on the image and world points.

```
I = readimage(images,10);
imageSize = [size(I,1) size(I,2)];
params = estimateFisheyeParameters(imagePoints,worldPoints,imageSize);
```

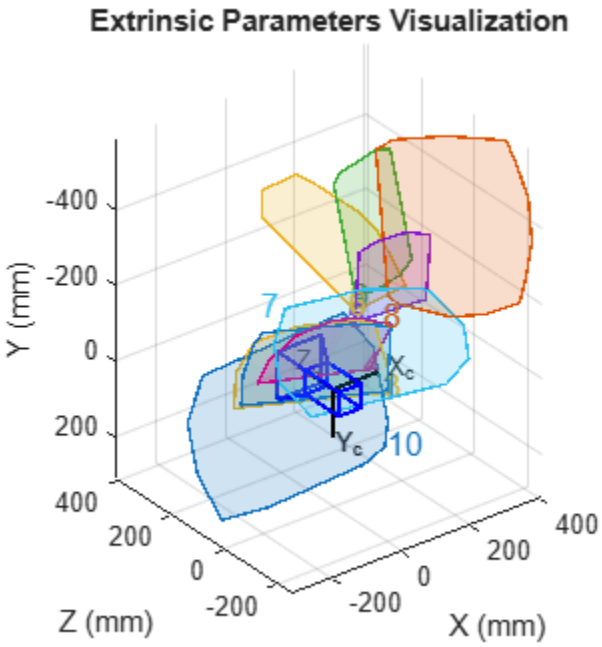
Visualize the calibration accuracy.

```
figure
showReprojectionErrors(params);
```



Visualize the camera extrinsics.

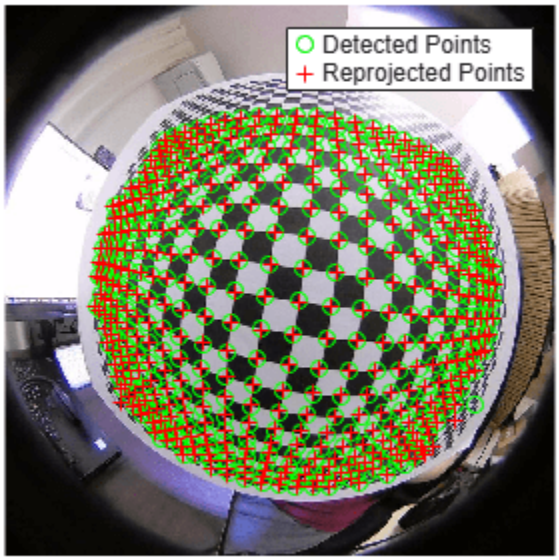
```
figure
showExtrinsics(params);
```




```
drawnow
```

Plot the detected and reprojected points.

```
figure
imshow(I);
hold on
plot(imagePoints(:,1,10),imagePoints(:,2,10),"go");
plot(params.ReprojectedPoints(:,1,10),params.ReprojectedPoints(:,2,10),"r+");
legend("Detected Points","Reprojected Points");
hold off
```



Input Arguments

`imagePoints` — Key points of calibration pattern

M-by-2-by-*numImages* array

Key points of calibration pattern, specified as an *M*-by-2-by-*numImages* array of [*x*,*y*] intrinsic image coordinates. The number of images, *numImages*, must be greater than 2. The number of keypoint coordinates in each pattern, *M*, must be greater than 3. To include partially detected patterns in the estimate, use `[NaN,NaN]` as *x*-*y* coordinates for missing keypoints.

Data Types: `single` | `double`

`worldPoints` — Key points of calibration pattern in world coordinates

M-by-2 matrix

Key points of calibration pattern in world coordinates, specified as an *M*-by-2 matrix of *M* [*x*,*y*] world coordinates. Because the pattern must be planar, the *z*-coordinates are zero.

Data Types: `single` | `double`

`imageSize` — Image size

`[mrows ncols]` vector

Image size, specified as an `[mrows ncols]` vector.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `WorldUnits="mm"` sets the world point units to millimeters.

`EstimateAlignment` — Estimate axes alignment
`false` or `0` (default) | `true` or `1`

Estimate the axes alignment, specified as `false` (`0`) or `true` (`1`). Set the value to `true` if the optical axis of the fisheye lens is not perpendicular to the image plane.

`WorldUnits` — World point units
`"mm"` (default) | character vector | string scalar

World point units, specified as a character vector or string scalar. This name-value argument is used to store the unit type and does not affect any calculations.

Output Arguments

`fisheyeParams` — Fisheye camera parameters
`fisheyeParameters` object

Fisheye camera parameters, returned as a `fisheyeParameters` object.

`imagesUsed` — Images used to estimate camera parameters
numImages-by-1 logical array

Images used to the estimate camera parameters, returned as a *numImages*-by-1 logical array. *numImages* corresponds to the number of images. A logical `true` value indicates the index of an image used to estimate the camera parameters.

`estimationErrors` — Standard errors of estimated parameters
`fisheyeCalibrationErrors` object

Standard errors of estimated parameters, returned as a `fisheyeCalibrationErrors` object.

Version History

Introduced in R2017b

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

References

[1] Scaramuzza, D., A. Martinelli, and R. Siegwart. "A Toolbox for Easy Calibrating Omnidirectional Cameras." *Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006)*. Beijing, China, October 7–15, 2006.

[2] Urban, S., J. Leitloff, and S. Hinz. "Improved Wide-Angle, Fisheye and Omnidirectional Camera Calibration." *ISPRS Journal of Photogrammetry and Remove Sensing*. Vol. 108, 2015, pp.72–79.

See Also

Objects
`fisheyeParameters` | `fisheyeIntrinsics` | `fisheyeCalibrationErrors` | `fisheyeIntrinsicsEstimationErrors` | `extrinsicsEstimationErrors`

Functions
`undistortFisheyePoints` | `detectCheckerboardPoints` | `generateCheckerboardPoints` | `showExtrinsics` | `showReprojectionErrors`

Topics
“Fisheye Calibration Basics”
“Stereo Fisheye Camera Calibration”
“Configure Monocular Fisheye Camera” (Automated Driving Toolbox)

undistortFisheyeImage

Correct fisheye image for lens distortion

Syntax

```
J = undistortFisheyeImage(I,intrinsics)
[J,camIntrinsics] = undistortFisheyeImage(I,intrinsics)
[___] = undistortFisheyeImage(___,interp)
[___] = undistortFisheyeImage(___,Name=Value)
```

Description

`J = undistortFisheyeImage(I,intrinsics)` removes lens distortion for image `I` and returns the result as image `J`.

`[J,camIntrinsics] = undistortFisheyeImage(I,intrinsics)` also returns a `cameraIntrinsics` object, which corresponds to a virtual pinhole camera.

`[___] = undistortFisheyeImage(___,interp)` specifies the interpolation method, `interp`, using the preceding syntaxes.

`[___] = undistortFisheyeImage(___,Name=Value)` specifies options using one or more name-value arguments in addition to any combination of arguments from previous syntaxes. For example, `"ScaleFactor"=2` increases the zoom in the camera view by a factor of 2.

Examples

Correct Fisheye Image for Lens Distortion

Remove lens distortion from a fisheye image by detecting a checkboard calibration pattern and calibrating the camera. Then, display the results.

Gather a set of checkerboard calibration images.

```
images = imageDatastore('calibrationImages');
```

Detect the calibration pattern from the images. The 'PartialDetections' Name-Value argument is set to true by default allowing detection of partial checkerboards.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files, 'HighDistortion', true);
```

Generate world coordinates for the corners of the checkerboard squares.

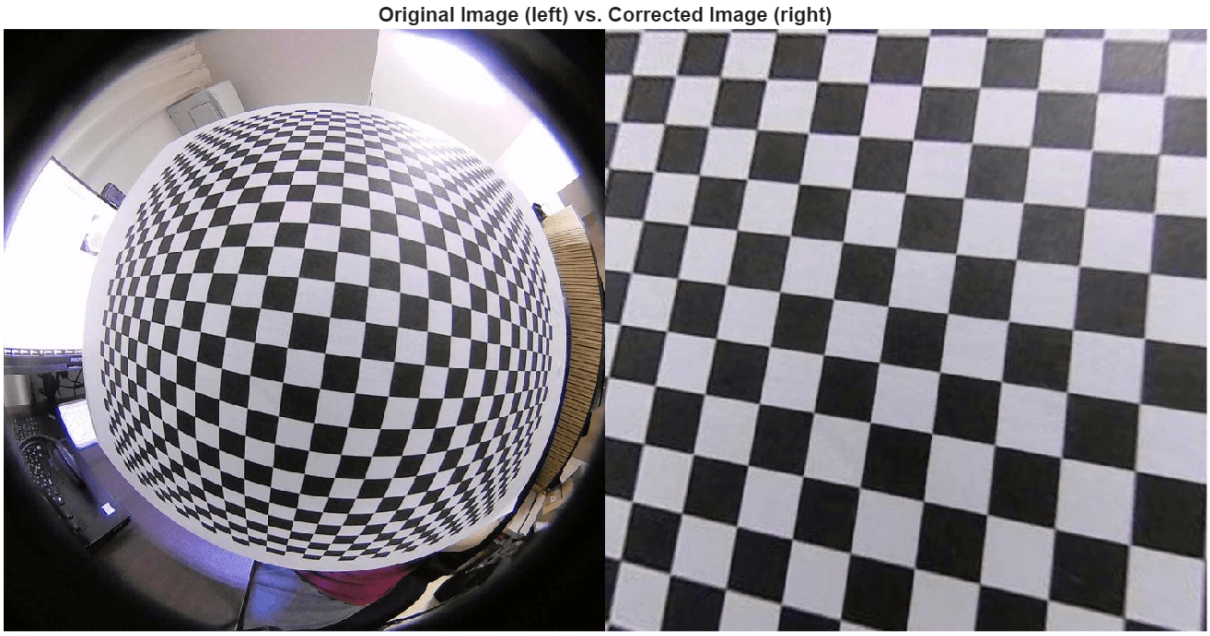
```
squareSize = 20; % millimeters
worldPoints = patternWorldPoints('checkerboard',patternDims,squareSize);
```

Estimate the fisheye camera calibration parameters based on the image and world points. Use the first image to get the image size.

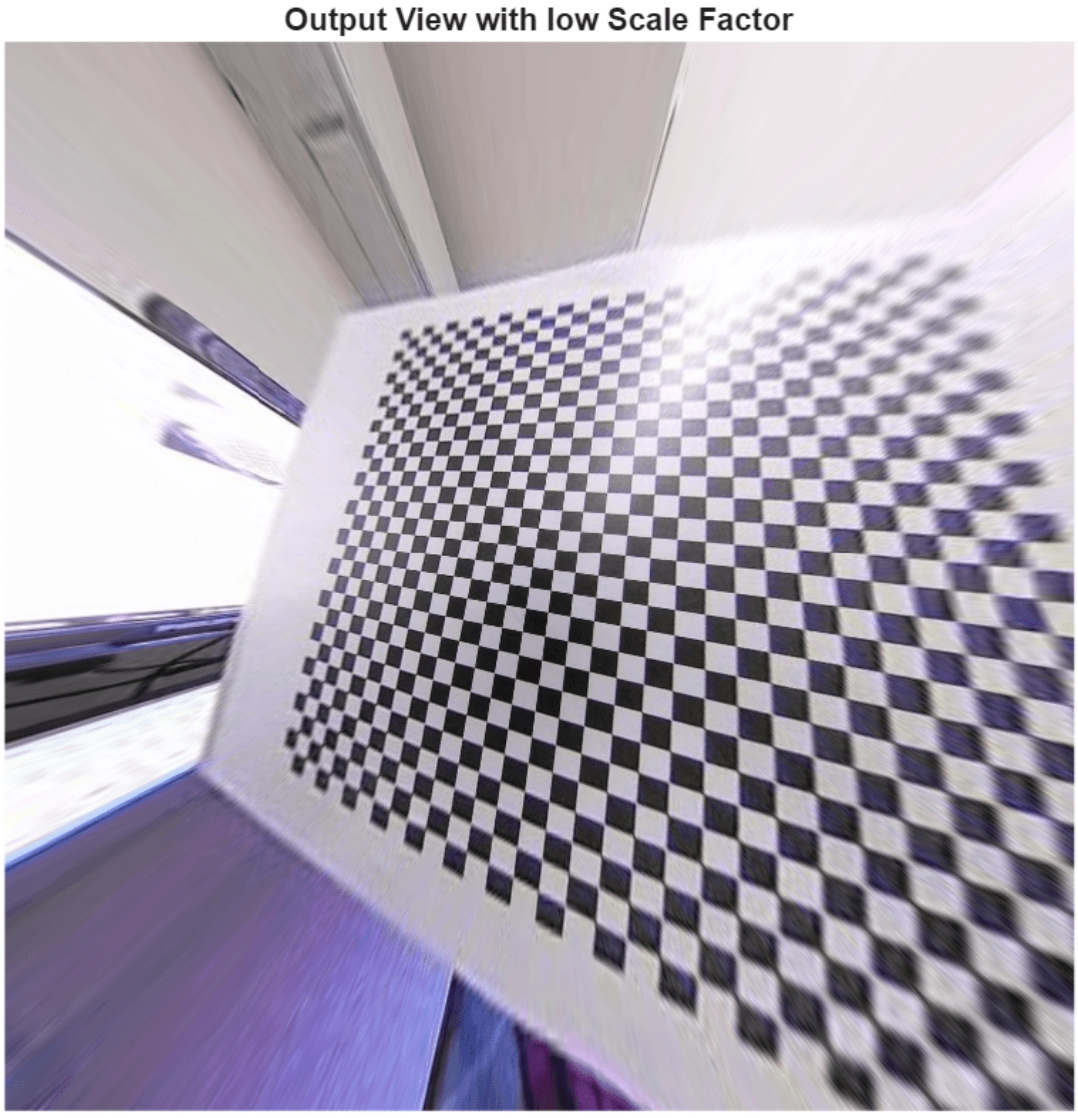
```
I = readimage(images,10);
imageSize = [size(I,1) size(I,2)];
params = estimateFisheyeParameters(imagePoints,worldPoints,imageSize);
```

Remove lens distortion from the first image `I` and display the results.

```
J1 = undistortFisheyeImage(I,params.Intrinsics);
figure
imshowpair(I,J1,'montage')
title('Original Image (left) vs. Corrected Image (right)')
```




```
J2 = undistortFisheyeImage(I,params.Intrinsics,'OutputView','same', 'ScaleFactor', 0.2);
figure
imshow(J2)
title('Output View with low Scale Factor')
```



Input Arguments

`I` — Input image

M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Input image, specified as an *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale image. The input image must be real and nonsparse.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`intrinsics` — Fisheye intrinsic camera parameters

`fisheyeIntrinsics` object

Fisheye intrinsic camera parameters, specified as a `fisheyeIntrinsics` object.

`interp` — Interpolation method
`'bilinear'` (default) | `'nearest'` | `'cubic'`

Interpolation method to use on the input image, specified as `'bilinear'`, `'nearest'` , or `'cubic'`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"ScaleFactor"=2` increases the zoom in the camera view by a factor of 2

`OutputView` — Size of the output image
`"same"` (default) | `"full"` | `"valid"`

Size of the output image, specified as either `"same"`, `"full"`, or `"valid"`.

The `"full"` output image size is not suitable for fisheye images created with wide-angle lenses, including catadioptic lenses, which generate spherical images.

`ScaleFactor` — Scale factor for focal length
1 (default) | scalar | `[sx sy]` vector

Scale factor for the focal length of a virtual camera perspective, in pixels, specified as a scalar or an `[sx sy]` vector. Specify a vector to scale the *x* and *y* axes individually. Increase the scale to zoom in the perspective of the camera view.

`FillValues` — Output pixel fill values
`0` (default) | scalar | 3-element vector

Output pixel fill values, specified as the comma-separated pair consisting of '`FillValues`' and scalar or 3-element vector. When the corresponding inverse-transformed location in the input image lies completely outside the input image boundaries, you use the fill values for output pixels. When you use a 2-D grayscale input image, `FillValues` must be a scalar. When you use a truecolor image, `FillValues` can be a scalar or a 3-element vector of RGB values.

Output Arguments

`J` — Undistorted image
M-by-*N*-by-3 truecolor image | *M*-by-*N* 2-D grayscale image

Undistorted image, returned as an *M*-by-*N*-by-3 truecolor or *M*-by-*N* 2-D grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`camIntrinsics` — Undistorted intrinsics of virtual camera
`cameraIntrinsics` object

Undistorted intrinsics of a virtual camera, returned as a `cameraIntrinsics` object. The `camIntrinsics` object represents a virtual pinhole camera. You can use this object with the pinhole

model calibration workflow functions. These intrinsics are for a camera that has a perspective that produces the undistorted image.

Tips

- The Computer Vision Toolbox calibration algorithm uses the fisheye camera model proposed by Scaramuzza [1] on page 3-1527.

Version History

Introduced in R2017b

R2024a: Generate C/C++ code using MATLAB Coder

`undistortFisheyeImage` now supports the generation of C/C++ code (requires MATLAB Coder).

References

[1] Scaramuzza, D., A. Martinelli, and R. Siegwart. "A Toolbox for Easy Calibrating Omnidirectional Cameras." *Proceedings to IEEE International Conference on Intelligent Robots and Systems, (IROS)*. Beijing, China, October 7–15, 2006.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`undistortFisheyePoints` | `fisheyeIntrinsics` | `estimateFisheyeParameters` | `cameraIntrinsics`

Topics

“Fisheye Calibration Basics”
“What Is Camera Calibration?”
“3-D Motion Reconstruction Using Multiple Cameras”

undistortFisheyePoints

Correct point coordinates for fisheye lens distortion

Syntax

```
undistortedPoints = undistortFisheyePoints(points,intrinsics)
undistortedPoints = undistortFisheyePoints( ___ ,scaleFactor)
[ ___ ,camIntrinsics] = undistortFisheyePoints( ___ )
[ ___ ,reprojectionErrors] = undistortFisheyePoints( ___ )
```

Description

`undistortedPoints = undistortFisheyePoints(points,intrinsics)` returns point coordinates corrected for fisheye lens distortion.

`undistortedPoints = undistortFisheyePoints( ___ ,scaleFactor)` returns corrected point coordinates using the `scaleFactor` and the previous inputs.

`[ ___ ,camIntrinsics] = undistortFisheyePoints( ___ )` also returns a `cameraIntrinsics` object, which corresponds to a virtual perspective camera that produces undistorted points.

`[ ___ ,reprojectionErrors] = undistortFisheyePoints( ___ )` also returns `reprojectionErrors` used to evaluate the accuracy of undistorted points. The function computes the reprojection errors by applying distortion to the points, and taking the distances between the result and the corresponding input points.

Examples

Undistort Checkerboard Points from Fisheye Image

Undistort and translate checkerboard points detected in a calibration image, and then display the results.

Create an `imageDatastore` object containing checkerboard calibration images.

```
images = imageDatastore(fullfile(toolboxdir('vision'),'visiondata',...
   'calibration','gopro'));
imageFileNames = images.Files;
```

Detect the calibration pattern from the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imageFileNames);
```

Generate world coordinates for the corners of the checkerboard squares.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints('checkerboard',patternDims,squareSize);
```

Estimate the fisheye parameters from the image and world points. Get the image size from the first image.

```
I = readimage(images,10);
imageSize = [size(I,1) size(I,2)];
params = estimateFisheyeParameters(imagePoints,worldPoints,imageSize);
```

In the first image, detect the checkerboard points.

```
points = detectCheckerboardPoints(I);
```

Undistort the points and image.

```
[undistortedPoints,intrinsics1] = undistortFisheyePoints(points,params.Intrinsics);
[J, intrinsics2] = undistortFisheyeImage(I,params.Intrinsics,'OutputView','full');
```

Translate the undistorted points.

```
newOrigin = intrinsics2.PrincipalPoint - intrinsics1.PrincipalPoint;
undistortedPoints = [undistortedPoints(:,1) + newOrigin(1), ...
                     undistortedPoints(:,2) + newOrigin(2)];
```

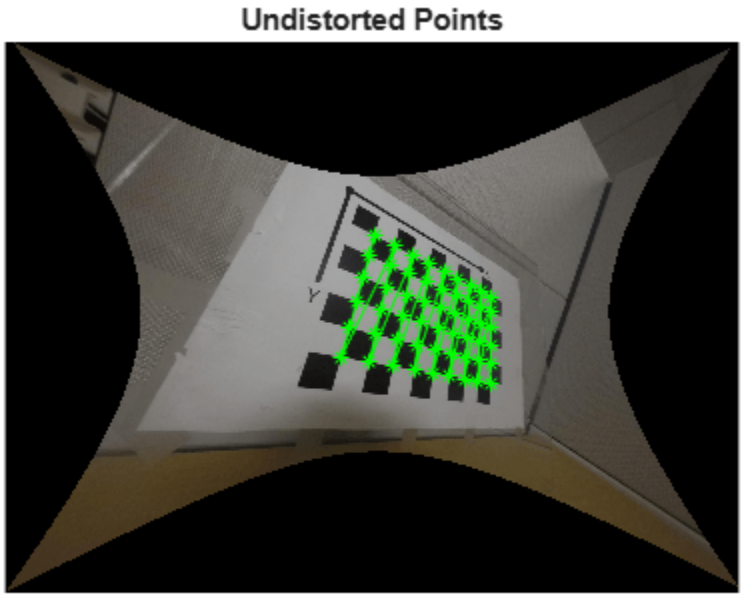
Display the results.

```
figure
imshow(I)
hold on
plot(points(:,1),points(:,2),'r*-')
title('Detected Points')
hold off
```




```
figure
imshow(J)
```

```
hold on
plot(undistortedPoints(:, 1),undistortedPoints(:, 2),'g*-')
title('Undistorted Points')
hold off
```



Input Arguments

`points` — Input points

M-by-2 matrix

Input points, specified as an *M*-by-2 matrix of *M* [x y] coordinates.

`intrinsics` — Fisheye intrinsic camera parameters

`fisheyeIntrinsics` object

Fisheye intrinsic camera parameters, specified as a `fisheyeIntrinsics` object.

`scaleFactor` — Scale factor for points

1 (default) | scalar | `[sx sy]` vector

Scale factor for points, specified as a scalar or an `[sx sy]` vector. Specify a vector to scale the *x* and *y* axes individually. Increase the scale to zoom in the perspective of the camera view.

Output Arguments

`undistortedPoints` — Undistorted points

M-by-2 matrix

Undistorted points, returned as an *M*-by-2 matrix of *M* number of [x y] coordinates. If `points` is `double`, then `undistortedPoints` is `double`. Otherwise, `undistortedPoints` is `single`.

Data Types: `single` | `double`

`camIntrinsics` — Undistorted intrinsics of virtual camera
`cameraIntrinsics` object

Undistorted intrinsics of a virtual camera, returned as a `cameraIntrinsics` object. These intrinsics are for a camera that has a perspective that produces the undistorted image.

`reprojectionErrors` — Reprojection errors
M-by-1 vector

Reprojection errors, returned as an *M*-by-1 vector. The function computes the reprojection errors by applying distortion to the undistorted points and taking the distances between the results and the corresponding input points. Errors are expressed in pixels.

Version History

Introduced in R2017b

R2025a: Generate CUDA code using GPU Coder

The `undistortFisheyePoints` function now supports the generation of CUDA code (requires GPU Coder).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`undistortFisheyeImage` | `fisheyeIntrinsics`

Topics
“Fisheye Calibration Basics”
“3-D Motion Reconstruction Using Multiple Cameras”

pointsToWorld

(Not recommended) Determine world coordinates of image points

Note `pointsToWorld` is not recommended. Use the `img2world2d` function instead. For more information, see "Version History".

Syntax

```
worldPoints = pointsToWorld(intrinsics,tform,imagePoints)
worldPoints = pointsToWorld(intrinsics,rotationMatrix,translationVector,
imagePoints)
```

Description

`worldPoints = pointsToWorld(intrinsics,tform,imagePoints)` maps undistorted image points `imagePoints`, onto points on the *X*-*Y* plane in world coordinates, `worldPoints` using the `rigid3d` transformation `tform`.

`worldPoints = pointsToWorld(intrinsics,rotationMatrix,translationVector, imagePoints)` returns world points on the *X*-*Y* plane, which correspond to the input image points. Points are converted using the input rotation matrix, translation vector, and camera intrinsics.

Examples

Map Image Points of Fisheye Image to World Coordinates

Map the points of a fisheye image to world coordinates and compare these points to the ground truth points. A series of checkerboard pattern images are used to estimate the fisheye parameters and calibrate the camera.

Create a set of checkerboard calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata" ,...
      "calibration","gopro"));
```

Detect the checkerboard corners in the images. Leave the last image for testing.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files(1:end-1));
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0).

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Estimate the fisheye camera parameters from the image and world points. Use the first image to get image size.

```
I = imread(images.Files{end});
imageSize = [size(I,1) size(I,2)];
```

```
fisheyeParams = estimateFisheyeParameters(imagePoints,worldPoints,imageSize);
intrinsics = fisheyeParams.Intrinsics;
```

Find the reference object in the new image.

```
imagePoints = detectCheckerboardPoints(I,PartialDetections=false);
```

Compute new extrinsics.

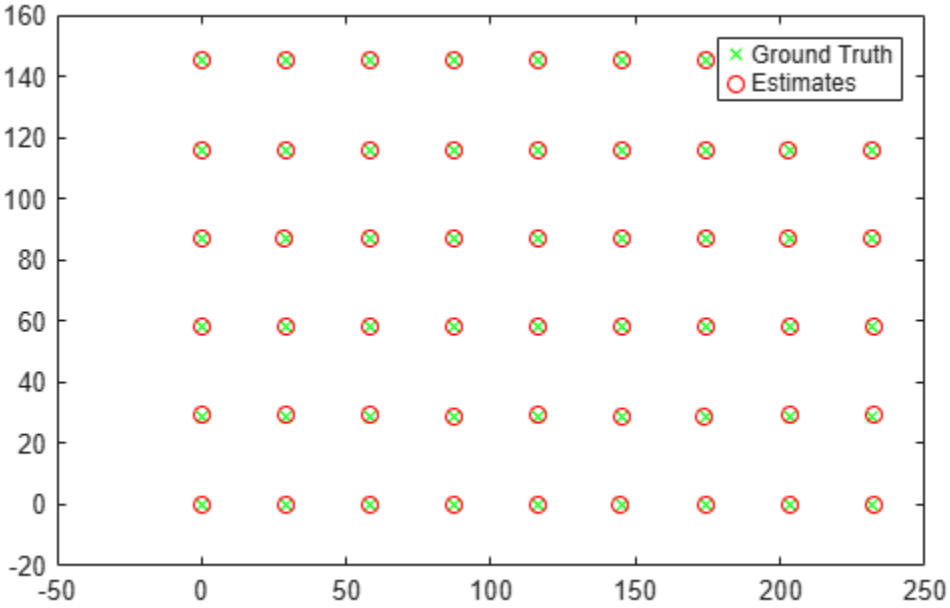
```
[R,t] = extrinsics(imagePoints,worldPoints,intrinsics);
```

Map image points to world coordinates in the X-Y plane.

```
newWorldPoints = pointsToWorld(intrinsics,R,t,imagePoints);
```

Compare estimated world points to the ground truth points.

```
plot(worldPoints(:,1),worldPoints(:,2),"gx");
hold on
plot(newWorldPoints(:,1),newWorldPoints(:,2),"ro");
legend("Ground Truth","Estimates");
hold off
```



Input Arguments

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | `fisheyeIntrinsics` object

Camera parameters, specified as a `cameraIntrinsics` or a `fisheyeIntrinsics` object. The objects store information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

`tform` — Transformation
`rigid3d` object

Transformation of the camera in world coordinates, specified as a `rigid3d` object.

`rotationMatrix` — 3-D rotation
3-by-3 matrix

3-D rotation of the world coordinates relative to the image coordinates, specified as a 3-by-3 matrix. The rotation matrix, together with the translation vector, enable you to transform points from the world coordinate system to the camera coordinate system. The `rotationMatrix` and `translationVector` inputs must be the same data type.



Data Types: `double` | `single`

`translationVector` — 3-D translation
1-by-3 vector

3-D translation of the world coordinates relative to the image coordinates, specified as a 1-by-3 vector. The translation vector, together with the rotation matrix, enable you to transform points from the world coordinate system to the camera coordinate system. The `rotationMatrix` and `translationVector` inputs must be the same data type.



Data Types: `double` | `single`

`imagePoints` — Image points
M-by-2 matrix

Image points, specified as an *M*-by-2 matrix containing *M* [*x*, *y*] coordinates of image points.

When using the `cameraParameters` object as the `cameraParams` input, `pointsToWorld` does not account for lens distortion. Therefore, the `imagePoints` input must contain image points detected in the undistorted image, or they must be undistorted using the `undistortPoints` function. For a `fisheyeIntrinsics` object, the image points are distorted.

Output Arguments

`worldPoints` — World coordinates
M-by-2 matrix

World coordinates, returned as an *M*-by-2 matrix. *M* represents the number of undistorted points in [*x*, *y*] world coordinates.

Version History

Introduced in R2016a

R2022b: Not recommended

Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `pointsToWorld` function uses the postmultiply convention. Although there are no plans to remove `pointsToWorld` at this time, you can streamline your geometric transformation workflows by switching to the `img2world2d` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `pointsToWorld` to `img2world2d`.
- Switch the order of the `intrinsics` and the `imagePoints` arguments.
- Specify the transformation as a `rigidtform3d` object using the `tform` argument. `img2world2d` does not support the `rotationMatrix` and `translationVector` input arguments. Note that you create the `rigidtform3d` object using the transpose of `rotationMatrix` or the transpose of the transformation matrix in the `T` property of `tform`.

Discouraged Usage	Recommended Replacement

This example specifies the transformation as a rotation matrix and a translation vector, then determines the world coordinates of image points using the `pointsToWorld` function.
 	
`worldPoints = pointsToWorld(intrinsics, ...`
`    rotationMatrix,translationVector,imagePoints)`
	This example specifies the transformation as a `rigidtform3d` object, then determines the world coordinates of image points using the `img2world2d` function. Note that you create the `rigidtform3d` object using the transpose of the rotation matrix, `rotationMatrix`.
	
	`tform = rigidtform3d(rotationMatrix',translationVecto`
	`worldPoints = img2world2d(imagePoints,tform,intrinsic`

This example specifies the transformation as a `rigid3d` object using the transpose of the geometric transformation matrix `A` in the premultiply convention, then determines the world coordinates of image points using the `pointsToWorld` function.
 	
`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
`tformOld = rigid3d(A');`
`worldPoints = pointsToWorld(intrinsics,tformOld,imagePoints);`
	This example specifies the transformation as a `rigidtform3d` object using the geometric transformation matrix `A`, then determines the world coordinates of image points using the `img2world2d` function
	
	`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
	`tform = rigidtform3d(A);`
	`worldPoints = img2world2d(imagePoints,tform,intrinsic`
	If instead you start with an existing `rigid3d` object `tformOld`, then you can get the geometric transformation matrix in the postmultiply convention by querying the property `T` of `tformOld`.
	
	`T = tformOld.T;`
	`tform = rigidtform3d(T');`
	`worldPoints = img2world2d(imagePoints,tform,intrinsic`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`img2world2d` | `world2img` | `undistortImage` | `undistortPoints` | `estimateCameraParameters` | `estimateExtrinsics` | `extr2pose` | `pose2extr` | `estworldpose` | `estrelpose`

Objects
`cameraParameters` | `fisheyeParameters` | `cameraIntrinsics` | `fisheyeIntrinsics`

img2world2d

Determine world coordinates of image points

Syntax

```
worldPoints = img2world2d(imagePoints,tform,intrinsics)
```

Description

`worldPoints = img2world2d(imagePoints,tform,intrinsics)` maps undistorted image points `imagePoints`, onto points on the *X*-*Y* plane in world coordinates `worldPoints`, using the rigid transformation `tform`.

Examples

Map Image Points of Fisheye Image to World Coordinates

Map the points of a fisheye image to world coordinates and compare these points to the ground truth points. A series of checkerboard pattern images are used to estimate the fisheye parameters and calibrate the camera.

Load a set of checkerboard calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata" ,...
      "calibration","gopro"));
```

Detect the checkerboard corners in the images. Leave the last image for testing.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files(1:end-1));
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0).

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Estimate the fisheye camera parameters from the image and world points. Use the first image to get image size.

```
I = imread(images.Files{end});
imageSize = [size(I,1) size(I,2)];
fisheyeParams = estimateFisheyeParameters(imagePoints,worldPoints,imageSize);
intrinsics = fisheyeParams.Intrinsics;
```

Find the reference object in the new image.

```
imagePoints = detectCheckerboardPoints(I,PartialDetections=false);
```

Calculate new extrinsics.

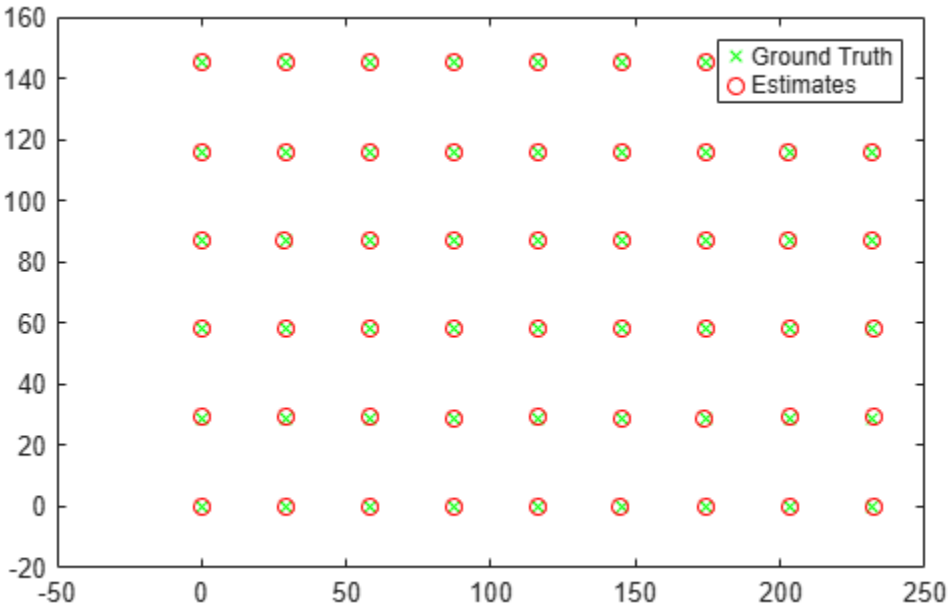
```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,intrinsics);
```

Map image points to world coordinates in the X-Y plane.

```
newWorldPoints = img2world2d(imagePoints,camExtrinsics,intrinsics);
```

Compare estimated world points to the ground truth points.

```
plot(worldPoints(:,1),worldPoints(:,2),"gx");
hold on
plot(newWorldPoints(:,1),newWorldPoints(:,2),"ro");
legend("Ground Truth","Estimates");
hold off
```



Input Arguments

`imagePoints` — Image points

M-by-2 matrix

Image points, specified as an *M*-by-2 matrix containing *M* [*x*, *y*] coordinates of image points.

When you specify the `cameraParams` argument as a `cameraIntrinsics` object, `img2world2d` does not account for lens distortion. Therefore, the `imagePoints` input must contain image points detected in the undistorted image, or they must be undistorted using the `undistortPoints` function. For a `fisheyeIntrinsics` object, the image points are distorted.

`tform` — Transformation

`rigidtform3d` object

Transformation of the camera in world coordinates, specified as a `rigidtform3d` object.

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | `fisheyeIntrinsics` object | `cameraIntrinsicsKB` object

Camera intrinsics, specified as a `cameraIntrinsics`, `fisheyeIntrinsics`, or a `cameraIntrinsicsKB` object. The object stores information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

Output Arguments

`worldPoints` — World coordinates

M-by-2 matrix

World coordinates, returned as an *M*-by-2 matrix. *M* represents the number of undistorted points in [*x*, *y*] world coordinates.

Version History

Introduced in R2022b

R2022b: Recommended over `pointsToWorld`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `pointsToWorld` function uses the postmultiply convention. Although there are no plans to remove `pointsToWorld` at this time, you can streamline your geometric transformation workflows by switching to the `img2world2d` function, which supports the premultiply convention. For more information, see "Migrate Geometric Transformations to Premultiply Convention".

R2024a: Added support for the `cameraIntrinsicsKB` object

Support added for the `cameraIntrinsicsKB` object as the `cameraParams` input.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Code generation not supported for `cameraIntrinsicsKB` object input.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps

Camera Calibrator | Stereo Camera Calibrator

Functions
world2img | undistortImage | undistortPoints | estimateCameraParameters | estimateExtrinsics | triangulate | triangulateMultiview | reconstructScene

Objects
cameraIntrinsics | cameraIntrinsicsKB | fisheyeIntrinsics

worldToImage

(Not recommended) Project world points into image

Note `worldToImage` is not recommended. Use the `world2img` function instead. For more information, see “Version History”.

Syntax

```
imagePoints = worldToImage(intrinsics,tform,worldPoints)
imagePoints = worldToImage(intrinsics,rotationMatrix,translationVector,
worldPoints)
[imagePoints,validIndex] = worldToImage( ___ )
[ ___ ] = worldToImage( ___ 'ApplyDistortion',distort)
```

Description

`imagePoints = worldToImage(intrinsics,tform,worldPoints)` projects 3-D world points `worldPoints` into points on an image, `imagePoints`. `intrinsics` can be a `cameraIntrinsics` or a `fisheyeIntrinsics` object. `tform` is a `rigid3d` object.

`imagePoints = worldToImage(intrinsics,rotationMatrix,translationVector, worldPoints)` returns the projection of 3-D world points into an image given the camera intrinsics, the rotation matrix, and the translation vector.

`[imagePoints,validIndex] = worldToImage( ___ )` also returns the indices of valid image points that are within the boundary of the image using any of the previous syntax inputs.

`[ ___ ] = worldToImage( ___ 'ApplyDistortion',distort)` returns the projection with the option of applying distortion. This syntax is supported for nonfisheye camera parameters.

Examples

Project World Points into Image

Create a set of calibration images.

```
images = imageDatastore(fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration","slr"));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0).

```
squareSize = 29; % in millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

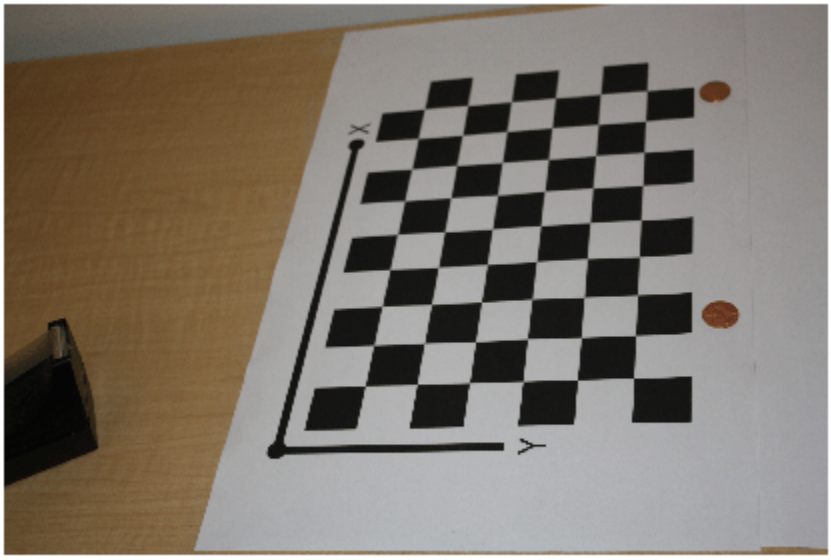
Calibrate the camera.

```
imageSize = size(readimage(images,1));
cameraParams = estimateCameraParameters(imagePoints,worldPoints, ...
    ImageSize=imageSize(1:2));
```

Load the image at a new location.

```
imOrig = imread(fullfile(toolboxdir("vision"),"visiondata", ...
        "calibration","slr","image9.jpg"));

imshow(imOrig);
```



Undistort the image.

```
imUndistorted = undistortImage(imOrig,cameraParams);
```

Find a reference object in the new image.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imUndistorted);
```

Compute new extrinsics.

```
[R,t] = extrinsics(imagePoints,worldPoints,cameraParams);
```

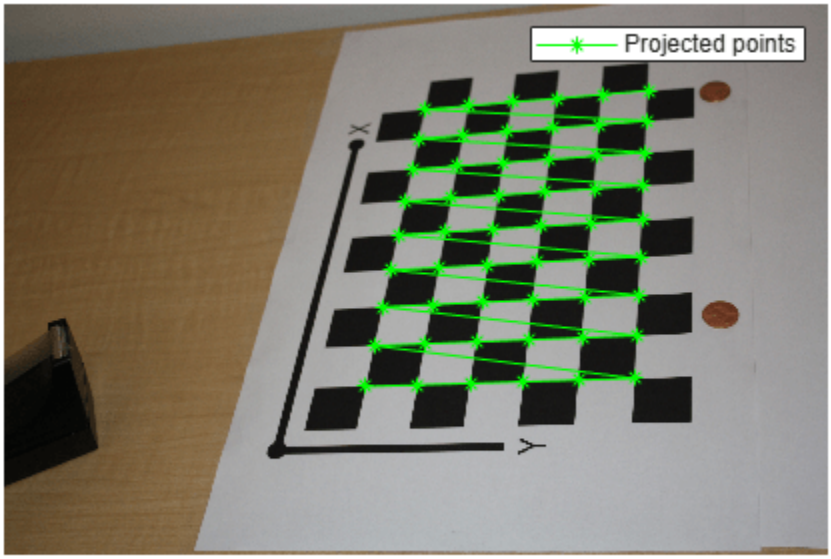
Add a *z*-coordinate to the world points.

```
zCoord = zeros(size(worldPoints,1),1);
worldPoints = [worldPoints zCoord];
```

Project the world points back into the original image.

```
projectedPoints = worldToImage(cameraParams,R,t,worldPoints);
hold on
```

```
plot(projectedPoints(:,1),projectedPoints(:,2),"g*-");
legend("Projected points");
hold off
```



Input Arguments

`tform` — Transformation

`rigid3d` object

Transformation of the camera in world coordinates, specified as a `rigid3d` object.

