Noise Removal

Digital images are prone to various types of noise. Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene.

Different types of noise:

Salt and pepper: random pixels being set to black or white (the extremes of the data range).MEDIAN

Gaussian:

Speckle:MEDIAN

# **imnoise**

Add noise to image

[collapse all in page](javascript:void(0);)

## Syntax

[J = imnoise(I,'gaussian')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165276?browser=F1help)

[J = imnoise(I,'gaussian',m)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165300?browser=F1help)

[J = imnoise(I,'gaussian',m,var\_gauss)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165319?browser=F1help)

[J = imnoise(I,'localvar',var\_local)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165344?browser=F1help)

[J = imnoise(I,'localvar',intensity\_map,var\_local)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165363?browser=F1help)

[J = imnoise(I,'poisson')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165392?browser=F1help)

[J = imnoise(I,'salt & pepper')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165407?browser=F1help)

[J = imnoise(I,'salt & pepper',d)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165420?browser=F1help)

[J = imnoise(I,'speckle')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165447)

[J = imnoise(I,'speckle',var\_speckle)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165466)

## Description

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165530),'gaussian') adds zero-mean, Gaussian white noise with variance of 0.01 to grayscale image I.

You optionally can add noise using a GPU (requires Parallel Computing Toolbox™).

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165530),'gaussian',[m](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165602)) adds Gaussian white noise with mean m and variance of 0.01.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165530),'gaussian',[m](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165602),[var\_gauss](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165627)) adds Gaussian white noise with mean m and variance var\_gauss.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165530),'localvar',[var\_local](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165652)) adds zero-mean, Gaussian white noise of local variance var\_local.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165530),'localvar',[intensity\_map](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165689),[var\_local](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165652)) adds zero-mean, Gaussian white noise. The local variance of the noise,var\_local, is a function of the image intensity values in I. The mapping of image intensity value to noise variance is specified by the vector intensity\_map.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165530),'poisson') generates Poisson noise from the data instead of adding artificial noise to the data. See [Algorithms](https://localhost:31515/static/help/images/ref/imnoise.html" \l "mw_226e1fb2-f53a-4e49-9bb1-6b167fc2eac1)for more information.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165530),'salt & pepper') adds salt and pepper noise, with default noise density 0.05. This affects approximately 5% of pixels.

[example](https://localhost:31515/static/help/images/ref/imnoise.html#bt56upa)

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165530),'salt & pepper',[d](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165720)) adds salt and pepper noise, where d is the noise density. This affects approximatelyd\*numel(I) pixels.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165530?browser=F1help),'speckle') adds multiplicative noise using the equation J = I+n\*I, where n is uniformly distributed random noise with mean 0 and variance 0.05.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser#d117e165530),'speckle',[var\_speckle](https://localhost:31515/static/help/images/ref/imnoise.html?snc=HGMHPF&browser=F1help&container=jshelpbrowser" \l "d117e165748?browser=F1help)) adds multiplicative noise with variance var\_speckle

Noise removal by average and medium filter

# filter2

2-D digital filter

[collapse all in page](javascript:void(0);)

## Syntax

[Y = filter2(H,X)](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#d117e411871)

[Y = filter2(H,X,shape)](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#d117e411894)

## Description

[example](https://localhost:31515/static/help/matlab/ref/filter2.html#bvjn16f-5)

Y = filter2([H](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#bvjn16f-1-H),[X](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#bvjn16f-1-X)) applies a finite impulse response filter to a matrix of data X according to coefficients in a matrix H.

[example](https://localhost:31515/static/help/matlab/ref/filter2.html#bvjn16f-5)

Y = filter2([H](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1" \l "bvjn16f-1-H),[X](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#bvjn16f-1-X),[shape](https://localhost:31515/static/help/matlab/ref/filter2.html?searchHighlight=filter2&searchResultIndex=1#bvjn16f-1-shape)) returns a subsection of the filtered data according to shape. For example, Y = filter2(H,X,'valid') returns only filtered data computed without zero-padded edges.

## Syntax

[h = fspecial(type)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81160)

[h = fspecial('average',hsize)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81192)

[h = fspecial('disk',radius)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81210)

[h = fspecial('gaussian',hsize,sigma)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81225)

[h = fspecial('laplacian',alpha)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81252)

[h = fspecial('log',hsize,sigma)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81267)

[h = fspecial('motion',len,theta)](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81288)

[h = fspecial('prewitt')](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81322)

[h = fspecial('sobel')](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81340)

## Description

[example](https://localhost:31515/static/help/images/ref/fspecial.html#mw_cca61637-6836-473e-aab4-f1ad39945549)

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial([type](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81414)) creates a two-dimensional filter h of the specified type. Some of the filter types have optional additional parameters, shown in the following syntaxes. fspecial returns h as a correlation kernel, which is the appropriate form to use with imfilter.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('average',[hsize](https://localhost:31515/static/help/images/ref/fspecial.html" \l "d117e81594)) returns an averaging filter h of size hsize.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('disk',[radius](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81645)) returns a circular averaging filter (pillbox) within the square matrix of size 2\*radius+1.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('gaussian',[hsize](https://localhost:31515/static/help/images/ref/fspecial.html" \l "d117e81594),[sigma](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81676)) returns a rotationally symmetric Gaussian lowpass filter of size hsize with standard deviation sigma. Not recommended. Use [imgaussfilt](https://localhost:31515/static/help/images/ref/imgaussfilt.html) or [imgaussfilt3](https://localhost:31515/static/help/images/ref/imgaussfilt3.html) instead.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('laplacian',[alpha](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81707)) returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator, alpha controls the shape of the Laplacian.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('log',[hsize](https://localhost:31515/static/help/images/ref/fspecial.html" \l "d117e81594),[sigma](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81676)) returns a rotationally symmetric Laplacian of Gaussian filter of size hsize with standard deviation sigma.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('motion',[len](https://localhost:31515/static/help/images/ref/fspecial.html" \l "d117e81739),[theta](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81770)) returns a filter to approximate, once convolved with an image, the linear motion of a camera. len specifies the length of the motion and theta specifies the angle of motion in degrees in a counter-clockwise direction. The filter becomes a vector for horizontal and vertical motions. The default len is 9 and the default theta is 0, which corresponds to a horizontal motion of nine pixels.

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('prewitt') returns a 3-by-3 filter that emphasizes horizontal edges by approximating a vertical gradient. To emphasize vertical edges, transpose the filter h'.

[ 1 1 1

0 0 0

-1 -1 -1 ]

[h](https://localhost:31515/static/help/images/ref/fspecial.html#d117e81806) = fspecial('sobel') returns a 3-by-3 filter that emphasizes horizontal edges using the smoothing effect by approximating a vertical gradient. To emphasize vertical edges, transpose the filter h'.

[ 1 2 1

0 0 0

-1 -2 -1 ]

ONLY WHEN IMG IS IN GRAYSCALE:

Average filter:

Kaverage = filter2(fspecial('average',3),J)/255;

figure

imshow(Kaverage)

## Input Arguments

[collapse all](javascript:void(0);)

### type — Type of filter 'average' | 'disk' | 'gaussian' | 'laplacian' | 'log' | 'motion' | 'prewitt' | 'sobel'

Type of filter, specified as one of the following values:

| **Value** | **Description** |
| --- | --- |
| 'average' | Averaging filter |
| 'disk' | Circular averaging filter (pillbox) |
| 'gaussian' | Gaussian lowpass filter. Not recommended. Use [imgaussfilt](https://localhost:31515/static/help/images/ref/imgaussfilt.html) or [imgaussfilt3](https://localhost:31515/static/help/images/ref/imgaussfilt3.html) instead. |
| 'laplacian' | Approximates the two-dimensional Laplacian operator |
| 'log' | Laplacian of Gaussian filter |
| 'motion' | Approximates the linear motion of a camera |
| 'prewitt' | Prewitt horizontal edge-emphasizing filter |
| 'sobel' | Sobel horizontal edge-emphasizing filter |

**Data Types:**char | string

### hsize — Size of the filter positive integer | 2-element vector of positive integers

Size of the filter, specified as a positive integer or 2-element vector of positive integers. Use a vector to specify the number of rows and columns in [h](https://localhost:31515/static/help/images/ref/fspecial.html?searchHighlight=fspecial&searchResultIndex=1#d117e81806). If you specify a scalar, then h is a square matrix.

When used with the 'average' filter type, the default filter size is [3 3]. When used with the Laplacian of Gaussian ('log') filter type, the default filter size is [5 5].

**Data Types:**double

### radius — Radius of a disk-shaped filter 5 (default) | positive number

Radius of a disk-shaped filter, specified as a positive number.

**Data Types:**double

### sigma — Standard deviation 0.5 (default) | positive number

Standard deviation, specified as a positive number.

**Data Types:**double

### alpha — Shape of the Laplacian 0.2 (default) | scalar in the range [0 1]

Shape of the Laplacian, specified as a scalar in the range [0 1].

**Data Types:**double

### len — Linear motion of camera 9 (default) | numeric scalar

Linear motion of camera, specified as a numeric scalar, measured in pixels.

**Data Types:**double

### theta — Angle of camera motion 0 (default) | numeric scalar

Angle of camera motion, specified as a numeric scalar, measured in degrees, in a counter-clockwise direction.

**Data Types:**double

## Output Arguments

[collapse all](javascript:void(0);)

### h — Correlation kernel matrix

Correlation kernel, returned as a matrix.