`intrinsics` — Camera intrinsics

`cameraIntrinsics` object | `fisheyeIntrinsics` object

Camera parameters, specified as a `cameraIntrinsics` or a `fisheyeIntrinsics` object. The objects store information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

`rotationMatrix` — 3-D rotation

3-by-3 matrix

3-D rotation of the world coordinates relative to the image coordinates, specified as a 3-by-3 matrix. The rotation matrix, together with the translation vector, enable you to transform points from the world coordinate system to the camera coordinate system. The `rotationMatrix` and `translationVector` inputs must be the same data type.



Data Types: `double` | `single`

`translationVector` — 3-D translation
1-by-3 vector

3-D translation of the world coordinates relative to the image coordinates, specified as a 1-by-3 vector. The translation vector, together with the rotation matrix, enable you to transform points from the world coordinate system to the camera coordinate system. The `rotationMatrix` and `translationVector` inputs must be the same data type.



Data Types: `double` | `single`

`worldPoints` — 3-D world points
M-by-3 matrix

3-D world points, specified as an *M*-by-3 matrix containing *M* [*x*,*y*,*z*] coordinates of 3-D world points.

If you use the `tform` input argument then the `worldPoints` coordinates must be in the same units as the `Translation` property of the `tform` object.

`distort` — Apply lens distortion
`false` (default) | `true`

Option to apply lens distortion, specified as `false` or `true`. When you set this argument to `true`, the function applies lens distortion to the output `imagePoints`.

This argument is valid only when using a `cameraParameters` object as the `cameraParams` input.

Output Arguments

`imagePoints` — Image points
M-by-2 matrix

Image points, returned as an *M*-by-2 matrix of *M* [*x*,*y*] point coordinates.

`validIndex` — Valid index
M-by-1 logical array

Valid index returned as an *M*-by-1 logical array that specify the indices of valid image points in the `imagePoints` output that are within the boundary of the image. The world points that correspond to the indices are inside the field of view of the camera.

Version History

Introduced in R2016b

R2022b: Not recommended
Not recommended starting in R2022b

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `worldToImage` function uses the postmultiply convention. Although there are no plans to remove `worldToImage` at this time, you can streamline your geometric transformation workflows by switching to the `world2img` function, which supports the premultiply convention. For more information, see “Migrate Geometric Transformations to Premultiply Convention”.

To update your code:

- Change instances of the function name `worldToImage` to `world2img`.
- Switch the order of the `intrinsics` and the `worldPoints` arguments.
- Specify the transformation as a `rigidtform3d` object using the `tform` argument. `world2img` does not support the `rotationMatrix` and `translationVector` input arguments. Note that you create the `rigidtform3d` object using the transpose of `rotationMatrix` or the transpose of the transformation matrix in the `T` property of `tform`.

Discouraged Usage	Recommended Replacement

This example specifies the transformation as a rotation matrix and a translation vector, then projects 3-D world points into points on an image using the `worldToImage` function.
`imagePoints = worldToImage(intrinsics, ...`
`    rotationMatrix,translationVector,worldPoints);`
	This example specifies the transformation as a `rigidtform3d` object, then projects 3-D world points into points on an image using the `world2img` function. Note that you create the `rigidtform3d` object using the transpose of the rotation matrix, `rotationMatrix`.
	`tform = rigidtform3d(rotationMatrix',translationVecto`
	`imagePoints = world2img(worldPoints,tform,intrinsics)`

This example specifies the transformation `rigid3d` object using the transpose of the geometric transformation matrix `A` in the premultiply convention, then projects 3-D world points into points on an image using the `worldToImage` function.
`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
`tformOld = rigid3d(A');`
`imagePoints = worldToImage(intrinsics,tformOld,worldPoints);`
	This example specifies the transformation as a `rigidtform3d` object using the geometric transformation matrix `A`, then projects 3-D world points into points on an image using the `world2img` function.
	`A = [1 0 0 0; 0 1 0 0; 0 0 1 -10; 0 0 0 1];`
	`tform = rigidtform3d(A);`
	`imagePoints = world2img(worldPoints,tform,intrinsics)`
	If instead you start with an existing `rigid3d` object `tformOld`, then you can get the geometric transformation matrix in the postmultiply convention by querying the property `T` of `tformOld`.
	`T = tformOld.T;`
	`tform = rigidtform3d(T');`
	`imagePoints = world2img(worldPoints,tform,intrinsics)`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`world2img` | `img2world2d` | `undistortImage` | `undistortPoints` | `estimateCameraParameters` | `estimateExtrinsics` | `fisheyeIntrinsics` | `pose2extr` | `extr2pose` | `estrelpose` | `estworldpose`

Objects
`cameraParameters` | `fisheyeParameters` | `cameraIntrinsics` | `fisheyeIntrinsics`

world2img

Project world points into image

Syntax

```
imagePoints = world2img(worldPoints,tform,intrinsics)
imagePoints = world2img(worldPoints,tform,intrinsics,ApplyDistortion=distort)
[imagePoints,validIndex] = world2img( ___ )
```

Description

`imagePoints = world2img(worldPoints,tform,intrinsics)` projects 3-D world points `worldPoints` into points on an image, `imagePoints`, according to a rigid geometric transformation, `tform`. `intrinsics` can be a `cameraIntrinsics`, `cameraIntrinsicsKB`, or a `fisheyeIntrinsics` object.

`imagePoints = world2img(worldPoints,tform,intrinsics,ApplyDistortion=distort)` returns the projection with distortion applied. Use this syntax only with `cameraIntrinsics` and `cameraIntrinsicsKB` objects.

`[imagePoints,validIndex] = world2img( ___ )` also returns the indices of valid image points that are within the boundary of the image.

Examples

Project World Points into Image

Create a set of calibration images.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata", ...
      "calibration","slr");
images = imageDatastore(dataDir);
imageSize = size(readimage(images,1));
```

Detect the checkerboard corners in the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate the world coordinates of the checkerboard corners in the pattern-centric coordinate system, with the upper-left corner at (0,0).

```
squareSize = 29; % in millimeters
worldPoints = patternWorldPoints("checkerboard",patternDims,squareSize);
```

Calibrate the camera.

```
cameraParams = estimateCameraParameters(imagePoints,worldPoints, ...
    ImageSize=imageSize(1:2));
intrinsics = cameraParams.Intrinsics;
```

Load the image at a new location.

```
imOrig = readimage(images,9);
imshow(imOrig)
```

Undistort the image.

```
imUndistorted = undistortImage(imOrig,intrinsics);
```

Find a reference object in the new image.

```
[imagePoints,patternDims] = detectCheckerboardPoints(imUndistorted);
```

Calculate new extrinsics.

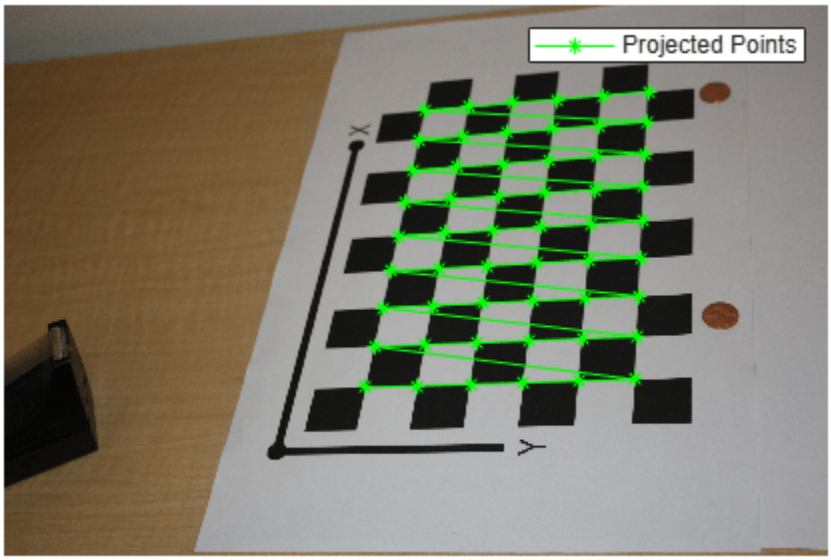
```
camExtrinsics = estimateExtrinsics(imagePoints,worldPoints,intrinsics);
```

Add a *z*-coordinate to the world points.

```
zCoord = zeros(size(worldPoints,1),1);
worldPoints = [worldPoints zCoord];
```

Project the world points back into the original image.

```
projectedPoints = world2img(worldPoints,camExtrinsics,intrinsics);
hold on
plot(projectedPoints(:,1),projectedPoints(:,2),"g*-");
legend("Projected Points");
hold off
```



Input Arguments

`worldPoints` — 3-D world points
M-by-3 matrix

3-D world points, specified as an *M*-by-3 matrix containing *M* [*x*,*y*,*z*] coordinates of 3-D world points. The `worldPoints` coordinates must be in the same units as the `Translation` property of the `tform` object.

`tform` — Transformation
`rigidtform3d` object

Transformation of the camera in world coordinates, specified as a `rigidtform3d` object.

`intrinsics` — Camera intrinsics
`cameraIntrinsics` object | `fisheyeIntrinsics` object | `cameraIntrinsicsKB` object

Camera intrinsics, specified as a `cameraIntrinsics`, `fisheyeIntrinsics` or a `cameraIntrinsicsKB` object. The object stores information about a camera's intrinsic calibration parameters, including the lens distortion parameters.

`distort` — Apply lens distortion
`false` (default) | `true`

Apply lens distortion, specified as `false` or `true`. When you set this argument to `true`, the function applies lens distortion to the output `imagePoints`.

This argument is valid only when the `intrinsics` argument is a `cameraIntrinsics` or `cameraIntrinsicsKB` object.

Output Arguments

`imagePoints` — Image points
M-by-2 matrix

Image points, returned as an *M*-by-2 matrix of *M* [*x*,*y*] point coordinates.

`validIndex` — Valid index
M-by-1 logical vector

Valid index returned as an *M*-by-1 logical vector that specify the indices of valid image points in the `imagePoints` output that are within the boundary of the image. The world points that correspond to the indices are inside the field of view of the camera.

Version History

Introduced in R2022b

R2022b: Recommended over `worldToImage`

Starting in R2022b, most Computer Vision Toolbox functions create and perform geometric transformations using the premultiply convention. However, the `worldToImage` function uses the postmultiply convention. Although there are no plans to remove `worldToImage` at this time, you can streamline your geometric transformation workflows by switching to the `world2img` function, which supports the premultiply convention. For more information, see "Migrate Geometric Transformations to Premultiply Convention".

R2024a: Added support for the `cameraIntrinsicsKB` object

Support added for the `cameraIntrinsicsKB` object as the `cameraParams` input.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Code generation not supported for `cameraIntrinsicsKB` object input.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

Apps
Camera Calibrator | Stereo Camera Calibrator

Functions
`img2world2d` | `undistortImage` | `undistortPoints` | `estimateCameraParameters` | `estimateExtrinsics` | `estworldpose` | `estrelpose`

Objects
`cameraIntrinsics` | `cameraIntrinsicsKB` | `fisheyeIntrinsics`

toStruct

Convert a trained aggregate channel features (ACF) object detector into structure

Syntax

```
detectorStruct = toStruct(acfObjectDetector)
```

Description

`detectorStruct = toStruct(acfObjectDetector)` returns a structure that stores the properties of the input `acfObjectDetector` object in fields. The output structure contains fields `Classifier` and `TrainingOptions`. Use the structure to create an identical `acfObjectDetector` object for C code generation. You can call the `toStruct`, and then pass the resulting structure to a MATLAB function, which recreates the `acfObjectDetector` object.

Examples

Convert ACF Object Detector to Struct

Load the pretrained ACF stop sign detector from the `stopSignDetector.mat` file, which is present in the current working folder as a supporting file.

```
stopSignDetector= load('stopSignDetectorACF.mat');
detector = stopSignDetector.detector
```

```
detector =

  acfObjectDetector with properties:

            ModelName: 'stopSign'
    ObjectTrainingSize: [34 31]
       NumWeakLearners: 61
```

Convert the detector to a structure.

```
detectorStruct = toStruct(detector)
```

```
detectorStruct =

  struct with fields:

            ModelName: 'stopSign'
    ObjectTrainingSize: [34 31]
       NumWeakLearners: 61
            Classifier: [1×1 struct]
```

```
    TrainingOptions: [1×1 struct]
```

Input Arguments

`acfObjectDetector` — ACF object detector

`acfObjectDetector` object

ACF object detector, specified as an `acfObjectDetector` object. To create this object, call the `trainACFObjectDetector` function with training data as input.

Output Arguments

`detectorStruct` — ACF object properties

structure

ACF object detector, returned as a structure containing the object properties into fields.

Version History

Introduced in R2017a

See Also

Topics

“Generate Code for Detecting Objects in Images by Using ACF Object Detector”

displayErrors

Display standard errors of camera parameter estimates

Syntax

```
displayErrors(estimationErrors,cameraParams)
```

Description

`displayErrors(estimationErrors,cameraParams)` displays the camera parameters and corresponding standard errors.

Examples

Display Fisheye Camera Calibration Errors

Gather a set of checkerboard calibration images.

```
images = imageDatastore(fullfile(toolboxdir('vision'),'visiondata',...
 'calibration','gopro'));
```

Detect the calibration pattern from the images.

```
[imagePoints,patternDims] = detectCheckerboardPoints(images.Files);
```

Generate world coordinates for the corners of the checkerboard squares.

```
squareSize = 29; % millimeters
worldPoints = patternWorldPoints('checkerboard',patternDims,squareSize);
```

Estimate the fisheye parameters using image and world points. Use the first image to get the image size. Also, store the errors from the calibration.

```
I = readimage(images,1);
imageSize = [size(I,1) size(I,2)];
[params,~,errors] = estimateFisheyeParameters(imagePoints, ...
                                worldPoints,imageSize);
```

Display the standard errors of the estimated camera parameters.

```
displayErrors(errors,params);
```

```
            Standard Errors of Estimated Camera Parameters
            ----------------------------------------------

Intrinsics
----------
Mapping coefficients:    [  875.0746 +/- 0.9451       -0.0003 +/- -0.0000      -0.0000 +/- 0.0000
Distortion center (pixels):[ 1005.8175 +/- 0.6871      743.0343 +/- 0.5578  ]
Stretch matrix parameters:[    1.0000 +/- 0.0000        0.0000 +/- 0.0000        0.0000 +/- 0.000

Extrinsics
```

```
----------
Rotation vectors:
                          [   -0.0699 +/- 0.0010       -0.0267 +/- 0.0009        0.0258 +/- 0.0002
                          [    0.3628 +/- 0.0010        0.2950 +/- 0.0009       -0.1967 +/- 0.0003
                          [   -0.2159 +/- 0.0009        0.3442 +/- 0.0009       -0.1941 +/- 0.0003
                          [    0.0282 +/- 0.0009       -0.3784 +/- 0.0009        0.0829 +/- 0.0003
                          [    0.0146 +/- 0.0008        0.4575 +/- 0.0009       -0.1215 +/- 0.0003
                          [    0.6775 +/- 0.0008        0.1089 +/- 0.0008       -0.0386 +/- 0.0004
                          [   -0.4936 +/- 0.0008        0.0063 +/- 0.0008        0.0486 +/- 0.0003
                          [    0.3823 +/- 0.0008        0.2797 +/- 0.0008        0.1509 +/- 0.0003
                          [    0.5171 +/- 0.0008       -0.3295 +/- 0.0008        0.0541 +/- 0.0003
                          [   -0.1896 +/- 0.0008       -0.3543 +/- 0.0009        0.2637 +/- 0.0003
                          [   -0.2911 +/- 0.0008        0.3680 +/- 0.0008       -0.1329 +/- 0.0003

Translation vectors (mm):
                          [ -132.9184 +/- 0.1609      -82.6066 +/- 0.1356      195.1098 +/- 0.2311
                          [ -178.9934 +/- 0.1905      -15.7749 +/- 0.1712      241.7116 +/- 0.2795
                          [ -183.7960 +/- 0.2168      -56.7377 +/- 0.1884      269.9729 +/- 0.2790
                          [  -17.6297 +/- 0.1315      -70.2875 +/- 0.1041      157.0820 +/- 0.1933
                          [ -161.9827 +/- 0.1807      -46.9680 +/- 0.1569      228.4052 +/- 0.2302
                          [ -122.4242 +/- 0.1309      -16.0260 +/- 0.1153      162.6239 +/- 0.2072
                          [ -112.4270 +/- 0.1745     -125.5876 +/- 0.1428      212.8047 +/- 0.2150
                          [ -148.7139 +/- 0.1387      -72.5409 +/- 0.1260      173.7608 +/- 0.2080
                          [  -49.5393 +/- 0.0919      -24.8329 +/- 0.0745      104.3536 +/- 0.1500
                          [   -3.4047 +/- 0.1274      -93.4073 +/- 0.1010      155.8242 +/- 0.1693
                          [ -160.7347 +/- 0.1855      -51.9152 +/- 0.1600      234.4066 +/- 0.2318
```

Input Arguments

`estimationErrors` — Standard errors of estimated parameters

`cameraCalibrationErrors` object | `fisheyeCalibrationErrors` object

Standard errors of estimated parameters, specified as a `cameraCalibrationErrors` or `fisheyeCalibrationErrors` object.

`cameraParams` — Camera parameters

`cameraParameters` object | `fisheyeParameters` object

Camera parameters, specified as a `cameraParameters` or `fisheyeParameters` object. These objects contain the intrinsic, extrinsic, and lens distortion parameters of a camera.

- To create a `cameraParameters`, use the `estimateCameraParameters` function or the Camera Calibrator app.
- To create a `fisheyeParameters`, use the `estimateFisheyeParameters` function.

Version History

Introduced in R2017b

See Also

Functions

`estimateCameraParameters` | `estimateFisheyeParameters`

Objects
`fisheyeParameters` | `cameraParameters`

Apps
Camera Calibrator

Topics
“Evaluating the Accuracy of Single Camera Calibration”

info

Information about specified video file

Syntax

```
S = info(videoFReader)
```

Description

`S = info(videoFReader)` returns a MATLAB structure, `S`, with information about the video file specified in the `Filename` property.

Input Arguments

`videoFReader` — Video file reader object

object (default)

Video file reader, specified as a `vision.VideoFileReader` System object.

Output Arguments

`S` — Information about input file

structure

Information about input file, returned as a structure. The fields and possible values for the structure `S` are described below:

Audio	Logical value indicating if the file has audio content.
Video	Logical value indicating if the file has video content.
VideoFrameRate	Frame rate of the video stream in frames per second. The value may vary from the actual frame rate of the recorded video, and takes into consideration any synchronization issues between audio and video streams when the file contains both audio and video content. This implies that video frames may be dropped if the audio stream leads the video stream by more than 1/(actual video frames per second).
VideoSize	Video size as a two-element numeric vector of the form: `[VideoWidthInPixels, VideoHeightInPixels]`
VideoFormat	Video signal format.

Version History

Introduced in R2012a

isDone

End-of-file status (logical)

Syntax

```
status = isDone(videoFReader)
```

Description

`status = isDone(videoFReader)` returns a logical value indicating that the `VideoFileReader` System object `videoFReader` , has reached the end of the multimedia file after playing it `PlayCount` number of times. After the object plays the file the number of times set by the `PlayCount` property, it sets the status to `true`.

Examples

Read and Play a Video File

Load the video using a video reader object.

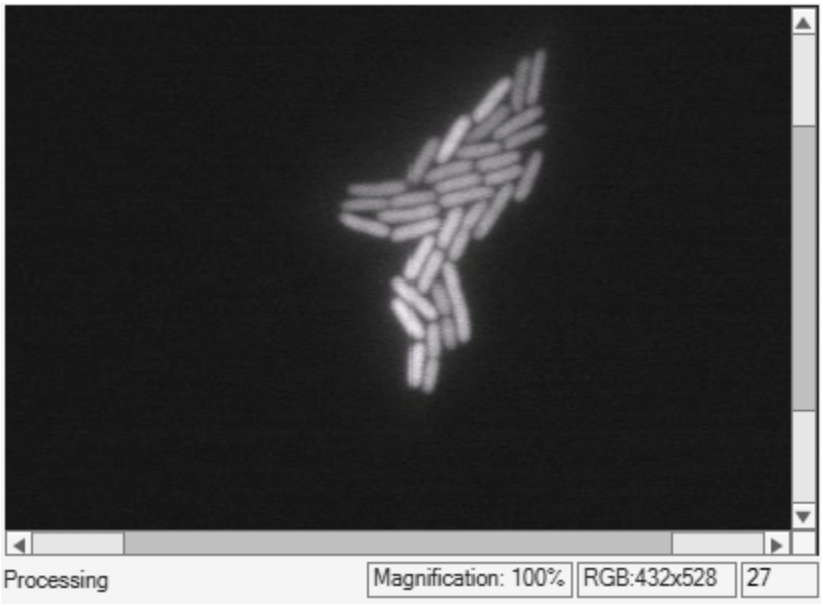
```
vidReader = VideoReader('ecolicells.avi');
```

Create a video player object to play the video file.

```
videoPlayer = vision.VideoPlayer;
```

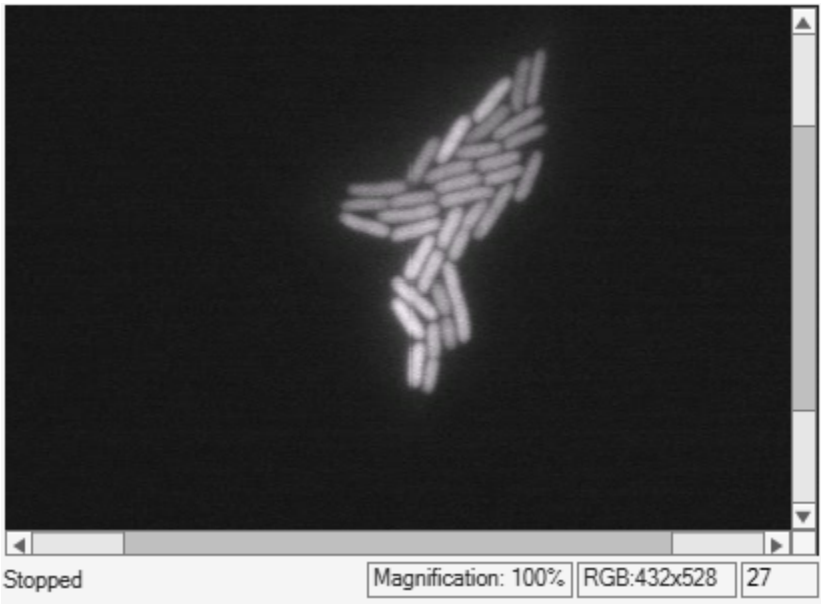
Use a while loop to read and play the video frames. Pause for 0.1 seconds after displaying each frame.

```
while hasFrame(vidReader)
    videoFrame = readFrame(vidReader);
    videoPlayer(videoFrame);
    pause(0.1)
end
```



Release the objects.

```
release(videoPlayer);
```



Input Arguments

`videoFReader` — Video file reader object

object (default)

Video file reader, specified as a `vision.VideoFileReader` System object.

Output Arguments

`status` — Status of reader file
`true` | `false`

Status of reader file, returned as `true` or `false`.

Version History

Introduced in R2012a

isOpen

Visible or hidden status for player

Syntax

```
isOpen(player)
```

Description

`isOpen(player)` returns `true` or `false` to indicate whether the player is visible.

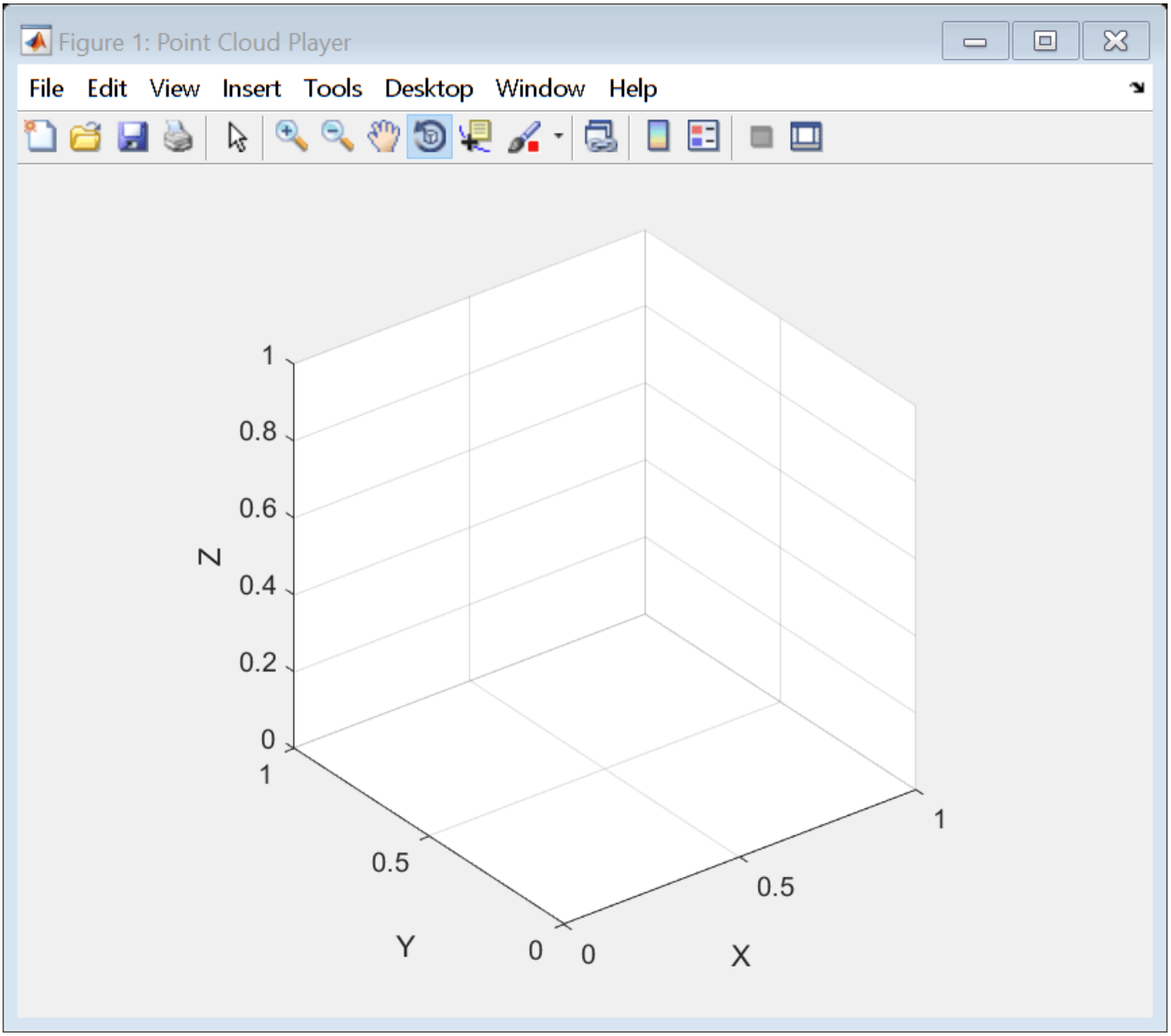
Examples

Terminate a Point Cloud Processing Loop

Close the display of continuous point cloud player

Add data to the point cloud player.

```
player = pcplayer([0 1],[0 1],[0 1]);
```



Display continuous player figure. Use the `isOpen` method to check if player figure window is open.

```
while isOpen(player)
   ptCloud = pointCloud(rand(1000,3,'single'));
   view(player, ptCloud);
end
```

Terminate the while-loop by closing `pcplayer` figure window.

Input Arguments

`player` — Player
object

Video player, specified as a `pcplayer`, or `vision.DeployableVideoPlayer` object.

Version History

Introduced in R2012a

segmentGroundFromLidarData

Segment ground points from organized lidar data

Syntax

```
groundPtsIdx = segmentGroundFromLidarData(ptCloud)
groundPtsIdx = segmentGroundFromLidarData(ptCloud,Name=Value)
```

Description

`groundPtsIdx = segmentGroundFromLidarData(ptCloud)` segments organized 3-D lidar data, `ptCloud`, into ground and nonground parts. The lidar sensor must be mounted horizontally such that all ground points are observed in the lidar scan closest to the sensor.

To use this function, the ground plane must be along the *x*-*y* plane. You can use the `pctransform` to rotate the sensor data before using `segmentGroundFromLidarData`.

`groundPtsIdx = segmentGroundFromLidarData(ptCloud,Name=Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `segmentGroundFromLidarData(ptCloud,ElevationAngleDelta=5)`

Examples

Segment and Plot Organized Lidar Data

Segment ground points and nonground points from an organized lidar point cloud. Create organized point clouds from these segmentations, and display them.

Load an organized lidar, point cloud.

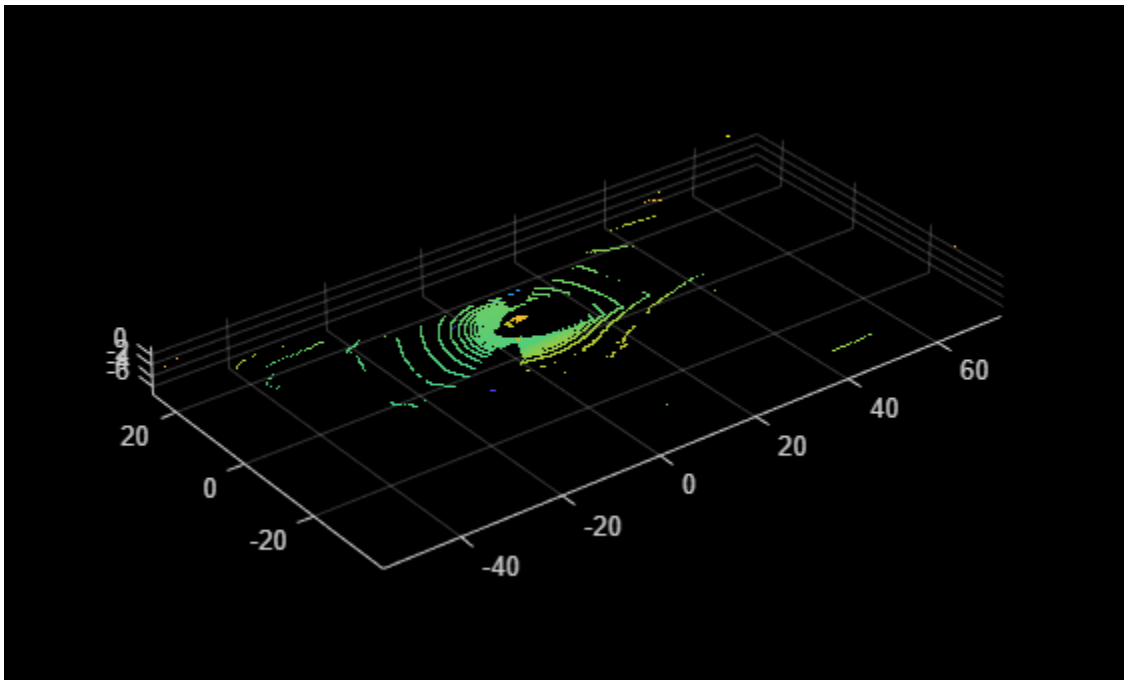
```
ld = load('drivingLidarPoints.mat');
```

Segment ground points from the organized lidar point cloud.

```
groundPtsIdx = segmentGroundFromLidarData(ld.ptCloud);
```

Create an organized point cloud containing only these ground points by using the `select` function. Display this point cloud.

```
groundPtCloud = select(ld.ptCloud,groundPtsIdx);
figure
pcshow(groundPtCloud)
```

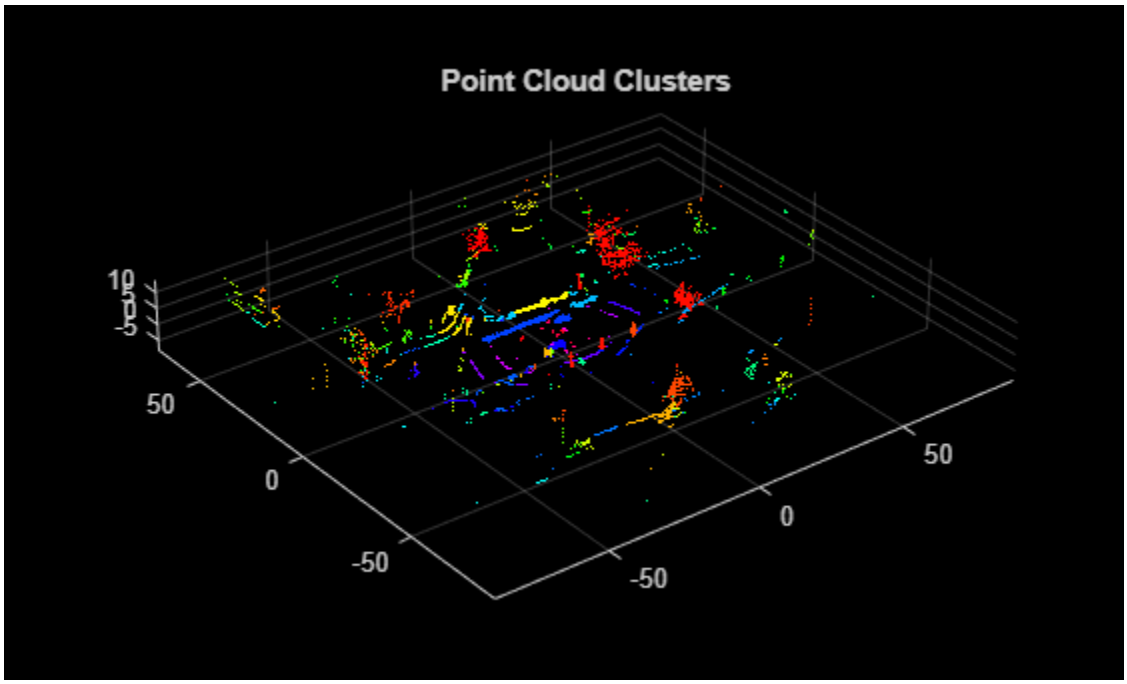


Create an organized point cloud containing only the nonground points. Specify a threshold of 0.5 meters.

```
nonGroundPtCloud = select(ld.ptCloud,~groundPtsIdx,'OutputSize','full');
distThreshold = 0.5;
[labels,numClusters] = segmentLidarData(nonGroundPtCloud,distThreshold);
```

Display the nonground points cloud clusters.

```
figure
colormap(hsv(numClusters))
pcshow(nonGroundPtCloud.Location,labels)
title('Point Cloud Clusters')
```



Segment and Plot Ground Plane Using PCAP File

Load Velodyne PCAP® to the workspace.

```
velodyneFileReaderObj = velodyneFileReader('lidarData_ConstructionRoad.pcap','HDL32E');
```

Create a point cloud player using pcplayer. Define its *x*-, *y*-, and *z*-axes limits, in meters, and label its axes.

```
xlimits = [-40 40];
ylimits = [-15 15];
zlimits = [-3 3];
player = pcplayer(xlimits,ylimits,zlimits);
```

Label the pcplayer axes.

```
xlabel(player.Axes,'X (m)')
ylabel(player.Axes,'Y (m)')
zlabel(player.Axes,'Z (m)')
```

Set the colormap for labeling points. Use RGB triplets to specify green for ground-plane points, and red for obstacle points.

```
colors = [0 1 0; 1 0 0];
greenIdx = 1;
redIdx = 2;
```

Iterate through the point clouds in the Velodyne PCAP file, using `readFrame` to read in the data. Segment the ground points from each point cloud. Color all ground points green and nonground points red. Plot the resulting lidar point cloud.

```
colormap(player.Axes,colors)
title(player.Axes,'Segmented Ground Plane of Lidar Point Cloud');

for i = 1:velodyneFileReaderObj.NumberOfFrames
    % Read current frame.
    ptCloud = velodyneFileReaderObj.readFrame(i);

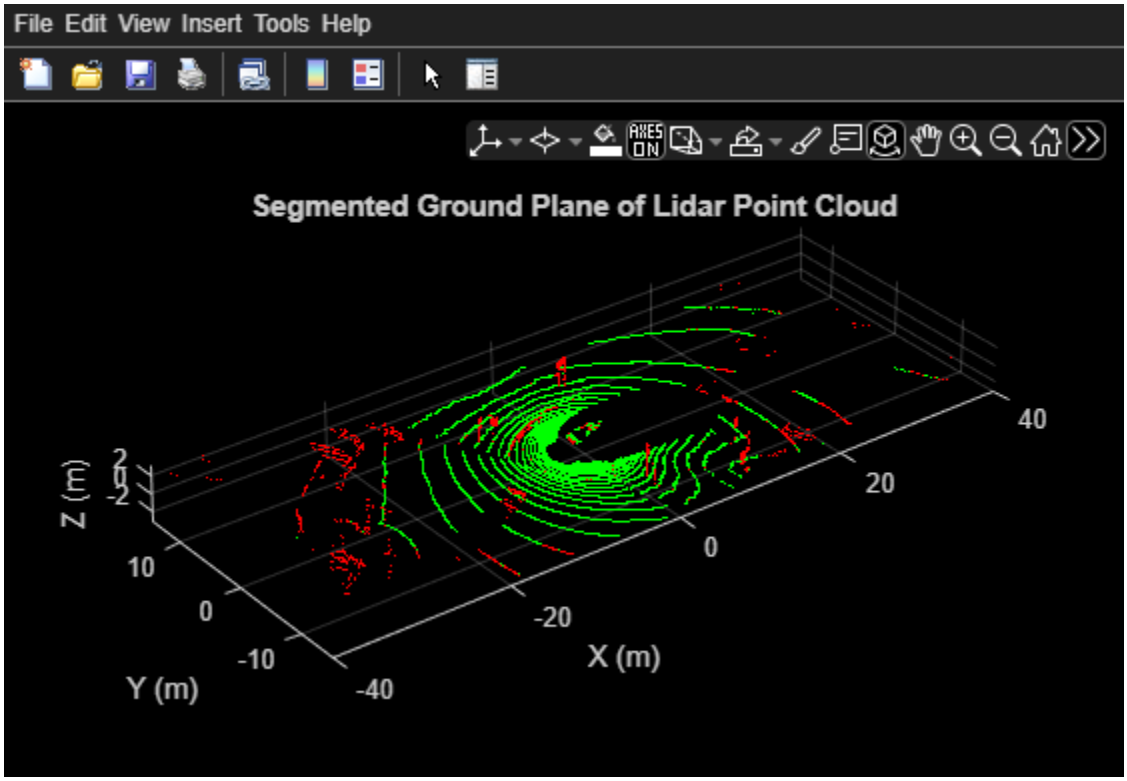
    % Create label array.
    colorLabels = zeros(size(ptCloud.Location,1),size(ptCloud.Location,2));

    % Find the ground points.
    groundPtsIdx = segmentGroundFromLidarData(ptCloud);

    % Map color ground points to green.
    colorLabels(groundPtsIdx (:)) = greenIdx;

    % Map color nonground points to red.
    colorLabels(~groundPtsIdx (:)) = redIdx;

    % Plot the results.
    view(player,ptCloud.Location,colorLabels)
end
```



Input Arguments

ptCloud — Point cloud
`pointCloud` object

Point cloud, specified as a `pointCloud` object. `ptCloud` is an organized point cloud that stores [*x*,*y*,*z*] point coordinates in an *M*-by-*N*-by-3 matrix.

Name-Value Pair Arguments

Example: `ElevationAngleDelta=5` sets the elevation angle threshold to 5 degrees.

`ElevationAngleDelta` — Elevation angle difference threshold

5 (default) | nonnegative scalar

Elevation angle difference threshold to identify ground points, specified as a nonnegative scalar. The function computes the elevation angle difference between one labeled ground point and its 4-connected neighbors. The neighborhood point is labeled as ground if the difference is below the threshold. Typical values for `ElevationAngleDelta` are in the range of [5,15] degrees. Increase this value to encompass more points from uneven ground surfaces.

`InitialElevationAngle` — Initial elevation angle threshold

`30` (default) | non-negative scalar

Initial elevation angle threshold to identify the ground point in the scanning line closest to the lidar sensor, specified as a non-negative scalar. The function marks a point as ground when the elevation angle falls below this value. Typical values for `InitialElevationAngle` are in the range of 15 and 30 degrees.

Output Arguments

`groundPtsIdx` — Ground points index

logical matrix

Ground points index, returned as an *M*-by-*N* logical matrix. Elements with a `true` value, `1`, indicate ground points. Elements with a `false` value, `0`, indicate nonground points.

Version History

Introduced in R2018b

References

[1] Bogoslavskyi, I. “Efficient Online Segmentation for Sparse 3D Laser Scans.” *Journal of Photogrammetry, Remote Sensing and Geoinformation Science*. Vol. 85, Number 1, 2017, pp. 41–52.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Because of architectural differences between the CPU and GPU, numerical verification does not always match. The GPU floating-point units use fused Floating-Point Multiply-Add (FMAD)

instructions while the CPU does not use these instructions. The CUDA compiler performs these instruction-level optimizations by default impacting the accuracy of the computed results. For example, the CUDA compiler fuses floating-point multiply and add instructions into a single instruction. This Floating-point Multiply-Add (FMAD) operation executes twice as fast compared to two single instructions but results in the loss of numerical accuracy. You can achieve tighter control over these optimizations by using compiler flags. For example, passing `--fmad=false` to the `Compiler Flags` property of the code configuration object instructs the `nvcc` compiler to disable contraction of floating-point multiply and add to a single Floating-Point Multiply-Add (FMAD) instruction. For more information, see “Generate Code by Using the GPU Coder App” (GPU Coder).

See Also

Objects
`pointCloud` | `velodyneFileReader` | `planeModel`

Functions
`pcsegdist` | `pcfitplane` | `segmentLidarData` | `pcshow`

focalCrossEntropy

Compute focal cross-entropy loss

Syntax

```
dlY = focalCrossEntropy(dlX,targets)
dlY = focalCrossEntropy(dlX,targets,"DataFormat",FMT)
dlY = focalCrossEntropy( ___ ,Name=Value)
```

Description

`dlY = focalCrossEntropy(dlX,targets)` computes the focal cross-entropy between network predictions and target values for single-label and multi-label classification tasks. The classes are mutually-exclusive classes. The focal cross-entropy loss weights towards poorly classified training samples and ignores well-classified samples. The focal cross-entropy loss is computed as the average logarithmic loss divided by number of non-zero targets.

`dlY = focalCrossEntropy(dlX,targets,"DataFormat",FMT)` also specifies the dimension format `FMT` when `dlX` is not a formatted `dlarray`.

`dlY = focalCrossEntropy( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntaxes. For example, `ClassificationMode="multilabel"` computes the cross-entropy loss for a multi-label classification task.

Examples

Compute Focal Cross-Entropy Loss Using Formatted `dlarray`

Create the input classification data as 32 observations of random variables belonging to 10 classes or categories.

```
numCategories = 10;
observations = 32;
X = rand(numCategories,observations);
```

Create a formatted deep learning array that has a data format with the labels `'C'` and `'B'`.

```
dlX = dlarray(X,'CB');
```

Use the `softmax` function to set all values in the input data to values between `0` and `1` that sum to `1` over all channels. The values specify the probability of each observation to belong to a particular category.

```
dlX = softmax(dlX);
```

Create the target data as unformatted deep learning array, which holds the correct category for each observation in dlX. Set the targets belonging to the second category as one-hot encoded vectors.

```
targets = dlarray(zeros(numCategories,observations));
targets(2,:) = 1;
```

Compute the focal cross-entropy loss between each prediction and the target.

```
dlY = focalCrossEntropy(dlX,targets,'Reduction','none');
```

Compute Average Focal Cross-Entropy Loss Using Unformatted `dlarray`

Create the input classification data as 32 observations of random variables belonging to 10 classes or categories.

```
numCategories = 10;
observations = 32;
X = rand(numCategories,observations);
```

Create an unformatted deep learning array.

```
dlX = dlarray(X);
```

Use the `softmax` function to set all values in the input data to values between `0` and `1` that sum to `1` over all channels. The values specify the probability for each observation to belong to a particular category.

```
dlX = softmax(dlX,'DataFormat','CB');
```

Create the target data. Set the targets belonging to the second category as one-hot encoded vectors.

```
targets = zeros(numCategories,observations);
targets(2,:) = 1;
```

Compute the average of focal cross-entropy loss computed between the predictions and the targets.

```
dlY = focalCrossEntropy(dlX,targets,'DataFormat','CB')
```

```
dlY =
  1×1 dlarray

    0.4769
```

Compute Average Focal Cross-Entropy Loss for Multi-Label Classification

Create the input classification data as 32 observations of random variables belonging to 10 classes or categories.

```
numCategories = 10;
observations = 32;
X = rand(numCategories,observations);
```

Create a formatted deep learning array that has a data format with the labels 'C' and 'B'.

```
dlX = dlarray(X,'CB');
```

Use the `sigmoid` function to set all values in the input data to values between `0` and `1` that sum to `1` over all channels. The values specify the probability of each observation to belong to a particular category.

```
dlX = sigmoid(dlX);
```

Create the target data, which holds the correct category for each observation in `dlX`. Set the targets belonging to the second and sixth category as one-hot encoded vectors.

```
targets = zeros(numCategories,observations);
targets(2,:) = 1;
targets(6,:) = 1;
```

Compute the average of focal cross-entropy loss computed between the predictions and the targets. Set the '`ClassificationMode`' value to '`multilabel`' for multi-label classification.

```
dlY = focalCrossEntropy(dlX,targets,'ClassificationMode','multilabel')
```

```
dlY =
  1×1 dlarray

    2.4362
```

Input Arguments

`dlX` — Predictions

`dlarray` | numeric array

Predictions, specified as a `dlarray` with or without dimension labels or a numeric array. When `dlX` is not a formatted `dlarray`, you must specify the dimension format using the `DataFormat` name-value argument. If `dlX` is a numeric array, `targets` must be a `dlarray`.

Data Types: `single` | `double`

`targets` — Target classification labels

`dlarray` | numeric array

Target classification labels, specified as a formatted or unformatted `dlarray` or a numeric array.

If `targets` is a formatted `dlarray`, its dimension format must be the same as the format of `dlX`, or the same as the `DataFormat` name-value argument if `dlX` is unformatted.

If `targets` is an unformatted `dlarray` or a numeric array, the size of `targets` must exactly match the size of `dlX`. The format of `dlX` or the value of `DataFormat` is implicitly applied to `targets`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `ClassificationMode="multilabel",DataFormat="CB"` evaluates the focal cross-entropy loss for multi-label classification tasks and specifies the dimension order of the input data as `"CB"`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"ClassificationMode","multilabel","DataFormat","CB"` evaluates the focal cross-entropy loss for multi-label classification tasks and specifies the dimension order of the input data as `"CB"`

`Gamma` — Focusing parameter
`2` (default) | positive real number

Focusing parameter of the focal loss function, specified as a positive real number. Increasing the value of `Gamma` increases the sensitivity of the network to misclassified observations.

`Alpha` — Balancing parameter
`0.25` (default) | positive real number

Balancing parameter of the focal loss function, specified as a positive real number. The `Alpha` value scales the loss function linearly and is typically set to `0.25`. If you decrease `Alpha`, increase `Gamma`.

`Reduction` — Type of output loss
`"mean"` (default) | `"none"`

Type of output loss, specified as one of the following:

- `"mean"` — Average of output loss for each prediction. The function computes the average of loss values computed for each predictions in input `dlX`. The function returns the average loss as an unformatted `dlarray`. Observations with all zero target values along the channel dimension are excluded from computing the average loss.
- `"none"` — Output loss for each prediction. The function returns the loss values for each observation in `dlX`. The samples for computing focal cross-entropy loss also contains observations whose target values are all zeros along the channel dimension. If `dlX` is a formatted `dlarray`, output `dlY` is a formatted `dlarray` with same dimension labels as `dlX`. If `dlX` is an unformatted `dlarray`, output `dlY` is an unformatted `dlarray`.

Data Types: `char` | `string`

`ClassificationMode` — Type of classification task
`"single-label"` (default) | `"multilabel"`

Type of classification task, specified as one of these values:

- `"single-label"` — Each observation in the predictions `dlX` is exclusively assigned to one category (single-label classification).
- `"multilabel"`— Each observation in the predictions `dlX` can be assigned to one or more independent categories (multilabel classification).

Note To select the classification mode for binary classification, you must consider the output layer of the network:

- If the final layer has an output size of one, such as with a sigmoid layer, use `"multilabel"`.

- If the final layer has an output size of two, such as with a softmax layer, use `"single-label"`.

`DataFormat` — Dimension order of unformatted data
char vector | string

Dimension order of unformatted input data, specified as a character vector or string `FMT` that provides a label for each dimension of the data. Each character in `FMT` must be one of the following:

- `"S"` — Spatial
- `"C"` — Channel
- `"B"` — Batch (for example, samples and observations)
- `"T"` — Time (for example, sequences)
- `"U"` — Unspecified

You can specify multiple dimensions labeled `"S"` or `"U"`. You can use the labels `"C"`, `"B"`, and `"T"` at most once.

You must specify `DataFormat` when the input data `dlX` is not a formatted `dlarray`.

Example: `DataFomat="SSCB"`

Data Types: `char` | `string`

Output Arguments

`dlY` — Focal cross-entropy loss
`dlarray` scalar

Focal cross-entropy loss, returned as a `dlarray` scalar without dimension labels. The output `dlY` has the same underlying data type as the input `dlX`.

Version History

Introduced in R2020b

R2023b: `TargetCategories` is not recommended
Not recommended starting in R2023b

`TargetCategories` is not recommended. Use `ClassificationMode` instead. To update your code, replace all instances of `TargetCategories="exclusive"` with `ClassificationMode="single-label"` and all instances of `TargetCategories="independent"` with `ClassificationMode="multilabel"`. There are no differences between the properties that require additional updates to your code. The default behavior of the `focalcrossentropy` function remains the same.

See Also

`softmax` | `sigmoid` | `crossentropy` | `mse`

Topics
“Lidar 3-D Object Detection Using PointPillars Deep Learning” (Lidar Toolbox)

bboxerase

Remove bounding boxes

Syntax

```
bboxB = bboxerase(bboxA,window)
[bboxB,indices] = bboxerase(bboxA,window)
___ = bboxerase( ___ ,EraseThreshold=threshold)
```

Description

`bboxB = bboxerase(bboxA,window)` removes bounding boxes in the input `bboxA` that lie within a region of interest (ROI) specified by `window`. The output is the set of bounding boxes retained from the input `bboxA`. This function supports 2-D and 3-D bounding boxes.

Note To perform random erase or cutout data augmentation, use `bboxerase` along with the `imerase` function.

`[bboxB,indices] = bboxerase(bboxA,window)` also returns the indices of the bounding boxes retained from the input set of bounding boxes `bboxIn`.

`___ = bboxerase( ___ ,EraseThreshold=threshold)` additionally specifies the threshold for the amount of overlap between a bounding box region and the specified ROI. A bounding box is removed if the overlap between the bounding box region and the ROI is equal to or greater than the specified threshold.

Examples

Perform Random Erase Augmentation

Read an image.

```
I = imread('peppers.png');
```

Define bounding boxes and labels.

```
bboxA = [410 230 100 90;
         186 78 80 60];
labelsA = ["garlic";"onion"];
```

Find the size of the input image

```
inputSize = size(I);
```

To randomly select a region of interest (ROI), specify a range for the scale and the aspect ratio of the ROI. The scale value for the ROI is set to lie in the range 0.2 and 0.3. Similarly, the minimum value for the aspect ratio is set to lie in the range 1:10 and the maximum value for the aspect ratio is set to lie in the range 30:100.

```
scale = [0.2,0.3];
dimensionRatio = [1,10;30,100];
```

Select the ROI by using the `randomWindow2d` function.

```
window = randomWindow2d(inputSize,'Scale',scale,'DimensionRatio',dimensionRatio);
```

Remove the pixels and the bounding boxes that lie within the randomly selected ROI.

```
J = imerase(I,window);
[bboxB,indices] = bboxerase(bboxA,window);
labelsB = labelsA(indices);
```

Display the original and augmented image.

```
annotatedI = insertObjectAnnotation(I,'Rectangle',bboxA,labelsA);
annotatedJ = insertObjectAnnotation(J,'Rectangle',bboxB,labelsB);
figure
montage({annotatedI,annotatedJ})
title(['Input | Random Erase Output'])
```



Perform Cutout Data Augmentation

Read an image.

```
I = imread('visionteam1.jpg');
```

Define bounding boxes and labels.

```
bboxA = [64 101 117 440;
193 101 67 309;
```

```
282 86 114 375;
618 118 79 345;
486 55 131 528;
475 109 66 361];

labelsA = ["Person 1";"Person 2";"Person 3";"Person 4";"Person 5";"Person 6"];
```

Specify a rectangular region of interest.

```
window = [470 100 90 360];
```

Erase the pixels that lie within the rectangular region by using `imerase` function.

```
J = imerase(I,window);
```

Erase the bounding boxes that lie within the rectangular region. Set the erase threshold value to 0.7. The `bboxerase` function returns the retained bounding boxes and the corresponding indices.

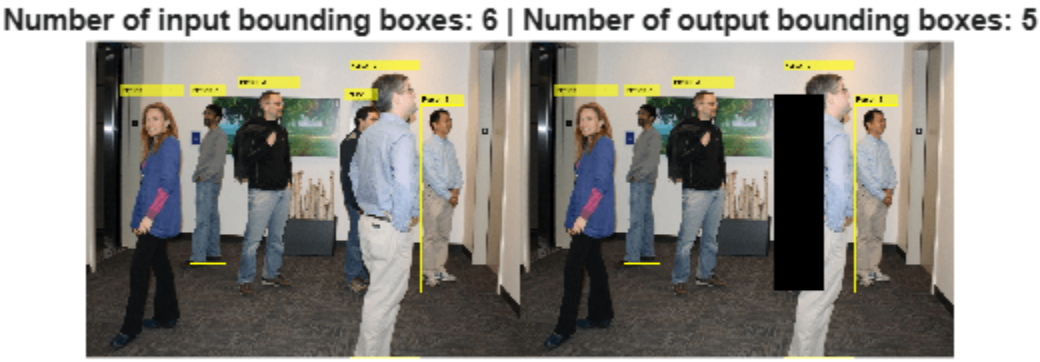
```
[bboxB,indices] = bboxerase(bboxA,window,'EraseThreshold',0.7);
```

Read the class labels corresponding to the retained bounding boxes.

```
labelsB = labelsA(indices);
```

Display the results.

```
figure
I = insertObjectAnnotation(I,'Rectangle',bboxA,labelsA);
J = insertObjectAnnotation(J,'Rectangle',bboxB,labelsB);
imshowpair(I,J,'montage')
title(['Number of input bounding boxes: ', num2str(length(labelsA)), ' | ', 'Number of output bou
```



Input Arguments

`bboxA` — Input set of bounding boxes

M-by-4 matrix

Input set of bounding boxes, specified as *M*-by-4 matrix. *M* is the number of bounding boxes. Each row in the matrix of form [x_{min} y_{min} *width height*]. [x_{min} y_{min}] are the top left coordinates of the bounding box. *width* and *height* are the width and the height of the bounding box respectively.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`window` — Region of interest

4-element vector | `Rectangle` object

Region of interest, specified as one of these values:

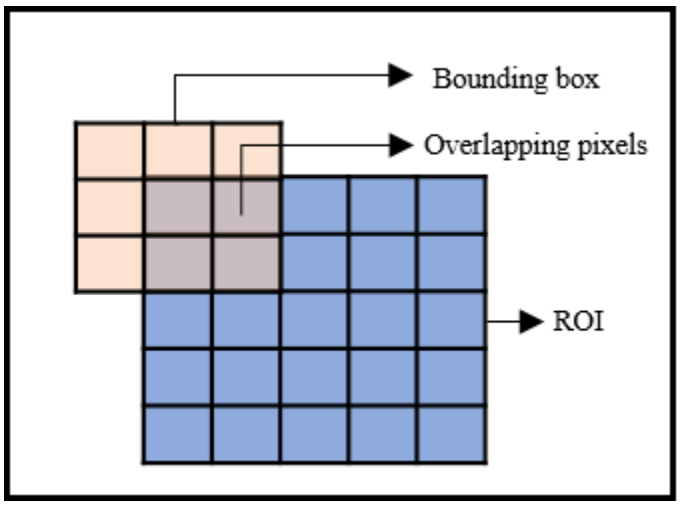
- 4 -element vector of form [*x y width height*]. [*x y*] are the top left coordinates of the ROI. *width* and *height* are the width and the height of the ROI respectively.
- `Rectangle` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`threshold` — Erase threshold

0.8 (default) | positive scalar

Erase threshold, specified as a positive scalar less than or equal to 1. The erase threshold sets the criteria for removing a bounding box from the input set `bboxA`. The value is the ratio of number of pixels in a bounding box that lie inside the ROI (overlapping pixels) to the total number of pixels in the bounding box.



$$threshold = \frac{Number\ of\ overlapping\ pixels}{Total\ number\ of\ pixels\ in\ bounding\ box}$$

The default value is 0.8. This implies that, a bounding box is removed if 80 % of the pixels in the bounding box lie within the specified ROI.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

bboxB — Output bounding boxes retained from the input
N-by-4 matrix

Output bounding boxes retained from the input, returned as a *N*-by-4 matrix. *N* is the number of bounding boxes retained from the input. Each row of the matrix defines one bounding box of the same type as the input `bboxA`. The value of *N* is always less than or equal to *M*, the number of input bounding boxes.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`indices` — Indices of retained bounding boxes
vector

Indices of retained bounding boxes, returned as a vector of integers. The indices indicate which bounding boxes in the input, `bboxA`, are retained and returned at the output.

Version History

Introduced in R2021a

See Also

`imerase` | `Rectangle` | `randomWindow2d` | `imref2d` | `affineOutputView` | `insertObjectAnnotation`

Object Detector Analyzer

Interactively visualize and evaluate object detection results against ground truth

Description

The Object Detector Analyzer app enables you to visualize and evaluate object detection results against ground truth data. Using the app, you can:

- Evaluate object detection performance either without ground truth data or by comparing detections to ground truth and generating performance metrics. To get started, see Get Started with Object Detector Analyzer App.
- Run a pretrained object detector in the app, or import precomputed object detection results from the workspace. For a list of supported object detectors to run in the app, see "Run Supported Object Detectors" on page 3-1581. For information about the format of precomputed detection results, see "Import Precomputed Detection Results" on page 3-1582.
- Visualize and compare detections and ground truth annotations in an interactive image browser. Inspect individual detections, and overlay their confidence scores, false positives, and false negatives overlaid on images.
- Compute and visualize detector performance metrics: precision-recall curves, confusion matrices, AP, mAP, and mLAMR across data sets, classes, and overlap thresholds, and detection performance by object area. For more information on performance metrics, see "Evaluate Object Detector Performance".
- Interactively adjust the detection threshold and overlap (IoU) threshold to analyze how stricter or more lenient thresholds impact detector performance, enabling you to tune your detector for the optimal trade-off between false negatives and false positives.
- Export all detections or filtered results to the workspace for further analysis.
- Export computed performance metrics as an `objectDetectionMetrics` object. You can use this object to create custom visualizations, compare different detector models, or perform further performance analysis.

To learn more about this app, see Get Started with Object Detector Analyzer App.

Open the Object Detector Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `objectDetectorAnalyzer`.

Examples

Visualize Object Detections on a Video

This example shows how to visualize object detections on a video using the Object Detector Analyzer App.

Load detector.

```
detector = peopleDetector();
```

Load a video using `VideoReader`.

```
videoURL = "https://ssd.mathworks.com/supportfiles/vision/data/PedestrianTrackingVideo.avi";
videoFilename = "PedestrianTrackingVideo.avi";
if ~exist("PedestrianTrackingVideo.avi","file")
disp("Downloading Pedestrian Tracking Video (35 MB)")
    websave(videoFilename,videoURL);
end

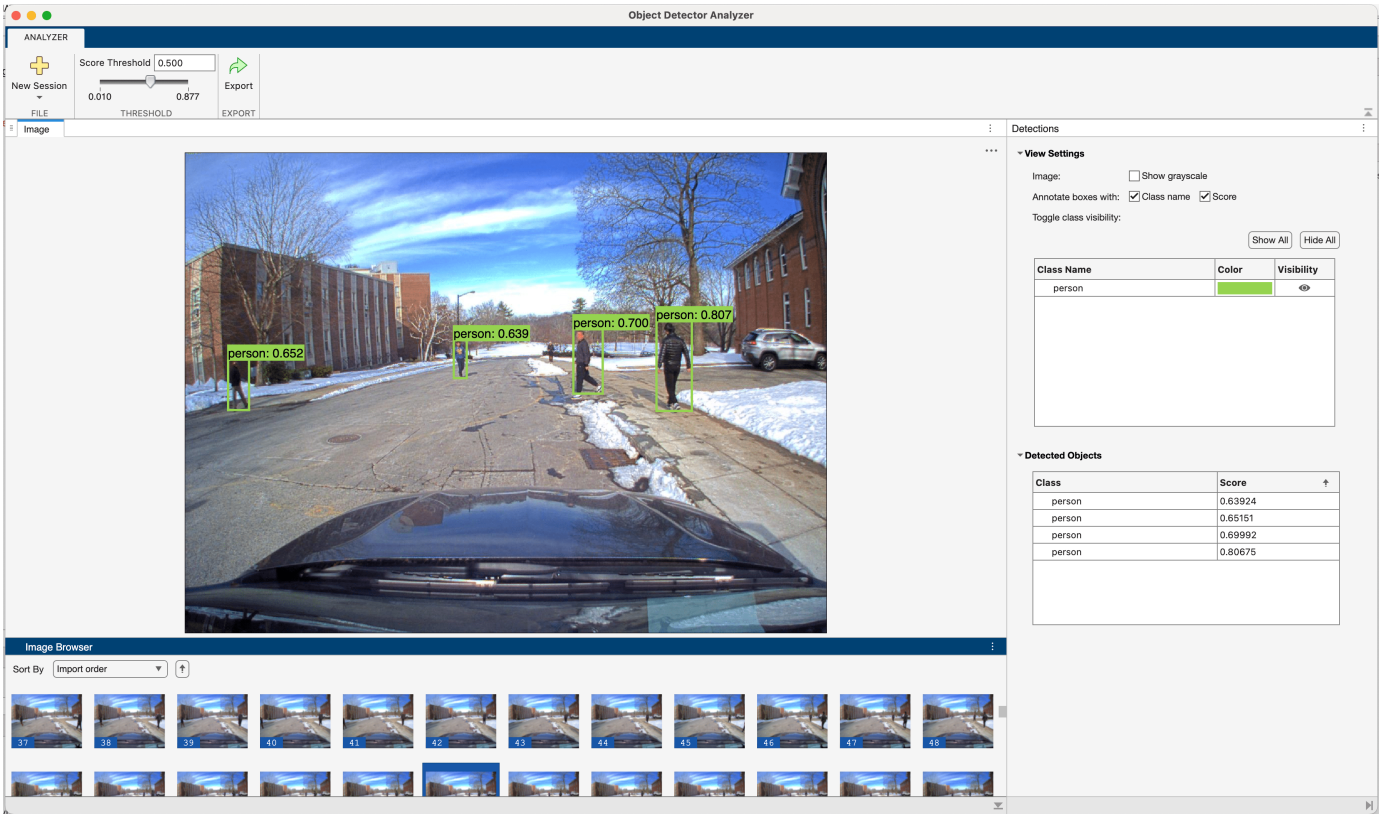
reader = VideoReader(videoFilename);
```

The Object Detector Analyzer App can import any datastore that produces image data. Use an `arrayDatastore` to produce video frame indices and create a `transform` datastore that calls `VideoReader` `read` object function with the frame indices returned by the `arrayDatastore`. This will read one frame of video data every time data is read from the datastore.

```
frameIndexDS = arrayDatastore((1:reader.NumFrames)',IterationDimension=1,OutputType="same");
videoFrameDS = transform(frameIndexDS,@(frameIdx)read(reader,frameIdx));
```

Launch Object Detector Analyzer App with the detector and the datastore as input. The app will run the detector on the video frames and display the detections on each frame. By default, the video frames are sorted by the number of detections in each frame. You can sort the video frames by Import order using the Sort By dropdown to visualize the video frames in sequence from start to finish.

```
objectDetectorAnalyzer(detector,videoFrameDS)
```



- Get Started with Object Detector Analyzer App
- “Multiclass Object Detection Using YOLO v2 Deep Learning”
- “Visualize Object Detection Results from Pretrained PyTorch Model”

Programmatic Use

`objectDetectorAnalyzer` opens a new session of the app. To start a new analysis session, select **New Session**, then **Evaluate detections without ground truth** or **Evaluate detections against ground truth**.

`objectDetectorAnalyzer(results,imds)` opens the app and visualizes the object detection results, `results`, on all of the test images from the `ImageDatastore` object `imds`. You must specify `results` as a table with *N* rows, where *N* is the number of images, and three columns in this order:

- Bounding boxes for the corresponding image, specified as an *M*-by-4 matrix. Each row of the matrix contains a bounding box of the form [*x y width height*], where *x*, `y`, *width*, *height* specify the upper-left corner coordinates, width, and height of each bounding box, respectively.
- Detection confidence scores for the corresponding image, specified as an *M*-by-1 vector.
- Object labels for the corresponding image, specified as an *M*-by-1 categorical array of class names which correspond to each bounding box.

M is the number of bounding boxes in each image.

`objectDetectorAnalyzer(results,groundTruthData)` opens the app, evaluates the object detection results, `results`, and visualizes them on all of the ground truth images, `groundTruthData`. Specify the ground truth data as a datastore or a `groundTruth` object. You can create a `groundTruth` object by exporting labels you create using the Image Labeler app into the workspace. If `groundTruthData` is a datastore, you must format the datastore such that calling the `read` function on it returns a cell array with these columns.

- The first column must contain the RGB, grayscale, or binary image data for each image.
- The second column must contain the bounding boxes for the corresponding image, specified as an *M*-by-4 matrix. *M* is the number of ground truth objects.
- The third column must contain the object class names for the corresponding image, specified as a categorical vector of size *M*-by-1. *M* is the number of ground truth objects.

More About

Run Supported Object Detectors

This table shows the supported pretrained detectors that you can run directly in the Object Detector Analyzer app. To select a pretrained object detector for your application, see “Choose an Object Detector”.

Detector	Object
Grounding DINO Object Detector	`groundingDinoObjectDetector`
RTM Object Detector	`rtmdetObjectDetector`

Detector	Object
YOLOX Object Detector	`yoloxObjectDetector`
YOLOv4 Object Detector	`yolov4ObjectDetector`
YOLOv3 Object Detector	`yolov3ObjectDetector`
YOLOv2 Object Detector	`yolov2ObjectDetector`
SSD Object Detector	`ssdObjectDetector`
People Detector	`peopleDetector`
RetinaFace Face Detector	`faceDetector`

Import Precomputed Detection Results

You can import precomputed detection results into the Object Detector Analyzer app.

Specify the precomputed detection results as a table with the same number of rows as the number of ground truth images, and these columns:

Bounding Boxes	Scores	Labels
A cell array containing an *M*-by-4 with rows in the format [*x y w h*]. *x* and *y* specify the upper-left corner of each rectangular bounding box. *w* specifies the box width, which is its length along the *x*-axis, and *h* specifies the box height, which is its length along the *y*-axis. *M* is the number of detected bounding boxes.	A cell array containing an *M*-by-1 numeric vector of detection confidence scores. *M* is the number of detected bounding boxes.	A cell array containing an *M*-by-1 categorical array of class labels. *M* is the number of detected bounding boxes.

To learn more, see Get Started with Object Detector Analyzer App.

Version History

Introduced in R2026a

See Also

Apps

Image Labeler | Video Labeler

Functions

`evaluateObjectDetection`

Objects

`groundTruth` | `rtmdetObjectDetector` | `yoloxObjectDetector` | `yolov4ObjectDetector` | `yolov3ObjectDetector` | `yolov2ObjectDetector` | `ssdObjectDetector` | `peopleDetector` | `faceDetector` | `groundTruthDataSource` | `labelDefinitionCreator`

Topics

Get Started with Object Detector Analyzer App
“Multiclass Object Detection Using YOLO v2 Deep Learning”
“Visualize Object Detection Results from Pretrained PyTorch Model”
“Evaluate Object Detector Performance”
“Automatically Label Ground Truth Using Segment Anything Model”
“Choose an Object Detector”
“Get Started with Object Detection Using Deep Learning”
“Training Data for Object Detection and Semantic Segmentation”

xyzquat

Convert transformation or rotation to compact 3-D pose representation

Syntax

```
pose = xyzquat(transformation)
pose = xyzquat(rotation)
```

Description

`pose = xyzquat(transformation)` converts a transformation `transformation` to a compact 3-D pose representation `pose`.

`pose = xyzquat(rotation)` converts a rotation `rotation` to a compact 3-D pose representation `pose` with no translation.

Examples

Convert SE(3) Transformation to 3-D Compact Pose

Create SE(3) transformation with an *xyz*-position of `[2 3 1]` and a rotation defined by a numeric quaternion. Use the `eul2quat` function to create the numeric quaternion.

```
trvec = [2 3 1];
quat1 = [0.9659 0.2588 0 0];
pose1 = [trvec quat1]
```

```
pose1 = 1×7

    2.0000    3.0000    1.0000    0.9659    0.2588         0         0
```

```
T = se3(pose1,"xyzquat")
```

```
T = se3
    1.0000         0         0    2.0000
         0    0.8660   -0.5000    3.0000
         0    0.5000    0.8660    1.0000
         0         0         0    1.0000
```

Convert the transformation back into a compact pose.

```
pose2 = xyzquat(T)
```

```
pose2 = 1×7

    2.0000    3.0000    1.0000    0.9659    0.2588         0         0
```

Convert SO(3) Rotation to 3-D Compact Pose

Create SO(3) rotation defined by a numeric quaternion. Use the `eul2quat` function to create the numeric quaternion.

```
quat1 = [0.9659 0.2588 0 0]
```

```
quat1 = 1×4

    0.9659    0.2588         0         0
```

```
R = so3(quat1,"quat")
```

```
R = so3
    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660
```

Convert the rotation into a 3-D compact pose.

```
pose1 = xyzquat(R)
```

```
pose1 = 1×7

         0         0         0    0.9659    0.2588         0         0
```

Input Arguments

`transformation` — Transformation

`se3` object | *N*-element array of `se3` objects

Transformation, specified as an `se3` object or as an *N*-element array of `se3` objects. *N* is the total number of transformations.

`rotation` — Rotation

`so3` object | *N*-element array of `so3` objects

Rotation, specified as an `so3` object or as an *N*-element array of `so3` objects. *N* is the total number of rotations.

Output Arguments

`pose` — 3-D compact pose

M-by-3 matrix

3-D compact pose, returned as an *M*-by-3 matrix, where each row is of the form [*x y z qw qx qy qz*]. *M* is the total number of transformations specified. *x*, *y*, *z* comprise the xyz-position and *qw*, *qx*, *qy*, and *qz* comprise the quaternion rotation.

See Also

`se3` | `so3`

anomalyMap

Predict per-pixel anomaly score map

Syntax

```
map = anomalyMap(detector,I)
map = anomalyMap(detector,I,ExecutionEnvironment=executionEnvironment)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`map = anomalyMap(detector,I)` creates an anomaly score map consisting of per-pixel anomaly scores predicted for test image `I`.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`map = anomalyMap(detector,I,ExecutionEnvironment=executionEnvironment)` also specifies the hardware resource on which to run the detector.

Examples

Display Anomaly Map over Test Image

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies.

```
load("digit8AnomalyDetector.mat");
```

Load test images and corresponding labels, then convert the test images to a formatted `dlarray` with the format label "SSCB" for spatial-spatial-channel-batch. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
```

Select a test image of the digit 9.

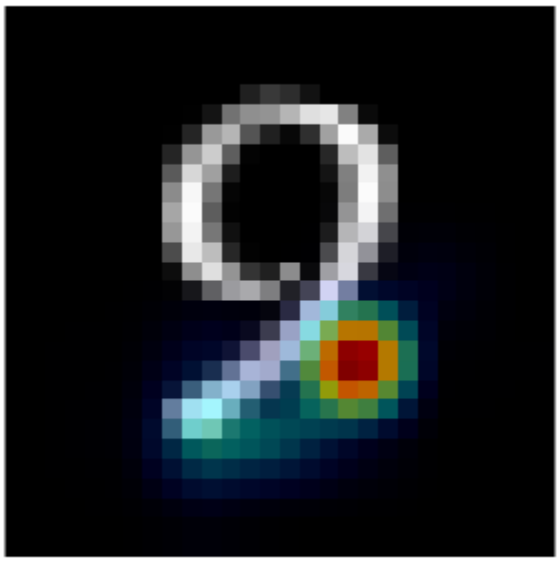
```
digitToTest = "9";
digitIdx = find(gtLabels==digitToTest,1);
imTest = Xtest(:,:,:,digitIdx);
```

Predict the per-pixel anomaly scores for the test image.

```
map = anomalyMap(detector,imTest);
```

Display the anomaly map over the test image.

```
overlay = anomalyMapOverlay(imTest,map);
imagesc(overlay)
axis image off
```



Input Arguments

`detector` — Anomaly detector

`efficientADAnomalyDetector` object | `fcddAnomalyDetector` object | `fastFlowAnomalyDetector` object | `patchCoreAnomalyDetector` object

Anomaly detector, specified as an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, a `fastFlowAnomalyDetector` object, or a `patchCoreAnomalyDetector` object.

`I` — Test image

numeric array | formatted `dlarray` object

Test image, specified in one of these formats:

Format	Supported Detectors
M-by-*N*-by-3 numeric array representing a truecolor image.	EfficientAD, FCDD, FastFlow, PatchCore
M-by-*N*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.	EfficientAD, FCDD, FastFlow, PatchCore
Formatted `dlarray` object with two spatial dimensions and one channel dimension. You can specify multiple test images by including a batch dimension.	EfficientAD, FCDD, FastFlow

FCDD anomaly detectors also support grayscale test images, with one color channel instead of three.

`executionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`map` — Anomaly score map
M-by-*N* numeric matrix | *M*-by-*N*-by-1-by-*B* numeric array | formatted `dlarray` object

Anomaly score map, returned in one of these formats:

- *M*-by-*N* numeric matrix, when the test image `I` is a 2-D grayscale or truecolor image.
- *M*-by-*N*-by-1-by-*B* numeric array representing a batch of images, when `I` is numeric array representing a batch of images.
- Formatted `dlarray` object, when `I` is a formatted `dlarray` object.

The values in the map are unnormalized. Larger values indicate a greater chance of a pixel being an anomaly. To normalize an anomaly map, use the `percentileNormalizer` object.

Data Types: `single`

Version History

Introduced in R2022b

R2024b: Support for EfficientAD anomaly detector

You can now specify the `detector` argument as an `efficientADAnomalyDetector` object.

R2023a: Support for FastFlow and PatchCore anomaly detectors

You can now specify the `detector` argument as a `fastFlowAnomalyDetector` object or a `patchCoreAnomalyDetector` object.

R2023a: Support for C and GPU code generation

`anomalyMap` now supports the generation of C code (requires MATLAB Coder) and optimized CUDA code (requires GPU Coder).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • The `executionEnvironment` argument is not supported.	• The datastore data format of `I` is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • GPU code generation does not support `gpuArray` inputs. • GPU code generation supports inputs that are defined as half-precision floating point data types. For more information, see `half`. • The `executionEnvironment` argument is not supported.	• The datastore data format of `I` is not supported.

See Also

`anomalyMapOverlay` | `percentileNormalizer` | `efficientADAnomalyDetector` | `fcddAnomalyDetector` | `fastFlowAnomalyDetector` | `patchCoreAnomalyDetector` | `classify` | `predict`

anomalyMapOverlay

Overlay heatmap on image using per-pixel anomaly scores

Syntax

```
B = anomalyMapOverlay(I,map)
B = anomalyMapOverlay(I,map,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`B = anomalyMapOverlay(I,map)` fuses an overlay of the per-pixel anomaly scores in `map` as a heatmap on a grayscale or RGB background image, `I`.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`B = anomalyMapOverlay(I,map,Name=Value)` specifies options using one or more name-value arguments. For example, `anomalyMapOverlay(I,map,Blend="equal")` blends the score map and the background image with equal weighting.

Examples

Display Anomaly Map over Test Image

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies.

```
load("digit8AnomalyDetector.mat");
```

Load test images and corresponding labels, then convert the test images to a formatted `dlarray` with the format label "SSCB" for spatial-spatial-channel-batch. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
```

Select a test image of the digit 9.

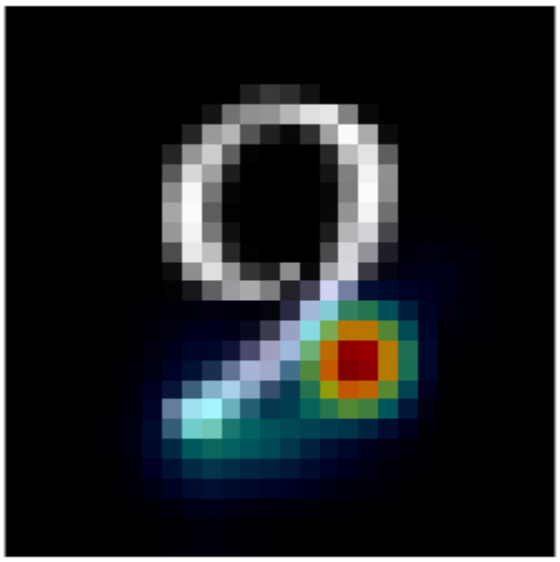
```
digitToTest = "9";
digitIdx = find(gtLabels==digitToTest,1);
imTest = Xtest(:,:,:,digitIdx);
```

Predict the per-pixel anomaly scores for the test image.

```
map = anomalyMap(detector,imTest);
```

Display the anomaly map over the test image.

```
overlay = anomalyMapOverlay(imTest,map);
imagesc(overlay)
axis image off
```



Input Arguments

`I` — Background image

M-by-*N*-by-3 numeric array | *M*-by-*N* numeric matrix | *M*-by-*N* logical matrix

Background image, specified in one of these formats:

- *M*-by-*N*-by-3 numeric array representing a truecolor image
- *M*-by-*N* numeric matrix representing a grayscale image
- *M*-by-*N* logical matrix representing a binary image

`map` — Anomaly score map

M-by-*N* numeric matrix

Anomaly score map, specified as an *M*-by-*N* numeric matrix. You can get an anomaly score map using the `anomalyMap` function. Larger values indicate a greater chance that a pixel is an anomaly.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `B = anomalyMapOverlay(I,map,Blend="equal")` blends the score map and the background image with equal weighting.

MapRange — Display range
two-element vector

Display range of the anomaly score map, `map`. The default value is `[min(map(:)) max(map(:))]`.

Blend — Weights to blend map with background image
`"proportional"` (default) | `"equal"` | numeric scalar

Weights to blend the map with background image `I`, specified as `"proportional"`, `"equal"`, or a numeric scalar in the range [0, 1].

Value	Description
`"proportional"`	Overlay `I` and `map` using alpha blending with per-pixel weights proportional to the corresponding values in `map`. The heatmap is more opaque over regions of interest, which have large anomaly scores. The heatmap is more transparent over normal regions, which have small anomaly scores.
`"equal"`	Overlay `I` and `map` using alpha blending with equal weights for the map and background image.
numeric scalar	Overlay `I` and `map` using alpha blending with weight *w* for the map, where *w* is the value of the `Blend` argument. When *w* is `0`, the heatmap is fully transparent and only the background image is visible. When you increase the value of *w*, the heatmap becomes more opaque.

For all values of `Blend`, the weight of the heatmap has an additional multiplicative scale factor, specified by `BlendFactor`. When you account for the additional scale factor, the weight of the heatmap is *w**`BlendFactor` and the weight of the background image is 1 – *w**`BlendFactor`.

BlendFactor — Constant multiplicative scale factor
`1` (default) | numeric scalar

Constant multiplicative factor to apply to the weights of the `Blend` argument, specified as a numeric scalar. Increasing the value of `BlendFactor` increases the brightness of the heatmap over the image.

Example: `BlendFactor=1.2` brightens the heatmap by increasing the weighting by a factor of `1.2`.

Example: `BlendFactor=0.6` darkens the heatmap by decreasing the weighting by a factor of `0.6`.

Colormap — Colormap
`jet(256)` (default) | 256-by-3 numeric matrix

Colormap, specified as a 256-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `single` | `double`

Output Arguments

B — RGB image with heatmap overlay
M-by-*N*-by-3 numeric array

RGB image with heatmap overlay, returned as an *M*-by-*N*-by-3 numeric array.

Data Types: `uint8`

Version History

Introduced in R2022b

See Also

`anomalyMap` | `percentileNormalizer` | `viewAnomalyDetectionResults` | `fcddAnomalyDetector`

anomalyThreshold

Optimal anomaly threshold for set of anomaly scores and corresponding labels

Syntax

```
t = anomalyThreshold(gtLabels,scores,anomalyLabels)
t = anomalyThreshold(gtLabels,scores,anomalyLabels,optimMethod)
t = anomalyThreshold(gtLabels,scores,anomalyLabels,MaxFalsePositiveRate=
maxFPR)
t = anomalyThreshold(gtLabels,scores,anomalyLabels,MaxFalseNegativeRate=
maxFNR)
[t,anomalyROC] = anomalyThreshold( ___ )
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`t = anomalyThreshold(gtLabels,scores,anomalyLabels)` calculates the optimal anomaly threshold given per-image anomaly scores and corresponding ground truth labels. `anomalyLabels` indicates which class labels in `gtLabels` belong to the anomaly (positive) class. When performing anomaly detection, images with scores below the calculated threshold are considered normal images and images with scores above the threshold are considered anomalous images.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`t = anomalyThreshold(gtLabels,scores,anomalyLabels,optimMethod)` also specifies the optimization method.

`t = anomalyThreshold(gtLabels,scores,anomalyLabels,MaxFalsePositiveRate=maxFPR)` also specifies the maximum false positive rate.