**Data Types:**double

## Algorithms

Averaging filters:

ones(n(1),n(2))/(n(1)\*n(2))

Gaussian filters:

*hg*(*n*1,*n*2)=*e*−(*n*21+*n*22)2*σ*2

*h*(*n*1,*n*2)=*hg*(*n*1,*n*2)Ξ*n*1Ξ*n*2*hg*

Laplacian filters:

∇2=∂2∂*x*2+∂2∂*y*2

∇2=4(*α*+1)2666666664*α*41−*α*4*α*41−*α*4−11−*α*4*α*41−*α*4*α*43777777775

Laplacian of Gaussian (LoG) filters:

*hg*(*n*1,*n*2)=*e*−(*n*21+*n*22)2*σ*2

*h*(*n*1,*n*2)=(*n*21+*n*22−2*σ*2)*hg*(*n*1,*n*2)*σ*4Ξ*n*1Ξ*n*2*hg*

Note that fspecial shifts the equation to ensure that the sum of all elements of the kernel is zero (similar to the Laplace kernel) so that the convolution result of homogeneous regions is always zero.

Motion filters:

1. Construct an ideal line segment with the length and angle specified by the arguments [len](https://localhost:31515/static/help/images/ref/fspecial.html?searchHighlight=fspecial&searchResultIndex=1" \l "d117e81739) and [theta](https://localhost:31515/static/help/images/ref/fspecial.html?searchHighlight=fspecial&searchResultIndex=1#d117e81770), centered at the center coefficient of h.
2. For each coefficient location (i,j), compute the nearest distance between that location and the ideal line segment.
3. h = max(1 - nearest\_distance, 0);
4. Normalize h: h = h/(sum(h(:)))

# medfilt2

2-D median filtering

[collapse all in page](javascript:void(0);)

## Syntax

[J = medfilt2(I)](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#d117e232987)

[J = medfilt2(I,[m n])](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#d117e233016)

[J = medfilt2(**\_\_\_**,padopt)](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#d117e233038)

## Description

[example](https://localhost:31515/static/help/images/ref/medfilt2.html#bupa2nx-2)

[J](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-B) = medfilt2([I](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-A)) performs median filtering of the image I in two dimensions. Each output pixel contains the median value in a 3-by-3 neighborhood around the corresponding pixel in the input image.

You optionally can compute the normalized cross-correlation using a GPU (requires Parallel Computing Toolbox™).

[J](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-B) = medfilt2([I](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-A),[[m n]](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-mn)) performs median filtering, where each output pixel contains the median value in the m-by-n neighborhood around the corresponding pixel in the input image.

[J](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser#bupa05x-1-B) = medfilt2(**\_\_\_**,[padopt](https://localhost:31515/static/help/images/ref/medfilt2.html?snc=NWFNEX&container=jshelpbrowser" \l "bupa05x-1-padopt)) controls how medfilt2 pads the image boundaries.

he median filter in image processing is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

### What is Median Filter ? Definition

**Median Filter** is a simple and powerful non-linear filter.

* It is used for reducing the amount of intensity variation between one pixel and the other pixel.
* In this filter, we replaces pixel value with the median value .
* The median is calculated by first sorting all the pixel values into ascending order and then replace the pixel being calculated with the middle pixel value
* Salt and pepper noise.

**Mean filtering** is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images.

### How to work the Median Filter

The median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings.

Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the **median** of those values.

The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.)

#### **Example 1: 3×3 Median Filter**

2D Median filtering example using a 3 x 3 sampling window:

Keeping border values unchanged

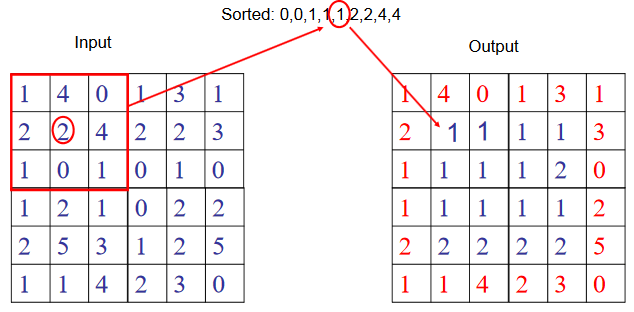


IMAGE FILTERING

## What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. (See [Neighborhood or Block Processing: An Overview](https://in.mathworks.com/help/images/neighborhood-or-block-processing-an-overview.html) for a general discussion of neighborhood operations.) Linear filtering is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

KERNEL

Box blur

Gaussian

Identity

Sharpen

### Convolution

Linear filtering of an image is accomplished through an operation called convolution. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the filter. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

A = [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]

and the correlation kernel is

h = [8 1 6

3 5 7

4 9 2]

You would use the following steps to compute the output pixel at position (2,4):

1. Rotate the correlation kernel 180 degrees about its center element to create a convolution kernel.
2. Slide the center element of the convolution kernel so that it lies on top of the (2,4) element of A.
3. Multiply each weight in the rotated convolution kernel by the pixel of A underneath.
4. Sum the individual products from step 3.

Hence the (2,4) output pixel is



Shown in the following figure.

**Computing the (2,4) Output of Convolution**

### Correlation

The operation called correlation is closely related to convolution. In correlation, the value of an output pixel is also computed as a weighted sum of neighboring pixels. The difference is that the matrix of weights, in this case called the correlation kernel, is not rotated during the computation. The Image Processing Toolbox™ filter design functions return correlation kernels.

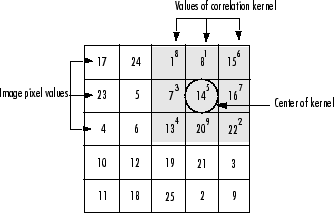
The following figure shows how to compute the (2,4) output pixel of the correlation of A, assuming h is a correlation kernel instead of a convolution kernel, using these steps:

1. Slide the center element of the correlation kernel so that lies on top of the (2,4) element of A.
2. Multiply each weight in the correlation kernel by the pixel of A underneath.
3. Sum the individual products.

The (2,4) output pixel from the correlation is



**Computing the (2,4) Output of Correlation**



For

## Mean or Average Filter in Image Processing

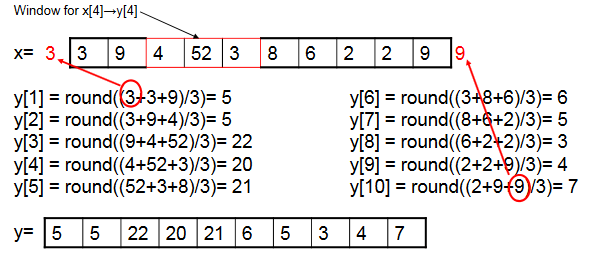
The average filter works by moving through the image pixel by pixel, replacing each value with the average value of neighboring pixels, including itself.

**There are some potential problems:**

* A single pixel with a very unrepresentative value can significantly affect the average value of all the pixels in its neighborhood.
* When the filter neighborhood straddles an edge, the filter will interpolate new values for pixels on the edge and so will blur that edge. This may be a problem if sharp edges are required in the output.

#### The following example shows the application of an average filter to a simple one dimensional signal.

A window size of three is used, with one entry immediately preceding and following each entry and following each entry.



## Functions

|  |  |
| --- | --- |
| [imageinfo](https://in.mathworks.com/help/images/ref/imageinfo.html) | Image Information tool |
| [imcolormaptool](https://in.mathworks.com/help/images/ref/imcolormaptool.html) | Choose Colormap tool |
| [imcontrast](https://in.mathworks.com/help/images/ref/imcontrast.html) | Adjust Contrast tool |
| [imcrop](https://in.mathworks.com/help/images/ref/imcrop.html) | Crop image |
| [imcrop3](https://in.mathworks.com/help/images/ref/imcrop3.html) | Crop 3-D image |
| [imdisplayrange](https://in.mathworks.com/help/images/ref/imdisplayrange.html) | Display Range tool |
| [imdistline](https://in.mathworks.com/help/images/ref/imdistline.html) | Distance tool |
| [impixelinfo](https://in.mathworks.com/help/images/ref/impixelinfo.html) | Pixel Information tool |
| [impixelinfoval](https://in.mathworks.com/help/images/ref/impixelinfoval.html) | Pixel Information tool without text label |
| [impixelregion](https://in.mathworks.com/help/images/ref/impixelregion.html) | Pixel Region tool |
| [impixelregionpanel](https://in.mathworks.com/help/images/ref/impixelregionpanel.html) | Pixel Region tool panel |
| [immagbox](https://in.mathworks.com/help/images/ref/immagbox.html) | Magnification box for image displayed in scroll panel |
| [imoverview](https://in.mathworks.com/help/images/ref/imoverview.html) | Overview tool for image displayed in scroll panel |
| [imoverviewpanel](https://in.mathworks.com/help/images/ref/imoverviewpanel.html) | Overview tool panel for image displayed in scroll panel |
| [imsave](https://in.mathworks.com/help/images/ref/imsave.html) | Save Image Tool |
| [imscrollpanel](https://in.mathworks.com/help/images/ref/imscrollpanel.html) | Scroll panel for interactive image navigation |
| [getimage](https://in.mathworks.com/help/images/ref/getimage.html) | Image data from axes |
| [getimagemodel](https://in.mathworks.com/help/images/ref/getimagemodel.html) | Image model object from image object |
| [imagemodel](https://in.mathworks.com/help/images/ref/imagemodel.html) | Image Model object |