`t = anomalyThreshold(gtLabels,scores,anomalyLabels,MaxFalseNegativeRate=maxFNR)` also specifies the maximum false negative rate.

`[t,anomalyROC] = anomalyThreshold( ___ )` also returns the receiver operating characteristic (ROC) curve and performance metrics.

Examples

Calculate Optimal Anomaly Threshold

Load calibration images and corresponding labels, then create a datastore that reads the calibration data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xcal,gtLabels] = digitTest4DArrayData;
dsCal = arrayDatastore(Xcal,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of anomaly labels as the set of digits between 0 and 9, excluding 8.

```
load("digit8AnomalyDetector.mat");
anomalyLabels = setdiff(string(0:9),"8");
```

Predict the anomaly score of each calibration image.

```
scores = predict(detector,dsCal);
```

Calculate the optimal threshold and corresponding ROC metrics from the anomaly scores and ground truth labels.

```
[T,roc] = anomalyThreshold(gtLabels,scores,anomalyLabels)
```

```
T = single

2.9632e-04

roc =
  rocmetrics with properties:

    Metrics: [4980×4 table]

Properties, Methods
```

Set the `Threshold` property of the FCDD anomaly detector as the optimal threshold.

```
net.Threshold = T;
```

Input Arguments

`gtLabels` — Ground truth labels

numeric vector | logical vector | categorical vector

Ground truth labels for each image, specified as a numeric vector, logical vector, or categorical vector. The `anomalyThreshold` function converts the labels into a logical vector according to the set of anomaly labels in `anomalyLabels`.

`scores` — Predicted anomaly scores

numeric vector

Predicted anomaly score for each image, specified as a numeric vector of the same length as `gtLabels`.

`anomalyLabels` — Anomaly labels

numeric vector | logical vector | categorical vector | string vector

Anomaly labels, specified as a vector of the same data type as `gtLabels`. When `gtLabels` is categorical, `anomalyLabels` can be of data type string whose values correspond to categories in `gtLabels`.

The `anomalyThreshold` function converts all ground truth labels in `gtLabels` that belong to the set of anomaly labels to a logical `true`, indicating an anomaly (positive detection). The function converts all other ground truth labels to a logical `false`, indicating normality (negative detection).

`optimMethod` — Optimization method
`"YoudensIndexROC"` (default) | `"NearestZeroOneROC"` | `"MaxF1Score"`

Optimization method, specified as one of the values in the table.

Optimization Method	Description
`"YoudensIndexROC"`	The optimal threshold is the point on the ROC curve with the maximum difference between the true positive rate (TPR) and false positive rate (FPR). The difference between TPR and FPR is also known as Youden's index or Youden's J statistic.
`"NearestZeroOneROC"`	The optimal threshold is the point on the ROC curve nearest to the coordinate (0, 1), where the FPR is 0 and the TPR is 1. This point minimizes the Euclidean distance between the ROC curve and the (0,1) point.
`"MaxF1Score"`	The optimal threshold is the point on the precision-recall (PR) curve that maximizes the F1 score. This value is the harmonic mean of the precision and recall values.

Data Types: `char` | `string`

`maxFPR` — Maximum false positive rate
`0` (default) | number in the range [0, 1]

Maximum false positive rate (FPR), specified as a number in the range [0, 1]. The optimal threshold is the point on the ROC curve where the FPR is less than or equal to `maxFPR`.

`maxFNR` — Maximum false negative rate
`0` (default) | number in the range [0, 1]

Maximum false negative rate (FNR), specified as a number in the range [0, 1]. The optimal threshold is the point on the ROC curve where the FNR is less than or equal to `maxFNR`.

Output Arguments

`t` — Optimal threshold
numeric scalar

Optimal threshold, returned as a numeric scalar of the same data type as `scores`.

`anomalyROC` — ROC curve and performance metrics
`rocmetrics` object

ROC curve and performance metrics, returned as a `rocmetrics` object.

Tips

- You can plot ROC and PR curves returned by `anomalyROC` using the `plot` function.

Version History

Introduced in R2022b

See Also

`fcddAnomalyDetector` | `patchCoreAnomalyDetector` | `fastFlowAnomalyDetector` | `efficientADAnomalyDetector` | `rocmetrics` | `anomalyMap`

Topics

“Identify Defects in Air Compressors Using Spectrogram Images”
“Getting Started with Anomaly Detection Using Deep Learning”
“ROC Curve and Performance Metrics” (Deep Learning Toolbox)

classify

Classify image as normal or anomalousJAB

Syntax

```
isAnomaly = classify(detector,I)
isAnomaly = classify(detector,I,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`isAnomaly = classify(detector,I)` predicts whether test image `I` is anomalous or normal, given an anomaly detector with an anomaly threshold.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`isAnomaly = classify(detector,I,Name=Value)` specifies options using one or more name-value arguments. For example, `classify(detector,I,MiniBatchSize=32)` limits the batch size to 32.

Examples

Calculate Metrics for Trained FCDD Anomaly Detector

Load test images and corresponding labels, then create a datastore that reads the test data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
dsTest = arrayDatastore(Xtest,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of anomaly labels as the set of digits between 0 and 9, excluding 8.

```
load("digit8AnomalyDetector.mat");
anomalyLabels = setdiff(string(0:9),"8");
```

Classify each test image as normal or anomalous.

```
predLabels = classify(detector,dsTest);
```

Calculate metrics for the anomaly detection results.

```
metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels)
```

```
Evaluating anomaly detection results
------------------------------------
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    Precision    Recall    Specificity    F1Score    FalsePosi
    ______________    ____________    _________    ______    ___________    _______    _________

        0.7662            0.839         0.98971     0.748        0.93        0.85204          0.07

metrics =
  anomalyDetectionMetrics with properties:

              ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
               DataSetMetrics: [1×8 table]
                 ClassMetrics: [2×2 table]
```

Display the confusion matrix.

```
cm = metrics.ConfusionMatrix
```

```
cm=2×2 table
               Normal    Anomaly
               ______    _______

    Normal       465         35
    Anomaly     1134       3366
```

Input Arguments

`detector` — Anomaly detector

`efficientADAnomalyDetector` object | `fcddAnomalyDetector` object | `fastFlowAnomalyDetector` object | `patchCoreAnomalyDetector` object

Anomaly detector, specified as an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, a `fastFlowAnomalyDetector` object, or a `patchCoreAnomalyDetector` object. You must set the `Threshold` property of the detector before calling this function.

`I` — Test image

numeric array | datastore | formatted `dlarray` object

Test image, specified in one of these formats:

Format	Supported Detectors
M-by-*N*-by-3 numeric array representing a truecolor image.	EfficientAD, FCDD, FastFlow, PatchCore
M-by-*N*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.	EfficientAD, FCDD, FastFlow, PatchCore

Format	Supported Detectors
Datastore that reads and returns truecolor images. The images must all have the same size.	EfficientAD, FCDD, FastFlow, PatchCore
Formatted `dlarray` object with two spatial dimensions and one channel dimension. You can specify multiple test images by including a batch dimension.	EfficientAD, FCDD, FastFlow

FCDD anomaly detectors also support grayscale test images, with one color channel instead of three.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `isAnomaly = classify(detector,I,MiniBatchSize=32)` limits the batch size to 32.

`MiniBatchSize` — Size of batches
128 (default) | positive integer

Size of batches for calculating predictions, specified as a positive integer. Larger batch sizes lead to faster processing but take up more memory.

`ScoreFunction` — Function for computing scalar score from anomaly map
function handle

Function for computing a scalar score from an anomaly map when `detector` is an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, or a `fastFlowAnomalyDetector` object, specified as a function handle. The function handle `ScoreFunction` must represent a function that accepts the input data `I` and returns an output in the form of a numeric scalar. `I` must be 2-D numeric image data, `dlarray` data with two spatial dimensions, or a datastore. The default value of `ScoreFunction` depends on the type of detector specified:

Detector Specified in `detector` Argument	Default `ScoreFunction` Value
FastFlow	`@(I)max(I,[],[1 2])`
EfficientAD	`@(I)max(I,[],[1 2 3])`
FCDD	`@(I)mean(I,[1 2])`
PatchCore	Not supported

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.

Resource	Action
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`isAnomaly` — Test image is anomaly
N-element logical vector

Test image is anomaly, returned as an *N*-element logical vector, where *N* is the number of test images in `I`. The value `true` indicates an anomaly (positive detection). The value `false` indicates normality (negative detection).

Data Types: `logical`

Version History

Introduced in R2022b

R2024b: Support for EfficientAD anomaly detector

You can now specify the `detector` argument as an `efficientADAnomalyDetector` object.

R2023a: Support for FastFlow and PatchCore anomaly detectors

You can now specify the `detector` argument as a `fastFlowAnomalyDetector` object or a `patchCoreAnomalyDetector` object.

R2023a: Support for C and GPU code generation

`classify` now supports the generation of C code (requires MATLAB Coder) and optimized CUDA code (requires GPU Coder).

R2024a: Option to specify custom anomaly score function for FastFlow and FCDD detectors

Specify a custom anomaly score function, which the `classify` function uses to compute a scalar score from the anomaly map, by using the `ScoreFunction` name-value argument.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • The only name-value argument supported for code generation is `MiniBatchSize`. The name-value argument must be a compile-time constant.	• The datastore data format of `I` is not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • GPU code generation does not support `gpuArray` inputs. • GPU code generation supports inputs that are defined as half-precision floating point data types. For more information, see `half`. • The only name-value argument supported for code generation is `MiniBatchSize`. The name-value argument must be a compile-time constant.	• The datastore data format of `I` is not supported.

See Also

`efficientADAnomalyDetector` | `trainEfficientADAnomalyDetector` | `trainFCDDAnomalyDetector` | `trainPatchCoreAnomalyDetector` | `evaluateAnomalyDetection`

evaluateAnomalyDetection

Evaluate anomaly detection results against ground truth

Syntax

```
metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels)
metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels,Verbose=
tf)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels)` calculates various metrics to evaluate the quality of the anomaly detection results `predLabels` against the ground truth `gtLabels`. `anomalyLabels` indicates which class labels in `gtLabels` belong to the anomaly (positive) class

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels,Verbose=tf)` also displays the evaluation progress information in the command window.

Examples

Calculate Metrics for Trained FCDD Anomaly Detector

Load test images and corresponding labels, then create a datastore that reads the test data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
dsTest = arrayDatastore(Xtest,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of anomaly labels as the set of digits between 0 and 9, excluding 8.

```
load("digit8AnomalyDetector.mat");
anomalyLabels = setdiff(string(0:9),"8");
```

Classify each test image as normal or anomalous.

```
predLabels = classify(detector,dsTest);
```

Calculate metrics for the anomaly detection results.

```
metrics = evaluateAnomalyDetection(predLabels,gtLabels,anomalyLabels)

Evaluating anomaly detection results
------------------------------------
* Finalizing... Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    Precision    Recall    Specificity    F1Score    FalsePosi
    ______________    ____________    _________    ______    ___________    _______    _________

        0.7662           0.839         0.98971     0.748        0.93        0.85204          0.07

metrics =
  anomalyDetectionMetrics with properties:

              ConfusionMatrix: [2×2 table]
    NormalizedConfusionMatrix: [2×2 table]
               DataSetMetrics: [1×8 table]
                 ClassMetrics: [2×2 table]
```

Display the confusion matrix.

```
cm = metrics.ConfusionMatrix

cm=2×2 table
                Normal    Anomaly
                ______    _______

    Normal       465         35
    Anomaly     1134       3366
```

Input Arguments

`predLabels` — Predicted labels

logical vector

Predicted labels, specified as a logical vector. The label value `0` (`false`) corresponds to the normal (negative) class. The value `1` (`true`) corresponds to the anomaly (positive) class. You can get the predicted labels using the `classify` function.

Data Types: `logical`

`gtLabels` — Ground truth labels

numeric vector | logical vector | categorical vector

Ground truth labels for each image, specified as a numeric vector, logical vector, or categorical vector. The `evaluateAnomalyDetection` function converts the labels into a logical vector according to the set of anomaly labels in `anomalyLabels`.

`anomalyLabels` — Anomaly labels

numeric vector | logical vector | categorical vector | string vector

Anomaly labels, specified as a vector of the same data type as `gtLabels`. When `gtLabels` is categorical, `anomalyLabels` can be of data type string whose values correspond to categories in `gtLabels`.

The `evaluateAnomalyDetection` function converts all ground truth labels in `gtLabels` that belong to the set of anomaly labels to a logical `true`, indicating an anomaly (positive detection). The function converts all other ground truth labels to a logical `false`, indicating normality (negative detection).

`tf` — Display evaluation progress
`1` (default) | `0`

Display evaluation progress information in the command window, specified as a numeric or logical `1` (`true`) or `0` (`false`). The displayed information includes a progress bar, elapsed time, estimated time remaining, and data set metrics.

Data Types: `logical`

Output Arguments

`metrics` — Anomaly detection metrics
`anomalyDetectionMetrics` object

Anomaly detection metrics, returned as an `anomalyDetectionMetrics` object.

Version History

Introduced in R2022b

See Also

`anomalyDetectionMetrics`

Topics
“Identify Defects in Air Compressors Using Spectrogram Images”
“Getting Started with Anomaly Detection Using Deep Learning”

predict

Predict unnormalized anomaly scores

Syntax

```
scores = predict(detector,I)
scores = predict(detector,I,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`scores = predict(detector,I)` calculates the unnormalized anomaly scores predicted by an anomaly detector during inference for a set of test images, `I`. Use this function to get predictions from the output layers of the detector during inference.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`scores = predict(detector,I,Name=Value)` specifies options using one or more name-value arguments. For example, `predict(detector,I,MiniBatchSize=32)` limits the batch size to 32.

Examples

Calculate Optimal Anomaly Threshold

Load calibration images and corresponding labels, then create a datastore that reads the calibration data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xcal,gtLabels] = digitTest4DArrayData;
dsCal = arrayDatastore(Xcal,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of anomaly labels as the set of digits between 0 and 9, excluding 8.

```
load("digit8AnomalyDetector.mat");
anomalyLabels = setdiff(string(0:9),"8");
```

Predict the anomaly score of each calibration image.

```
scores = predict(detector,dsCal);
```

Calculate the optimal threshold and corresponding ROC metrics from the anomaly scores and ground truth labels.

```
[T,roc] = anomalyThreshold(gtLabels,scores,anomalyLabels)

T = single

2.9632e-04

roc =
  rocmetrics with properties:

    Metrics: [4980×4 table]

Properties, Methods
```

Set the `Threshold` property of the FCDD anomaly detector as the optimal threshold.

```
net.Threshold = T;
```

Input Arguments

`detector` — Anomaly detector

`efficientADAnomalyDetector` object | `fcddAnomalyDetector` object | `fastFlowAnomalyDetector` object | `patchCoreAnomalyDetector` object

Anomaly detector, specified as an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, a `fastFlowAnomalyDetector` object, or a `patchCoreAnomalyDetector` object.

`I` — Test image

numeric array | datastore | formatted `dlarray` object

Test image, specified in one of these formats:

Format	Supported Detectors
M-by-*N*-by-3 numeric array representing a truecolor image.	EfficientAD, FCDD, FastFlow, PatchCore
M-by-*N*-by-3-by-*B* numeric array representing a batch of *B* truecolor images.	EfficientAD, FCDD, FastFlow, PatchCore
Datastore that reads and returns truecolor images. The images must all have the same size.	EfficientAD, FCDD, FastFlow, PatchCore
Formatted `dlarray` object with two spatial dimensions and one channel dimension. You can specify multiple test images by including a batch dimension.	EfficientAD, FCDD, FastFlow

FCDD anomaly detectors also support grayscale test images, with one color channel instead of three.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `isAnomaly = predict(detector,I,MiniBatchSize=32)` limits the batch size to 32.

`MiniBatchSize` — Size of batches
128 (default) | positive integer

Size of batches for calculating predictions, specified as a positive integer. Larger batch sizes lead to faster processing but take up more memory.

`ScoreFunction` — Function for computing scalar score from anomaly map
function handle

Function for computing a scalar score from an anomaly map when `detector` is an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, or a `fastFlowAnomalyDetector` object, specified as a function handle. The function handle `ScoreFunction` must represent a function that accepts the input data `I` and returns an output in the form of a numeric scalar. `I` must be 2-D numeric image data, `dlarray` data with two spatial dimensions, or a datastore. The default value of `ScoreFunction` depends on the type of detector specified:

Detector Specified in `detector` Argument	Default `ScoreFunction` Value
FastFlow	`@(I)max(I,[],[1 2])`
EfficientAD	`@(I)max(I,[],[1 2 3])`
FCDD	`@(I)mean(I,[1 2])`
PatchCore	Not supported

`ExecutionEnvironment` — Hardware resource
`"auto"` (default) | `"gpu"` | `"cpu"`

Hardware resource on which to run the detector, specified as `"auto"`, `"gpu"`, or `"cpu"`. The table shows the valid hardware resource values.

Resource	Action
`"auto"`	Use a GPU if it is available. Otherwise, use the CPU.
`"gpu"`	Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
`"cpu"`	Use the CPU.

Output Arguments

`scores` — Predicted anomaly scores
numeric scalar | numeric vector | formatted `dlarray` object

Predicted anomaly scores, returned as a numeric scalar, numeric vector, or formatted `dlarray` object. `scores` contains one element for each image in `I`.

Version History

Introduced in R2022b

R2024b: Support for EfficientAD anomaly detector

You can now specify the `detector` argument as an `efficientADAnomalyDetector` object.

R2023a: Support for FastFlow and PatchCore anomaly detectors

You can now specify the `detector` argument as a `fastFlowAnomalyDetector` object or a `patchCoreAnomalyDetector` object.

R2023a: Support for C and GPU code generation

`predict` now supports the generation of C code (requires MATLAB Coder) and optimized CUDA code (requires GPU Coder).

R2024a: Option to specify custom anomaly score function for FastFlow and FCDD detectors

Specify a custom anomaly score function, which the `predict` function uses to compute a scalar score from the anomaly map, by using the `ScoreFunction` name-value argument.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • The only name-value argument supported for code generation is `MiniBatchSize`. The name-value argument must be a compile-time constant.	• The datastore data format of `I` is not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

FCDD, FastFlow, PatchCore	EfficientAD
• The size of `I` must be fixed at code generation time. • GPU code generation does not support `gpuArray` inputs. • GPU code generation supports inputs that are defined as half-precision floating point data types. For more information, see `half`. • The only name-value argument supported for code generation is `MiniBatchSize`. The name-value argument must be a compile-time constant.	• The datastore data format of `I` is not supported.

See Also

`efficientADAnomalyDetector` | `trainEfficientADAnomalyDetector` | `trainFCDDAnomalyDetector` | `evaluateAnomalyDetection` | `fastFlowAnomalyDetector` | `patchCoreAnomalyDetector`

Topics

“Localize Industrial Defects Using PatchCore Anomaly Detector”
“Detect Image Anomalies Using Explainable FCDD Network”

normalize

Normalize anomaly score map using percentiles

Syntax

```
normalizedMap = normalize(normalizer,map)
normalizedMap = normalize(normalizer,map,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`normalizedMap = normalize(normalizer,map)` normalizes an anomaly map, `map`, using the map percentile statistics computed by the `percentileNormalizer` object `normalizer`.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`normalizedMap = normalize(normalizer,map,Name=Value)` specifies options for anomaly map normalization using one or more name-value arguments. For example, `NormalOutputRange=[0 0.2]` specifies the normalized map output range of a normal image as `[0, 0.2]`.

Input Arguments

`normalizer` — Percentile normalizer

`percentileNormalizer` object

Percentile normalizer, specified as a `percentileNormalizer` object.

`map` — Anomaly score map

M-by-*N* numeric matrix

Anomaly score map, specified as an *M*-by-*N* numeric matrix. You can produce an anomaly score map using the `anomalyMap` function. Larger values indicate a greater chance that the corresponding pixel is an anomaly.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `normalize(normalizer,map,NormalOutputRange=[0 0.2])` specifies the normalized map output range of a normal image as `[0, 0.2]`.

`NormalOutputRange` — Normalized map output range of normal image

`[0 0.1]` (default) | two-element row vector

Normalized map output range of a normal image, specified as a two-element row vector. The `percentileNormalizer` function linearly maps this range to the range specified by the `NormalPercentileRange` name-value argument. To learn more, see “Linear Mapping for Anomaly Map Normalization” on page 2-1648.

`OutputClipRange` — Clipping of normalized heat map output range
`[ ]` (default) | two-element row vector

Clipping of the normalized heat map output range, specified by a two-element row vector. This argument specifies the limits of the output normalized anomaly scores, and both values must be within the range specified by the `NormalOutputRange` name-value argument.

Output Arguments

`normalizedMap` — Normalized anomaly score map
M-by-*N* numeric matrix

Normalized anomaly score map, returned as an *M*-by-*N* numeric matrix.

Tips

- The normalized anomaly score values returned in the `normalizedMap` argument are unbounded. To limit the range of the normalized anomaly scores, such as to [0, 1] for use with `anomalyMapOverlay`, specify the range using the `OutputClipRange` name-value argument.

Version History

Introduced in R2024b

See Also

`percentileNormalizer` | `anomalyMap` | `viewAnomalyDetectionResults`

splitAnomalyData

Split data into training, validation and testing sets for anomaly detection

Syntax

```
[dsTrain,dsVal,dsTest] = splitAnomalyData(ds,anomalyLabels)
[dsTrain,trainLabels,dsVal,valLabels,dsTest,testLabels] = splitAnomalyData(
ds,gtLabels,anomalyLabels)
[ ___ ] = splitAnomalyData( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`[dsTrain,dsVal,dsTest] = splitAnomalyData(ds,anomalyLabels)` automatically splits the input image datastore, `ds`, into three datastores for training, validation and testing of anomaly detection networks. `anomalyLabels` indicates which class labels in `gtLabels` belong to the anomaly (positive) class. By default, 70% of the total input data is used for training, 10% are used for validation, and 20% are used for testing. By default, the training datastore does not include anomaly images.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`[dsTrain,trainLabels,dsVal,valLabels,dsTest,testLabels] = splitAnomalyData(ds,gtLabels,anomalyLabels)` automatically splits the input datastore, `ds`, into three datastores and their labels for training, validation and testing of anomaly detection networks. `gtLabels` represents the ground-truth labels for each corresponding image in the datastore and `anomalyLabels` determine which labels in `gtLabels` belong to the anomaly class.

`[ ___ ] = splitAnomalyData( ___ ,Name=Value)` uses name-value arguments to perform custom proportioning of the images. You can specify either the proportion of images in each datastore on page 3-0 or the proportion of anomaly and normal images on page 3-0 .

If you specify name-value arguments for both proportioning strategies, then `splitAnomalyData` uses the arguments that specify the proportion of images in each datastore. In this case, the function ignores the arguments that specify the proportion of anomaly and normal images.

Examples

Split Datastore of Labeled Anomaly Images

Load a data set that consists of images of digits from 0 to 9.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","digits","synthetic");
ds = imageDatastore(dataDir,IncludeSubfolders=true, ...
    LabelSource="foldernames");
```

Specify the digits that count as anomalous. For instance, consider images of the digit 8 to be normal, and all other digits to be anomalous.

```
anomalyLabels = ["0","1","2","3","4","5","6","7","9"];
```

Split the training data into training, validation, and testing datastores.

```
[dsTrain,dsVal,dsTest] = splitAnomalyData(ds,anomalyLabels);
```

```
Splitting anomaly dataset
-------------------------
* Finalizing... Done.
* Number of files and proportions per class in all the datasets:
```

	Input		Train		Validation		Test	
	NumFiles	Ratio	NumFiles	Ratio	NumFiles	Ratio	NumFiles	Ratio
0	101	0.1	0	0	34	0.10863	67	0.10686
1	101	0.1	0	0	34	0.10863	67	0.10686
2	101	0.1	0	0	34	0.10863	67	0.10686
3	101	0.1	0	0	34	0.10863	67	0.10686
4	101	0.1	0	0	33	0.10543	68	0.10845
5	101	0.1	0	0	34	0.10863	67	0.10686
6	101	0.1	0	0	33	0.10543	68	0.10845
7	101	0.1	0	0	34	0.10863	67	0.10686
8	101	0.1	70	1	10	0.031949	21	0.033493
9	101	0.1	0	0	33	0.10543	68	0.10845

Input Arguments

`ds` — Data set

datastore

Data set of normal and anomaly images, specified as a datastore. If you do not specify the `gtLabels` argument, then `ds` must be an image datastore containing labeled images.

`gtLabels` — Ground truth labels

numeric vector | logical vector | categorical vector

Ground truth labels for each image, specified as a numeric vector, logical vector, or categorical vector. The `splitAnomalyData` function converts the labels into a logical vector according to the set of anomaly labels in `anomalyLabels`.

`anomalyLabels` — Anomaly labels

numeric vector | logical vector | categorical vector | string vector

Anomaly labels, specified as a vector of the same data type as `gtLabels`. When `gtLabels` is categorical, `anomalyLabels` can be of data type string whose values correspond to categories in `gtLabels`.

The `splitAnomalyData` function converts all ground truth labels in `gtLabels` that belong to the set of anomaly labels to a logical `true`, indicating an anomaly (positive detection). The function converts all other ground truth labels to a logical `false`, indicating normality (negative detection).

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[dsTrain,dsVal,dsTest] = splitAnomalyData(ds,anomalyLabels,DataAllocationRatio=[0.8 0.1 0.1])` assigns 80% of the total input data for training, 10% for validation, and 10% for testing

Specify Proportion of Images in Each Datastore

`DataAllocationRatio` — Proportion of input data to include in each datastore
`[0.7 0.1 0.2]` (default) | 1-by-3 numeric vector

Proportion of input data to include in the train, validation and test datastores respectively, specified as a 1-by-3 numeric vector. The function ignores class labels when splitting the data. The elements of the vector must sum to a number between 0 and 1. If the elements sum to a number less than 1, then the `splitAnomalyData` function does not allocate the remaining data to a training, validation, or testing datastore.

`KeepAnomalyLabelsInTrainingDatastore` — Include anomaly images in training datastore
`false` or `0` (default) | `true` or `1`

Include anomaly images in the training datastore, specified as a numeric or logical `false` (`0`) or `true` (`1`).

`KeepUnusedAnomalyLabels` — Include all anomaly images in validation and testing datastores
`true` or `1` (default) | `false` or `0`

Include all anomaly images in the validation and testing datastores, specified as a numeric or logical `true` (`1`) or `false` (`0`). This argument applies only when the value of `KeepAnomalyLabelsInTrainingDatastore` is `false`.

By default, `KeepUnusedAnomalyLabels` is `true` and the `splitAnomalyData` function distributes all anomaly images to the validation and testing datastores. Therefore, the validation datastore `dsVal` and testing datastore `dsTest` have a higher proportion of anomaly to normal labels than the input datastore `ds`.

When `KeepUnusedAnomalyLabels` is `false`, `splitAnomalyData` function omits some anomaly files from `dsVal` and `dsTest`. Therefore, `dsVal` and `dsTest` use the same proportion of anomaly to normal images as `ds`. Specify `KeepUnusedAnomalyLabels` as `false` for stratified partitions.

Specify Proportion of Normal and Anomaly Images

`NormalLabelsRatio` — Ratio of files with normal class labels
`[]` (default) | 1-by-3 numeric vector

Ratio of files with normal class labels to include in the training, validation, and testing datastores, specified as a 1-by-3 numeric vector. The elements of the vector must sum to a number between 0 and

1. If the elements sum to a number less than 1, then the `splitAnomalyData` function does not allocate the remaining normal data to a training, validation, or testing datastore.

Example: `NormalLabelsRatio=[0.8 0.1 0.1]` assigns 80% of the normal data for training, 10% for validation, and 10% for testing

`AnomalyLabelsRatio` — Ratio of files with anomaly class labels
`[]` (default) | 1-by-3 numeric vector

Ratio of files with normal class labels to include in the training, validation, and testing datastores, specified as a 1-by-3 numeric vector. The elements of the vector must sum to a number between 0 and 1. If the elements sum to a number less than 1, then the `splitAnomalyData` function does not allocate the remaining anomaly data to a training, validation, or testing datastore.

Example: `AnomalyLabelsRatio=[0.1 0.1 0.5]` assigns 10% of the anomaly data for training, 10% for validation, and 50% for testing, and omits the remaining 30% of the anomaly data

Other Partitioning Options

`Verbose` — Display statistics
`true` or `1` (default) | `false` or `0`

Display statistics of the partitioned data for each class, specified as a numeric or logical `true` (`1`) or `false` (`0`).

Output Arguments

`dsTrain` — Training datastore
datastore

Training datastore, returned as a datastore of the same type as `ds`.

`dsVal` — Validation datastore
datastore

Validation datastore, returned as a datastore of the same type as `ds`.

`dsTest` — Testing datastore
datastore

Testing datastore, returned as a datastore of the same type as `ds`.

`trainLabels` — Labels of data in training datastore
numeric vector | logical vector | categorical vector

Labels of data in training datastore, returned as a numeric vector, logical vector, or categorical vector.

`valLabels` — Labels of data in validation datastore
numeric vector | logical vector | categorical vector

Labels of data in validation datastore, returned as a numeric vector, logical vector, or categorical vector.

`testLabels` — Labels of data in testing datastore
numeric vector | logical vector | categorical vector

Labels of data in testing datastore, returned as a numeric vector, logical vector, or categorical vector.

Version History

Introduced in R2023a

See Also

`trainFCDDAnomalyDetector` | `trainFastFlowAnomalyDetector`

trainEfficientADAnomalyDetector

Train EfficientAD anomaly detection network

Syntax

```
detector = trainEfficientADAnomalyDetector(normalData,detectorIn,options)
detector = trainEfficientADAnomalyDetector( ___ ,NormalizationDataRatio=Ratio)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`detector = trainEfficientADAnomalyDetector(normalData,detectorIn,options)` trains the input EfficientAD anomaly detection network `detectorIn`. The training data consists of the specified normal images `normalData`.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Note It is recommended that you also have Parallel Computing Toolbox to use with a CUDA enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`detector = trainEfficientADAnomalyDetector( ___ ,NormalizationDataRatio=Ratio)` specifies the data ratio to use for anomaly map normalization, in addition to the input arguments from the previous syntax.

Input Arguments

`detectorIn` — EfficientAD anomaly detector to train

`efficientADAnomalyDetector` object

EfficientAD anomaly detector to train, specified as an `efficientADAnomalyDetector` object.

`normalData` — Training data

datastore

Training data, specified as a datastore. The training data consists of samples of normal images. Do not include anomaly images in the training data.

`options` — Training options

`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

Tip When you perform tile-based training and GPU memory is limited, reduce the mini-batch size using the `MiniBatchSize` argument of the `trainingOptions` function, at the expense of slower computation speed.

`Ratio` — Proportion of data to use for anomaly map normalization

`0.1` (default) | numeric scalar in range [0, 1]

Proportion of training data to use for anomaly map normalization and optionally, optimization of anomaly score for logical anomalies, specified as a numeric scalar in the range [0, 1]. For example, a data ratio of 0.1 specifies that the detector must use only 10% of the training data for anomaly map normalization. To not perform the normalization, set the value to 0. Use a higher percentage of training data for normalization when your training data size is large.

Note When you specify the `OptimizeScoreForLogicalAnomalies` name-value argument as `true`, the `trainEfficientADAnomalyDetector` function additionally optimizes the anomaly score for improved detection of logical anomalies using the specified portion of training data.

Output Arguments

`detector` — Trained EfficientAD anomaly detector

`efficientADAnomalyDetector` object

Trained EfficientAD anomaly detector, returned as an `efficientADAnomalyDetector` object.

Tips

- For a given training image size and number of training images, if the peak memory usage for creating a memory bank exceeds the available memory, EfficientAD outputs a warning. To address insufficient memory, try reducing image resolution, using fewer training images, or enhancing GPU performance.

Version History

Introduced in R2024b

R2025a: MATLAB Compiler support will be removed

Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

`efficientADAnomalyDetector` | `percentileNormalizer` | `trainingOptions`

Topics

“Detect Defects Using Tiled Training of EfficientAD Anomaly Detector”

“Identify Defects in Air Compressors Using Spectrogram Images”
“Getting Started with Anomaly Detection Using Deep Learning”

trainFastFlowAnomalyDetector

Train FastFlow anomaly detection network

Syntax

```
detector = trainFastFlowAnomalyDetector(normalData,detectorIn,options)
[detector,info] = trainFastFlowAnomalyDetector(normalData,detectorIn,options)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`detector = trainFastFlowAnomalyDetector(normalData,detectorIn,options)` trains the input FastFlow anomaly detection network `detectorIn`. The training data consists of normal images in `normalData`. The `options` argument controls options for training.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Note It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`[detector,info] = trainFastFlowAnomalyDetector(normalData,detectorIn,options)` also returns information on the training progress, such as the training accuracy and learning rate for each iteration.

Examples

Train FastFlow Anomaly Detector

Load a data set that consists of images of digits from 0 to 9. Consider images of the digit 8 to be normal, and all other digits to be anomalous.

```
trainDir = fullfile(toolboxdir("vision"),"visiondata","digits","synthetic");
dsNormal = imageDatastore(fullfile(trainDir,"8"));
```

Create a `fastFlowAnomalyDetector` object.

```
untrainedDetector = fastFlowAnomalyDetector;
```

Specify training options for Adam optimization.

```
options = trainingOptions("adam", ...
   InitialLearnRate = 1e-2, ...
```

```
    MaxEpochs=40, ...
    VerboseFrequency=4, ...
    MiniBatchSize=250, ...
    Shuffle="every-epoch", ...
    Plots="none");
```

Train the anomaly detector.

```
detector = trainFastFlowAnomalyDetector(dsNormal,untrainedDetector,options);
```

```
Computing Input Normalization Statistics.

    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss
    _____    _________    ___________    _________    ____________
      4          4          00:00:13        0.01        2.0351e+11
      8          8          00:00:20        0.01        9.2107e+08
     12         12          00:00:26        0.01        5.9771e+08
     16         16          00:00:31        0.01        5.4185e+10
     20         20          00:00:36        0.01        1.436e+10
     24         24          00:00:41        0.01        8.415e+12
     28         28          00:00:47        0.01        5.7707e+09
     32         32          00:00:52        0.01        4.0718e+09
     36         36          00:00:57        0.01        1.8575e+09
     40         40          00:01:02        0.01        9.3897e+08
```

Set the anomaly threshold of the detector using a calibration data set.

```
calDir = fullfile(toolboxdir("vision"),"visiondata","digits","handwritten");
dsCal = imageDatastore(calDir,IncludeSubfolders=true,LabelSource="foldernames");
gtLabels = dsCal.Labels;
anomalyLabels = setdiff(string(0:9),"8");
scores = predict(detector,dsCal);
[T,roc] = anomalyThreshold(gtLabels,scores,anomalyLabels)
```

```
T = single

-0.0833

roc =
  rocmetrics with properties:

    Metrics: [111×4 table]

Properties, Methods
```

```
detector.Threshold = T;
```

Input Arguments

`detectorIn` — FastFlow anomaly detector to train

`fastFlowAnomalyDetector` object

FastFlow anomaly detector to train, specified as a `fastFlowAnomalyDetector` object.

`normalData` — Training data

datastore

Training data, specified as a datastore. The training data consists of samples of normal images. Do not include anomaly images in the training data.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function. You must set the `BatchNormalizationStatistics` property of the object as `"moving"`.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

Output Arguments

`detector` — Trained FastFlow anomaly detector
`fastFlowAnomalyDetector` object

Trained FastFlow anomaly detector, returned as a `fastFlowAnomalyDetector` object.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with these fields.

- `Epoch` — Epoch
- `Iteration` — Iteration
- `TimeElapsed` — Total elapsed duration
- `LearnRate` — Learning rate for each iteration
- `TrainingLoss` — Loss at the end of each iteration
- `ValidationLoss` — Loss on the validation data

If you specify AUC as a metric to track using the `Metrics` property of the `trainingOptions` function, then the structure array will additionally include this field:

- `ValidationAUC` — Area under the ROC curve (AUC) metric values for the validation data

---

Note If you specify the `ValidationData` training option to return `ValidationLoss` and `ValidationAUC`, you must specify your data in one of these formats:

- A two-element cell array `{dsNormal, dsDefect}`, where `dsNormal` is a datastore of normal images, and `dsDefect` is a datastore of anomalous images.
- A single datastore. You must organize the datastore so that calling the `read` and `readall` functions on it returns a table or two-element cell array of the form `{I, Labels}`, where `I` is the image data and `Labels` is the corresponding label data. In the label data, each element is a logical `1` (`true`) for anomaly images, or a logical `0` (`false`) for normal images.

Version History

Introduced in R2023a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

`fastFlowAnomalyDetector` | `trainPatchCoreAnomalyDetector` | `trainFCDDAnomalyDetector` | `trainingOptions` | `percentileNormalizer`

Topics
“Getting Started with Anomaly Detection Using Deep Learning”

trainFCDDAnomalyDetector

Train fully convolutional data description (FCDD) anomaly detection network

Syntax

```
detector = trainFCDDAnomalyDetector(normalData,anomalyData,untrainedDetector,
options)
detector = trainFCDDAnomalyDetector(normalData,anomalyData,untrainedDetector,
options,Name=Value)
[detector,info] = trainFCDDAnomalyDetector( ___ )
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`detector = trainFCDDAnomalyDetector(normalData,anomalyData,untrainedDetector, options)` trains the FCDD anomaly detection network `untrainedDetector`. The training data consists of normal images in `normalData` and anomaly images in `anomalyData`. The `options` argument controls options for training.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Note It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`detector = trainFCDDAnomalyDetector(normalData,anomalyData,untrainedDetector, options,Name=Value)` uses additional options specified by one or more name-value arguments.

`[detector,info] = trainFCDDAnomalyDetector( ___ )` also returns information on the training progress, such as the training accuracy and learning rate for each iteration.

Examples

Train FCDD Anomaly Detector

Load a data set that consists of images of digits from `0` to `9`.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","digits","synthetic");
dsTrain = imageDatastore(dataDir,IncludeSubfolders=true, ...
    LabelSource="foldernames");
```

Split the training data into normal and anomaly images. For instance, set the images of the digit `8` to be normal, and all other digits to be anomalous.

```
trainingLabels = dsTrain.Labels;
normalDataIdx = find(trainingLabels == "8");
anomalyDataIdx = find(trainingLabels ~= "8");
dsNormal = subset(dsTrain,normalDataIdx);
dsAnomaly = subset(dsTrain,anomalyDataIdx);
```

Create an `fcddAnomalyDetector` object with a ResNet-18 backbone.

```
backbone = pretrainedEncoderNetwork("resnet18",3);
untrainedDetector = fcddAnomalyDetector(backbone);
```

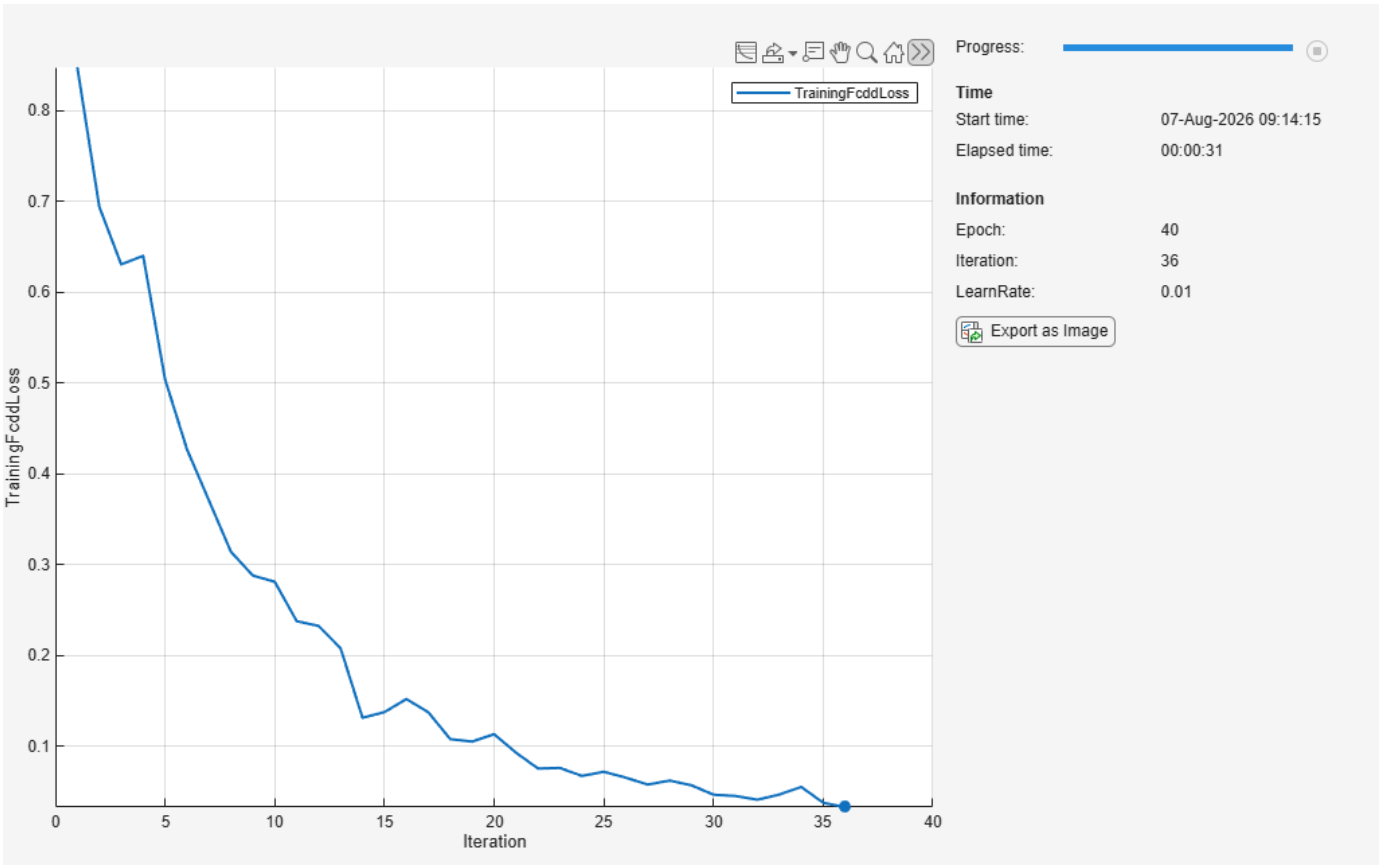
Specify training options for SGDM optimization.

```
options = trainingOptions("sgdm", ...
    Plots="training-progress", ...
    InitialLearnRate = 0.01, ...
    MaxEpochs=40, ...
    MiniBatchSize=300, ...
    ResetInputNormalization=false, ...
    Shuffle="every-epoch");
```

Train the anomaly detector.

```
detector = trainFCDDAnomalyDetector(dsNormal,dsAnomaly,untrainedDetector,options);
```

```
    Epoch    Iteration    TimeElapsed    LearnRate    TrainingFcddLoss
    _____    _________    ___________    _________    ________________
```



Input Arguments

`untrainedDetector` — Untrained FCDD anomaly detector
`fcddAnomalyDetector` object

Untrained FCDD anomaly detector, specified as an `fcddAnomalyDetector` object.

`normalData` — Labeled ground truth normal images
datastore

Labeled ground truth normal images, specified as a datastore.

`anomalyData` — Labeled ground truth anomaly images
datastore

Labeled ground truth anomaly images, specified as a datastore.

`options` — Training options
`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function.

You must set the `BatchNormalizationStatistics` property of the object as `"moving"`.

Note If you specify the `OutputFcn` function handle using the `OutputFcn` name-value argument, it must use a per-epoch `info` structure with these fields:

- `Epoch`
- `Iteration`
- `TimeElapsed`
- `LearnRate`
- `TrainingLoss`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `detector = trainFCDDAnomalyDetector(normalData,anomalyData,untrainedDetector,options,FreezeBackbone=false)` does not freeze the backbone of the FCDD network during training.

`UseMaskData` — Use mask data
`false` or `0` (default) | `true` or `1`

Use mask data in the anomaly training data to form the pixelwise loss, specified as a numeric or logical `false` (`0`) or `true` (`1`). When `true`, the anomaly training data in `anomalyData` must be a

datastore that returns pairs of images as a two-element cell array of the form `{anomalyImage,anomalyMask}`. The first element, `anomalyImage`, is a sample image of an anomaly. The second element, `anomalyMask`, is a logical image of the same size as `anomalyImage` and has `true` valued pixels at locations where `anomalyImage` has anomaly content.

`FreezeBackbone` — Freeze backbone of FCDD network
`true` or `1` (default) | `false` or `0`

Freeze the backbone of the FCDD network during training, specified as a numeric or logical `true` (`1`) or `false` (`0`). When `true`, the weight of layers in the backbone does not change during training.

Output Arguments

`detector` — Trained FCDD anomaly detector
`fcddAnomalyDetector` object

Trained FCDD anomaly detector, returned as an `fcddAnomalyDetector` object.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with these fields.

- `Epoch` — Epoch
- `Iteration` — Iteration
- `TimeElapsed` — Total elapsed duration
- `LearnRate` — Learning rate for each iteration
- `TrainingFcddLoss` — FCDD loss at the end of each iteration
- `OutputNetworkIteration` — Iteration number of returned network

Version History

Introduced in R2022b

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

`fcddAnomalyDetector` | `trainFastFlowAnomalyDetector` | `trainPatchCoreAnomalyDetector` | `trainingOptions` | `percentileNormalizer`

Topics
“Detect Image Anomalies Using Explainable FCDD Network”
“Getting Started with Anomaly Detection Using Deep Learning”

trainPatchCoreAnomalyDetector

Train PatchCore anomaly detection network

Syntax

```
detector = trainPatchCoreAnomalyDetector(normalData,detectorIn)
detector = trainPatchCoreAnomalyDetector( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`detector = trainPatchCoreAnomalyDetector(normalData,detectorIn)` trains the input PatchCore anomaly detection network `detectorIn`. The training data consists of normal images in `normalData`.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Note It is recommended that you also have Parallel Computing Toolbox to use with a CUDA-enabled NVIDIA GPU. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).

`detector = trainPatchCoreAnomalyDetector( ___ ,Name=Value)` specifies options that control aspects of network creation and training as one or more name-value arguments, in addition to all input arguments from the previous syntax.

Examples

Train PatchCore Anomaly Detector

Load a data set that consists of images of digits from 0 to 9. Consider images of the digit 8 to be normal, and all other digits to be anomalous.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","digits","synthetic");
dsNormal  = imageDatastore(fullfile(dataDir,"8"));
```

Create a `patchCoreAnomalyDetector` object.

```
untrainedDetector = patchCoreAnomalyDetector(Backbone="resnet18");
```

Train the anomaly detector.

```
detector = trainPatchCoreAnomalyDetector(dsNormal,untrainedDetector);
```

```
Computing Input Normalization Statistics.
Training PatchCore Model.
------------------------

Step 1: Computing embeddings for each minibatch
Done creating uncompressed train embeddings.
Step 2: Compress train embeddings at 0.1 factor to create a coreset...

Computing coreset samples
Done compressing train embeddings (coreset).
Done training PatchCore model.
```

Set the anomaly threshold of the detector using a calibration data set.

```
calDir = fullfile(toolboxdir("vision"),"visiondata","digits","handwritten");
dsCal = imageDatastore(calDir,IncludeSubfolders=true,LabelSource="foldernames");
gtLabels = dsCal.Labels;
anomalyLabels = setdiff(string(0:9),"8");
scores = predict(detector,dsCal);
[T,roc] = anomalyThreshold(gtLabels,scores,anomalyLabels)
```

```
T = single

1.1555

roc =
  rocmetrics with properties:

    Metrics: [121×4 table]

Properties, Methods
```

```
detector.Threshold = T;
```

Input Arguments

`detectorIn` — PatchCore anomaly detector to train

`patchCoreAnomalyDetector` object

PatchCore anomaly detector to train, specified as a `patchCoreAnomalyDetector` object.

`normalData` — Training data

datastore

Training data, specified as a datastore. The training data consists of samples of normal images. Do not include anomaly images in the training data.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:
`trainPatchCoreAnomalyDetector(normalData,detectorIn,CompressionRatio=0.1)`
specifies that the detector must use only 10% of the features extracted from training images.

`CompressionRatio` — Compression ratio
`0.1` (default) | numeric scalar in range `[0, 1]`

Compression ratio, specified as a number in the range `[0, 1]`. This value indicates the decimal amount of compression used on the features extracted from training images to construct a memory bank. For example, a compression ratio of 0.1 specifies that the detector must use only 10% of the features. This ratio is a trade off between memory and accuracy. A smaller `CompressionRatio` value increases compression, but decreases accuracy.

`MiniBatchSize` — Size of mini-batch
`128` (default) | positive integer

Size of the mini-batch to use for each training iteration, specified as a positive integer. A mini-batch is a subset of the training set that the training function uses to evaluate the gradient of the loss function and update the weights.

`ExecutionEnvironment` — Hardware resource for running neural network
`"auto"` (default) | `"cpu"` | `"gpu"`

Hardware resource for running the neural network, specified as one of these options:

- `"auto"` — Use a GPU if one is available. Otherwise, use the CPU.
- `"gpu"` — Use the GPU. To use a GPU, you must have Parallel Computing Toolbox and a CUDA-enabled NVIDIA GPU. If a suitable GPU is not available, the function returns an error. For information about the supported compute capabilities, see “GPU Computing Requirements” (Parallel Computing Toolbox).
- `"cpu"` — Use the CPU.

For more information on when to use the different execution environments, see “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud” (Deep Learning Toolbox).

`ResetInputNormalization` — Reset input layer normalization
`1` or `true` (default) | `0` or `false`

Reset input layer normalization using training images, specified as a logical `1` (`true`) or `0` (`false`):

- `1` (`true`) — Reset the input layer normalization statistics using the `normalData` data set, and recalculate them at training time.
- `0` (`false`) — Recalculate the input layer normalization statistics at training time.

`SubsamplingStrategy` — Subsampling method
`"greedycoreset"` (default) | `"random"`

Subsampling method used to reduce redundancy in the feature bank that occurs during training, specified as one of these options:

- `"greedycoreset"` — Greedy coreset subsampling method.
- `"random"` — Random subsampling method.

The greedy coreset subsampling method has higher accuracy at the expense of slower run time. The random subsampling method can increase run speed at the expense of a decrease in comparative

accuracy. When using the random subsampling method, increasing the number of training images can improve accuracy, but also increases the model size and inference time.

`Verbose` — Display training progress information
`1` or `true` (default) | `0` or `false`

Display training progress information in the command window, specified as a logical `1` (`true`) or `0` (`false`).

Output Arguments

`detector` — Trained PatchCore anomaly detector
`patchCoreAnomalyDetector` object

Trained PatchCore anomaly detector, returned as a `patchCoreAnomalyDetector` object.

Tips

- For a given training image size and number of training images, if the peak memory usage for creating a memory bank exceeds the available memory, PatchCore outputs a warning. To decrease memory usage, try reducing image resolution, using fewer training images, or enhancing GPU performance (in this order to balance the ease of implementation with the target outcome of lowering memory usage).

Version History

Introduced in R2023a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

R2024a: Option to specify subsampling strategy

Specify the subsampling method by using the `SubsamplingStrategy` name-value argument. For example, `SubsamplingStrategy="greedycoreset"` specifies the greedy coreset subsampling method.

See Also

Functions
`trainFastFlowAnomalyDetector` | `trainFCDDAnomalyDetector`

Objects
`patchCoreAnomalyDetector` | `percentileNormalizer`

Topics
“Localize Industrial Defects Using PatchCore Anomaly Detector”
“Getting Started with Anomaly Detection Using Deep Learning”

viewAnomalyDetectionResults

View anomaly detection results

Syntax

```
viewAnomalyDetectionResults(detector,I,gtLabels,scores,anomalyLabels)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`viewAnomalyDetectionResults(detector,I,gtLabels,scores,anomalyLabels)` displays the anomaly detection results for a set of test images `I`. The function displays the predicted scores against the ground truth `gtLabels`. `anomalyLabels` indicates which class labels in `gtLabels` belong to the anomaly (positive) class

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

View Anomaly Detection Results

Load test images and corresponding labels, then create a datastore that reads the test data. The data set consists of grayscale images of handwritten digits 0–9.

```
[Xtest,gtLabels] = digitTest4DArrayData;
dsTest = arrayDatastore(Xtest,IterationDimension=4);
```

Load a pretrained FCDD anomaly detector. This detector has been trained to classify the digit 8 as normal and all other digits as anomalies. Therefore, specify the set of positive labels as the set of digits between 0 and 9, excluding 8.

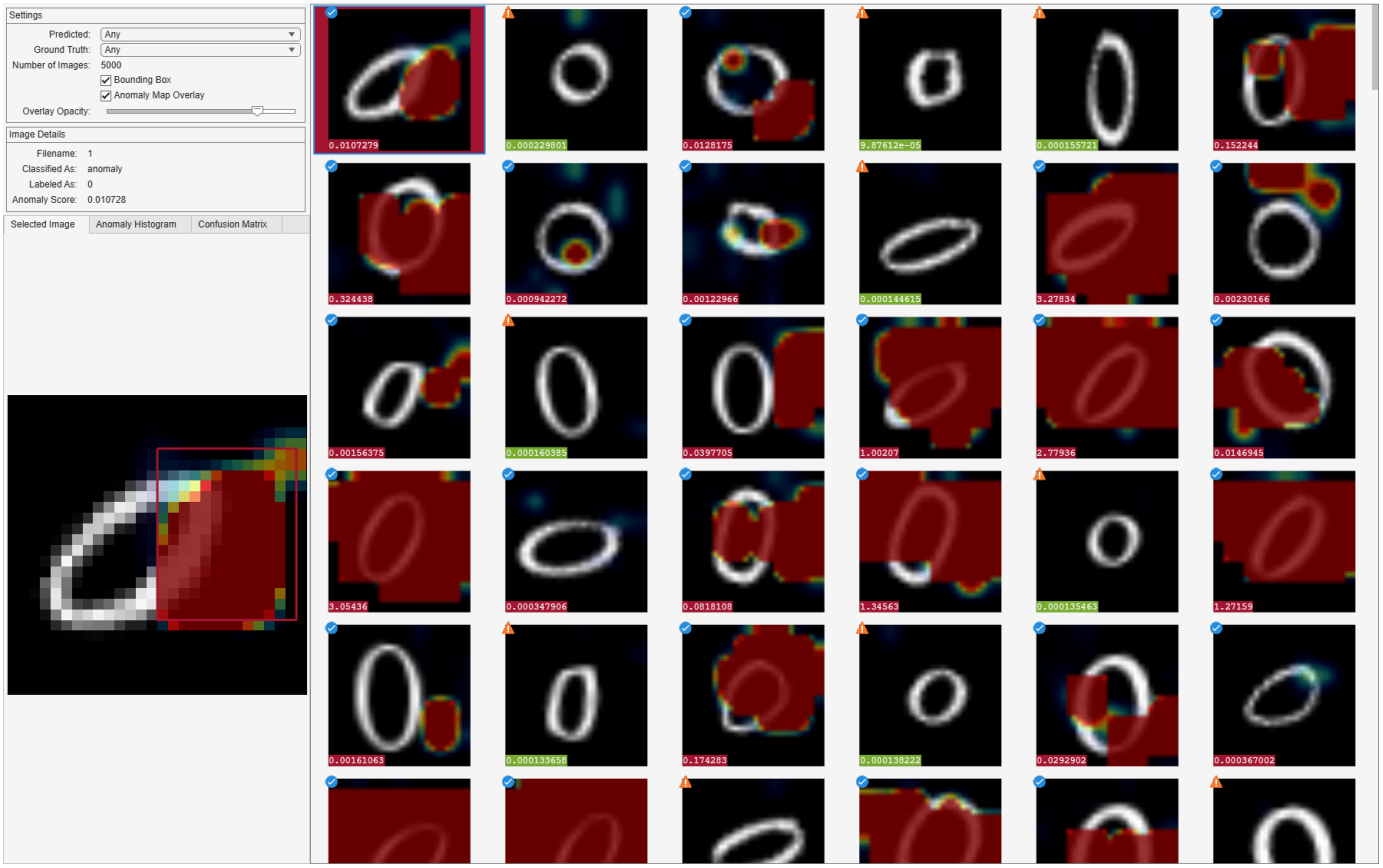
```
load("digit8AnomalyDetector.mat")
anomalyLabels = setdiff(string(0:9),"8");
```

Predict anomaly scores for each test image using the FCDD anomaly detector.

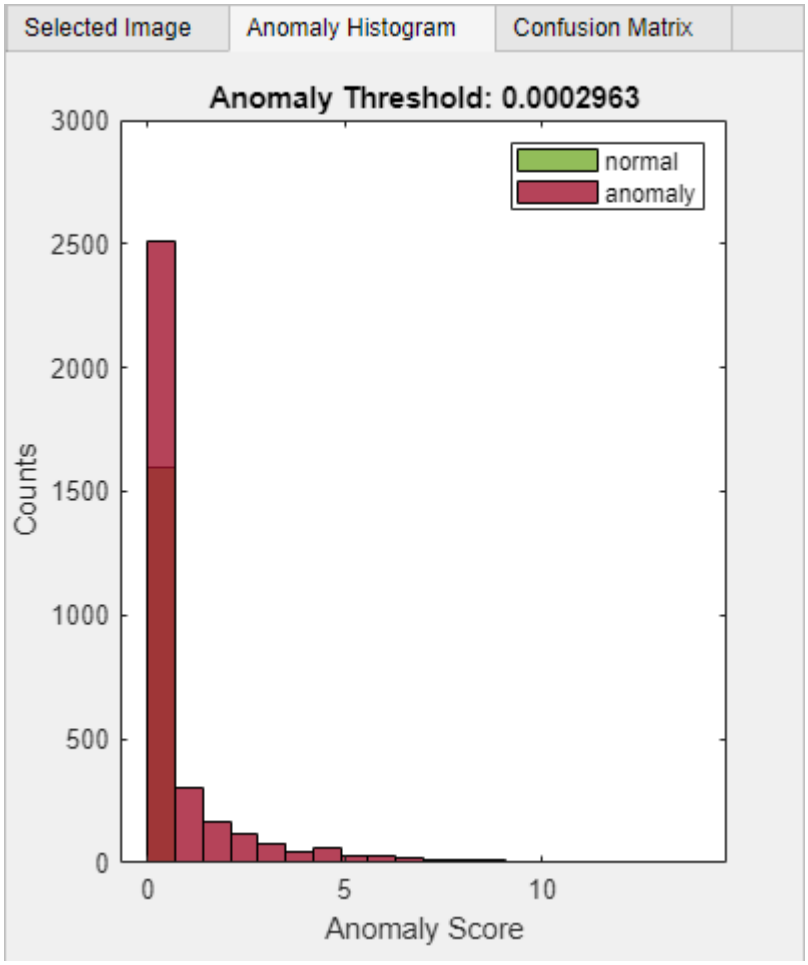
```
predScores = predict(detector,dsTest);
```

View the anomaly detection results.

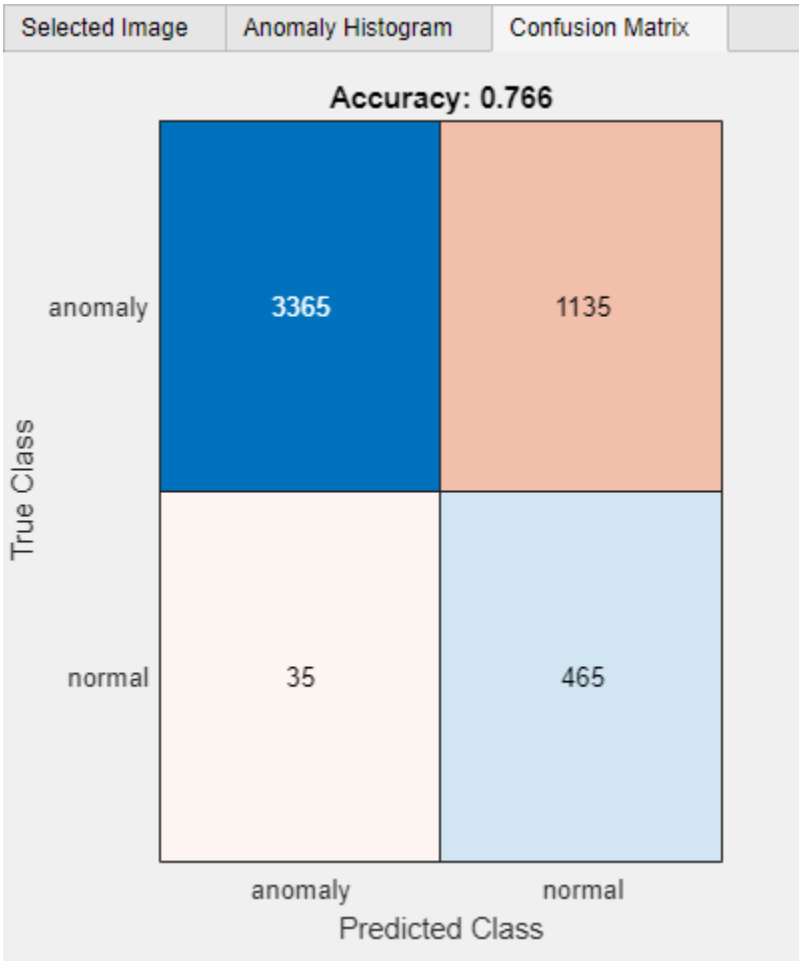
```
viewAnomalyDetectionResults(detector,dsTest,gtLabels,predScores,anomalyLabels);
```



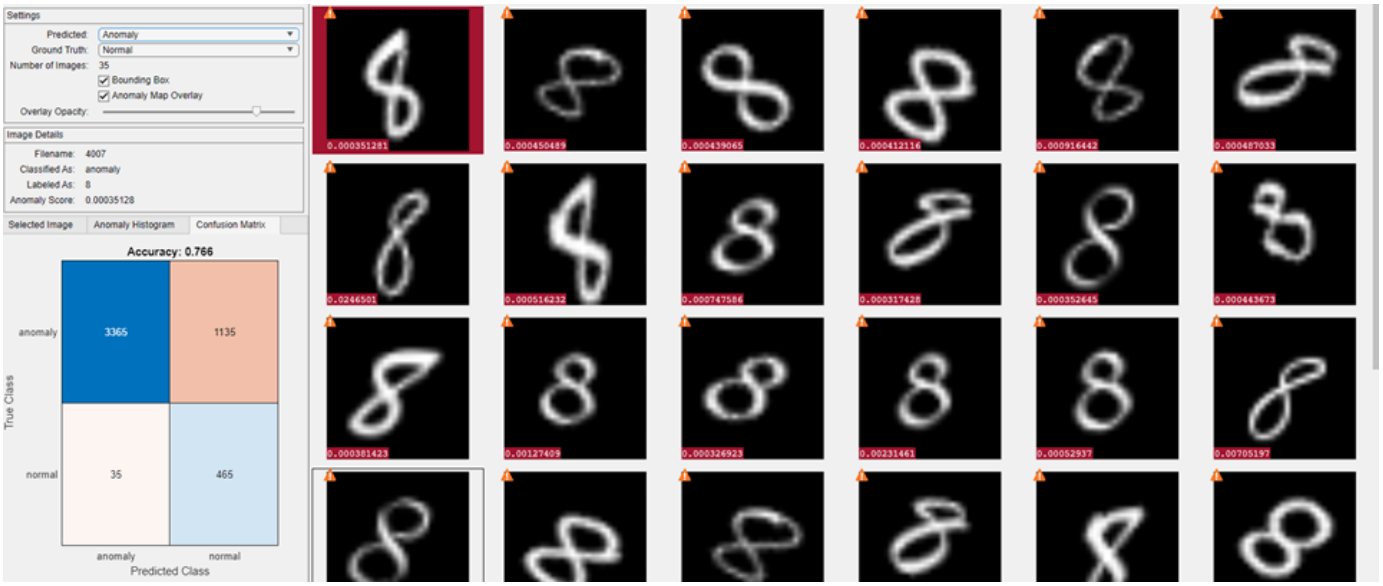
Select the **Anomaly Histogram** tab to view the **Anomaly Score vs. Counts** histogram plot. Note the **Anomaly Threshold** of `0.0002963` on the anomaly histogram plot. Above this threshold, the detector classifies the test image as an anomaly.



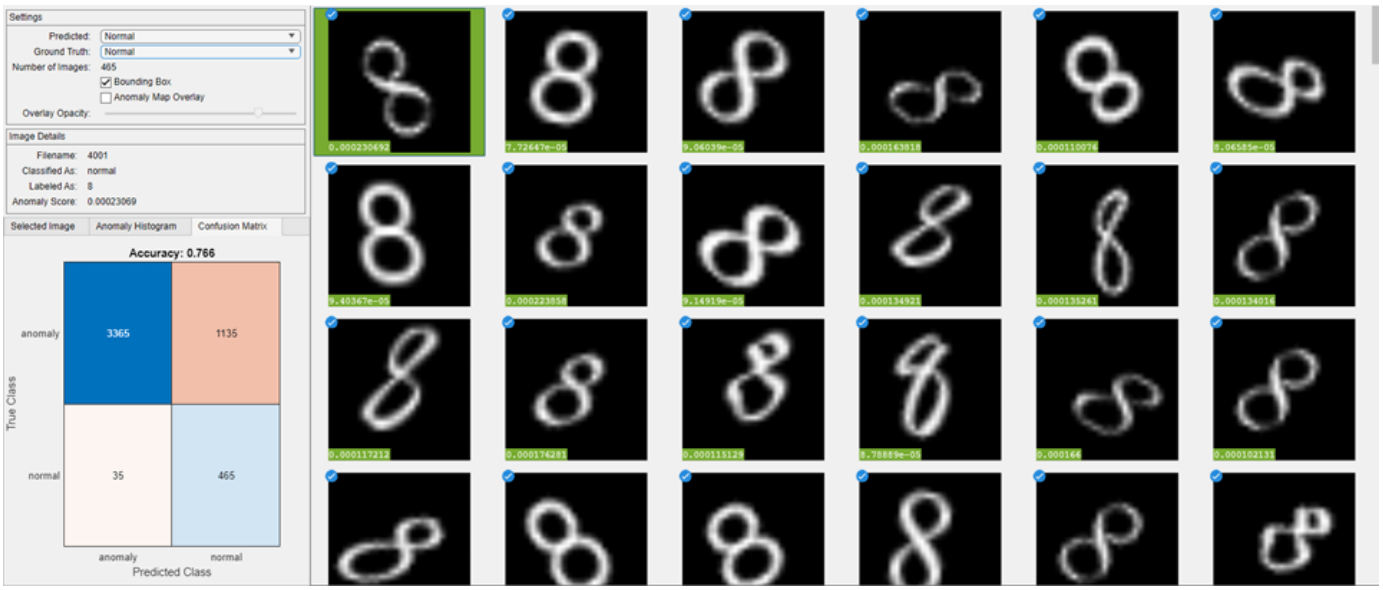
Select the **Confusion Matrix** tab to view the performance of the trained anomaly detector. You can also use this matrix to set predicted and ground truth filters. The accuracy of `0.766` represents the ratio of correctly classified images to the total number of images in the data set.



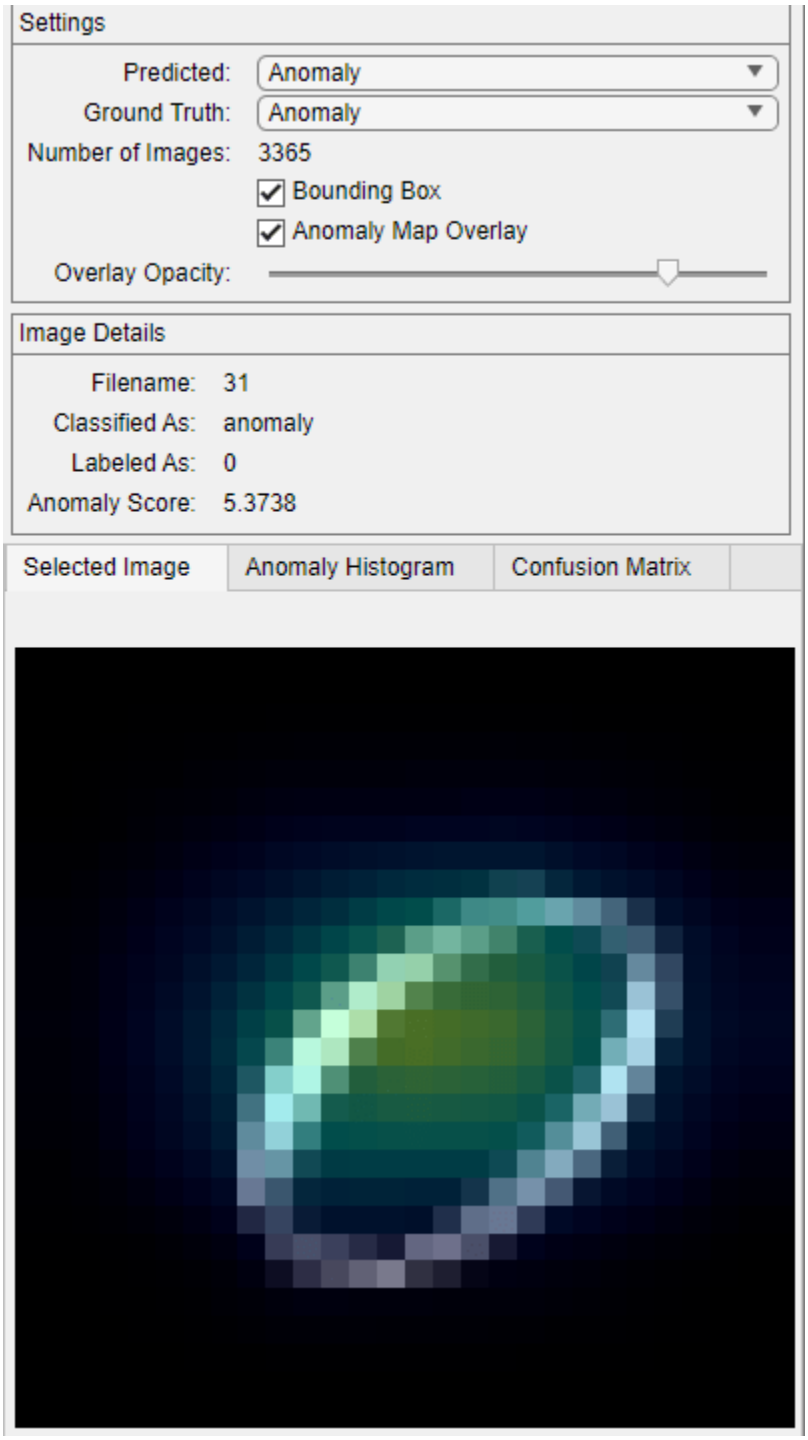
Analyze the false positive images by setting the **Predicted** parameter to `Anomaly` and the **Ground Truth** parameter to `Normal` in the **Settings** pane. On each thumbnail of false positive images, the score appears against a red background because of the positive classification. The icon in the top left corner of the thumbnail is a yellow triangle with a white exclamation mark, which indicates the image is misclassified.



Analyze the true negative images by setting the **Predicted** parameter to `Normal` and the **Ground Truth** parameter to `Normal`. On each thumbnail of true negative images, the score appears against a green background because of the negative classification. The icon in the top left corner of the thumbnail is a blue circle with a white checkmark, which indicates the image is correctly classified.



Analyze the true positive images by setting the **Predicted** parameter to `Anomaly` and the **Ground Truth** parameter to `Anomaly`. Select the image with the filename `31`. The anomaly map overlay displays the anomaly heatmap over the image. Clear **Anomaly Map Overlay** in the **Settings** pane and observe the difference in the **Selected Image** tab.


Settings
Predicted:
Anomaly
Ground Truth:
Anomaly
Number of Images:
3365
Bounding Box
Anomaly Map Overlay
Overlay Opacity:
Image Details
Filename:
31
Classified As:
anomaly
Labeled As:
0
Anomaly Score:
5.3738
Selected Image
Anomaly Histogram
Confusion Matrix
Settings
Predicted:
Anomaly
Ground Truth:
Anomaly
Number of Images:
3365
Bounding Box
Anomaly Map Overlay
Overlay Opacity:
Image Details
Filename:
31

Input Arguments

`detector` — Anomaly detector

`efficientADAnomalyDetector` object | `fcddAnomalyDetector` object | `fastFlowAnomalyDetector` object | `patchCoreAnomalyDetector` object

Anomaly detector, specified as an `efficientADAnomalyDetector` object, an `fcddAnomalyDetector` object, a `fastFlowAnomalyDetector` object, or a `patchCoreAnomalyDetector` object. You must set the `Threshold` property of the detector before calling this function.

`I` — Images

datastore

Images, specified as a datastore.

`gtLabels` — Ground truth labels

numeric vector | logical vector | categorical vector

Ground truth labels for each image, specified as a numeric vector, logical vector, or categorical vector. The `viewAnomalyDetectionResults` function converts the labels into a logical vector according to the set of anomaly labels in `anomalyLabels`.

`scores` — Predicted anomaly scores

numeric scalar | numeric vector | formatted `dlarray` object

Predicted anomaly scores, returned as a numeric scalar, numeric vector, or formatted `dlarray` object. `scores` contains one element for each image in `I`.

`anomalyLabels` — Anomaly labels

numeric vector | logical vector | categorical vector | string vector

Anomaly labels, specified as a vector of the same data type as `gtLabels`. When `gtLabels` is categorical, `anomalyLabels` can be of data type string whose values correspond to categories in `gtLabels`.

The `viewAnomalyDetectionResults` function converts all ground truth labels in `gtLabels` that belong to the set of anomaly labels to a logical `true`, indicating an anomaly (positive detection). The function converts all other ground truth labels to a logical `false`, indicating normality (negative detection).

More About

Anomaly Detection Figure Window

The `viewAnomalyDetectionResults` function opens an interactive figure window that enables you to explore the classification results and anomaly score maps for a set of images. Annotations on the image thumbnails can help you interpret the classification.

- The number in the colored box indicates the overall anomaly score of the image. When the number is in a green box, the score is below the anomaly threshold and the image is predicted to be normal, or negative. When the number is in a red box, the score is above the anomaly threshold and the image is predicted to be anomalous, or positive.

- The icon in the upper right corner indicates whether the predicted and ground truth classification agree. When the icon is a blue circle with a white check mark, the prediction and ground truth agree, and the prediction is a true positive or a true negative. When the icon is a yellow triangle with a white exclamation mark, the prediction and ground truth disagree, and the prediction is a false positive or false negative.

Version History

Introduced in R2022b

R2023a: Support for FastFlow and PatchCore anomaly detectors

You can now specify the `detector` argument as a `fastFlowAnomalyDetector` object or a `patchCoreAnomalyDetector` object.

See Also

`anomalyMapOverlay`

vision.loadFastFlowAnomalyDetector

Load FastFlow anomaly detector model for code generation

Syntax

```
net = vision.loadFastFlowAnomalyDetector(filename)
```

Description

`net = vision.loadFastFlowAnomalyDetector(filename)` loads a pretrained `fastFlowAnomalyDetector` object saved in the `filename` MAT file. `filename` must be a compile-time constant.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Generate C Code from MAT File Containing FastFlow Anomaly Detector

Use the `vision.loadFastFlowAnomalyDetector` function to load a FastFlow anomaly detector and generate C code for this network. This example assumes you have a pretrained `fastFlowAnomalyDetector` object saved as a variable named `detector` in the file `trainedFastFlow8Detector.mat`. In order to run the example, you must train the detector and save the trained detector in a MAT file with this name. For an example of how to train a FastFlow anomaly detector, see `trainFastFlowAnomalyDetector`.

Load a pretrained `fastFlowAnomalyDetector` object from a MAT file.

```
load("trainedFastFlow8Detector.mat");
```

Create an entry-point function `myFastFlow.m` that accepts an image as input. The entry-point function performs these operations:

- Define a persistent variable called `myDetector`. The persistent variable prevents reconstructing and reloading the network object during subsequent calls to the `myFastFlow` function.
- Load the detector in the file `trainedFastFlow8Detector.mat` into the `myDetector` variable using the `vision.loadFastFlowAnomalyDetector` function.
- Perform operations, such as prediction and classification, on the input image using the detector.

```
function [score,label] = myFastFlow(in)
  persistent myDetector;
    if isempty(myDetector)
        myDetector = vision.loadFastFlowAnomalyDetector("trainedFastFlow8Detector.mat");
    end
  score = predict(myDetector,in);
```

```
  if score < myDetector.Threshold
    label = "normal";
  else
    label = "anomaly";
  end
end
```

Specify the size of the training images as the size of the input to the entry-point function. The example assumes the network was trained on truecolor images of size 16-by-16 pixels.

```
inputSize = [16 16 3];
```

Create a `coder.config` configuration object for MEX code generation and set the target language to C. On the configuration object, set `DeepLearningConfig` with no target library. The `codegen` function must determine the size, class, and complexity of MATLAB function inputs. Use the `-args` option to specify the size of the input to the entry-point function. Use the `-config` option to pass the code configuration object.

```
cfg = coder.config("mex");
cfg.TargetLang = "C";
cfg.DeepLearningConfig = coder.DeepLearningConfig;
codegen -args {ones(inputSize,"single")} -config cfg myFastFlow -report;
```

The `codegen` command places all the generated files in the `codegen` folder. The folder contains the C code for the entry-point function `myFastFlow.c`, header and source files containing the C class definitions for the network, weight, and bias files.

Read a test image from a graphics file. The image must be the same size as `inputSize`. Convert the image to data type `single` as expected by the generated code.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","digits","handwritten","4");
imTest = imread(dataDir+filesep+"digit_4_5.png");
imTest = im2single(imTest);
```

Predict the anomaly score and label of the test image by calling the generated code.

```
[predScore,predLabel] = myFastFlow_mex(imTest)
```

```
predScore =

  single

     -0.0033


predLabel =

    "anomaly"
```

Input Arguments

`filename` — Filename

string scalar

Filename of the MAT file containing the pretrained `fastFlowAnomalyDetector` object, specified as a string scalar. The MAT file must exist on the MATLAB path and contain only the network to be loaded.

This input argument must be a compile-time constant.

Data Types: `string`

Output Arguments

net — Network

`fastFlowAnomalyDetector` object

Network, returned as a `fastFlowAnomalyDetector` object.

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`coder.loadDeepLearningNetwork` | `coder.DeepLearningConfig`

vision.loadFCDDAnomalyDetector

Load FCDD anomaly detector model for code generation

Syntax

```
net = vision.loadFCDDAnomalyDetector(filename)
```

Description

`net = vision.loadFCDDAnomalyDetector(filename)` loads a pretrained `fcddAnomalyDetector` object saved in the `filename` MAT file. `filename` must be a compile-time constant.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Generate C Code from MAT File Containing FCDD Anomaly Detector

Load a pretrained `fcddAnomalyDetector` object from a MAT file. This detector was trained on grayscale images of size 28-by-28.

```
load("digit8AnomalyDetector.mat");
```

Create an entry-point function `myFCDD` that accepts an image as input. The entry-point function performs these operations:

- Define a persistent variable called `myDetector`. The persistent variable prevents reconstructing and reloading the network object during subsequent calls to the `myFCDD` function.
- Load the detector in the file `digit8AnomalyDetector.mat` into the `myDetector` variable using the `vision.loadFCDDAnomalyDetector` function.
- Perform operations, such as prediction and classification, on the input image using the detector.

Show the contents of the entry-point function. This function is attached to the example as a supporting file.

```
type myFCDD.m
```

```
function [score,label] = myFCDD(in)
  persistent myDetector;
    if isempty(myDetector)
        myDetector = vision.loadFCDDAnomalyDetector("digit8AnomalyDetector.mat");
    end
  score = predict(myDetector,in);
  if score < myDetector.Threshold
```

```
    label = "normal";
  else
    label = "anomaly";
  end
end
```

Specify the size of the training images as the size of the input to the entry-point function.

```
inputSize = [28 28];
```

Create a `coder.config` configuration object for MEX code generation and set the target language to C. On the configuration object, set the `DeepLearningConfig` property with no target library. The `codegen` function must determine the size, class, and complexity of MATLAB function inputs. Use the `-args` option to specify the size of the input to the entry-point function. Use the `-config` option to pass the code configuration object.

```
cfg = coder.config("mex");
cfg.TargetLang = "C";
cfg.DeepLearningConfig = coder.DeepLearningConfig;
codegen -args {ones(inputSize,"single")} -config cfg myFCDD -report;
```

```
Code generation successful: View report
```

The `codegen` command places all the generated files in the `codegen` folder. The folder contains the C code for the entry-point function `myFCDD.c`, header and source files containing the C class definitions for the convoluted neural network (CNN), weight, and bias files.

Load the test data. The test images must be the same size as the training images.

```
[XTest,YTest] = digitTest4DArrayData;
```

Select a test image. Convert the image to data type `single`, as expected by the generated code.

```
idxToTest = 1;
imTest = XTest(:,:,:,idxToTest);
imTest = im2single(imTest);
```

Predict the anomaly score and label of the test image by calling the generated code.

```
[predScore,predLabel] = myFCDD_mex(imTest)
```

```
predScore = single

0.0107

predLabel =
"anomaly"
```

Input Arguments

`filename` — Filename

string scalar

Filename of the MAT file containing the pretrained `fcddAnomalyDetector` object, specified as a string scalar. The MAT file must exist on the MATLAB path and contain only the network to be loaded.

This input argument must be a compile-time constant.

Data Types: `string`

Output Arguments

net — Network
`fcddAnomalyDetector` object

Network, returned as an `fcddAnomalyDetector` object.

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`coder.loadDeepLearningNetwork` | `coder.config` | `coder.DeepLearningConfig`

vision.loadPatchCoreAnomalyDetector

Load PatchCore anomaly detector model for code generation

Syntax

```
net = vision.loadPatchCoreAnomalyDetector(filename)
```

Description

`net = vision.loadPatchCoreAnomalyDetector(filename)` loads a pretrained `patchCoreAnomalyDetector` object saved in the `filename` MAT file. `filename` must be a compile-time constant.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Generate C Code from MAT File Containing PatchCore Anomaly Detector

Use the `vision.loadPatchCoreAnomalyDetector` function to load a PatchCore anomaly detector and generate C code for this network. This example assumes you have a pretrained `patchCoreAnomalyDetector` object saved as a variable named `detector` in the file `trainedPatchCore8Detector.mat`. For an example of training a PatchCore anomaly detector, see `trainPatchCoreAnomalyDetector`.