|  |  |
| --- | --- |
| [axes2pix](https://in.mathworks.com/help/images/ref/axes2pix.html) | Convert axes coordinates to pixel coordinates |
| [imattributes](https://in.mathworks.com/help/images/ref/imattributes.html) | Information about image attributes |
| [imgca](https://in.mathworks.com/help/images/ref/imgca.html) | Get current axes containing image |
| [imgcf](https://in.mathworks.com/help/images/ref/imgcf.html) | Get current figure containing image |
| [imgetfile](https://in.mathworks.com/help/images/ref/imgetfile.html) | Display Open Image dialog box |
| [imputfile](https://in.mathworks.com/help/images/ref/imputfile.html) | Display Save Image dialog box |
| [imhandles](https://in.mathworks.com/help/images/ref/imhandles.html) | Get all image objects |
| [iptaddcallback](https://in.mathworks.com/help/images/ref/iptaddcallback.html) | Add function handle to callback list |
| [iptcheckmap](https://in.mathworks.com/help/images/ref/iptcheckmap.html) | Check validity of colormap |
| [iptcheckhandle](https://in.mathworks.com/help/images/ref/iptcheckhandle.html) | Check validity of handle |
| [iptgetapi](https://in.mathworks.com/help/images/ref/iptgetapi.html) | Get Application Programmer Interface (API) for handle |
| [iptGetPointerBehavior](https://in.mathworks.com/help/images/ref/iptgetpointerbehavior.html) | Retrieve pointer behavior from graphics object |
| [ipticondir](https://in.mathworks.com/help/images/ref/ipticondir.html) | Directories containing Image Processing Toolbox and MATLAB icons |
| [iptPointerManager](https://in.mathworks.com/help/images/ref/iptpointermanager.html) | Create pointer manager in figure |
| [iptremovecallback](https://in.mathworks.com/help/images/ref/iptremovecallback.html) | Delete function handle from callback list |
| [iptSetPointerBehavior](https://in.mathworks.com/help/images/ref/iptsetpointerbehavior.html) | Store pointer behavior structure in graphics object |
| [iptwindowalign](https://in.mathworks.com/help/images/ref/iptwindowalign.html) | Align figure windows |
| [makeConstrainToRectFcn](https://in.mathworks.com/help/images/ref/makeconstraintorectfcn.html) | Create rectangularly bounded drag constraint function |
| [truesize](https://in.mathworks.com/help/images/ref/truesize.html) | Adjust display size of image |

## Functions

|  |  |
| --- | --- |
| [imcrop](https://in.mathworks.com/help/images/ref/imcrop.html) | Crop image |
| [imcrop3](https://in.mathworks.com/help/images/ref/imcrop3.html) | Crop 3-D image |
| [imresize](https://in.mathworks.com/help/images/ref/imresize.html) | Resize image |
| [imresize3](https://in.mathworks.com/help/images/ref/imresize3.html) | Resize 3-D volumetric intensity image |
| [imrotate](https://in.mathworks.com/help/images/ref/imrotate.html) | Rotate image |
| [imrotate3](https://in.mathworks.com/help/images/ref/imrotate3.html) | Rotate 3-D volumetric grayscale image |
| [imtranslate](https://in.mathworks.com/help/images/ref/imtranslate.html) | Translate image |
| [impyramid](https://in.mathworks.com/help/images/ref/impyramid.html) | Image pyramid reduction and expansion |

## Functions

|  |  |
| --- | --- |
| [imwarp](https://in.mathworks.com/help/images/ref/imwarp.html) | Apply geometric transformation to image |
| [affineOutputView](https://in.mathworks.com/help/images/ref/affineoutputview.html) | Create output view for warping images |
| [fitgeotrans](https://in.mathworks.com/help/images/ref/fitgeotrans.html) | Fit geometric transformation to control point pairs |
| [findbounds](https://in.mathworks.com/help/images/ref/findbounds.html) | Find output bounds for spatial transformation |
| [fliptform](https://in.mathworks.com/help/images/ref/fliptform.html) | Flip input and output roles of spatial transformation structure |
| [makeresampler](https://in.mathworks.com/help/images/ref/makeresampler.html) | Create resampling structure |
| [maketform](https://in.mathworks.com/help/images/ref/maketform.html) | Create spatial transformation structure (TFORM) |
| [tformarray](https://in.mathworks.com/help/images/ref/tformarray.html) | Apply spatial transformation to N-D array |
| [tformfwd](https://in.mathworks.com/help/images/ref/tformfwd.html) | Apply forward spatial transformation |
| [tforminv](https://in.mathworks.com/help/images/ref/tforminv.html) | Apply inverse spatial transformation |

## Syntax

[J = imnoise(I,'gaussian')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165276?browser=F1help)

[J = imnoise(I,'gaussian',m)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165300?browser=F1help)

[J = imnoise(I,'gaussian',m,var\_gauss)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165319?browser=F1help)

[J = imnoise(I,'localvar',var\_local)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165344?browser=F1help)

[J = imnoise(I,'localvar',intensity\_map,var\_local)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165363?browser=F1help)

[J = imnoise(I,'poisson')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165392?browser=F1help)

[J = imnoise(I,'salt & pepper')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165407?browser=F1help)

[J = imnoise(I,'salt & pepper',d)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165420?browser=F1help)

[J = imnoise(I,'speckle')](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165447?browser=F1help)

[J = imnoise(I,'speckle',var\_speckle)](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165466?browser=F1help)

## Description

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165530),'gaussian') adds zero-mean, Gaussian white noise with variance of 0.01 to grayscale image I.

You optionally can add noise using a GPU (requires Parallel Computing Toolbox™).

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165530),'gaussian',[m](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165602)) adds Gaussian white noise with mean m and variance of 0.01.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165530),'gaussian',[m](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165602),[var\_gauss](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165627)) adds Gaussian white noise with mean m and variance var\_gauss.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165530),'localvar',[var\_local](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165652)) adds zero-mean, Gaussian white noise of local variance var\_local.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165530),'localvar',[intensity\_map](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165689?browser=F1help),[var\_local](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165652)) adds zero-mean, Gaussian white noise. The local variance of the noise,var\_local, is a function of the image intensity values in I. The mapping of image intensity value to noise variance is specified by the vector intensity\_map.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165530),'poisson') generates Poisson noise from the data instead of adding artificial noise to the data. See [Algorithms](https://localhost:31515/static/help/images/ref/imnoise.html" \l "mw_226e1fb2-f53a-4e49-9bb1-6b167fc2eac1)for more information.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165530),'salt & pepper') adds salt and pepper noise, with default noise density 0.05. This affects approximately 5% of pixels.

[example](https://localhost:31515/static/help/images/ref/imnoise.html#bt56upa)

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165530),'salt & pepper',[d](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165720)) adds salt and pepper noise, where d is the noise density. This affects approximatelyd\*numel(I) pixels.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165530),'speckle') adds multiplicative noise using the equation J = I+n\*I, where n is uniformly distributed random noise with mean 0 and variance 0.05.

[J](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165778) = imnoise([I](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser#d117e165530),'speckle',[var\_speckle](https://localhost:31515/static/help/images/ref/imnoise.html?snc=MQNZWP&browser=F1help&container=jshelpbrowser" \l "d117e165748?browser=F1help)) adds multiplicative noise with variance var\_speckle.

image and display it.

Smoothing Image

Filter the image with isotropic Gaussian smoothing kernels of increasing standard deviations. Gaussian filters are generally isotropic, that is, they have the same standard deviation along both dimensions. An image can be filtered by an isotropic Gaussian filter by specifying a scalar value for sigma.

Iblur1 = imgaussfilt(I,2);

Iblur2 = imgaussfilt(I,4);

Iblur3 = imgaussfilt(I,8);

Display the original image and all the filtered images.

figure

imshow(I)

title('Original image')



figure

imshow(Iblur1)

title('Smoothed image, \sigma = 2')



figure

imshow(Iblur2)

title('Smoothed image, \sigma = 4')



figure

imshow(Iblur1)

title('Smoothed image, \sigma = 2')



figure

imshow(Iblur2)

title('Smoothed image, \sigma = 4')



figure

imshow(Iblur1)

title('Smoothed image, \sigma = 2')



figure

imshow(Iblur2)

title('Smoothed image, \sigma = 4')



figure

imshow(Iblur1)

title('Smoothed image, \sigma = 2')



figure

imshow(Iblur2)

title('Smoothed image, \sigma = 4')



Filter the image with anisotropic Gaussian smoothing kernels. imgaussfilt allows the Gaussian kernel to have different standard deviations along row and column dimensions. These are called axis-aligned anisotropic Gaussian filters. Specify a 2-element vector for sigma when using anisotropic filters.

IblurX1 = imgaussfilt(I,[4 1]);

IblurX2 = imgaussfilt(I,[8 1]);

IblurY1 = imgaussfilt(I,[1 4]);

IblurY2 = imgaussfilt(I,[1 8]);

Noise Removing using gradient

Even with the use of Sobel, Roberts or Prewitts gradient operators, the image gradient may be too noisy. To overcome this, smooth the image using a Gaussian smoothing filter before computing image gradients. Use the imgaussfilt function to smooth the image. The standard deviation of the Gaussian filter varies the extent of smoothing. Since smoothing is taken care of by Gaussian filtering, the central or intermediate differencing gradient operators can be used.

# Reduce Noise in Image Gradients

This example demonstrates how to reduce noise associated with computing image gradients. Image gradients are used to highlight interesting features in images and are used in many feature detection algorithms like edge/corner detection. Reducing noise in gradient computations is crucial to detecting accurate features.

Read an image into the workspace and convert it to grayscale.

originalImage = imread('yellowlily.jpg');

originalImage = rgb2gray(originalImage);

imshow(originalImage)

To simulate noise for this example, add some Gaussian noise to the image.

noisyImage = imnoise(originalImage,'gaussian');

imshow(noisyImage)

Compute the magnitude of the gradient by using the imgradient and imgradientxy functions. imgradient finds the gradient magnitude and direction, and imgradientxy finds directional image gradients.

sobelGradient = imgradient(noisyImage);

imshow(sobelGradient,[])

title('Sobel Gradient Magnitude')

Looking at the gradient magnitude image, it is clear that the image gradient is very noisy. The effect of noise can be minimized by smoothing before gradient computation. imgradient already offers this capability for small amounts of noise by using the Sobel gradient operator. The Sobel gradient operators are 3x3 filters as shown below. They can be generated using the fspecial function.

hy = -fspecial('sobel')

hx = hy'

The hy filter computes a gradient along the vertical direction while smoothing in the horizontal direction. hx smooths in the vertical direction and computes a gradient along the horizontal direction. The 'Prewitt' and 'Roberts' method options also provide this capability.

Even with the use of Sobel, Roberts or Prewitts gradient operators, the image gradient may be too noisy. To overcome this, smooth the image using a Gaussian smoothing filter before computing image gradients. Use the imgaussfilt function to smooth the image. The standard deviation of the Gaussian filter varies the extent of smoothing. Since smoothing is taken care of by Gaussian filtering, the central or intermediate differencing gradient operators can be used.

sigma = 2;

smoothImage = imgaussfilt(noisyImage,sigma);

smoothGradient = imgradient(smoothImage,'CentralDifference');

imshow(smoothGradient,[])

title('Smoothed Gradient Magnitude')

# mgradient

Find gradient magnitude and direction of 2-D image

[collapse all in page](javascript:void(0);)

## Syntax

[[Gmag,Gdir] = imgradient(I)](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#d117e154137)

[[Gmag,Gdir] = imgradient(I,method)](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#d117e154170)

[[Gmag,Gdir] = imgradient(Gx,Gy)](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#d117e154197)

## Description

[[Gmag](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gmag),[Gdir](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-Gdir)] = imgradient([I](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-I)) returns the gradient magnitude, Gmag, and the gradient direction, Gdir, of the 2-D grayscale or binary image I.

You optionally can compute the gradient magnitude and direction using a GPU (requires Parallel Computing Toolbox™).

[example](https://localhost:31515/static/help/images/ref/imgradient.html#bthh53d)

[[Gmag](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gmag),[Gdir](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-Gdir)] = imgradient([I](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-I),[method](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-method)) returns the gradient magnitude and direction using the specified method.

[example](https://localhost:31515/static/help/images/ref/imgradient.html#bthh56q-1)

[[Gmag](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gmag),[Gdir](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-Gdir)] = imgradient([Gx](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gx),[Gy](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-Gy)) returns the gradient magnitude and direction from the directional gradients Gx and Gy in the x and y directions, respectively.

## Examples

[collapse all](javascript:void(0);)

### Calculate Gradient Magnitude and Direction Using Prewitt Method

[Open Live Script](matlab:openExample('images/CalculateGradientMagnitudeAndGradientDirectionExample'))

Read an image into workspace.

I = imread('coins.png');

Calculate the gradient magnitude and direction, specifying the Prewitt gradient operator.

[Gmag, Gdir] = imgradient(I,'prewitt');

Display the gradient magnitude and direction.

figure

imshowpair(Gmag, Gdir, 'montage');

title('Gradient Magnitude, Gmag (left), and Gradient Direction, Gdir (right), using Prewitt method')

### Calculate Gradient Magnitude and Direction Using Directional Gradients

[Open Live Script](matlab:openExample('images/CalculateDirGradientsAndMagnitudeAndDirectionExample'))

Read an image into workspace.

I = imread('coins.png');

Calculate the x- and y-directional gradients. By default, imgradientxy uses the Sobel gradient operator.

[Gx,Gy] = imgradientxy(I);

Display the directional gradients.

imshowpair(Gx,Gy,'montage')

title('Directional Gradients Gx and Gy, Using Sobel Method')

Calculate the gradient magnitude and direction using the directional gradients.

[Gmag,Gdir] = imgradient(Gx,Gy);

Display the gradient magnitude and direction.

imshowpair(Gmag,Gdir,'montage')

title('Gradient Magnitude (Left) and Gradient Direction (Right)')

## Input Arguments

[collapse all](javascript:void(0);)

### I — Input image 2-D grayscale image | 2-D binary image

Input image, specified as a 2-D grayscale or 2-D binary image.

**Data Types:**single | double | int8 | int32 | uint8 | uint16 | uint32 | logical

### method — Gradient operator 'sobel' (default) | 'prewitt' | 'central' | 'intermediate' | 'roberts'

Gradient operator, specified as one of the following values.

| **Method** | **Description** |
| --- | --- |
| 'sobel' | Sobel gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. For gradients in the vertical (y) direction, the weights are:  [ 1 2 1  0 0 0  -1 -2 -1 ]  In the x direction, the weights are transposed. |
| 'prewitt' | Prewitt gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. For gradients in the vertical (y) direction, the weights are:  [ 1 1 1  0 0 0  -1 -1 -1 ]  In the x direction, the weights are transposed. |
| 'central' | Central difference gradient. The gradient of a pixel is a weighted difference of neighboring pixels. In the y direction, dI/dy = (I(y+1) - I(y-1))/2. |
| 'intermediate' | Intermediate difference gradient. The gradient of a pixel is the difference between an adjacent pixel and the current pixel. In the y direction, dI/dy = I(y+1) - I(y). |
| 'roberts' | Roberts gradient operator. The gradient of a pixel is the difference between diagonally adjacent pixels. For gradients in one direction, the weights are:  [ 1 0  0 -1 ]  In the orthogonal direction, the weights are flipped along the vertical axis. |

**Data Types:**char | string

### Gx — Horizontal gradient numeric matrix

Horizontal gradient, specified as a numeric matrix. The horizontal (x) axis points in the direction of increasing column subscripts. You can use the [imgradientxy](https://localhost:31515/static/help/images/ref/imgradientxy.html) function to calculate Gx.

**Data Types:**single | double | int8 | int32 | uint8 | uint16 | uint32

### Gy — Vertical gradient numeric matrix

Vertical gradient, specified as a numeric matrix of the same size as [Gx](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gx). The vertical (y) axis points in the direction of increasing row subscripts. You can use the [imgradientxy](https://localhost:31515/static/help/images/ref/imgradientxy.html) function to calculate Gy.

**Data Types:**single | double | int8 | int32 | uint8 | uint16 | uint32

## Output Arguments

[collapse all](javascript:void(0);)

### Gmag — Gradient magnitude numeric matrix

Gradient magnitude, returned as a numeric matrix of the same size as image [I](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-I) or the directional gradients [Gx](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gx) and [Gy](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gy). Gmag is of class double, unless the input image or directional gradients are of classsingle, in which case it is of class single.

**Data Types:**double | single

### Gdir — Gradient direction numeric matrix

Gradient direction, returned as a numeric matrix of the same size as gradient magnitude [Gmag](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gmag). Gdir contains angles in degrees within the range [-180, 180] measured counterclockwise from the positive x-axis. (The x-axis points in the direction of increasing column subscripts.) Gdir is of class double, unless the input image [I](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1#bthgcil-I) or directional gradients are of class single, in which case it is of class single.

**Data Types:**double | single

## Tips

* When applying the gradient operator at the boundaries of the image, values outside the bounds of the image are assumed to equal the nearest image border value. This is similar to the 'replicate' boundary option in [imfilter](https://localhost:31515/static/help/images/ref/imfilter.html).

## Algorithms

The algorithmic approach taken in imgradient for each of the listed gradient methods is to first compute directional gradients, [Gx](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gx) and [Gy](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-Gy), in the x and y directions, respectively. The horizontal (x) axis points in the direction of increasing column subscripts. The vertical (y) axis points in the direction of increasing row subscripts. The gradient magnitude and direction are then computed from their orthogonal components Gxand Gy.

imgradient does not normalize the gradient output. If the range of the gradient output image has to match the range of the input image, consider normalizing the gradient image, depending on the [method](https://localhost:31515/static/help/images/ref/imgradient.html?searchHighlight=imgradient&searchResultIndex=1" \l "bthgcil-method)argument used. For example, with a Sobel kernel, the normalization factor is 1/8, for Prewitt, it is 1/6, and for Roberts it is 1/2.

CONTRAST

## Gamma Correction

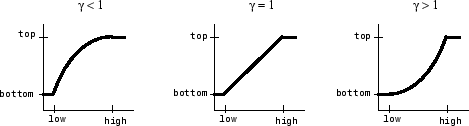
imadjust maps low to bottom, and high to top. By default, the values between low and high are mapped linearly to values between bottom and top. For example, the value halfway between low and high corresponds to the value halfway between bottom and top.

imadjust can accept an additional argument that specifies the gamma correction factor. Depending on the value of gamma, the mapping between values in the input and output images might be nonlinear. For example, the value halfway between low and high might map to a value either greater than or less than the value halfway between bottom and top.