Load the pretrained `patchCoreAnomalyDetector` object from the MAT file.

```
load("trainedPatchCore8Detector.mat");
```

Create an entry-point function `myPatchCore.m` that accepts an image as input. The entry-point function performs these operations:

- Define a persistent variable called `myDetector`. The persistent variable prevents reconstructing and reloading the network object during subsequent calls to the `myPatchCore` function.
- Load the detector in the file `trainedPatchCore8Detector.mat` into the `myDetector` variable using the `vision.loadPatchCoreAnomalyDetector` function.
- Perform operations, such as prediction and classification, on the input image using the detector.

```
function [score,label] = myPatchCore(in)
  persistent myDetector;
    if isempty(myDetector)
        myDetector = vision.loadPatchCoreAnomalyDetector("trainedPatchCore8Detector.mat");
    end
  score = predict(myDetector,in);
  if score < myDetector.Threshold
```

```
    label = "normal";
  else
    label = "anomaly";
  end
end
```

Specify the size of the training images as the size of the input to the entry-point function. The example assumes the network was trained on truecolor images of size 16-by-16 pixels.

```
inputSize = [16 16 3];
```

Create a `coder.config` configuration object for MEX code generation and set the target language to C. On the configuration object, set `DeepLearningConfig` with no target library. The `codegen` function must determine the size, class, and complexity of MATLAB function inputs. Use the `-args` option to specify the size of the input to the entry-point function. Use the `-config` option to pass the code configuration object.

```
cfg = coder.config("mex");
cfg.TargetLang = "C";
cfg.DeepLearningConfig = coder.DeepLearningConfig;
codegen -args {ones(inputSize,"single")} -config cfg myPatchCore -report;
```

The `codegen` command places all the generated files in the `codegen` folder. The folder contains the C code for the entry-point function `myPatchCore.c`, header and source files containing the C class definitions for the network, weight, and bias files.

Read a test image from a graphics file. The image must be the same size as `inputSize`. Convert the image to data type `single` as expected by the generated code.

```
dataDir = fullfile(toolboxdir("vision"),"visiondata","digits","handwritten","4");
imTest = imread(dataDir+filesep+"digit_4_5.png");
imTest = im2single(imTest);
```

Predict the anomaly score and label of the test image by calling the generated code.

```
[predScore,predLabel] = myPatchCore_mex(imTest)
```

```
predScore =

  single

    2.6989


predLabel =

    "anomaly"
```

Input Arguments

`filename` — Filename

string scalar

Filename of the MAT file containing the pretrained `patchCoreAnomalyDetector` object, specified as a string scalar. The MAT file must exist on the MATLAB path and contain only the network to be loaded.

This input argument must be a compile-time constant.

Data Types: `string`

Output Arguments

net — Network
`patchCoreAnomalyDetector` object

Network, returned as a `patchCoreAnomalyDetector` object.

Version History

Introduced in R2023a

See Also

`coder.loadDeepLearningNetwork` | `coder.DeepLearningConfig`

extractpatches

Extract patches from image at specified ROIs

Syntax

```
patches = extractpatches(I,bboxes)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`patches = extractpatches(I,bboxes)` extracts patches from image `I` at the ROI locations specified by the bounding boxes `bboxes`.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Select Exemplar Patches for Counting Objects

Load an image containing objects to count into the workspace.

```
I = imread("coins.png");
```

Display the image using the `imageshow` function.

```
hImage = imageshow(I);
```

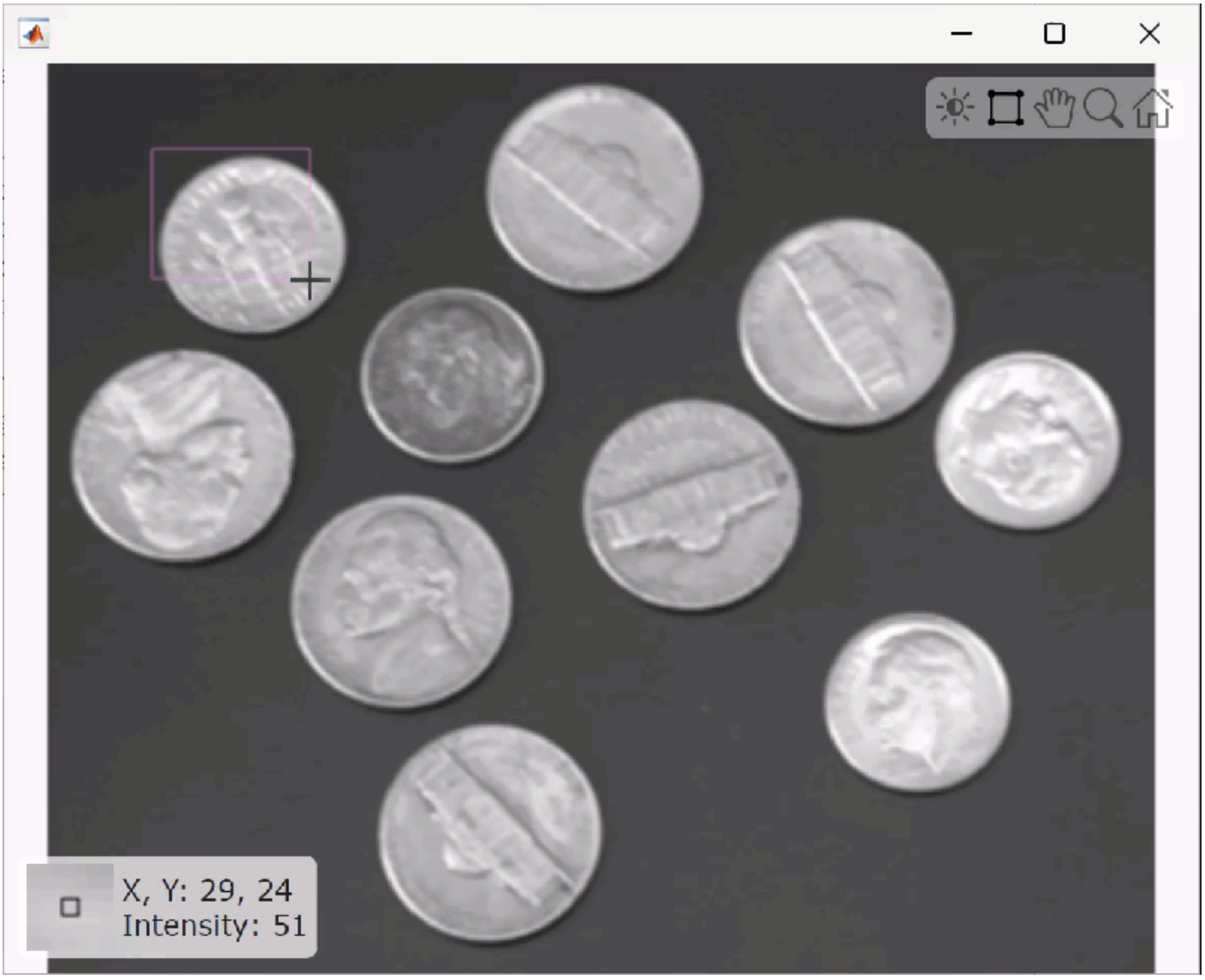
To interactively define bounding box ROIs on the image, use the `uiselectboxes` function. Click a start point in the image and drag boxes around the desired objects. Click **Accept** in the top-right corner of the figure window when you are satisfied with your selections.

```
bboxes = uiselectboxes(hImage)
```

```
bboxes = 4×4

    195.4484     28.4718    69.5974    63.5455
    321.0263    101.0952    55.9805    58.2500
    224.9516    116.2251    65.0584    61.2760
    165.1886     85.9653    55.2240    52.1980
```

This GIF shows how to interactively select bounding boxes by creating and dragging the boxes to different positions on the image.



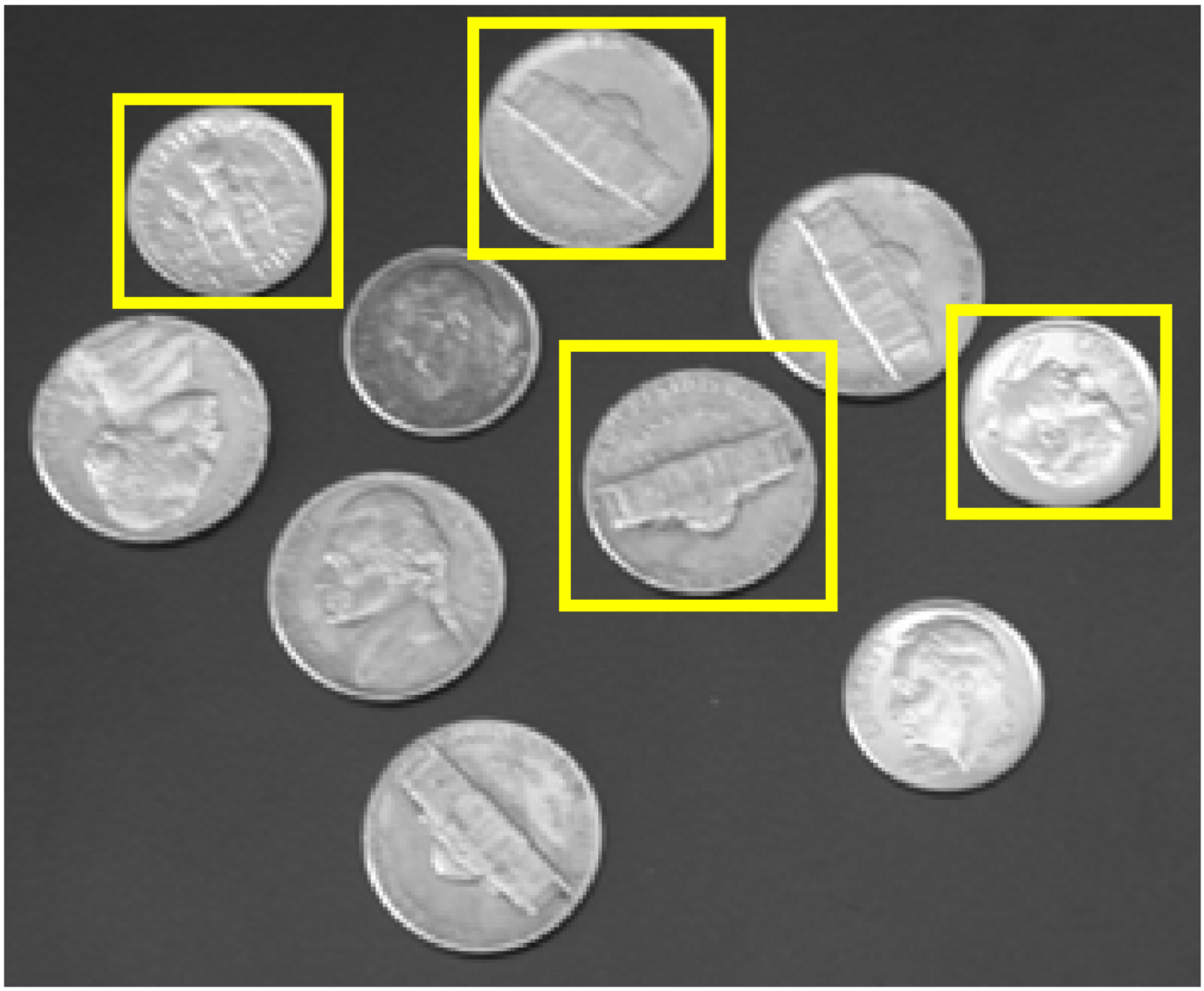
Extract the patch tiles from the image using the `extractpatches` function.

```
tiles = extractpatches(I,bboxes)
```

```
tiles=4×1 cell array
    64×70 uint8
    58×56 uint8
    62×65 uint8
    52×55 uint8
```

Display the bounding boxes defining the image patches overlaid on the image using the `insertShape` function.

```
annotatedImage = insertShape(I,"rectangle",bboxes,"LineWidth",3);
imshow(annotatedImage)
```



Input Arguments

I — Input image

H-by-*W* numeric matrix | *H*-by-*W*-by-3 numeric array

Input image from which to extract patches, specified as one of these options:

Image Type	Data Format
Grayscale	*H*-by-*W* numeric matrix
RGB	*H*-by-*W*-by-3 numeric array

bboxes — Axis-aligned rectangular bounding box ROIs

M-by-4 numeric matrix

Axis-aligned rectangular bounding box ROIs that enclose patches, specified as an *M*-by-4 numeric matrix. *M* is the number of bounding box ROIs. Each row of the matrix represents a bounding box, and is of the form [*x y w h*], where:

- *x* and `y` specify the upper-left corner of the rectangle.
- *w* specifies the width of the rectangle, which is its length along the *x*-axis.
- *h* specifies the height of the rectangle, which is its length along the *y*-axis.

You can interactively select bounding box ROIs for an image using the `uiselectboxes` function.

Output Arguments

`patches` — Image patches

M-by-1 cell array

Image patches, returned as an *M*-by-1 cell array. *M* is the number of bounding box ROIs specified using the `bboxes` name-value argument. Each element of the array is an image patch, returned as one of these values depending on the format of the input image `I`.

- RGB — Each patch is a *P*-by-*Q*-by-3 numeric array.
- Grayscale — Each patch is a *P*-by-*Q* numeric matrix.

P and *Q* are the height and width dimensions of each patch, respectively.

Version History

Introduced in R2026a

See Also

`uiselectboxes` | `counTRObjectCounter`

uiselectboxes

Select bounding box ROIs from image

Syntax

```
bboxes = uiselectboxes(hImage)
bboxes = uiselectboxes(hImage,Color=boxColor)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`bboxes = uiselectboxes(hImage)` enables you to interactively select rectangular bounding box ROIs on the displayed image `hImage`.

Note This functionality requires the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`bboxes = uiselectboxes(hImage,Color=boxColor)` specifies the color of created bounding box ROI boundaries.

Examples

Select Exemplar Patches for Counting Objects

Load an image containing objects to count into the workspace.

```
I = imread("coins.png");
```

Display the image using the `imageshow` function.

```
hImage = imageshow(I);
```

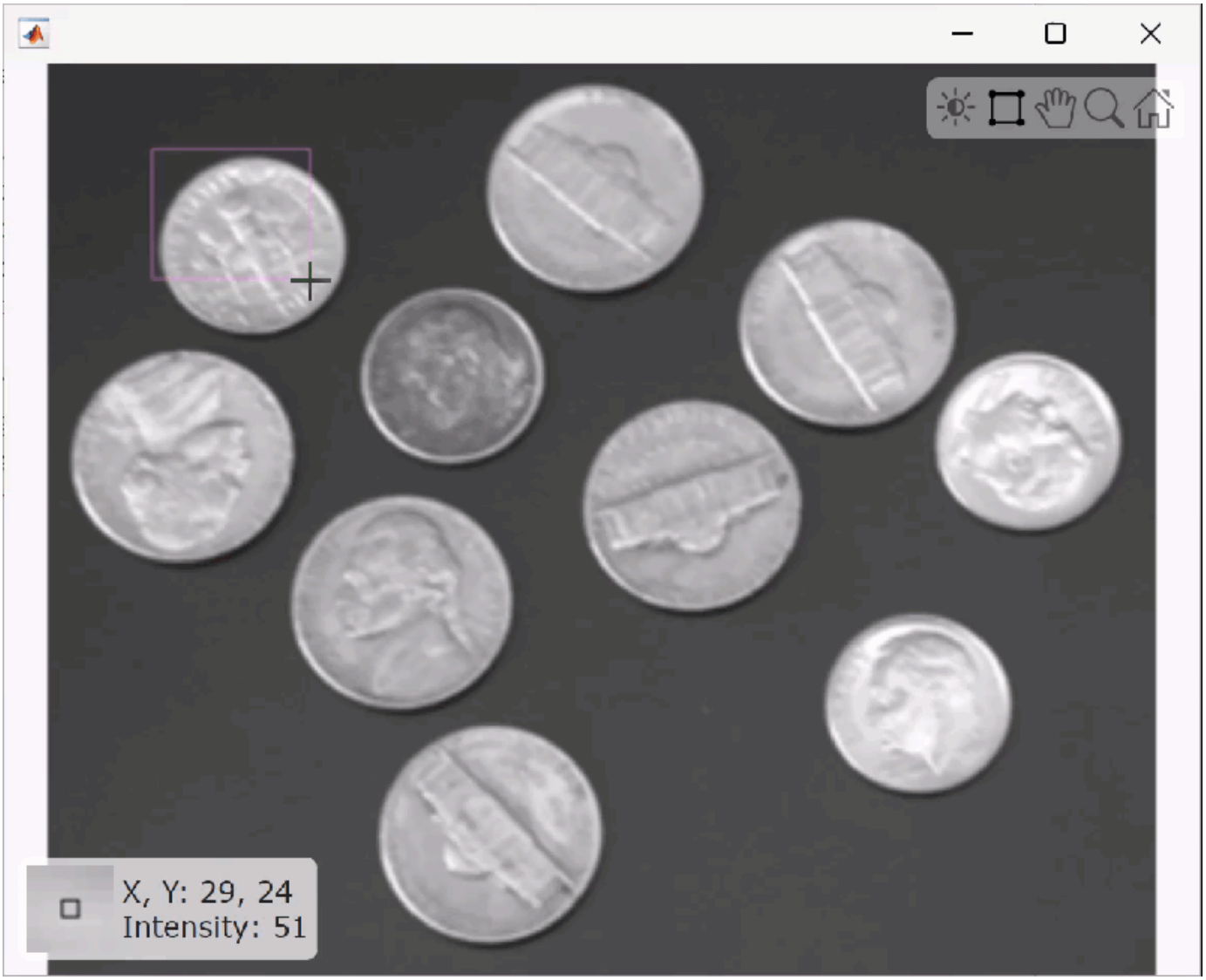
To interactively define bounding box ROIs on the image, use the `uiselectboxes` function. Click a start point in the image and drag boxes around the desired objects. Click **Accept** in the top-right corner of the figure window when you are satisfied with your selections.

```
bboxes = uiselectboxes(hImage)
```

```
bboxes = 4×4

    195.4484     28.4718    69.5974    63.5455
    321.0263    101.0952    55.9805    58.2500
    224.9516    116.2251    65.0584    61.2760
    165.1886     85.9653    55.2240    52.1980
```

This GIF shows how to interactively select bounding boxes by creating and dragging the boxes to different positions on the image.



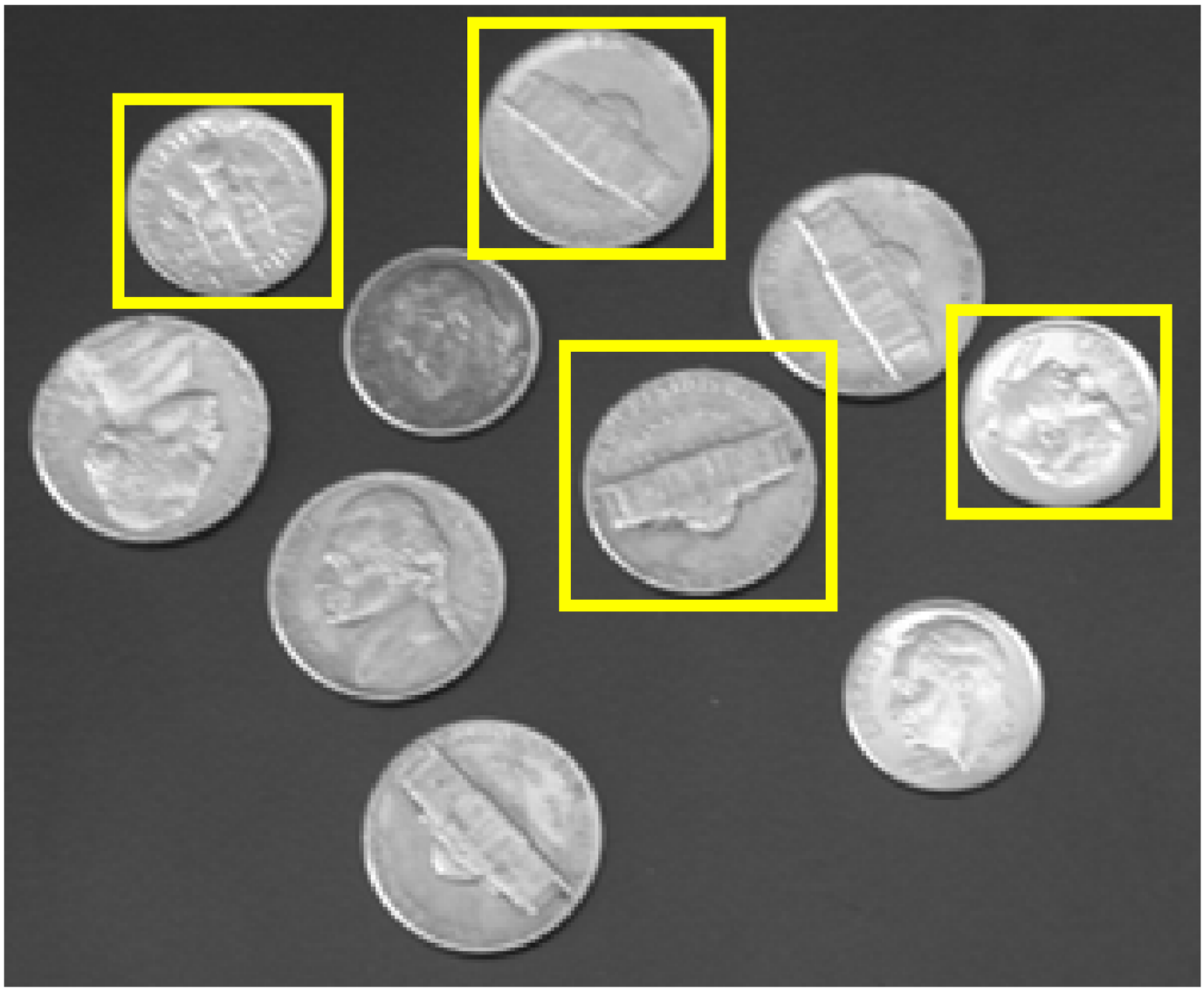
Extract the patch tiles from the image using the `extractpatches` function.

```
tiles = extractpatches(I,bboxes)
```

```
tiles=4×1 cell array
    64×70 uint8
    58×56 uint8
    62×65 uint8
    52×55 uint8
```

Display the bounding boxes defining the image patches overlaid on the image using the `insertShape` function.

```
annotatedImage = insertShape(I,"rectangle",bboxes,"LineWidth",3);
imshow(annotatedImage)
```



Input Arguments

`hImage` — Input image

`Image` object

Input image object, specified as an `Image` object. To create an `Image` object, use the `imageshow` function. The image must be RGB or grayscale.

`boxColor` — Bounding box boundary color

`[1.0 0.56 1.0]` (default) | RGB triplet | hexadecimal color code | short or long color name

Bounding box boundary color, specified as an RGB triplet, hexadecimal color code, or a short or long color name. For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]. For example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a string scalar or character vector that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Therefore, the color codes `"#FF8800"`, `"#ff8800"`, `"#F80"`, and `"#f80"` are equivalent.

Output Arguments

bboxes — Axis-aligned rectangular bounding box ROIs

M-by-4 numeric matrix

Axis-aligned rectangular bounding box ROIs, returned as an *M*-by-4 numeric matrix. *M* is the number of bounding box ROIs. Each row of the matrix represents a bounding box, returned as a 4-element row vector, and is of the form [*x y w h*], where:

- *x* and `y` specify the upper-left corner of the rectangle.
- *w* specifies the width of the rectangle, which is its length along the *x*-axis.
- *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Version History

Introduced in R2026a

See Also

`patchextractor` | `counTRObjectCounter`

visionTransformer

Pretrained vision transformer (ViT) neural network

Syntax

```
[net,classNames] = visionTransformer
[net,classNames] = visionTransformer(modelName)
[net,classNames] = visionTransformer( ___ ,Name=Value)
```

Description

`[net,classNames] = visionTransformer` returns a base-sized ViT neural network (86.8 million parameters) with a patch size of 16. The network is fine tuned using the ImageNet 2012 data set at a resolution of 384-by-384.

This feature requires a Deep Learning Toolbox license and the Computer Vision Toolbox Model for Vision Transformer Network support package. You can download this support package from the Add-On Explorer. For more information, see “Get and Manage Add-Ons”.

`[net,classNames] = visionTransformer(modelName)` returns the ViT neural network with the specified model name.

`[net,classNames] = visionTransformer( ___ ,Name=Value)` specifies additional options using one or more name-value arguments.

Examples

Load ViT Neural Network

Load a pretrained ViT neural network using the `visionTransformer` function. If the Computer Vision Toolbox™ Model for Vision Transformer Network support package is not installed, then the function provides a link to the required support package in the Add-On Explorer. To install the support package, click the link, and then click **Install**.

Load a pretrained ViT neural network and the class names. If the required support package is installed, then the function returns a `dlnetwork` object and a string array of class names.

```
[net,classNames] = visionTransformer;
```

View the neural network.

```
net
```

```
net =
  dlnetwork with properties:

         Layers: [143×1 nnet.cnn.layer.Layer]
    Connections: [167×2 table]
     Learnables: [200×3 table]
          State: [1×3 table]
```

```
     InputNames: {'imageinput'}
    OutputNames: {'softmax'}
    Initialized: 1

  View summary with summary.
```

View the number of classes.

```
numClasses = numel(classNames)
```

```
numClasses =
1000
```

Input Arguments

`modelName` — Model name

`"base-16-imagenet-384"` (default) | `"small-16-imagenet-384"` | `"tiny-16-imagenet-384"`

Model name, specified as one of these values:

Model Name	Neural Network Name	Parameter Memory	Parameters (Millions)	Image Input Size	Input Value Range	Input Layer Normalization
`"base-16-imagenet-384"`	Base-sized ViT with a patch size of 16, fine-tuned using the ImageNet 2012 data set at a resolution of 384-by-384.	331 MB	86.8	384-by-384	[0, 255]	`"zscore"`
`"small-16-imagenet-384"`	Small-sized ViT with a patch size of 16, fine-tuned using the ImageNet 2012 data set at a resolution of 384-by-384.	85 MB	22.1	384-by-384	[0, 255]	`"zscore"`

Model Name	Neural Network Name	Parameter Memory	Parameters (Millions)	Image Input Size	Input Value Range	Input Layer Normalization
`"tiny-16-imagenet-384"`	Tiny-sized ViT with a patch size of 16, fine-tuned using the ImageNet 2012 data set at a resolution of 384-by-384	22 MB	5.7	384-by-384	[0, 255]	`"zscore"`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `visionTransformer(DropoutProbability=0.2)` returns a pretrained vision transformer neural network with a dropout probability set of 0.2.

`DropoutProbability` — Probability of dropping out input elements in dropout layers
`0.1` (default) | scalar in the range [0, 1)

Probability of dropping out input elements in dropout layers, specified as a scalar in the range [0, 1).

When you train a neural network with dropout layers, the layer randomly sets input elements to zero using the dropout mask `rand(size(X)) < p`, where `X` is the layer input and `p` is the layer dropout probability. The layer then scales the remaining elements by `1/(1-p)`.

This operation helps to prevent the network from overfitting [2], [3]. A higher number results in the network dropping more elements during training. At prediction time, the output of the layer is equal to its input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`AttentionDropoutProbability` — Probability of dropping out input elements in attention layers
`0.1` (default) | scalar in the range [0, 1)

Probability of dropping out input elements in attention layers, specified as a scalar in the range [0, 1).

When you train a neural network with attention layers, the layer randomly sets attention scores to zero using the dropout mask `rand(size(scores)) < p`, where `scores` is the layer input and `p` is the layer dropout probability. The layer then scales the remaining elements by `1/(1-p)`.

This operation helps to prevent the network from overfitting [2], [3]. A higher number results in the network dropping more elements during training. At prediction time, the output of the layer is equal to its input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`net` — Pretrained ViT neural network
`dlnetwork` object

Pretrained ViT neural network, returned as a `dlnetwork` object.

`classNames` — Class names
string array

Class names, returned as a string array.

Version History

Introduced in R2023b

References

[1] Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An Image is Worth 16x16 words: Transformers for Image Recognition at Scale." Preprint, submitted June 3, 2021. https://doi.org/10.48550/arXiv.2010.11929.

[2] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *The Journal of Machine Learning Research* 15, no. 1 (January 1, 2014): 1929–58

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Communications of the ACM* 60, no. 6 (May 24, 2017): 84–90. https://doi.org/10.1145/3065386.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- You can generate generic C/C++ code that does not depend on third-party libraries and deploy the generated code to hardware platforms.

  Code generation that requires any third-party libraries is not supported. To generate code, save the pretrained network to a MAT file and load it using `coder.loadDeepLearningNetwork`. For more information, see “Load Pretrained Networks for Code Generation” (MATLAB Coder).

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- You can generate CUDA code that is independent of deep learning libraries and deploy the generated code to platforms that use NVIDIA GPU processors.

Code generation that requires any deep learning libraries is not supported. To generate code, save the pretrained network to a MAT file and load it using `coder.loadDeepLearningNetwork`. For more information, see “Load Pretrained Networks for Code Generation” (GPU Coder).

See Also

`patchEmbeddingLayer` | `trainnet` | `trainingOptions` | `dlnetwork`

Topics

“Train Vision Transformer Network for Image Classification”
“Deep Learning in MATLAB” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning Tips and Tricks” (Deep Learning Toolbox)
“Data Sets for Deep Learning” (Deep Learning Toolbox)

vision.loadYOLOXObjectDetector

Load YOLOX object detector model for code generation

Syntax

```
net = vision.loadYOLOXObjectDetector(filename)
```

Description

Add-On Required: This feature requires the Automated Visual Inspection Library for Computer Vision Toolbox add-on.

`net = vision.loadYOLOXObjectDetector(filename)` loads a pretrained YOLOX object detector model net from the MAT file `filename`. `filename` must be a compile-time constant.

Note This functionality requires Deep Learning Toolbox and the Automated Visual Inspection Library for Computer Vision Toolbox. You can install the Automated Visual Inspection Library for Computer Vision Toolbox from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Generate C++ Code from MAT File Containing YOLOX Object Detector

Use the `vision.loadYOLOXObjectDetector` function to load a YOLOX object detector and generate C code for the network. You must have a pretrained `yoloxObjectDetector` object saved as a variable named `detector` in the file `trainedYoloxObjectDetector.mat`. For an example of training a YOLOX object detector, see `trainYOLOXObjectDetector`.

Load the pretrained `yoloxObjectDetector` object from the MAT file.

```
load("trainedYoloxObjectDetector.mat");
```

Create an entry-point function `myYoloxObject.m` that accepts an image as input. The entry-point function performs these operations:

- Defines a persistent variable called `myDetector`. The persistent variable prevents reconstructing and reloading the network object during subsequent calls to the `myYoloxObject` function.
- Loads the detector from the file `trainedYoloxObjectDetector.mat` into the `myDetector` variable using the `vision.loadYOLOXObjectDetector` function.
- Detects object bounding boxes, scores, and labels on the input test image `imTest` using the detector.

```
function [bboxes,scores,labels] = myYoloxObject(imTest)
  persistent myDetector;
    if isempty(myDetector)
        myDetector = vision.loadYOLOXObjectDetector("trainedYoloxObjectDetector.mat");
    end
```

```
  [bboxes,scores,labels] = detect(myDetector,imTest);
end
```

Read a test image from a graphics file.

```
imTest = imread("highway.png");
```

Create a `coder.config` configuration object for MEX code generation and set the target language to C++. On the configuration object, set the `DeepLearningConfig` property to the target library `"mkldnn"`, for MKL-DNN targets. For more information about the supported target libraries for C++ code generation in deep learning networks, see “Generate Generic C/C++ Code for Deep Learning Networks” (MATLAB Coder).

The `codegen` function must determine the size, class, and complexity of MATLAB function inputs. Use the `-args` option to specify the input test image `imTest` to the entry-point function. Use the `-config` option to pass the code configuration object `myYoloxObject` to `codegen`.

```
cfg = coder.config("mex");
cfg.TargetLang = "C++";
cfg.DeepLearningConfig = coder.DeepLearningConfig(TargetLibrary = "mkldnn");
codegen -args {imTest} -config cfg myYoloxObject -report;
```

The `codegen` command places all the generated files in the `codegen` folder. The folder contains the C code for the entry-point function `myYoloxObject.c`, as well as header and source files containing the C class definitions for the network, weight, and bias files.

Note The YOLOX object detector deep learning network architecture contains the `spaceToDepthLayer`. Code generation for `spaceToDepthLayer` is not supported for the target library `"none"`.

Predict the bounding boxes, detection confidence scores, and object labels in the test image by calling the code generated mex file `myYoloxObject_mex`.

```
[predBboxes,predScores,predLabels] = myYoloxObject_mex(imTest)
```

```
predBboxes
```

```
predBboxes =

  3×4 single matrix

   97.9102  104.5984   39.9582   32.5561
   97.6418  104.9113   41.7848   32.5751
  146.5345   88.3480   82.4816   73.4598
```

```
predScores
```

```
predScores =

  3×1 single column vector

    0.4241
    0.7093
    0.9108
```

```
predLabels

predLabels =

  3×1 categorical array

     car
     truck
     truck
```

Input Arguments

`filename` — Filename

character vector | string scalar

Filename of the MAT file containing the pretrained YOLOX object detector, specified as a character vector or string scalar. The MAT file must exist on the MATLAB path and contain only the network to be loaded.

This input argument must be a compile-time constant.

Data Types: `char` | `string`

Output Arguments

`net` — YOLOX object detector network

`yoloxObjectDetector` object

YOLOX object detector network, returned as a `yoloxObjectDetector` object.

Version History

Introduced in R2023b

See Also

`yoloxObjectDetector` | `coder.loadDeepLearningNetwork` | `coder.DeepLearningConfig`

Topics

“Code Generation for Deep Learning Networks” (GPU Coder)
“Code Generation for Deep Learning Networks by Using cuDNN” (GPU Coder)
“Code Generation for Deep Learning Networks by Using TensorRT” (GPU Coder)
“Code Generation for Deep Learning Networks Targeting ARM Mali GPUs” (GPU Coder)

insertObjectKeypoints

Insert object keypoints in image

Syntax

```
RGB = insertObjectKeypoints(I,keypoints)
RGB = insertObjectKeypoints( ___ ,Name=Value)
```

Description

`RGB = insertObjectKeypoints(I,keypoints)` inserts a set of object keypoints, `keypoints` into the specified image, `I` and returns the result as a truecolor output image, `RGB`.

`RGB = insertObjectKeypoints( ___ ,Name=Value)` specifies one or more name-value arguments, in addition to any combination of input arguments from the previous syntax, to configure the keypoints to be inserted in the image. For example, `KeypointSize=3` inserts keypoints with the size of 3 pixels in the image.

Examples

Insert Object Keypoints and Connections

Read an input image.

```
I = imread("visionteam.jpg");
```

Run the `detect` method of YOLOv4 object detector to detect bounding boxes around persons in the image. Here, persons are considered as objects.

```
detector = yolov4ObjectDetector("csp-darknet53-coco");
[bboxes,scores,labels] = detect(detector,I,Threshold=0.4);
personIndices = labels=="person";
personBoxes = bboxes(personIndices,:);
```

Run the `detect` method of a pretrained HRNet object keypoint detector to detect keypoints of every person in the image.

```
keypointDetector = hrnetObjectKeypointDetector;
[keypoints,keypointScores,valid] = detect(keypointDetector,I,personBoxes);
```

Create an object skeleton from keypoint connections. Each row in the skeleton corresponds to categorical keypoint class indices of a keypoint connection.

```
objectSkeleton = keypointDetector.KeypointConnections
```

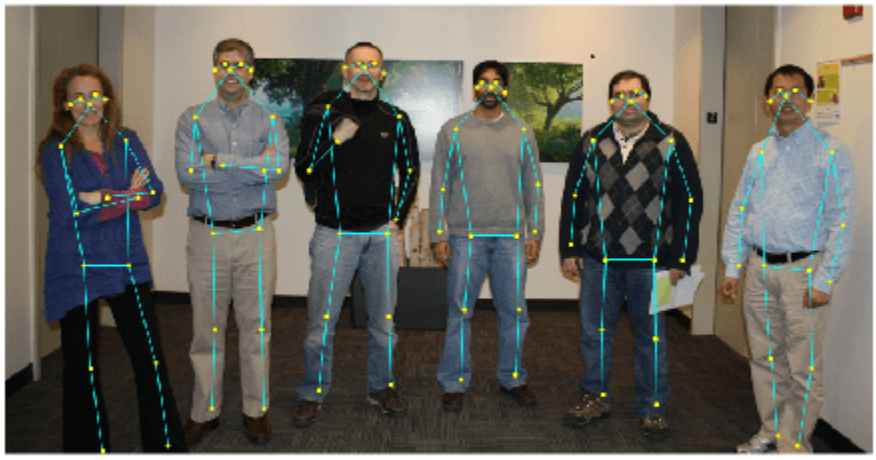
```
objectSkeleton = 17×2

     2     4
     2     1
     3     5
     3     1
```

```
     1     6
     1     7
     6     8
     8    10
     7     9
     9    11
     6    12
     7    13
    12    13
    12    14
    14    16
     ⋮
```

Insert the keypoints and skeleton on the respective person and display results.

```
detectedKeypoints = insertObjectKeypoints(I,keypoints,KeypointColor="yellow",KeypointSize=2,...
                                                  Connections=objectSkeleton,ConnectionColor="c",LineWid
imshow(detectedKeypoints);
```



Insert Object Keypoints with Label Annotations

Read an input image.

```
I = imread("visionPeople.jpg");
```

Specify a bounding box to detect the persons in the image.

```
bboxes = uint8([1.39,19.30,122.39,341.61;131.78,27.06,108.36,324.50]);
```

Detect keypoints of the person by using the `detect` method of pretrained HRNet object keypoint detector.

```
keypointDetector = hrnetObjectKeypointDetector("human-full-body-w32");
[keypoints,keypointScores,valid] = detect(keypointDetector,I,bboxes);
```

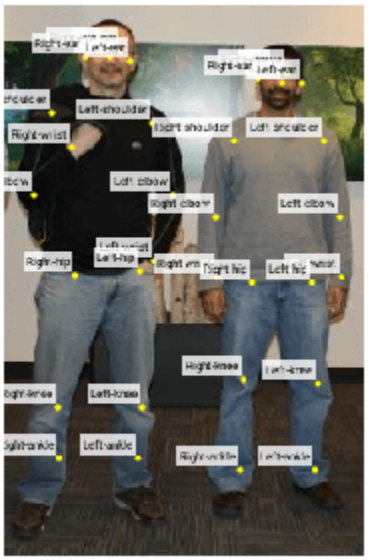
Read categorical class labels of keypoints from the keypoint detector object.

```
labels = keypointDetector.KeyPointClasses
```

```
labels = 17×1 categorical
     Nose
     Left-eye
     Right-eye
     Left-ear
     Right-ear
     Left-shoulder
     Right-shoulder
     Left-elbow
     Right-elbow
     Left-wrist
     Right-wrist
     Left-hip
     Right-hip
     Left-knee
     Right-knee
     Left-ankle
     Right-ankle
```

Insert and display detected keypoints along with the labels in the image.

```
detectedKeypoints = insertObjectKeypoints(I,keypoints,KeypointColor="yellow",KeypointSize=2,keypo
                                          TextBoxColor="white",TextBoxOpacity=0.8,FontSize=8);
imshow(detectedKeypoints);
```



Input Arguments

`I` — Input image

matrix | array

Input image, specified as an *X*-by-*Y* matrix for a grayscale image, or an *X*-by-*Y*-by-3 array for a RGB color image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`keypoints` — Object keypoints

array

Object keypoints, specified as an *M*-by-2-by-*P* array for *P* objects in the image. *M* is the number of keypoints corresponding to each object in the input image *I*. Each row of the *M*-by-2 matrix is a keypoint location in the format [*x y*].

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `insertObjectKeypoints(__,KeypointColor=[255 0 0])` inserts all the keypoints in red color.

`Connections` — Keypoint indices of keypoint connections

`[]` (default) | matrix

Keypoint indices of keypoint connections specified as an *L*-by-2 matrix. *L* is the number of keypoint connections in an object. Each row in the matrix has two indices of connected keypoints. These two keypoints are connected by a line segment.

Data Types: `double`

`KeypointVisibility` — Keypoint visibility value

`true` (default) | *M*-by-*P* matrix

Keypoint visibility value, specified as an *M*-by-*P* logical matrix. *M* is the number of keypoints in an object. *P* is the total number of objects in the image.

- Keypoints with the keypoint visibility value set to `true` or `1` in the matrix are inserted in the image.
- Keypoints with the keypoint visibility value set to `false` or `0` in the matrix are not inserted in the image.

By default, all keypoints have a keypoint visibility value set to `true` or `1`.

Data Types: `logical`

`KeypointColor` — Keypoint color

`[0.850 0.325 0.098]` (default) | character vector | cell array of character vectors | string scalar | string array | *M*-by-3 matrix

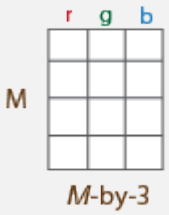
Keypoint color, specified as a character vector, cell array of character vectors, *M*- element vector, or *M*-by-3 matrix of RGB values.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color.

You can specify a different color for each keypoint or one color for all keypoints. To specify one color for all keypoints, set `KeypointColor` to a color string or an [*R G B*] vector.

Specification	Format	Example
One color for all keypoints	String or character color name	`"r"` `"red"`
	1-by-3 vector (RGB triplet)	 `[1 0 0]`
Different color for each keypoint	*M*-element vector	`["red","yellow","blue"]`

Specification	Format	Example

	M-by-3 matrix, as a list of RGB values	`[1 0 0`
		` 0 1 1`
		` 1 0 1]`
		

Supported colors are listed in the table.