Gamma can be any value between 0 and infinity. If gamma is 1 (the default), the mapping is linear. If gamma is less than 1, the mapping is weighted toward higher (brighter) output values. If gamma is greater than 1, the mapping is weighted toward lower (darker) output values.

The figure illustrates this relationship. The three transformation curves show how values are mapped when gamma is less than, equal to, and greater than 1. (In each graph, the x-axis represents the intensity values in the input image, and the y-axis represents the intensity values in the output image.)

**Plots Showing Three Different Gamma Correction Settings**



Adjust the contrast, specifying a gamma value of less than 1 (0.5). Notice that in the call to imadjust, the example specifies the data ranges of the input and output images as empty matrices. When you specify an empty matrix, imadjust uses the default range of [0,1]. In the example, both ranges are left empty. This means that gamma correction is applied without any other adjustment of the data.

J = imadjust(I,[],[],0.5);



### Make Contrast Adjustments Permanent

The Adjust Contrast tool adjusts the values of the pixels used to display the image in the Image Viewer but does not change the actual image data. To modify pixel values in the image to reflect the contrast adjustments you made, you must click the **Adjust Data** button.

The following example illustrates this process.

1. Display an image in the Image Viewer. The example opens an image from a file.

imtool('moon.tif');

1. Start the Adjust Contrast tool by clicking the **Adjust contrast** button, , or by selecting **Adjust Contrast** from the **Tools** menu in the Image Viewer.
2. Adjust the contrast of the image. Use one of the mechanisms provided by Adjust Contrast tool, such as resizing the window over the histogram. See [Adjust Image Contrast Using the Histogram Window](https://in.mathworks.com/help/images/adjust-image-contrast-in-image-viewer-app.html#brcolqe-1). You can also adjust contrast using the Window/Level tool, moving the pointer over the image.
3. Adjust the image data to reflect the contrast adjustment you just made. Click the Adjust Data button in the Adjust Contrast Tool. When you click the Adjust Data button, the histogram will update. You can then adjust the contrast again, if necessary. If you have other interactive modular tool windows open, they will update automatically to reflect the contrast adjustment.

### Specify Contast Adjustment Limits as Range

Try This Example

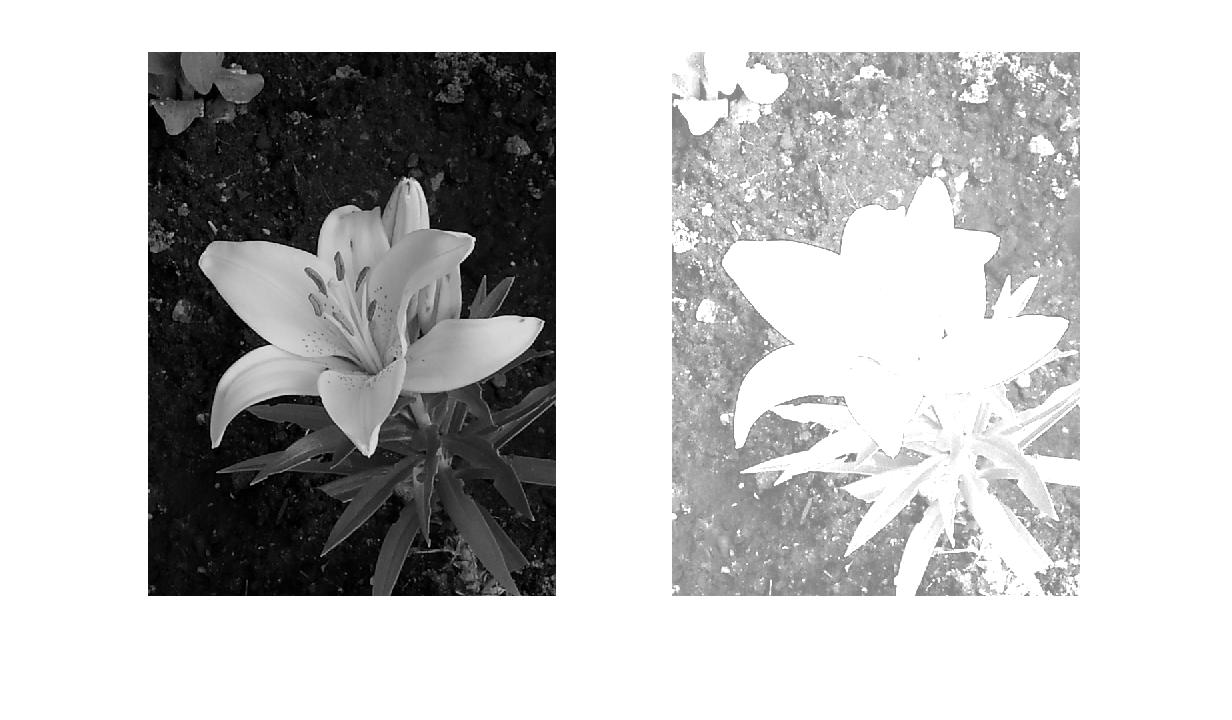
This example shows how to specify contast adjustment limits as a range using the imadjust function. This example decreases the contrast of an image by narrowing the range of the data.

Read an image into the workspace.

I = imread('cameraman.tif');

Adjust the contrast of the image, specifying the range of values used in the output image. In the example below, the man's coat is too dark to reveal any detail. imadjust maps the range [0,51] in the uint8 input image to [128,255] in the output image. This brightens the image considerably, and also widens the dynamic range of the dark portions of the original image, making it much easier to see the details in the coat. Note, however, that because all values above 51 in the original image are mapped to 255 (white) in the adjusted image, the adjusted image appears washed out.

J = imadjust(I,[0 0.2],[0.5 1]);



MORPHOLOGY

## Types of Morphological Operations

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors.

### Morphological Dilation and Erosion

The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion. This table lists the rules for both dilation and erosion.

**Rules for Dilation and Erosion**

| **Operation** | **Rule** | **Example (Original and Processed Image)** |
| --- | --- | --- |
| Dilation | The value of the output pixel is the maximum value of all pixels in the neighborhood. In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1.  Morphological dilation makes objects more visible and fills in small holes in objects. |  |
| Erosion | The value of the output pixel is the minimum value of all pixels in the neighborhood. In a binary image, a pixel is set to 0 if any of the neighboring pixels have the value 0.  Morphological erosion removes islands and small objects so that only substantive objects remain. |  |

### Operations Based on Dilation and Erosion

Dilation and erosion are often used in combination to implement image processing operations. For example, the definition of a morphological opening of an image is an erosion followed by a dilation, using the same structuring element for both operations. You can combine dilation and erosion to remove small objects from an image and smooth the border of large objects.

This table lists functions in the toolbox that perform common morphological operations that are based on dilation and erosion.

| **Function** | | **Morphological Definition** | **Example (Original and Processed Image)** | |
| --- | --- | --- | --- | --- |
| [imopen](https://in.mathworks.com/help/images/ref/imopen.html) | | Perform morphological opening. The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations.  Morphological opening is useful for removing small objects from an image while preserving the shape and size of larger objects in the image. For an example, see [Use Morphological Opening to Extract Large Image Features](https://in.mathworks.com/help/images/use-morphological-opening-to-extract-large-image-features.html). |  | |
| [imclose](https://in.mathworks.com/help/images/ref/imclose.html) | | Perform morphological closing. The closing operation dilates an image and then erodes the dilated image, using the same structuring element for both operations.  Morphological closing is useful for filling small holes from an image while preserving the shape and size of the objects in the image. |  | |
| [bwskel](https://in.mathworks.com/help/images/ref/bwskel.html) | *Skeletonize* objects in a binary image. The process of skeletonization erodes all objects to centerlines without changing the essential structure of the objects, such as the existence holes and branches. | | |  |
| [bwperim](https://in.mathworks.com/help/images/ref/bwperim.html) | Find perimeter of objects in a binary image. A pixel is part of the perimeter if it is nonzero and it is connected to at least one zero-valued pixel. | | |  |

bwhitmiss

|  |  |
| --- | --- |
| Perform binary hit-miss transform. The hit-miss transform preserves pixels in a binary image whose neighborhoods match the shape of one structuring element and do not match the shape of a second disjoint structuring element.  The hit-miss transforms can be used to detect patterns in an image. | This example uses one structuring element with a neighborhood above and to the right of center, and a second structuring element with a neighborhood below and to the left of center. The transform preserves pixels with neighbors only to the top and right. |
| [imtophat](https://in.mathworks.com/help/images/ref/imtophat.html) | Perform a morphological top-hat transform. The top-hat transform opens an image, then subtracts the opened image from the original image.  The top-hat transform can be used to enhance contrast in a grayscale image with nonuniform illumination. The transform can also isolate small bright objects in an image. |  |
| [imbothat](https://in.mathworks.com/help/images/ref/imbothat.html) | Perform morphological bottom-hat transform. The bottom-hat transform closes an image, then subtracts the original image form the closed image.  The bottom-hat transform can be used to find intensity troughs in a grayscale image. |  |

## escription

A strel object represents a flat morphological structuring element, which is an essential part of morphological dilation and erosion operations.

A flat structuring element is a binary valued neighborhood, either 2-D or multidimensional, in which the true pixels are included in the morphological computation, and the false pixels are not. The center pixel of the structuring element, called the origin, identifies the pixel in the image being processed. Use the strel function (described below) to create a flat structuring element. You can use flat structuring elements with both binary and grayscale images. The following figure illustrates a flat structuring element.

To create a nonflat structuring element, use [offsetstrel](https://localhost:31515/static/help/images/ref/offsetstrel.html).

## Creation

### Syntax

[SE = strel(nhood)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267347)

[SE = strel('arbitrary',nhood)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267359)

[SE = strel('diamond',r)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267372)

[SE = strel('disk',r,n)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267386)

[SE = strel('octagon',r)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267411)

[SE = strel('line',len,deg)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267428)

[SE = strel('rectangle',[m n])](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267453)

[SE = strel('square',w)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267468)

[SE = strel('cube',w)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267489)

[SE = strel('cuboid',[m n p])](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267503)

[SE = strel('sphere',r)](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267517)

### Description

SE = strel([nhood](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser" \l "mw_a493d9c9-affd-4d09-bdf9-7695025645eb)) creates a flat structuring element with specified neighborhood nhood.

You can also use the syntax SE = strel('arbitrary',[nhood](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser" \l "mw_a493d9c9-affd-4d09-bdf9-7695025645eb)) to create a flat structuring element with a specified neighborhood.

SE = strel('diamond',[r](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267633)) creates a diamond-shaped structuring element, where r specifies the distance from the structuring element origin to the points of the diamond.

[example](https://localhost:31515/static/help/images/ref/strel.html#bu7pnvx-1)

SE = strel('disk',[r](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser" \l "d117e267633),[n](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267695)) creates a disk-shaped structuring element, where r specifies the radius and n specifies the number of line structuring elements used to approximate the disk shape. Morphological operations using disk approximations run much faster when the structuring element uses approximations.

SE = strel('octagon',[r](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267633)) creates a octagonal structuring element, where r specifies the distance from the structuring element origin to the sides of the octagon, as measured along the horizontal and vertical axes. r must be a nonnegative multiple of 3.

[example](https://localhost:31515/static/help/images/ref/strel.html#bu7pnvk-1)

SE = strel('line',[len](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser" \l "mw_1190c8ef-ae7e-493d-9801-6416968858f7),[deg](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#mw_e12253e1-0651-4fe4-89f6-41762e25bbf5)) creates a linear structuring element that is symmetric with respect to the neighborhood center, with approximate length len and angle deg.

SE = strel('rectangle',[[m n]](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267879)) creates a rectangular structuring element of size [m n].

[example](https://localhost:31515/static/help/images/ref/strel.html#bu6giwk)

SE = strel('square',[w](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267912)) creates a square structuring element whose width is w pixels.

SE = strel('cube',[w](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267912)) creates a 3-D cubic structuring element whose width is w pixels.

SE = strel('cuboid',[[m n p]](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267941)) creates a 3-D cuboidal structuring element of size [m n p].

[example](https://localhost:31515/static/help/images/ref/strel.html#bu7pnv2-1)

SE = strel('sphere',[r](https://localhost:31515/static/help/images/ref/strel.html?snc=HODF9R&container=jshelpbrowser#d117e267633)) creates a 3-D spherical structuring element whose radius is r pixels.

#### Compatibility

The following syntaxes still work, but [offsetstrel](https://localhost:31515/static/help/images/ref/offsetstrel.html) is the preferred way to create these nonflat structuring element shapes:

* SE = strel('arbitrary',nhood,h)
* SE = strel('ball',r,h,n)

The following syntaxes still work, but are not recommended for use:

* SE = strel('pair',offset)
* SE = strel('periodicline',p,v)

MORPHOLOGICAL OPENING

## Use Morphological Opening to Extract Large Image Features

Try This Example

You can use morphological opening to remove small objects from an image while preserving the shape and size of larger objects in the image.

In this example, you use morphological opening on an image of a circuitboard to remove all the circuit lines from the image. The output image contains only the rectangular shapes of the microchips.

### Open an Image In One Step

You can use the [imopen](https://in.mathworks.com/help/images/ref/imopen.html) function to perform erosion and dilation in one step.

Read the image into the workspace, and display it.

BW1 = imread('circbw.tif');

figure

imshow(BW1)



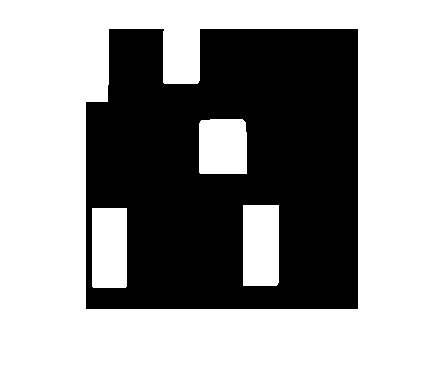
Create a structuring element. The structuring element should be large enough to remove the lines when you erode the image, but not large enough to remove the rectangles. It should consist of all 1s, so it removes everything but large contiguous patches of foreground pixels.

SE = strel('rectangle',[40 30]);

Open the image.

BW2 = imopen(BW1, SE);

imshow(BW2);



## Use Morphological Opening to Extract Large Image Features

Try This Example

You can use morphological opening to remove small objects from an image while preserving the shape and size of larger objects in the image.

In this example, you use morphological opening on an image of a circuitboard to remove all the circuit lines from the image. The output image contains only the rectangular shapes of the microchips.

### Open an Image In One Step

You can use the [imopen](https://in.mathworks.com/help/images/ref/imopen.html) function to perform erosion and dilation in one step.

Read the image into the workspace, and display it.

BW1 = imread('circbw.tif');

figure

imshow(BW1)



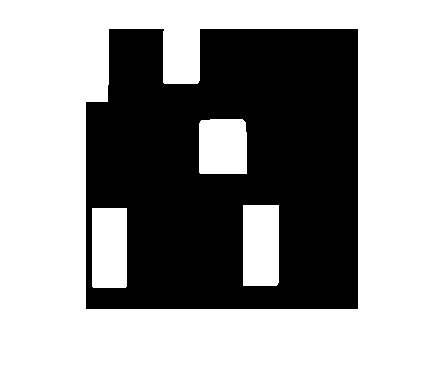
Create a structuring element. The structuring element should be large enough to remove the lines when you erode the image, but not large enough to remove the rectangles. It should consist of all 1s, so it removes everything but large contiguous patches of foreground pixels.

SE = strel('rectangle',[40 30]);

Open the image.

BW2 = imopen(BW1, SE);

imshow(BW2);



### Open an Image By Performing Erosion Then Dilation

You can also perform erosion and dilation sequentially.

Erode the image with the structuring element. This removes all the lines, but also shrinks the rectangles.

BW3 = imerode(BW1,SE);

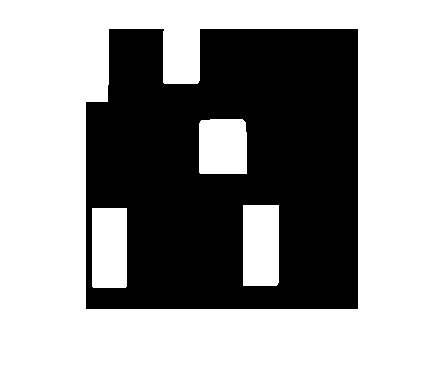
imshow(BW3)



To restore the rectangles to their original sizes, dilate the eroded image using the same structuring element, SE.

BW4 = imdilate(BW3,SE);

imshow(BW4)

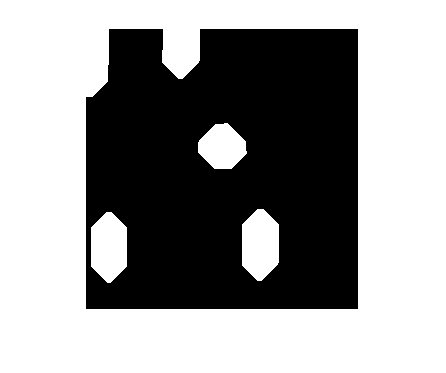


y performing the operations sequentially, you have the flexibility to change the structuring element. Create a different structuring element, and dilate the eroded image using the new structuring element.

SE = strel('diamond',15);

BW5 = imdilate(BW3,SE);

imshow(BW5)



# **imfill**

Fill image regions and holes

[collapse all in page](javascript:void(0);)

## Syntax

[BW2 = imfill(BW,locations)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146471)

[BW2 = imfill(BW,locations,conn)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146506)

[BW2 = imfill(BW,'holes')](https://in.mathworks.com/help/images/ref/imfill.html#d117e146531)

[BW2 = imfill(BW,conn,'holes')](https://in.mathworks.com/help/images/ref/imfill.html#d117e146552)

[I2 = imfill(I)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146581)

[I2 = imfill(I,conn)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146602)

[BW2 = imfill(BW)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146631)

[BW2 = imfill(BW,0,conn)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146665)

[[BW2, locations\_out] = imfill(BW)](https://in.mathworks.com/help/images/ref/imfill.html#d117e146684)

## Description

[example](https://in.mathworks.com/help/images/ref/imfill.html#buo3hpj-1)

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html" \l "buo3g4l-1-BW),[locations](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-locations)) performs a flood-fill operation on background pixels of the input binary image BW, starting from the points specified in locations.

You optionally can perform the flood-fill operation using a GPU (requires Parallel Computing Toolbox™).

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html" \l "buo3g4l-1-BW),[locations](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-locations),[conn](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-conn)) fills the area defined by locations, where conn specifies the connectivity.