Color Name	Short Name	RGB Triplet	Appearance
`"red"`	`"r"`	`[1 0 0]`	
`"green"`	`"g"`	`[0 1 0]`	
`"blue"`	`"b"`	`[0 0 1]`	
`"cyan"`	`"c"`	`[0 1 1]`	
`"magenta"`	`"m"`	`[1 0 1]`	
`"yellow"`	`"y"`	`[1 1 0]`	
`"black"`	`"k"`	`[0 0 0]`	
`"white"`	`"w"`	`[1 1 1]`	

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`KeypointLabel` — Keypoint labels
`[]` (default) | *M*-element vector

Keypoint labels, specified as an *M*-element categorical vector. *M* is the number of keypoints in an object.

Data Types: `categorical`

`KeypointSize` — Keypoint size
2 (default) | positive integer

Keypoint size, specified as a positive integer in pixels.

Data Types: `categorical`

`Opacity` — Opacity of keypoints and connections
1 (default) | scalar in the range `[0 1]`

Opacity of keypoints and connections, specified as a scalar value in the range [0 1]. The value `1` makes the keypoints and connections completely opaque and the value `0` makes them completely transparent.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConnectionColor` — Color of line segment connecting keypoints
`[0 0.447 0.741]` (default) | character vector | cell array of character vectors | string scalar | string array | *M*-by-3 matrix

Color of line segment connecting keypoints, specified as a character vector, cell array of character vectors, *M*- element vector, or *M*-by-3 matrix of RGB values.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color.

You can specify a different color for each keypoint connection or one color for all connections. To specify one color for all keypoint connections, set `ConnectionColor` to a color string or an [*R G B*] vector.

For a description of RGB color and how to specify it, see `KeypointColor`.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`LineWidth` — Line width of keypoint connection
2 (default) | positive integer

Line width of keypoint connection, specified as a positive integer. The unit is pixels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`TextBoxColor` — Text box color of keypoint label
`"yellow"` (default) | character vector | cell array of character vectors | string scalar | string array | *M*-by-3 matrix

Text box color of keypoint label, specified as a character vector, cell array of character vectors, *M*-element vector, or *M*-by-3 matrix of RGB values. *M* is the number of keypoints in an object.

An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color.

You can specify a different color for each text box or one color for all text boxes. To specify one color for all text boxes, set `TextBoxColor` to a color string or an [*R G B*] vector.

For a description of RGB color and how to specify it, see `KeypointColor`.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`TextBoxOpacity` — Text box background opacity of keypoint label
0.6 (default) | scalar in the range [`0` `1`]

Text box background opacity of keypoint label, specified as a scalar. Specify the opacity value in the range [`0` `1`].

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`FontColor` — Font color of keypoint label
`"black"` (default) | character vector | cell array of character vectors | string scalar | string array | *M*-by-3 matrix

Font color of keypoint label, specified as a character vector, cell array of character vectors, vector, or *M*-by-3 matrix of RGB values. *M* is the number of keypoints in an object.

You can specify a different font color for each keypoint label or one font color for all keypoint labels. To specify one color for all keypoint labels, set `FontColor` to a color string or an [*R G B*] vector. For a description of RGB color and how to specify it, see `KeypointColor`.

Data Types: `logical` | `uint8` | `uint16` | `int16` | `double` | `single`

`Font` — Font face of text
`"Roboto-Regular"` (default) | character vector | string scalar

Font face of text, specified as a character vector. The font face must be one of the available truetype fonts installed on your system. To get a list of available fonts on your system, type `listTrueTypeFonts` at the MATLAB command prompt.

Data Types: `char` | `string`

`FontSize` — Font size of keypoint label text
12 (default) | integer in the range [8 72]

Font size of keypoint label text, specified as an integer that corresponds to points in the range [8 72].

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`RGB` — Output image
truecolor image

Output image, returned as a RGB color image of same datatype as the input image, with the same *X*-by-*Y*-by-3 dimensions as `I`.

Limitations

Value to the `LineWidth`, `TextBoxColor`, `TextBoxOpacity`, `Font`, `FontColor`, `FontSize` arguments should be specified only when the `KeypointLabel` argument is non-empty.

Version History

Introduced in R2023b

R2026a: Default value of `Font` changed to `Roboto-Regular`
Behavior changed in R2026a

Starting in R2026a, the default value of the `Font` name-value argument is `Roboto-Regular`. In previous releases, the default value was `LucidaSansRegular`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `Font`, `FontSize`, `KeypointColor` and `KeypointConnections` name-value arguments must be compile-time constants.
- Non-ASCII characters are not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Refer to the usage notes and limitations in the C/C++ Code Generation section. The same usage notes and limitations apply to GPU code generation.

See Also

`hrnetObjectKeypointDetector` | `visibleKeypoints` | `detect` | `insertObjectAnnotation`

Topics

“Choose Function to Visualize Detected Objects”

loadHRNETObjectKeypointDetector

Load HRNet object keypoint detector model for code generation

Syntax

```
net = loadHRNETObjectKeypointDetector(filename)
```

Description

`net = loadHRNETObjectKeypointDetector(filename)` loads a pretrained HRNet object keypoint detector model net from the MAT file `filename`. `filename` must be a compile-time constant.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Object Keypoint Detection. You can download and install the Computer Vision Toolbox Model for Object Keypoint Detection from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Generate C++ Code from MAT File Containing HRNet Object Keypoint Detector

Use the `loadHRNETObjectDetector` function to load an HRNet object detector and generate C code for the network. You must have a pretrained `hrnetObjectKeypointDetector` object saved as a variable named `detector` in the file `trainedHRNetObjectKeypointDetector.mat`.

Load the pretrained `hrnetObjectKeypointDetector` object from the MAT file.

```
load("trainedHRNetObjectKeypointDetector.mat");
```

Create an entry-point function `myHRNetObject.m` that accepts an image and the bounding boxes of image objects as input. The entry-point function performs these operations:

- Defines a persistent variable called `myDetector`. The persistent variable prevents reconstructing and reloading the network object during subsequent calls to the `myHRNetObject` function.
- Loads the detector from the file `trainedHRNetObjectKeypointDetector.mat` into the `myDetector` variable using the `loadHRNETObjectDetector` function.
- Detects object keypoints, confidence scores, and validity of the objects in the input test image `imTest` using the detector.

```
function [keypoints,scores,valid] = myHRNetObject(imTest,bboxes)
  persistent myDetector;
    if isempty(myDetector)
        myDetector = loadHRNETObjectKeypointDetector("trainedHRNetObjectKeypointDetector.mat");
    end
  [keypoints,scores,valid] = detect(myDetector,imTest,bboxes);
end
```

Read a test image from a graphics file.

```
imTest = imread("visionTeam.jpg");
```

Specify the bounding boxes of objects in the test image.

```
bboxes = [28.999092    48.454803    136.47928    364.54520;
157.66484    34.789597    101.99452    375.76184;
261.84634    39.663254    123.77393    345.26978;
388.49371    49.597092    111.15561    328.06561;
506.54700    51.744110    140.14929    351.37390;
656.18372    49.514694    134.45197    363.48529];
```

Create a `coder.config` configuration object for MEX code generation and set the target language to C++. On the configuration object, set the `DeepLearningConfig` property to the target library `"mkldnn"`, for MKL-DNN targets. For more information about the supported target libraries for C++ code generation in deep learning networks, see “Generate Generic C/C++ Code for Deep Learning Networks” (MATLAB Coder).

Note The HRNet object keypoint detector deep learning network architecture contains the `spaceToDepthLayer`. Code generation for `spaceToDepthLayer` is not supported for target library `"none"`.

The `codegen` function must determine the size, class, and complexity of MATLAB function inputs. Use the `-args` option to specify the input test image `imTest` and object bounding boxes `bboxes` to the entry-point function. Use the `-config` option to pass the code configuration object `myHRNetObject` to `codegen`.

```
cfg = coder.config("mex");
cfg.TargetLang = "C++";
cfg.DeepLearningConfig = coder.DeepLearningConfig(TargetLibrary = "mkldnn");
codegen -args {imTest,bboxes} -config cfg myHRNetObject -report;
```

The `codegen` command places all the generated files in the `codegen` folder. The folder contains the C code for the entry-point function `myHRNetObject.c`, header and source files containing the C class definitions for the network, weight, and bias files.

Compute the object keypoint locations, keypoint confidence scores, and keypoint validity in the test image by calling the code-generated MEX file `myHRNetObject_mex`.

```
[objKeypoints,keypointScores,keypointValid] = myHRNetObject_mex(imTest,bboxes)
```

```
objKeypoints(:,:,1) =

   80.0250   88.0543
   84.5691   83.3077
   70.6329   83.3077
   93.9612   88.0543
   56.8991   92.8010
  113.2518  121.2811
   48.4184  130.7745
  137.5927  173.4946
   68.3165  192.4813
   95.8852  178.2413
  123.7577  178.2413
```

```
  115.7833  239.9482
   69.3291  239.9482
  140.9344  330.1351
   75.9997  334.8818
  151.8454  406.0820
   91.4549  406.0820


objKeypoints(:,:,2) =

  208.9805   60.3294
  218.4503   55.4367
  204.0877   55.4367
  228.1306   60.3294
  194.6179   60.3294
  248.2275  104.3640
  176.4147  104.3640
  253.9619  148.3986
  172.6793  153.2913
  201.1940  143.5059
  234.9171  153.2913
  236.0744  207.1114
  193.0919  212.0041
  237.9681  295.1806
  194.9856  300.0733
  239.6515  373.4643
  191.7762  373.4643


objKeypoints(:,:,3) =

  332.8377   58.8094
  337.1445   54.3137
  323.9408   54.3137
  346.1359   63.3051
  315.2329   58.8094
  360.1899  103.7664
  298.3838   94.7750
  374.4328  153.2192
  282.0071  153.2192
  362.1739  198.1762
  303.4464  126.2449
  353.6549  211.6633
  309.5482  207.1676
  355.2610  288.0902
  298.0451  288.0902
  352.1825  351.0300
  290.6599  355.5257


objKeypoints(:,:,4) =

  449.0307   76.9031
  453.1255   72.6314
  440.4873   68.3597
  461.5805   76.9031
  432.2976   76.9031
  479.1093  115.3483
```

```
  416.3604  115.3483
  488.5369  166.6085
  404.8717  166.6085
  489.4212  209.3254
  401.5726  209.3254
  472.7766  213.5971
  426.7607  213.5971
  474.1913  281.9441
  423.9922  281.9441
  471.2460  341.7477
  421.1353  346.0194


objKeypoints(:,:,5) =

  573.6482   89.4887
  582.4121   80.3383
  569.0730   84.9135
  591.4658   84.9135
  560.2125   89.4887
  614.7282  126.0901
  543.0713  126.0901
  633.8019  180.9923
  526.2200  176.4171
  625.9077  231.3193
  522.7077  222.1689
  599.1330  235.8945
  549.8689  235.8945
  600.5824  304.5222
  551.2217  299.9470
  601.9352  368.5747
  552.3812  354.8492


objKeypoints(:,:,6) =

  720.0562   83.7954
  729.2193   79.0625
  715.3233   79.0625
  743.3170   88.5283
  706.2611   88.5283
  762.8539  135.8571
  698.0060  135.8571
  773.5303  202.1174
  680.5878  187.9188
  760.7441  254.1791
  677.0656  239.9805
  742.1152  249.4462
  700.3264  244.7133
  743.9312  334.6381
  706.6735  325.1723
  731.5486  405.6313
  712.8188  396.1655


keypointScores =

    0.9671    0.9648    0.9650    0.9617    0.9565    0.9564
```

```
    0.9809    0.9810    0.9627    0.9769    0.9846    0.9839
    0.9762    0.9953    0.9727    0.9730    0.9699    0.9814
    0.9646    0.9943    0.9740    0.9795    0.9659    0.9850
    0.9305    0.9845    0.9636    0.9675    0.9577    0.9677
    0.9603    0.9220    0.9320    0.9558    0.9397    0.8990
    0.9044    0.9620    0.8772    0.9620    0.9502    0.9549
    0.9584    0.9620    0.9544    0.9133    0.9080    0.9309
    0.9316    0.9707    0.9516    0.9333    0.9222    0.9300
    0.7920    0.9553    0.9716    0.9492    0.9421    0.9558
    0.9104    0.7432    0.9154    0.9533    0.9711    0.9333
    0.8087    0.8730    0.8530    0.8723    0.8607    0.8483
    0.7811    0.8598    0.8277    0.8443    0.8671    0.8340
    0.8606    0.9081    0.9165    0.9512    0.8997    0.9069
    0.8365    0.9004    0.9179    0.9260    0.9611    0.8890
    0.8982    0.9391    0.9052    0.9220    0.9360    0.9669
    0.7941    0.9696    0.8999    0.9185    0.9210    0.9494


keypointValid =

  17×6 logical array

   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
   1   1   1   1   1   1
```

Input Arguments

`filename` — Filename

character vector | string scalar

Filename of the MAT file containing the pretrained HRNet object keypoint detector, specified as a character vector or string scalar. The MAT file must exist on the MATLAB path and contain only the network to be loaded.

This input argument must be a compile-time constant.

Data Types: `char` | `string`

Output Arguments

`net` — HRNet object keypoint detector network

`hrnetObjectKeypointDetector` object

HRNet object keypoint detector network, returned as a `hrnetObjectKeypointDetector` object.

Version History

Introduced in R2023b

See Also

`hrnetObjectKeypointDetector` | `coder.loadDeepLearningNetwork` | `coder.DeepLearningConfig`

Topics

“Code Generation for Deep Learning Networks” (GPU Coder)
“Code Generation for Deep Learning Networks by Using cuDNN” (GPU Coder)
“Code Generation for Deep Learning Networks by Using TensorRT” (GPU Coder)
“Code Generation for Deep Learning Networks Targeting ARM Mali GPUs” (GPU Coder)

trainHRNetObjectKeypointDetector

Train HRNet object keypoint detector

Syntax

```
trainedKeypointDetector = trainHRNetObjectKeypointDetector(trainingData,
keypointDetector,options)
trainedKeypointDetector = trainHRNetObjectKeypointDetector(trainingData,
checkpoint,options)
[trainedKeypointDetector,info] = trainHRNetObjectKeypointDetector( ___ )
[ ___ ] = trainHRNetObjectKeypointDetector( ___ ,Name=Value)
```

Description

`trainedKeypointDetector = trainHRNetObjectKeypointDetector(trainingData, keypointDetector,options)` trains an object keypoint detector using the specified high resolution deep learning network (HRNet) `keypointDetector`. You can specify `keypointDetector` as a pretrained or custom HRNet object keypoint detector. The `options` input specifies training parameters for the keypoint detection network.

You can also use this syntax to fine-tune a pretrained HRNet object keypoint detector.

`trainedKeypointDetector = trainHRNetObjectKeypointDetector(trainingData, checkpoint,options)` resumes training from the saved detector checkpoint.

You can use this syntax to:

- Add more training data and continue training a keypoint detector.
- Improve training accuracy by increasing the maximum number of iterations.

`[trainedKeypointDetector,info] = trainHRNetObjectKeypointDetector( ___ )` returns information on the training progress, such as the training loss and learning rate for each iteration, using any combination of input arguments from previous syntaxes.

`[ ___ ] = trainHRNetObjectKeypointDetector( ___ ,Name=Value)` specifies options using one or more name-value arguments. For example, `ExperimentMonitor=[]` specifies not to track metrics using Experiment Manager.

Note This functionality requires Deep Learning Toolbox and the Computer Vision Toolbox Model for Object Keypoint Detection. You can install the Computer Vision Toolbox Model for Object Keypoint Detection from the Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Train HRNet Keypoint Detection Network for Hand Keypoints Detection

Fine-tune a pretrained HRNet object keypoint detector to detect the keypoints of a human hand in an image. The keypoint detector uses an HRNet-W32 deep learning network, trained on the COCO keypoint detection data set.

Download and load the hand pose data set using `helperDownloadHandPoseDataset` helper function. The hand pose data set is a labeled data set that contains 2500 images from the Large-Scale Multiview Hand Pose Dataset [1 on page 3-1685]. Extract a subset of the downloaded hand pose data set that contains the first 100 images. Each image in the data set contains a human hand with 21 annotated keypoints.

```
downloadFolder = tempdir;
dataset = helperDownloadHandPoseDataset(downloadFolder);
data = load(dataset);
handPoseDataset = data.handPoseDataset(1:100,:);
```

The training data is a table with three columns in which the first, second, and third columns contain the image filenames, keypoint locations, and hand bounding boxes, respectively. Each entry in the keypoint locations column consists of an *N*-by-2 matrix, where *N* is the number of keypoints present in the hand in the corresponding image. Each image contains only one hand, which is one object. Therefore, each row represents one object in an image. Add the full data path to the locally stored hand pose data folder.

```
handPoseDataset.imageFilename = fullfile(downloadFolder,"2DHandPoseDataAndGroundTruth","2DHandPos
```

Create an `ImageDatastore` object for loading the image data.

```
handPoseImds = imageDatastore(handPoseDataset.imageFilename);
```

Create an `ArrayDatastore` object for loading the ground truth keypoint location data.

```
handPoseArrds = arrayDatastore(handPoseDataset(:,2));
```

Create a `boxLabelDatastore` object for loading the bounding box locations.

```
handPoseBlds = boxLabelDatastore(handPoseDataset(:,3));
```

Combine the image, array, and box label datastores into a single datastore.

```
trainingData = combine(handPoseImds,handPoseArrds,handPoseBlds);
```

Specify the keypoint classes in a human hand.

```
keypointClasses = ["forefinger3","forefinger4","forefinger2","forefinger1", ...
    "middleFinger3","middleFinger4","middleFinger2","middleFinger1", ...
    "pinkyFinger3","pinkyFinger4","pinkyFinger2","pinkyFinger1", ...
    "ringFinger3","ringFinger4","ringFinger2","ringFinger1", ...
    "thumb3","thumb4","thumb2","thumb1","wrist"]';
```

Create an `hrnetObjectKeypointDetector` object, and configure it to detect the specified keypoints classes.

```
handKeypointDetector = hrnetObjectKeypointDetector("human-full-body-w32",keypointClasses);
handKeypointDetector.Network
```

```
ans =
  dlnetwork with properties:
```

```
         Layers: [1036×1 nnet.cnn.layer.Layer]
    Connections: [1196×2 table]
     Learnables: [1170×3 table]
          State: [584×3 table]
     InputNames: {'input_1'}
    OutputNames: {'finallayer'}
    Initialized: 1

  View summary with summary.
```

Specify training options for the hand keypoint detector.

```
options = trainingOptions("adam", ...
    MaxEpochs=20, ...
    InitialLearnRate=0.001, ...
    MiniBatchSize=16, ...
    LearnRateSchedule="piecewise", ...
    LearnRateDropFactor=0.1, ...
    LearnRateDropPeriod=12, ...
    VerboseFrequency=25, ...
    BatchNormalizationStatistics="moving", ...
    ResetInputNormalization=false);
```

Fine-tune the pretrained HRNet object keypoint detector on the new data set by using the `trainHRNetObjectKeypointDetector` function.

```
[trainedHandKeypointDetector,info] = trainHRNetObjectKeypointDetector(trainingData,handKeypointDe

*************************************************************************
Training a HRNet Object Keypoint Detector for the following keypoint classes:

* forefinger3
* forefinger4
* forefinger2
* forefinger1
* middleFinger3
* middleFinger4
* middleFinger2
* middleFinger1
* pinkyFinger3
* pinkyFinger4
* pinkyFinger2
* pinkyFinger1
* ringFinger3
* ringFinger4
* ringFinger2
* ringFinger1
* thumb3
* thumb4
* thumb2
* thumb1
* wrist

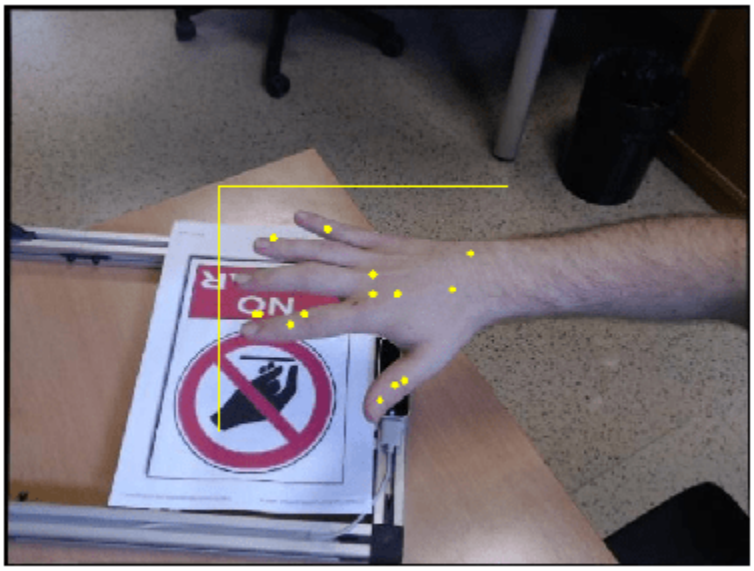
    Epoch    Iteration    TimeElapsed    LearnRate    TrainingLoss
    _____    _________    ___________    _________    ____________
```

```
      4          25          00:08:40         0.001        0.0018743
      8          50          00:16:21         0.001        0.0017478
     11          75          00:22:27         0.001        0.0014061
     15          100         00:28:27        0.0001        0.0012995
     18          125         00:34:44        0.0001        0.0013525

************************************************************************
Keypoint detector training complete.
************************************************************************
```

Read a test image. Use the trained HRNet hand keypoint detector to detect hand keypoints and display the detection results.

```
I = imread("test.jpg");
bbox= [185 156 249 211];
predictedKeypoints = detect(trainedHandKeypointDetector,I,bbox);
outputImg = insertObjectKeypoints(I,predictedKeypoints, ...
    KeypointColor="yellow",KeypointSize=3,LineWidth=3);
outputImg = insertShape(outputImg,rectangle=bbox);
figure
imshow(outputImg)
```



Supporting Function

```
function dataset = helperDownloadHandPoseDataset(downloadFolder)
dataFilename = "2DHandPoseDataAndGroundTruth.zip";
dataAndImageUrl = "https://ssd.mathworks.com/supportfiles/vision/data/2DHandPose/" + dataFilename
zipFile = fullfile(downloadFolder,dataFilename);
if ~exist(zipFile,"file")
    disp("Downloading hand pose dataset (98 MB)...")
    websave(zipFile,dataAndImageUrl);
end
unzip(zipFile,downloadFolder)
```

```
dataset = fullfile(downloadFolder,"2DHandPoseDataAndGroundTruth","2DHandPoseGroundTruth.mat");
end
```

References

[1] Gomez-Donoso, Francisco, Sergio Orts-Escolano, and Miguel Cazorla. "Large-Scale Multiview 3D Hand Pose Dataset." *Image and Vision Computing* 81 (January 2019): 25–33. https://doi.org/10.1016/j.imavis.2018.12.001.

Input Arguments

`trainingData` — Labeled ground truth images

datastore

Labeled ground truth images, specified as a datastore. Your data must be set up so that using the `read` or `readall` function on the datastore returns a cell array with three columns in the order {*ImageData*,*Keypoint*,*BoundingBox*}. This table describes the format of each cell in a row.

ImageData	Keypoint	BoundingBox
Grayscale, RGB, or multi-channel image that serves as a network input, specified as an *H*-by-*W* numeric matrix, *H*-by-*W*-by-3 numeric array, or *H*-by-*W*-by-*C* numeric array, respectively, where: • *H* is the height of the image. • *W* is the width of the image. • *C* is number of channels in the image.	Keypoint locations, defined in spatial coordinates as an *N*-by-2 or *N*-by-3 numeric matrix with rows of the form [*x y*] or [*x y v*], respectively, where: • *N* is the number of keypoint classes. • *x* and `y` specify the spatial coordinates of a keypoint. • *v* specifies the visibility of a keypoint.	Bounding box, defined in spatial coordinates as a 1-by-4 numeric vector of the form [*x y w h*], where: • *x* and `y` specify the upper-left corner of the rectangle. • *w* specifies the width of the rectangle, which is its length along the *x*-axis. • *h* specifies the height of the rectangle, which is its length along the *y*-axis.

Use the `combine` function on an `ImageDatastore`, an `arrayDatastore` and a `boxLabelDatastore` datastore, in that order, to create a combined datastore that returns these three data columns using `read`.

`keypointDetector` — Pretrained or untrained HRNet object keypoint detector

`hrnetObjectKeypointDetector` object

Pretrained or untrained HRNet object keypoint detector, specified as an `hrnetObjectKeypointDetector` object.

`options` — Training options

`TrainingOptionsSGDM` object | `TrainingOptionsRMSProp` object | `TrainingOptionsADAM` object

Training options, specified as a `TrainingOptionsSGDM`, `TrainingOptionsRMSProp`, or `TrainingOptionsADAM` object returned by the `trainingOptions` function. To specify the solver name and other options for network training, use the `trainingOptions` function. You must set the `BatchNormalizationStatistics` property of the object to `"moving"`.

`checkpoint` — Saved detector checkpoint
`hrnetObjectKeypointDetector` object

Saved detector checkpoint, specified as an `hrnetObjectKeypointDetector` object. To periodically save a detector checkpoint during training as a MAT file, specify a location for the file using the `CheckpointPath` property of the training options object `options`. To control how frequently the detector saves checkpoints, use the `CheckPointFrequency` and `CheckPointFrequencyUnit` properties of the training options object.

To load a checkpoint for a previously trained detector, load the MAT file from the checkpoint path. For example, this code loads the checkpoint MAT file of a detector from the `"checkpath"` folder in the current working directory to which the detector saves checkpoints during training.

```
data = load("checkpath/net_checkpoint__6__2023_11_17__16_03_08.mat");
checkpoint = data.net;
```

The name of each MAT file includes the iteration number and timestamp at which the detector saves the checkpoint. The file stores the detector in the `net` variable. To continue training the network, specify the detector extracted from the file to the `trainHRNetObjectKeypointDetector` function.

```
HRNetDetector = trainHRNetObjectKeypointDetector(trainingData,checkpoint,options);
```

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `trainHRNetObjectKeypointDetector(trainingData,ExperimentMonitor=[])` specifies not to track metrics with Experiment Manager.

`ExperimentMonitor` — Detector training experiment monitoring
`[]` (default) | `experiments.Monitor` object

Detector training experiment monitoring, specified as an `experiments.Monitor` object for use with the Experiment Manager app. You can use this object to track the progress of training, update information fields in the training results table, record values of the metrics used by the training, or produce training plots. For more information on using this app, see the “Train Object Detectors in Experiment Manager” example.

The app monitors this information during training:

- Training loss at each iteration
- Learning rate at each iteration
- Validation loss at each iteration, if the `options` input contains validation data

Output Arguments

`trainedKeypointDetector` — Trained HRNet object keypoint detector
`hrnetObjectKeypointDetector` object

Trained HRNet object keypoint detector, returned as an `hrnetObjectKeypointDetector` object.

`info` — Training progress information
structure array

Training progress information, returned as a structure array with these fields. Each field corresponds to a stage of training.

- `TrainingLoss` — Training loss at each iteration. The `trainHRNetObjectKeypointDetector` function computes the mean squared error (MSE) loss between predicted and ground truth keypoint heatmaps. For more information, see “Getting Started with HRNet”.
- `BaseLearnRate` — Learning rate at each iteration.
- `OutputNetworkIteration` — Iteration number of the returned network.

Each field is a numeric vector with one element per training iteration. If the function does not calculate a metric at a specific iteration, the corresponding element of that vector has a value of `NaN`.

Version History

Introduced in R2024a

R2025a: MATLAB Compiler support will be removed
Warns starting in R2025a

Support for using MATLAB Compiler will be removed in a future release.

See Also

Functions
`loadHRNETObjectKeypointDetector` | `trainingOptions`

Objects
`hrnetObjectKeypointDetector`

Topics
“Multi-Object Tracking and Human Pose Estimation”
“Estimate Anchor Boxes from Training Data”
“Train Object Detectors in Experiment Manager”
“Getting Started with HRNet”
“Datastores for Deep Learning” (Deep Learning Toolbox)

trainNerfacto

Create and train Nerfacto Neural Radiance Field (NeRF) model

Syntax

```
nerf = trainNerfacto(trainingData,cameraPoses,intrinsics,outputFolder)
nerf = trainNerfacto(pretrainedModelFolder)
nerf = trainNerfacto( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

`nerf = trainNerfacto(trainingData,cameraPoses,intrinsics,outputFolder)` creates and trains a Nerfacto Neural Radiance Field (NeRF) model [1] from the Nerfstudio Library [2] on a collection of images taken from different camera poses. The function also creates a model folder which contains the model weights and other parameters in the path that you specified using the `outputFolder` argument.

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

`nerf = trainNerfacto(pretrainedModelFolder)` resumes training of a pretrained Nerfacto NeRF model.

`nerf = trainNerfacto( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination input of arguments from previous syntaxes. For example, `MaxIterations=1000` specifies to perform a maximum of 1000 training iterations on the NeRF model.

Examples

Train Neural Radiance Field Model and Generate Novel Views

Training Data Requirements

To train a NeRF model [1 on page 3-1692] from the Nerfstudio Library [2 on page 3-1692] on a scene, your training set must contain:

- At least 100 images of the scene from multiple overlapping views of the region of interest.
- Camera poses associated with each image.
- Intrinsic parameters of camera used to capture the images.

To calculate the intrinsic parameters of camera through calibration, use the Camera Calibrator app. For information on estimating camera poses and related geometric information, see the “Structure from Motion from Multiple Views” example.

Download Training Data

The training data is located in the `sfmTrainingDataTUMRGBD` folder, which is stored in the `tum_rgbd_nerfacto.zip` file. Download and extract the `tum_rgbd_nerfacto.zip` file in the current directory by running this code.

The `tum_rgbd_nerfacto.zip` file contains the `sfmTrainingDataTUMRGBD` folder. The folder contains the `cameraInfo.mat` file which contains the camera poses associated with each image and the intrinsic parameter of camera, and the `images` folder which contains indoor images from the TUM RGB-D dataset [3 on page 3-1692].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_data.zip","file")
    websave("tum_rgbd_data.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_rgbd
    unzip(fullfile("tum_rgbd_data.zip"), pwd);
end
```

Set Up Training Data

Specify the path to the `images` folder, and then create an `imageDatastore` object using the path.

```
trainingImageFolder = fullfile("sfmTrainingDataTUMRGBD","images");
imds = imageDatastore(trainingImageFolder);
disp("Number of images: " + length(imds.Files))
```

```
Number of images: 104
```

Show the first training image.

```
figure
sampleTrainingImage = preview(imds);
imshow(sampleTrainingImage)
title("Sample Training Image")
```



Load the camera poses corresponding to the training images, and the intrinsic parameters of the camera used to capture the images.

```
camInfo = load(fullfile("sfmTrainingDataTUMRGBD","cameraInfo.mat"));
trainIntrinsics = camInfo.intrinsics;
trainPoses = camInfo.cameraPoses;
```

Display the camera poses. Observe that there are 104 camera poses, which corresponds to the 104 training images in the image datastore `imds`.

```
disp(trainPoses)
```

```
  1×104 rigidtform3d array with properties:

    Dimensionality
    Translation
    R
    A
```

Display the camera intrinsics. The `trainNerfacto` function assumes that all images were taken by a single camera, thus sharing the same intrinsics parameters for all training images.

```
disp(trainIntrinsics)
```

```
  cameraIntrinsics with properties:

            FocalLength: [535.1307 532.1860]
         PrincipalPoint: [323.3722 239.7986]
              ImageSize: [480 640]
       RadialDistortion: [-0.0032 0.0192]
    TangentialDistortion: [-0.0029 -0.0023]
                    Skew: 0
                       K: [3×3 double]
```

Train NeRF Model

Use the `trainNerfacto` function to create and train a NeRF model on the training images across up to 1000 iterations. Store intermediate model checkpoints and configurations in the `nerfTUMTrainingOutput` folder. Training requires about 30 minutes on a Linux machine with an NVIDIA GeForce RTX 3090 GPU that has 24 GB of memory.

```
outputFolder = fullfile(pwd,"nerfTUMTrainingOutput");
nerf = trainNerfacto(imds,trainPoses,trainIntrinsics,outputFolder,MaxIterations = 1000);
```

Write Novel Views from Trained NeRF Model to Image Files

Write the views captured by the first 20 camera poses to a local folder, `generatedTrainingViewsTUM`, by using the `writeViews` function.

```
imdsGen = writeViews(nerf,trainPoses(1:20),"generatedTrainingViewsTUM");
```

Display a side-by-side comparison between the training images and the generated images of the scene at a camera pose. Each comparison includes these metrics that quantify the similarity between the training images and the generated images:

- Structural Similarity Index Metrics (SSIM) — Value in the range [0,1], where a value of 1 indicates the perfect similarity.

- Peak Signal-to-Noise Ratio (PSNR) — Values are in decibels, where higher values indicate better similarity.

`imageIdx` selects the camera pose to compare.

```
numImg = numel(imdsGen.Files);

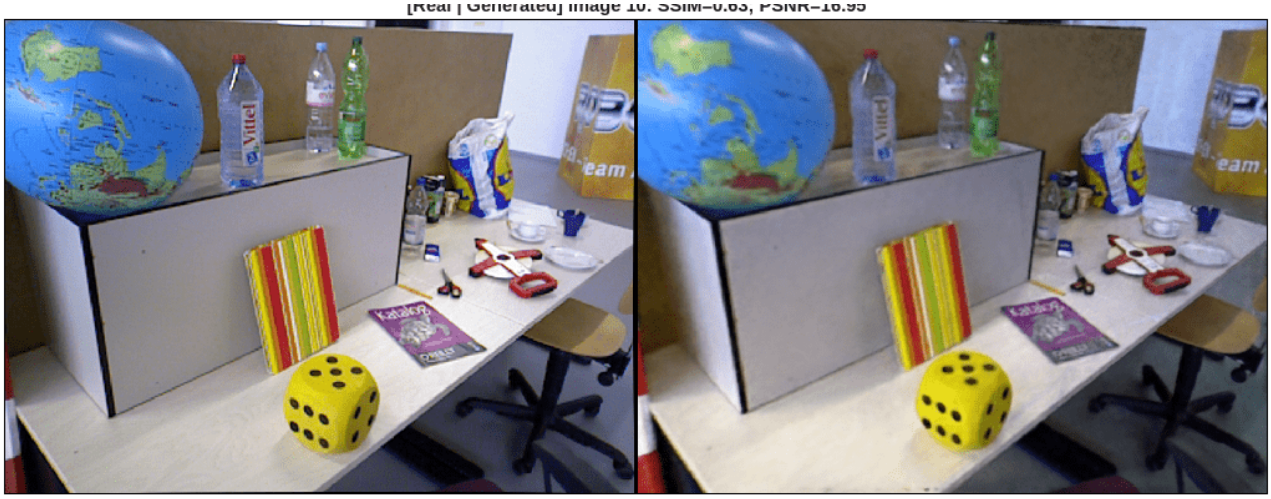
imageIdx =  10                                                  ;

% Load image generated by the trained NeRF
imageGen = readimage(imdsGen, imageIdx);

% Load the corresponding real image used in training
imageReal = readimage(imds,imageIdx);

% Compute image similarity metrics - SSIM and PSNR
ssimVal = ssim(rgb2gray(imageGen), rgb2gray(imageReal));
psnrVal = psnr(imageGen, imageReal);

% Visualize the images and their similarity maps
figure
montage({imageReal,imageGen}, BorderSize=[2 2])
title("[Real | Generated] Image " + imageIdx + ": SSIM="+num2str(ssimVal,2) + "; PSNR="+num2str(p
truesize;
```



Tips to Improve Training Results

Although the generated images are photorealistic, there are minor differences in the image texture and high-frequency details between the generated images and the training images. To achieve better quality results:

- Capture a larger set of high resolution images.
- Ensure the image set has good lighting.
- Ensure the image set has no motion blur or moving objects.
- Train the `nerfacto` object with a higher number of maximum training iterations (at the cost of longer computation time).

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

Input Arguments

`trainingData` — Training image collection

`imageDatastore` object

Training image collection, specified as an `imageDatastore` object. The image collection must contain three channel `uint8` RGB images in JPEG or PNG formats. All images must be in the same format.

Tip To improve the quality of the trained Nerfacto NeRF model, ensure that your training image collection:

- Contains at least 100 images that capture the subject from multiple overlapping views with good lighting
- Contains no motion blur or moving object

`cameraPoses` — Camera poses of training images

M-by-1 vector of `rigidtform3d` objects

Camera poses of the training images, specified as an *M*-by-1 vector of `rigidtform3d` objects, where *M* is the number images in the training image collection. Each element of the vector specifies the camera pose of the corresponding training image.

Tip For sequential training images, such as video frames, use the `monovslam` object to obtain the camera poses for each training image. To learn more about how to estimate camera poses from non-sequential training images, see “Structure from Motion from Multiple Views”.

`intrinsics` — Intrinsic parameters of camera

`cameraIntrinsics` object

Intrinsic parameters of the camera used to capture the training images, specified as a `cameraIntrinsics` object. Camera intrinsic parameters include the focal length, principal point, and skew of the camera.

Tip Use the Camera Calibrator app to obtain the camera intrinsic parameters from your training images. For an example of how to obtain camera intrinsic parameters using the app, see “Using the Single Camera Calibrator App”.

`outputFolder` — Output model folder path

string scalar | character vector

Output model folder path, specified as a string scalar or character vector. The function stores the trained NeRF model and the associated configuration file in this folder. The folder that you specify must be empty and writeable. If a folder that you specify in the path does not exist, the `trainNerfacto` function creates that folder.

Data Types: `char` | `string`

`pretrainedModelFolder` — Pretrained model folder path

string scalar | character vector

Pretrained model folder path, specified as a string scalar or a character vector. This path must match the path in the “ModelFolder” on page 2-0 property of the nerfacto object for the pretrained model.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `trainNerfacto(trainingData,cameraPoses intrinsics,OutputFolder,MaxIterations=1000)` specifies to perform a maximum of 1000 training iterations.

`MaxIterations` — Maximum training iterations

`30000` (default) | positive integer

Maximum number of training iterations, specified as a positive integer.

Tip This hyperparameter has the highest impact to the quality of the trained Nerfacto NeRF model. Increasing this value improves the quality of the scene reconstruction that the Nerfacto NeRF model generates, but increases the training time.

Data Types: `single` | `double`

`CheckpointFrequency` — Checkpoint saving interval

5 (default) | positive integer

Checkpoint saving interval, specified as a positive integer. This parameter specifies the iterations interval at which checkpoints are saved during model training.

Data Types: `single` | `double`

`SaveAllCheckpoints` — Keep all saved checkpoints
`false` or `0` (default) | `true` or `1`

Keep all saved checkpoints, specified as a logical `0` (`false`) or `1` (`true`). If you specify this value as `false`, the function only saves the most recent checkpoint and deletes all other checkpoints once training is complete.

Data Types: `logical`

`NumImagesToSample` — Number of images to sample in each iteration
inf (default) | positive integer

Number of images to sample in each training iteration, specified as a positive integer. By default, pixels are randomly sampled from all training images. Reducing this value reduces memory consumption during training.

Data Types: `single` | `double`

`OptimizeCameras` — Refine camera poses during training
`true` or `1` (default) | `false` or `0`

Refine camera poses during training, specified as a logical `1` (`true`) or `0` (`false`). Specify this value as `true` improves the training result If the initial camera poses of the training data are not accurate. If the initial camera poses are highly accurate, setting this value to `false` may improve the training result.

Data Types: `logical`

Output Arguments

`nerf` — Trained Nerfacto NeRF model
`nerfacto` object

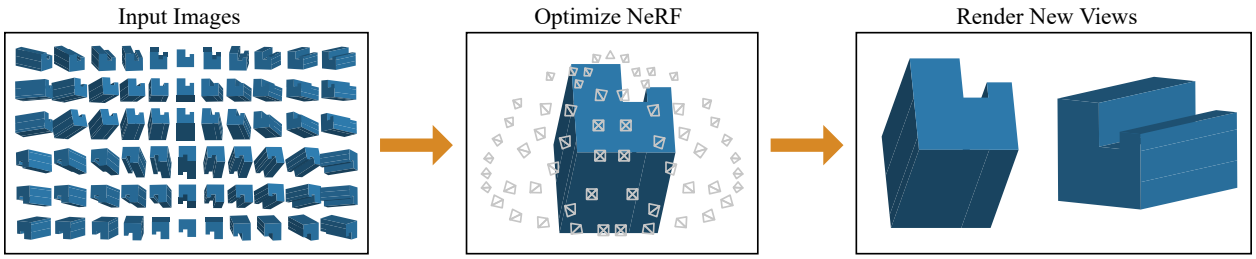
Trained Nerfacto NeRF model, returned as a `nerfacto` object.

Tips

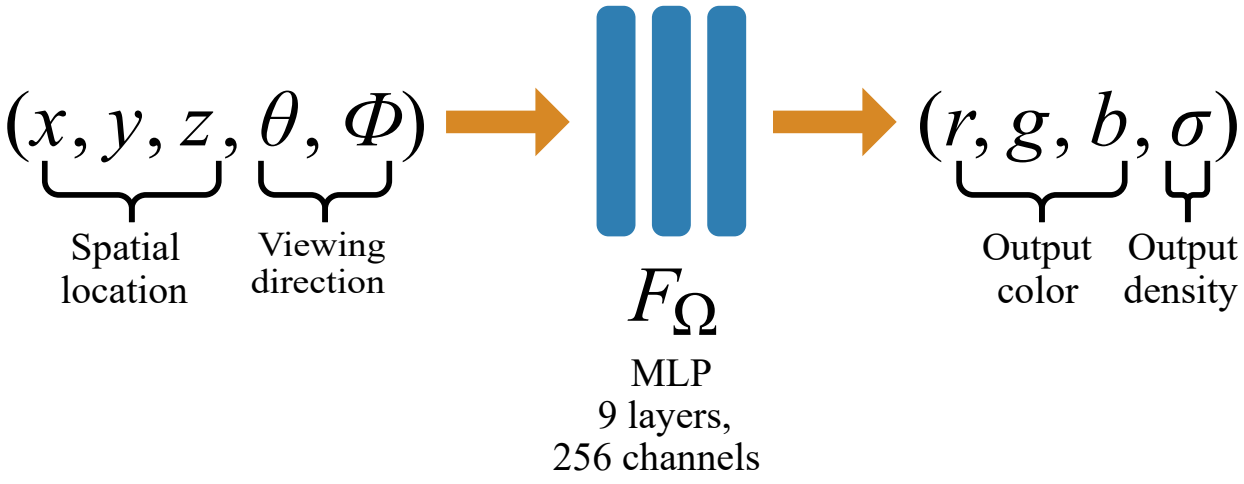
Training a Nerfacto NeRF model using high-resolution training images can cause out-of-memory errors. To resolve this, reduce the size of your training images using the `imresize` function, then obtain the camera poses and camera intrinsic parameters of the resized images.

Algorithms

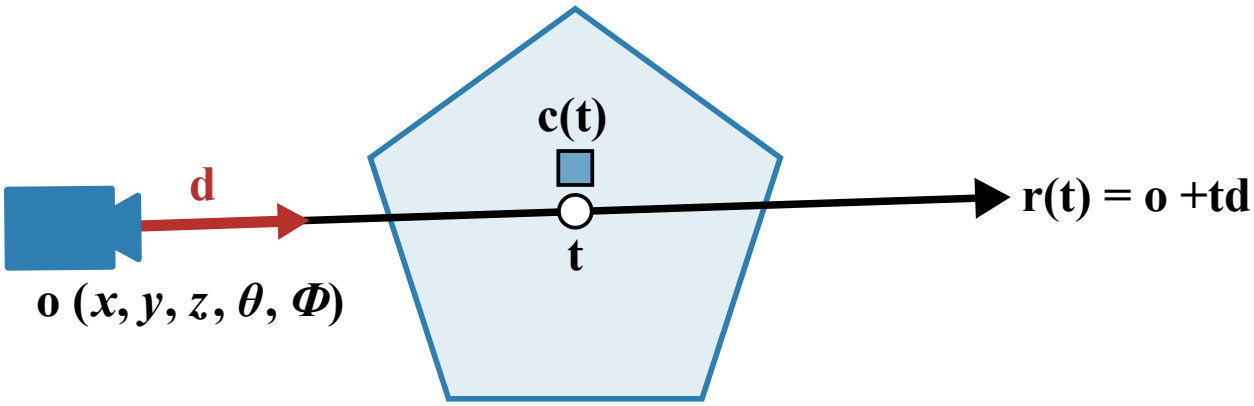
By training a NeRF model on a set of sparse input images and camera poses, you can create an internal representation of a 3D scene. You can use a trained NeRF model to generate a dense, colored point cloud, and render images from novel viewpoints.

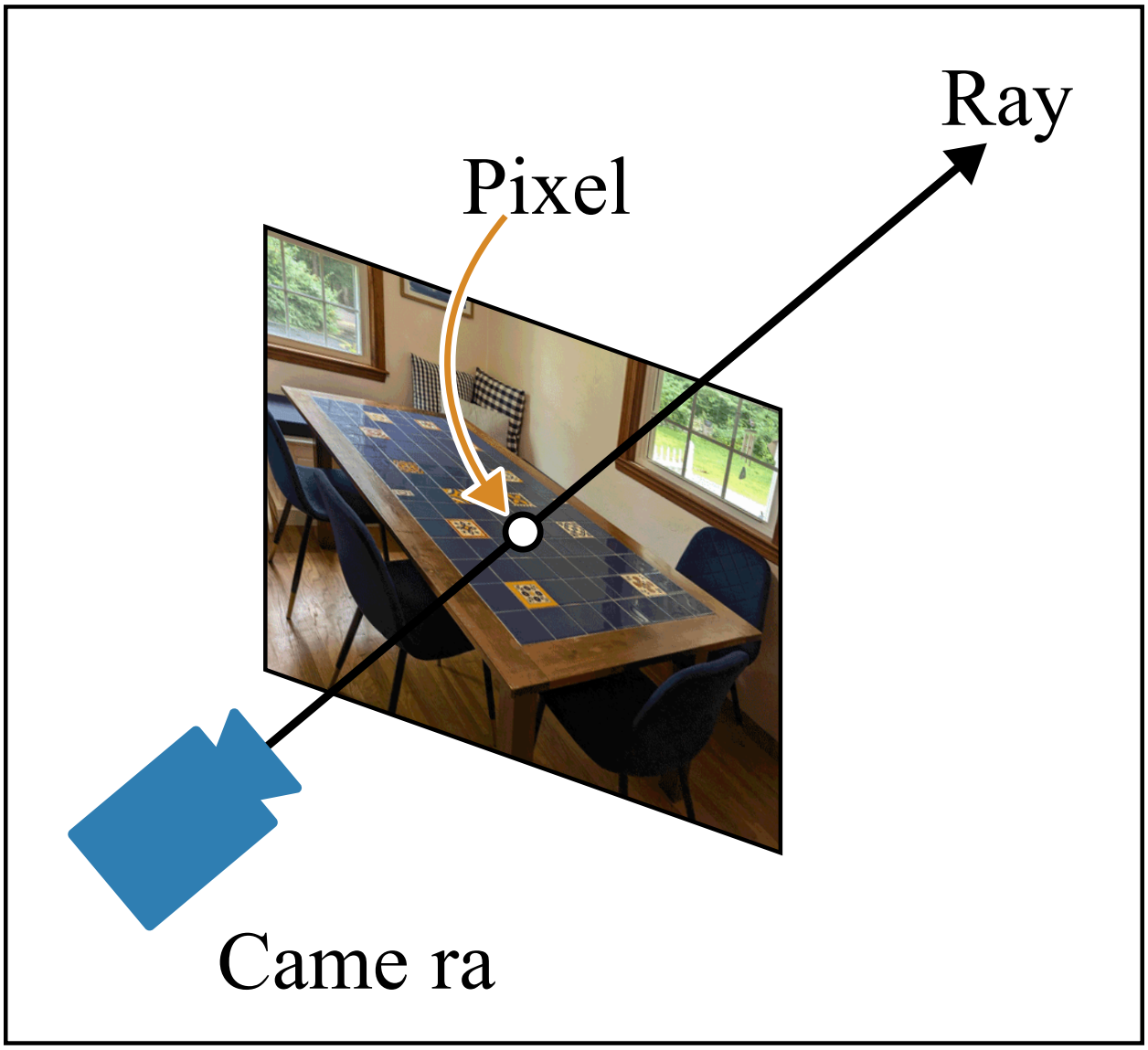

Input Images
Optimize NeRF
Render New Views

A NeRF model represents a scene as a continuous 5D vector-valued function instantiated as a neural network with weights Ω. The function accepts the position and direction of a light ray and returns the color and density of 3D points along the ray.

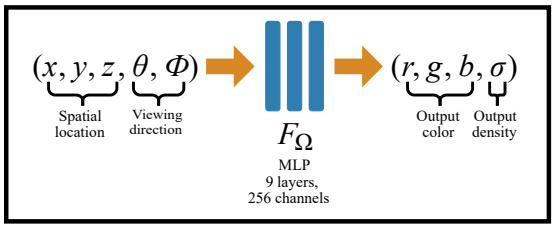

(x, y, z, θ, Φ)
Spatial location
Viewing direction
F_Ω
MLP
9 layers,
256 channels
(r, g, b, σ)
Output color
Output density

When generating a novel view, the NeRF model first uses the neural network to calculate the color and density of 3D points along each ray from a virtual camera. The ray is defined as $r(t) = o + td$, where o is the origin coordinates of the ray, and d is the unit vector in the ray direction defined by the camera angles θ and ϕ. The neural network return the color $c(t)$ if the ray $r(t)$ hits a particle at distance t along the ray. The model then uses volumetric rendering to project the output colors and densities into an image.

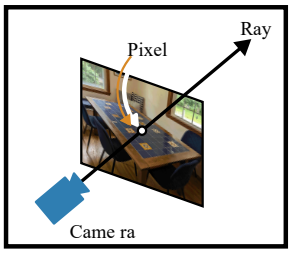

c(t)
d
r(t) = o +td
t
o (x, y, z, θ, Φ)


Ray
Pixel
Came ra

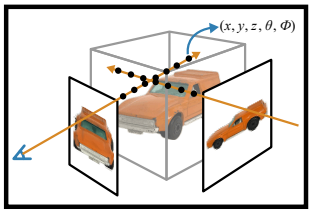
During training, a NeRF model first samples a batch of rays that consist of multiple spatial locations and viewing directions. The neural network F_Ω then predicts the color and density at 3D points along each ray, and the NeRF model uses volumetric rendering to project the output colors and densities onto an image. The model calculates the training loss by matching the image pixels of the rendered image against the captured images in the training data.



Predict color and density at 3D points along ray using neu ral net



Calculate color at each pixel using volumetric rendering



Calculate t raining loss by comparing rendered vs actual image pixels

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`monovslam` | `nerfacto` | `cameraIntrinsics` | `generateViews`

Topics

“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”
“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites

Nerfstudio Library GitHub Repository
Nerfstudio Library Documentation

generatePointCloud

Generate dense, colored point cloud using trained Nerfacto Neural Radiance Field (NeRF) model

Syntax

```
ptCloud = generatePointCloud(nerf)
ptCloud = generatePointCloud(nerf,MaxNumPoints=numPoints)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

`ptCloud = generatePointCloud(nerf)` generates a dense, colored point cloud using a trained Nerfacto NeRF model [1] from the Nerfstudio Library [2].

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

`ptCloud = generatePointCloud(nerf,MaxNumPoints=numPoints)` specifies the maximum number of points in the dense, colored point cloud

Examples

Generate Point Cloud Using Neural Radiance Field Model

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto` object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 3-1706], and the `nerfactoModelFolder` folder, which contains the data and configuration files required for the Nerfstudio Library [2 on page 3-1706] to load and execute the trained NeRF model [3 on page 3-1706].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
    unzip("tum_rgbd_nerfacto.zip",pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd, "trainedNerfactoTUMRGBD");
load(fullfile(modelRoot, "saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot, "nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

      ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex9089
    MaxIterations: 30000
       Intrinsics: [1×1 cameraIntrinsics]
```

Generate Point Cloud with Pretrained NeRF Model

Use the `generatePointCloud` function to generate a dense, colored point cloud of the scene. Set the maximum number of points in the point cloud to 100,000 by specifying the `MaxNumPoints` name-value argument. This value achieves a balance between the quality of the point cloud, the computing time, and storage space.

```
ptCloud = generatePointCloud(nerf, MaxNumPoints=100000);
```

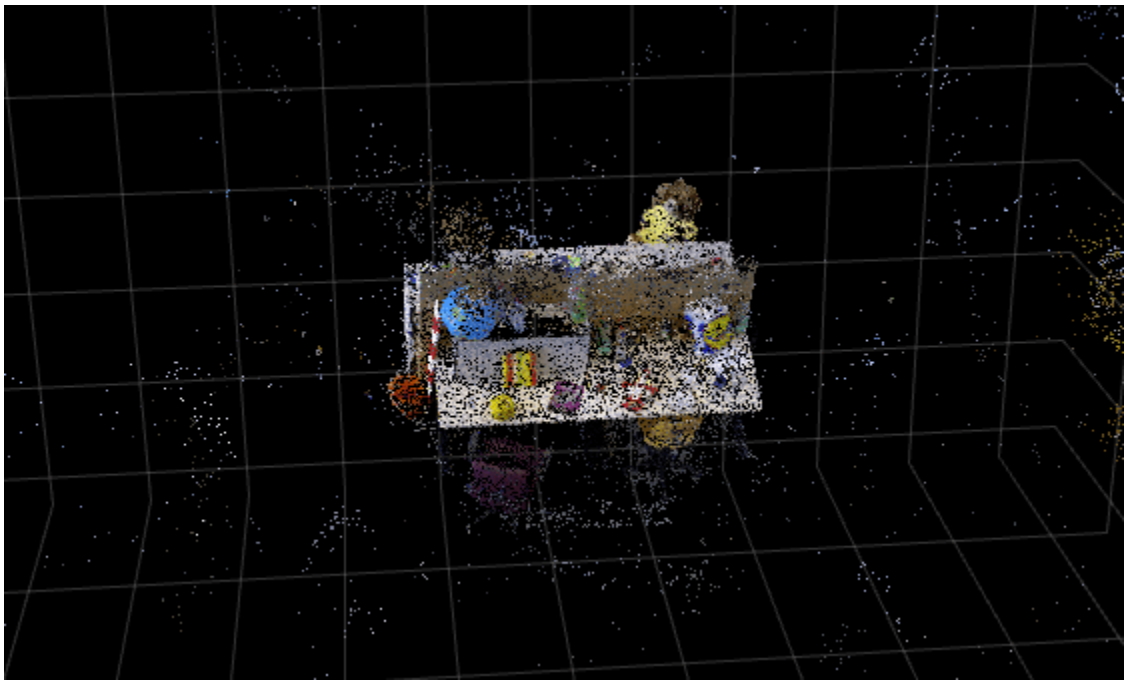
```
Generating point cloud from nerfacto object.
Loading latest checkpoint from load_dir
Setting up the Nerfstudio pipeline...
[12:27:23 PM] Auto image downscale factor of 1                                                  nerfs
Setting up training dataset...
Caching all 104 images.
Loading latest checkpoint from load_dir
nerfacto object point cloud generation complete.
```

Visualize Point Cloud

Visualize the point cloud by using the `pcshow` function, and specify the vertical axis and vertical axis direction to match the coordinate system of the TUM RGB-D data set used to train the `nerfacto` object. Modify the view of the point cloud visualization to focus on the scene region of interest by specifying the of low-level camera properties of the axes object. For more information on configuring these properties, see “Low-Level Camera Properties”.

```
figure
ax = pcshow(ptCloud, VerticalAxis="y",VerticalAxisDir="down");
ax.CameraPosition = [-1.5185 -6.5178 -11.9106];
ax.CameraUpVector = [0.0520 -0.8878 0.4572];
ax.CameraViewAngle = 15;
```

```
xlabel("X")
ylabel("Y")
zlabel("Z")
title("Point Cloud from NeRF")
```



Tip: You can use the `pcdenoise` function to clean up the outliers and noise in the point cloud.

References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Input Arguments

`nerf` — Trained Nerfacto NeRF model
`nerfacto` object

Trained Nerfacto NeRF model, specified as a `nerfacto` object.

`numPoints` — Maximum number of points in generated dense, colored point cloud
`500000` (default) | positive integer

Maximum number of points in the generated dense colored point cloud, specified as a positive integer. Increasing this value results in a higher quality point cloud, but increases the point cloud size and processing time.

Note The generated point cloud might contain fewer points than the maximum number of points specified because the Nerfacto NeRF model rejects outlier points during point cloud generation.

Data Types: `single` | `double`

Output Arguments

`ptCloud` — Dense, colored point cloud
`pointCloud` object

Dense, colored point cloud, returned as a `pointCloud` object. The object contains the location, color, and normal vector for each point in the point cloud.

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`nerfacto` | `generateViews` | `writeViews` | `pointCloud` | `monovslam`

Topics
“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”
“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites
Nerfstudio Library GitHub Repository

Nerfstudio Library Documentation

generateViews

Generate novel views using Nerfacto Neural Radiance Field (NeRF) model

Syntax

```
I = generateViews(nerf,cameraPoses)
I = generateViews( ___ ,Intrinsics=virtualCameraIntrinsics)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

`I = generateViews(nerf,cameraPoses)` generates novel views using the trained Nerfacto NeRF model [1] from the Nerfstudio library [2], for the specified virtual camera poses `cameraPoses`.

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

`I = generateViews( ___ ,Intrinsics=virtualCameraIntrinsics)` specifies the intrinsic parameters of the virtual camera in addition to all input arguments from the previous syntax.

Examples

Generate Novel Views using Neural Radiance Field Model

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto` object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 3-1712], and the `nerfactoModelFolder` folder, which contains the data and configuration files required for the Nerfstudio Library [2 on page 3-1712] to load and execute the trained NeRF model [3 on page 3-1712].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
    unzip("tum_rgbd_nerfacto.zip", pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd, "trainedNerfactoTUMRGBD");
load(fullfile(modelRoot, "saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot, "nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

     ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex39493
   MaxIterations: 30000
      Intrinsics: [1×1 cameraIntrinsics]
```

Load Camera Poses

This example contains a set of predefined camera poses that are stored as an array of `rigidtform3d` objects in the `camera-poses-tum.mat` file. These represent the position and orientation of the camera in the 3-D world coordinate system of the scene.

Load the `camera-poses-tum.mat` file into workspace, which contains the camera poses `camPoses`.

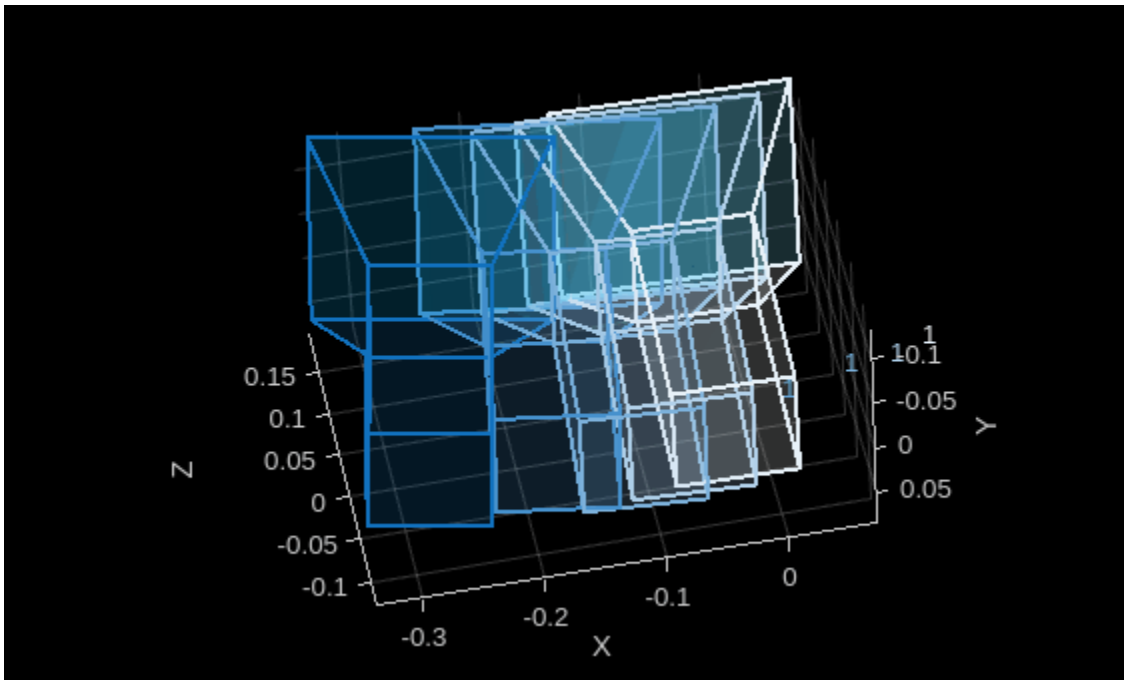
```
load("camera-poses-tum.mat","camPoses");
```

Display the camera poses by using the `plotCamera` function.

```
figure
pcshow([NaN NaN NaN], VerticalAxis="y", VerticalAxisDir="down");
xlabel("X")
ylabel("Y")
zlabel("Z")
hold on

colors = sky(length(camPoses));
for i = 1:length(camPoses)
plotCamera(camPoses(i), Size=0.1, Color=colors(i,:), Opacity=0.1);
end

hold off
```



Generate Novel Views with Pretrained NeRF Model

Generate the views captured by the camera poses `camPoses` by using the `generateViews` function.

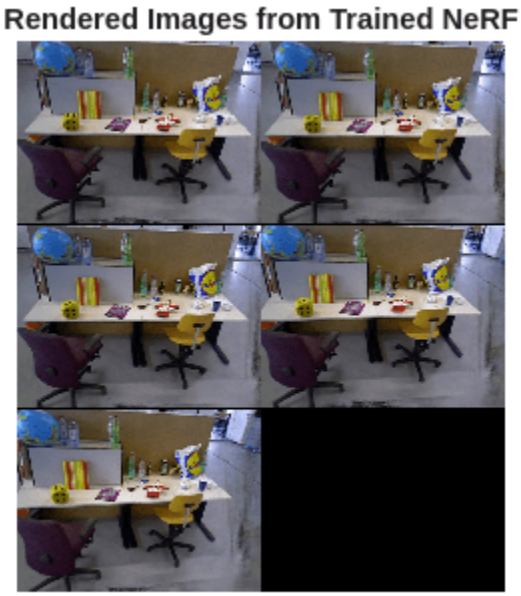
```
images = generateViews(nerf,camPoses);
```

```
Generating views from nerfacto object.
Loading latest checkpoint from load_dir
nerfacto object view generation complete.
```

Tip: If you want to generate a large number of novel views, use the `writeViews` function instead of `generateViews` to avoid an out-of-memory error.

Display the generated images. The images show the globe and chairs in front of the desk that are present in the images used to train this NeRF model.

```
figure
montage(images, BorderSize=[2 2])
title("Rendered Images from Trained NeRF")
```



References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Input Arguments

`nerf` — Trained Nerfacto NeRF model

`nerfacto` object

Trained Nerfacto NeRF model, specified as a `nerfacto` object.

`cameraPoses` — Virtual camera poses

M-by-1 vector of `rigidtform3d` objects

Virtual camera poses, specified as an *M*-by-1 vector of `rigidtform3d` objects, where *M* is the number novel views that you want to generate. Each element of the vector specifies a virtual camera

pose for a novel view that you want to generate in the world coordinate system. You must specify the virtual camera poses in the same world coordinate system as the camera poses of the training images.

Tip Generating a large number of views can cause an out of memory error because the `generateViews` function stores the generated images in memory. To generate a large number of views, use the `writeViews` function, which stores the generated images as files, instead.

`virtualCameraIntrinsics` — Virtual camera intrinsic parameters

`nerf.Instrinsics` (default) | `cameraIntrinsics` object

Virtual camera intrinsic parameters, specified as a `cameraIntrinsics` object. This function supports the intrinsic parameters of a pinhole camera model, which consist of focal length, principal point, and skew, and ignores other parameter values such as lens distortion.

By default, the `generateViews` function generates novel views using a virtual camera with the same intrinsic parameter as the camera that captures the training images. However, you can specify a virtual camera with intrinsic parameters that are different from the camera intrinsic parameters of the training images, such as to generate novel views with a higher resolution or wider field-of-view compared to the training images.

Output Arguments

`I` — Novel views

H-by-*W*-by-3-by-*M* array

Novel views, returned as an *H*-by-*W*-by-3-by-*M* array, where *M* is the number of novel views. Each image is of size *H*-by-*W*-by-3, where *H* and *W* are the height and width of the image, in pixels.

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`nerfacto` | `generatePointCloud` | `writeViews` | `monovslam` | `imageDatastore`

Topics
“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”
“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites
Nerfstudio Library GitHub Repository
Nerfstudio Library Documentation

writeViews

Write novel views from Nerfacto Neural Radiance Field (NeRF) model to image files

Syntax

```
imds = writeViews(nerf,cameraPoses)
imds = writeViews(nerf,cameraPoses,outputFolder)
imds = writeViews( ___ ,Name=Value)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

`imds = writeViews(nerf,cameraPoses)` writes novel views generated using the trained Nerfacto NeRF model [1] from the Nerfstudio Library [2] for the virtual camera poses `cameraPoses` to a folder named `render` in the current working directory. The function returns an image datastore that contains the paths to the image files.

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

`imds = writeViews(nerf,cameraPoses,outputFolder)` specifies the path of the folder to which to write the novel view images.

`imds = writeViews( ___ ,Name=Value)` specifies options using one or more name-value arguments in addition to any combination input of arguments from previous syntaxes.

For example, `ImageFormat="jpg"` specifies to write the generated novel view images to `.jpg` files.

Examples

Write Novel Views from Neural Radiance Field Model to Image Files

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto`object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 3-1718], and the `nerfactoModelFolder` folder, which contains the data and configuration files required for the Nerfstudio Library [2 on page 3-1718] to load and execute the trained NeRF model [3 on page 3-1718].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
```

```
    unzip("tum_rgbd_nerfacto.zip",pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd,"trainedNerfactoTUMRGBD");
load(fullfile(modelRoot,"saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot,"nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

      ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex6257
    MaxIterations: 30000
       Intrinsics: [1×1 cameraIntrinsics]
```

Load Camera Poses

This example contains a set of predefined camera poses that are stored as an array of `rigidtform3d` objects in the `camera-poses-tum.mat` file. These represent the position and orientation of the camera in the 3-D world coordinate system of the scene.

Load the `camera-poses-tum.mat` file into workspace, which contains the camera poses `camPoses`.

```
load("camera-poses-tum.mat","camPoses");
```

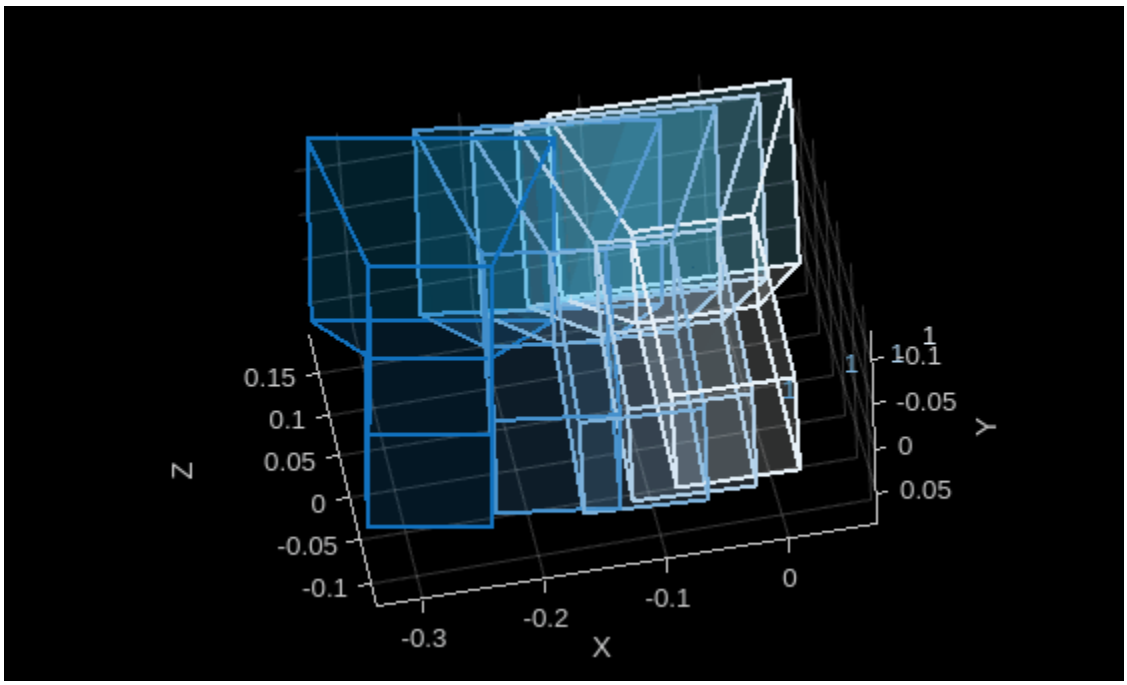
Plot the camera poses by using the `plotCamera` function.

```
figure
pcshow([NaN NaN NaN],VerticalAxis="y",VerticalAxisDir="down");
xlabel("X")
ylabel("Y")
zlabel("Z")
hold on

colors = sky(length(camPoses));
for i = 1:length(camPoses)
plotCamera(camPoses(i),Size=0.1,Color=colors(i,:),Opacity=0.1);
```

```
end

hold off
```



Write Novel Views from Pretrained NeRF Model to Image Files

Write the views captured by the camera poses `camPoses` to a local folder, `generatedNeRFViews`, by using the `writeViews` function.

```
imdsGen = writeViews(nerf,camPoses,"generatedNeRFViews");
```

```
Generating views from nerfacto object.
Loading latest checkpoint from load_dir
nerfacto object view generation complete.
```

Display the generated image of the scene at a camera pose. `imageIdx` selects the camera pose.

```
numImg = numel(imdsGen.Files);
```

```
imageIdx = 3 ;
```

```
imageGen = readimage(imdsGen,imageIdx);

figure
imshow(imageGen)
title("Generated Image " + imageIdx)
```



Tips to Improve Training Results

Although the generated images are photorealistic, there are minor differences in the image texture and high-frequency details between the generated images and the training images. To achieve better quality results:

- Capture a larger set of high resolution images.
- Ensure the image set has good lighting.
- Ensure the image set has no motion blur or moving objects.
- Train the `nerfacto` object with a higher number of maximum training iterations (at the cost of longer computation time).

References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer

Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Input Arguments

`nerf` — Trained Nerfacto NeRF model

`nerfacto` object

Trained Nerfacto NeRF model, specified as a `nerfacto` object.

`cameraPoses` — Virtual camera poses

M-by-1 vector of `rigidtform3d` objects

Virtual camera poses, specified as an *M*-by-1 vector of `rigidtform3d` objects, where *M* is the number novel views that you want to generate. Each element of the vector specifies a virtual camera pose for a novel view that you want to generate in the world coordinate system. You must specify the virtual camera poses in the same world coordinate system as the camera poses of the training images.

Tip Use the `generateViews` function to instead generate a small number of novel view images stored in memory.

`outputFolder` — Output images folder path

string scalar | character vector

Output images folder path, specified as a string scalar or character vector. If you specify a folder in the path that does not exist, the `writeViews` function creates that folder.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `writeViews(nerf,cameraPoses,ImageFormat="jpg")` writes the generated novel view images to the `.jpg` format.

`ImageFormat` — Output image file format

`"jpeg"` (default) | `"jpg"` | `"png"`

Output image file format, specified as one of these options:

- `"jpeg"` — Stores the output images as `.jpeg` files.
- `"jpg"` — Stores the output images as `.jpg` files.
- `"png"` — Stores the output images as `.png` files.

Data Types: `char` | `string`

`JPEGQuality` — Quality of generated JPEG images

`100` (default) | scalar in range `[0, 100]`

Quality of the generated JPEG images, specified as a scalar in the range `[0, 100]`. Specifying a smaller value results in images with lower quality and more compression, while specifying a larger value results in images with higher quality and less compression.

Dependencies

To specify this value, you must specify `ImageFormat` as `"jpeg"`

Data Types: `single` | `double`

`Intrinsics` — Virtual camera intrinsic parameters
`nerf.Instrinsics` (default) | `cameraIntrinsics` object

Virtual camera intrinsic parameters, specified as a `cameraIntrinsics` object. This function supports the intrinsic parameters of a pinhole camera model, which consist of focal length, principal point, and skew, and ignores other parameter values such as lens distortion.

By default, the `writeViews` function generates novel views using a virtual camera with the same intrinsic parameter as the camera that captures the training images. However, you can specify a virtual camera with intrinsic parameters that are different from the camera intrinsic parameters of the training images, such as to generate novel views with a higher resolution or wider field-of-view compared to the training images.

Output Arguments

`imds` — Output image datastore
`imageDatastore` object

Output image datastore, returned as an `imageDatastore` object. The `writeViews` function stores the images in the datastore in the corresponding order to the specified virtual camera poses.

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`nerfacto` | `generatePointCloud` | `generateViews` | `cameraIntrinsics` | `monovslam`

Topics

“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”

“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites
Nerfstudio Library GitHub Repository
Nerfstudio Library Documentation

changePath

Update model folder property of trained Nerfacto NeRF model

Syntax

```
nerfUpdated = changePath(nerf,newFolderPath)
```

Description

Add-On Required: This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on.

`nerfUpdated = changePath(nerf,newFolderPath)` updates the model folder property of a trained Nerfacto NeRF model [1] from the Nerfstudio library [2].

You can use this function to update the `ModelFolder` property of a trained `nerfacto` object , `nerf`, when you move the associated model folder to a new location, `newFolderPath`. This allows the object to access the model folder to generate a dense colored point cloud and create novel views of the scene.

Note This feature requires the Computer Vision Toolbox Interface for Nerfstudio Library add-on, a Deep Learning Toolbox license, a Parallel Computing Toolbox license, and a CUDA enabled NVIDIA GPU with at least 16 GB of available GPU memory.

Examples

Generate Novel Views using Neural Radiance Field Model

Download Pretrained NeRF Model

The `tum_rgbd_nerfacto.zip` file contains the `trainedNerfactoTUMRGBD` folder. The folder contains the `saved-nerfacto-tum-rgbd.mat` supporting file, which contains a `nerfacto` object that has been trained on a sequence of indoor images from the TUM RGB-D data set [1 on page 3-1725], and the `nerfactoModelFolder` folder, which contains the data and configuration files required for the Nerfstudio Library [2 on page 3-1725] to load and execute the trained NeRF model [3 on page 3-1725].

Download and extract `tum_rgbd_nerfacto.zip`.

```
if ~exist("tum_rgbd_nerfacto.zip","file")
    websave("tum_rgbd_nerfacto.zip","https://ssd.mathworks.com/supportfiles/3DReconstruction/tum_
    unzip("tum_rgbd_nerfacto.zip", pwd);
end
```

Specify the path to the folder containing the `saved-nerfacto-tum-rgbd` MAT file, and then load the pretrained `nerfacto` object into the workspace.

```
modelRoot = fullfile(pwd, "trainedNerfactoTUMRGBD");
load(fullfile(modelRoot, "saved-nerfacto-tum-rgbd.mat"))
```

```
Warning: The model folder for the nerfacto object could not be loaded. Update the model folder lo
  '\\mathworks\devel\sandbox\user\Shared\EX_25b\EX_nerfacto_feature\train_nerfacto_small'
```

MATLAB® displays a warning when you first load the pretrained `nerfacto` object into the workspace because the path to the model folder associated with the object, `nerfactoModelFolder`, changes when you extract the `nerfacto_saved_model.zip`.

To resolve this issue, use the `changePath` function to update the `ModelFolder` property of the trained `nerfacto` object to the current path of `nerfactoModelFolder`. This process can take a few minutes.

```
nerf = changePath(nerf, fullfile(modelRoot, "nerfactoModelFolder"));
```

```
Changing nerfacto object model folder path.
nerfacto object model folder path change complete.
```

Display the `nerfacto` object properties to verify that the `ModelFolder` property is set to the current path of `nerfactoModelFolder` on your system.

```
disp(nerf)
```

```
  nerfacto with properties:

      ModelFolder: "/home/user/Documents/MATLAB/ExampleManager/user.Bdoc.EX_26a_v1/vision-ex3949
    MaxIterations: 30000
       Intrinsics: [1×1 cameraIntrinsics]
```

Load Camera Poses

This example contains a set of predefined camera poses that are stored as an array of `rigidtform3d` objects in the `camera-poses-tum.mat` file. These represent the position and orientation of the camera in the 3-D world coordinate system of the scene.

Load the `camera-poses-tum.mat` file into workspace, which contains the camera poses `camPoses`.

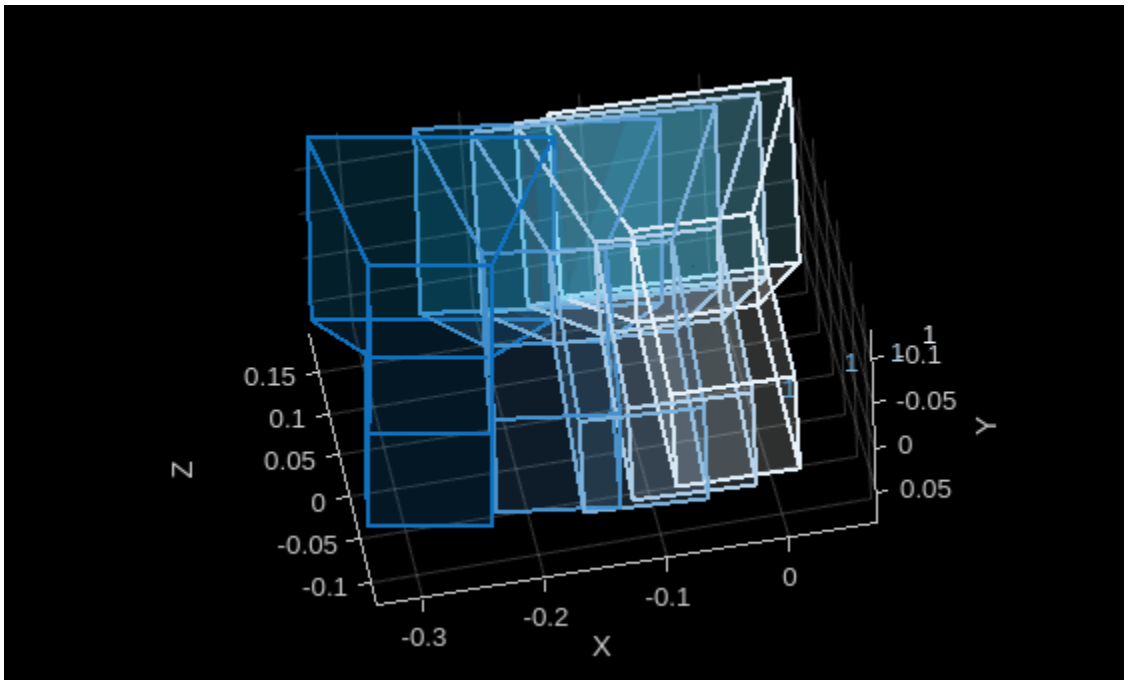
```
load("camera-poses-tum.mat","camPoses");
```

Display the camera poses by using the `plotCamera` function.

```
figure
pcshow([NaN NaN NaN], VerticalAxis="y", VerticalAxisDir="down");
xlabel("X")
ylabel("Y")
zlabel("Z")
hold on

colors = sky(length(camPoses));
for i = 1:length(camPoses)
plotCamera(camPoses(i), Size=0.1, Color=colors(i,:), Opacity=0.1);
end

hold off
```



Generate Novel Views with Pretrained NeRF Model

Generate the views captured by the camera poses `camPoses` by using the `generateViews` function.

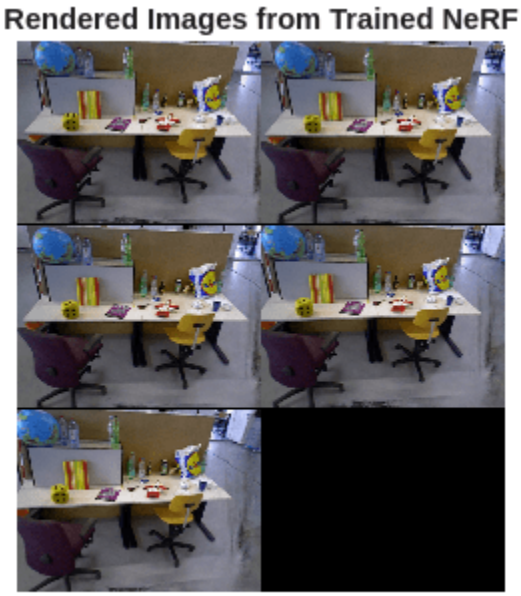
```
images = generateViews(nerf,camPoses);
```

```
Generating views from nerfacto object.
Loading latest checkpoint from load_dir
nerfacto object view generation complete.
```

Tip: If you want to generate a large number of novel views, use the `writeViews` function instead of `generateViews` to avoid an out-of-memory error.

Display the generated images. The images show the globe and chairs in front of the desk that are present in the images used to train this NeRF model.

```
figure
montage(images, BorderSize=[2 2])
title("Rendered Images from Trained NeRF")
```



References

[1] Sturm, Jürgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems.” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2012, 573–80. https://doi.org/10.1109/IROS.2012.6385773.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

[3] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." In Computer Vision - ECCV 2020, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

Input Arguments

`nerf` — Trained Nerfacto NeRF model

`nerfacto` object

Trained Nerfacto NeRF model, specified as a `nerfacto` object.

`newFolderPath` — New model folder path

string scalar | character vector

New model folder path, specified as a string scalar or character vector.

Data Types: `char` | `string`

Output Arguments

`nerfUpdated` — Trained Nerfacto NeRF model with updated model folder path

`nerfacto` object

Trained Nerfacto NeRF model with the updated model folder path, returned as a `nerfacto` object.

Version History

Introduced in R2026a

References

[1] Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” In *Computer Vision – ECCV 2020*, edited by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-58452-8_24.

[2] Tancik, Matthew, Ethan Weber, Evonne Ng, et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development.” *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, ACM, July 23, 2023, 1–12. https://doi.org/10.1145/3588432.3591516.

See Also

`trainNerfacto` | `nerfacto` | `generatePointCloud` | `generateViews`

Topics

“Reconstruct 3-D Scenes and Synthesize Novel Views Using Neural Radiance Field Model”
“What Is Structure from Motion?”
“Structure from Motion from Multiple Views”
“Monocular Visual Simultaneous Localization and Mapping”

External Websites

Nerfstudio Library GitHub Repository
Nerfstudio Library Documentation