[example](https://in.mathworks.com/help/images/ref/imfill.html#buo3hpj-2)

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html" \l "buo3g4l-1-BW),'holes') fills holes in the input binary image BW. In this syntax, a hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

[example](https://in.mathworks.com/help/images/ref/imfill.html#buo3hpj-2)

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html" \l "buo3g4l-1-BW),[conn](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-conn),'holes') fills holes in the binary image BW, where conn specifies the connectivity.

[example](https://in.mathworks.com/help/images/ref/imfill.html#buo3hpy-1)

[I2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-I2) = imfill([I](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-I)) fills holes in the grayscale image I. In this syntax, a hole is defined as an area of dark pixels surrounded by lighter pixels.

[example](https://in.mathworks.com/help/images/ref/imfill.html#buo3hpy-1)

[I2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-I2) = imfill([I](https://in.mathworks.com/help/images/ref/imfill.html" \l "buo3g4l-1-I),[conn](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-conn)) fills holes in the grayscale image I, where conn specifies the connectivity.

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW)) displays the binary image BW on the screen and lets you define the region to fill by selecting points interactively with the mouse. To use this syntax, BW must be a 2-D image.

Press **Backspace** or **Delete** to remove the previously selected point. Shift-click, right-click, or double-click to select a final point and start the fill operation. Press **Return** to finish the selection without adding a point.

This syntax is not supported on a GPU.

[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2) = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW),0,[conn](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-conn)) lets you override the default connectivity as you interactively specify locations.

This syntax is not supported on a GPU.

[[BW2](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW2), [locations\_out](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-locations_out)] = imfill([BW](https://in.mathworks.com/help/images/ref/imfill.html#buo3g4l-1-BW)) returns the locations of points selected interactively in locations\_out. To use this syntax, BW must be a 2-D image.

This syntax is not supported on a GPU.

DEBLURRING

# **Deblurring**

Deconvolution for deblurring

Images can be distorted by blur, such as motion blur or blur resulting from an out-of-focus lens. Blur is represented by a distortion operator, also called the point spread function (PSF). Different deblurring algorithms estimate and remove blur based on how much knowledge you have of the PSF and noise in the image.

## Functions

|  |  |
| --- | --- |
| [deconvblind](https://in.mathworks.com/help/images/ref/deconvblind.html) | Deblur image using blind deconvolution |
| [deconvlucy](https://in.mathworks.com/help/images/ref/deconvlucy.html) | Deblur image using Lucy-Richardson method |
| [deconvreg](https://in.mathworks.com/help/images/ref/deconvreg.html) | Deblur image using regularized filter |
| [deconvwnr](https://in.mathworks.com/help/images/ref/deconvwnr.html) | Deblur image using Wiener filter |
| [edgetaper](https://in.mathworks.com/help/images/ref/edgetaper.html) | Taper discontinuities along image edges |
| [otf2psf](https://in.mathworks.com/help/images/ref/otf2psf.html) | Convert optical transfer function to point-spread function |
| [psf2otf](https://in.mathworks.com/help/images/ref/psf2otf.html) | Convert point-spread function to optical transfer function |
| [padarray](https://in.mathworks.com/help/images/ref/padarray.html) | Pad array |

# **ROI-Based Processing**

Define and operate on regions of interest (ROI)

A region of interest (ROI) is a portion of an image that you want to filter or operate on in some way. The toolbox supports a set of ROI objects that you can use to create ROIs of many shapes, such circles, ellipses, polygons, rectangles, and hand-drawn shapes. After creation, you can use ROI object properties to customize their appearance and functioning. In addition, the ROI objects support object functions and events that you can use to implement interactive behavior. For example, using events, your application can execute custom code whenever the ROI changes position. As a convenience, the toolbox includes a parallel set of convenience functions for ROI creation. For example, to create a rectangular ROI, you can use images.roi.Rectangle or the corresponding convenience function drawrectangle. For more information about ROIs, see [ROI Creation Overview](https://in.mathworks.com/help/images/roi-creation-overview.html).

A common use of an ROI is to create a binary mask image. In the mask image, pixels that belong to the ROI are set to 1 and pixels outside the ROI are set to 0. To create a mask, use the createMask object functions supported by most of the ROI objects. You can also create a mask without requiring an image using the poly2mask function.

Starting in R2018b, a new set of ROI objects replaced the previous set of ROI objects. The new objects provide better performance and more functional capabilities, such as face color transparency. With the new objects, you can also receive notification of interactions with the object, such as clicks or movement, using events. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on migrating to the new ROIs, see [ROI Migration](https://in.mathworks.com/help/images/migrating-to-the-new-roi-objects.html).

## Functions

[collapse all](javascript:void(0);)

### ROI Objects

|  |  |
| --- | --- |
| [AssistedFreehand](https://in.mathworks.com/help/images/ref/images.roi.assistedfreehand.html) | Assisted freehand region of interest |
| [Circle](https://in.mathworks.com/help/images/ref/images.roi.circle.html) | Circular region of interest |
| [Crosshair](https://in.mathworks.com/help/images/ref/images.roi.crosshair.html) | Crosshair region of interest |
| [Cuboid](https://in.mathworks.com/help/images/ref/images.roi.cuboid.html) | Cuboidal region of interest |
| [Ellipse](https://in.mathworks.com/help/images/ref/images.roi.ellipse.html) | Elliptical region of interest |
| [Freehand](https://in.mathworks.com/help/images/ref/images.roi.freehand.html) | Freehand region of interest |
| [Line](https://in.mathworks.com/help/images/ref/images.roi.line.html) | Line region of interest |
| [Point](https://in.mathworks.com/help/images/ref/images.roi.point.html) | Point region of interest |
| [Polygon](https://in.mathworks.com/help/images/ref/images.roi.polygon.html) | Polygonal region of interest |
| [Polyline](https://in.mathworks.com/help/images/ref/images.roi.polyline.html) | Polyline region of interest |
| [Rectangle](https://in.mathworks.com/help/images/ref/images.roi.rectangle.html) | Rectangular region of interest |
| [draw](https://in.mathworks.com/help/images/ref/draw.html) | Begin drawing ROI interactively |

### ROI Creation Convenience Functions

|  |  |
| --- | --- |
| [drawassisted](https://in.mathworks.com/help/images/ref/drawassisted.html) | Create freehand ROI on image with assistance from image edges |
| [drawcircle](https://in.mathworks.com/help/images/ref/drawcircle.html) | Create customizable circular ROI |
| [drawcrosshair](https://in.mathworks.com/help/images/ref/drawcrosshair.html) | Create customizable crosshair ROI |
| [drawcuboid](https://in.mathworks.com/help/images/ref/drawcuboid.html) | Create customizable cuboidal ROI |
| [drawellipse](https://in.mathworks.com/help/images/ref/drawellipse.html) | Create customizable elliptical ROI |
| [drawfreehand](https://in.mathworks.com/help/images/ref/drawfreehand.html) | Create customizable freehand ROI |
| [drawline](https://in.mathworks.com/help/images/ref/drawline.html) | Create customizable linear ROI |
| [drawpoint](https://in.mathworks.com/help/images/ref/drawpoint.html) | Create customizable point ROI |
| [drawpolygon](https://in.mathworks.com/help/images/ref/drawpolygon.html) | Create customizable polygonal ROI |
| [drawpolyline](https://in.mathworks.com/help/images/ref/drawpolyline.html) | Create customizable polyline ROI |
| [drawrectangle](https://in.mathworks.com/help/images/ref/drawrectangle.html) | Create customizable rectangular ROI |

### ROI Object Customization

|  |  |
| --- | --- |
| [reduce](https://in.mathworks.com/help/images/ref/reduce.html) | Reduce density of points in ROI |
| [beginDrawingFromPoint](https://in.mathworks.com/help/images/ref/begindrawingfrompoint.html) | Begin drawing ROI from specified point |
| [inROI](https://in.mathworks.com/help/images/ref/inroi.html) | Query if points are located in ROI |
| [bringToFront](https://in.mathworks.com/help/images/ref/bringtofront.html) | Bring ROI to front of Axes stacking order |
| [wait](https://in.mathworks.com/help/images/ref/wait.html) | Block MATLAB command line until ROI operation is finished |

### Mask Creation

|  |  |
| --- | --- |
| [createMask](https://in.mathworks.com/help/images/ref/createmask.html) | Create binary mask image from ROI |
| [roipoly](https://in.mathworks.com/help/images/ref/roipoly.html) | Specify polygonal region of interest (ROI) |
| [poly2mask](https://in.mathworks.com/help/images/ref/poly2mask.html) | Convert region of interest (ROI) polygon to region mask |

### ROI Filtering

|  |  |
| --- | --- |
| [regionfill](https://in.mathworks.com/help/images/ref/regionfill.html) | Fill in specified regions in image using inward interpolation |
| [inpaintCoherent](https://in.mathworks.com/help/images/ref/inpaintcoherent.html) | Restore specific image regions using coherence transport based image inpainting |
| [inpaintExemplar](https://in.mathworks.com/help/images/ref/inpaintexemplar.html) | Restore specific image regions using exemplar-based image inpainting |
| [roicolor](https://in.mathworks.com/help/images/ref/roicolor.html) | Select region of interest (ROI) based on color |
| [roifilt2](https://in.mathworks.com/help/images/ref/roifilt2.html) | Filter region of interest (ROI) in image |
| [reducepoly](https://in.mathworks.com/help/images/ref/reducepoly.html) | Reduce density of points in ROI |

he basic workflow when using ROI convenience functions is use the function to create the object. The convenience function automatically calls the draw object method.

1. Read an image into the MATLAB workspace and display it in a figure.
2. I = imread('pears.png');

imshow(I)

1. Create the ROI. The example creates an elliptical ROI using the drawellipse convenience function. The example specifies theStripeColor parameter to customize the look of the edge.

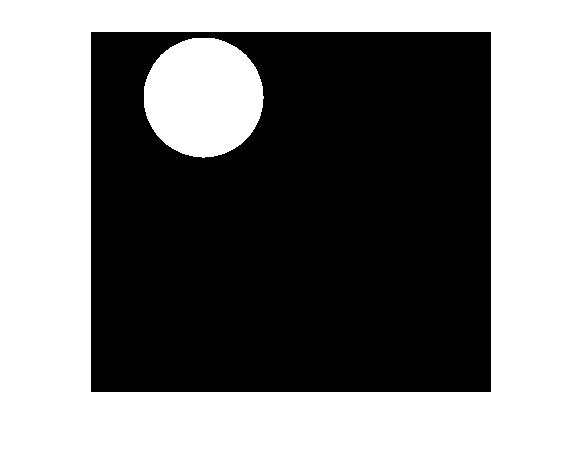
roi = drawEllipse('StripeColor','m');

I=imread('b&wlily.png');

>> C=drawcircle('Center',[113,66],'Radius',60);

>> BW=createMask(C);

>> imshow(BW)

>>

### Create a Binary Mask from a Grayscale Image

You can create a mask from a grayscale image by classifying each pixel as belonging to either the region of interest or the background. For example, suppose you want to filter the grayscale image I, filtering only those pixels whose values are greater than 0.5. You can create the appropriate mask with this command:

BW = (I > 0.5)

### Create Binary Mask Based on Color Values

You can use the [roicolor](https://in.mathworks.com/help/images/ref/roicolor.html) function to define an ROI based on color or intensity range.

### Create Binary Mask Without an Associated Image

You can use the [poly2mask](https://in.mathworks.com/help/images/ref/poly2mask.html) function to create a binary mask without having an associated image. Unlike the createMask method, poly2mask does not require an input image. You specify the vertices of the ROI in two vectors and specify the size of the binary mask returned. For example, the following creates a binary mask that can be used to filter an ROI in the pout.tif image.

c = [123 123 170 170];

r = [160 210 210 160];

m = 291; % height of pout image

n = 240; % width of pout image

BW = poly2mask(c,r,m,n);

imshow(BW)

## Overview of ROI Filtering

Filtering a region of interest (ROI) is the process of applying a filter to a region in an image, where a binary mask defines the region. For example, you can apply an intensity adjustment filter to certain regions of an image.

To filter an ROI in an image, use the [roifilt2](https://in.mathworks.com/help/images/ref/roifilt2.html) function. When you call roifilt2, you specify:

* Input grayscale image to be filtered
* Binary mask image that defines the ROI
* Filter (either a 2-D filter or function)

roifilt2 filters the input image and returns an image that consists of filtered values for pixels where the binary mask contains 1s and unfiltered values for pixels where the binary mask contains 0s. This type of operation is called masked filtering.

roifilt2 is best suited for operations that return data in the same range as in the original image, because the output image takes some of its data directly from the input image. Certain filtering operations can result in values outside the normal image data range (i.e., [0,1] for images of class double, [0,255] for images of class uint8, and [0,65535] for images of class uint16).

## Sharpen Region of Interest in an Image

Try This Example

Read a grayscale image into the workspace.

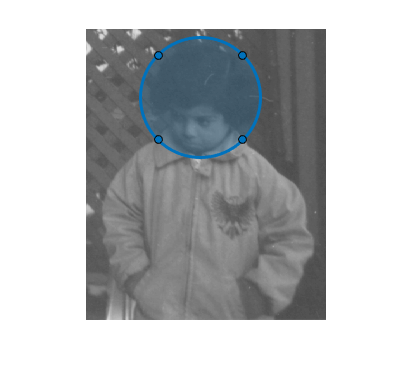
I = imread('pout.tif');

imshow(I)



Draw a region of interest over the image to specify the area you want to filter. Use the drawcircle function to create the region of interest, specifying the center of the circle and the radius of the circle. Alternatively, if you want to draw the circle interactively, then do not specify the center or radius of the circle.

hax = drawcircle(gca,'Center',[115 69],'Radius', 60);



Create the mask using the createMask function and specifying the ROI.

mask = createMask(hax);

Define the function you want to use as a filter. This function, named f, passes the input image x to the imsharpen function and specifies the strength of the sharpening effect by using the 'Amount' name-value pair argument.

f = @(x)imsharpen(x,'Amount',3)

f = *function\_handle with value:*

@(x)imsharpen(x,'Amount',3)

Filter the ROI using the roifilt2 function and specifying the image, mask, and filtering function.

J = roifilt2(I,mask,f);

Display the result.

imshow(J)

## Neighborhood or Block Processing: An Overview

Certain image processing operations involve processing an image in sections, called blocks or neighborhoods, rather than processing the entire image at once. Several functions in the toolbox, such as linear filtering and morphological functions, use this approach.

The toolbox includes several functions that you can use to implement image processing algorithms as a block or neighborhood operation. These functions break the input image into blocks or neighborhoods, call the specified function to process each block or neighborhood, and then reassemble the results into an output image. The following table summarizes these functions.

| **Function** | **Description** |
| --- | --- |
| [nlfilter](https://in.mathworks.com/help/images/ref/nlfilter.html) | Implements sliding neighborhood operations that you can use to process an input image in a pixel-wise fashion. For each pixel in the input image, the function performs the operation you specify on a block of neighboring pixels to determine the value of the corresponding pixel in the output image. For more information, see [Sliding Neighborhood Operations](https://in.mathworks.com/help/images/sliding-neighborhood-operations.html) |
| [blockproc](https://in.mathworks.com/help/images/ref/blockproc.html) | Implements distinct block operations that you can use to process an input image a block at a time. The function divides the image into rectangular blocks, and performs the operation you specify on each individual block to determine the values of the pixels in the corresponding block of the output image. For more information, see [Distinct Block Processing](https://in.mathworks.com/help/images/distinct-block-processing.html) |
| [colfilt](https://in.mathworks.com/help/images/ref/colfilt.html) | Implements column-wise processing operations which provide a way of speeding up neighborhood or block operations by rearranging blocks into matrix columns. For more information, see [Use Column-wise Processing to Speed Up Sliding Neighborhood or Distinct Block Operations](https://in.mathworks.com/help/images/use-columnwise-processing-to-speed-up-sliding-neighborhood-or-distinct-block-operations.html). |

# **Image Arithmetic**

Add, subtract, multiply, and divide images

Image arithmetic is the implementation of standard arithmetic operations, such as addition, subtraction, multiplication, and division, on images. Image arithmetic has many uses in image processing both as a preliminary step in more complex operations and by itself. For example, image subtraction can be used to detect differences between two or more images of the same scene or object.

## Functions

|  |  |
| --- | --- |
| [imabsdiff](https://in.mathworks.com/help/images/ref/imabsdiff.html) | Absolute difference of two images |
| [imadd](https://in.mathworks.com/help/images/ref/imadd.html) | Add two images or add constant to image |
| [imapplymatrix](https://in.mathworks.com/help/images/ref/imapplymatrix.html) | Linear combination of color channels |
| [imcomplement](https://in.mathworks.com/help/images/ref/imcomplement.html) | Complement image |
| [imdivide](https://in.mathworks.com/help/images/ref/imdivide.html) | Divide one image into another or divide image by constant |
| [imlincomb](https://in.mathworks.com/help/images/ref/imlincomb.html) | Linear combination of images |
| [immultiply](https://in.mathworks.com/help/images/ref/immultiply.html) | Multiply two images or multiply image by constant |
| [imsubtract](https://in.mathworks.com/help/images/ref/imsubtract.html) | Subtract one image from another or subtract constant from image |

# **Image Segmentation**

Segment images

Image segmentation is the process of partitioning an image into parts or regions. This division into parts is often based on the characteristics of the pixels in the image. For example, one way to find regions in an image is to look for abrupt discontinuities in pixel values, which typically indicate edges. These edges can define regions. Other methods divide the image into regions based on color values or texture.

## Apps

|  |  |
| --- | --- |
| [Color Thresholder](https://in.mathworks.com/help/images/ref/colorthresholder-app.html) | Threshold a color image |
| [Image Segmenter](https://in.mathworks.com/help/images/ref/imagesegmenter-app.html) | Segment an image by refining regions |

## Functions

|  |  |
| --- | --- |
| [activecontour](https://in.mathworks.com/help/images/ref/activecontour.html) | Segment image into foreground and background using active contours (snakes) |
| [imsegfmm](https://in.mathworks.com/help/images/ref/imsegfmm.html) | Binary image segmentation using Fast Marching Method |
| [imseggeodesic](https://in.mathworks.com/help/images/ref/imseggeodesic.html) | Segment image into two or three regions using geodesic distance-based color segmentation |
| [imsegkmeans](https://in.mathworks.com/help/images/ref/imsegkmeans.html) | K-means clustering based image segmentation |
| [imsegkmeans3](https://in.mathworks.com/help/images/ref/imsegkmeans3.html) | K-means clustering based volume segmentation |
| [watershed](https://in.mathworks.com/help/images/ref/watershed.html) | Watershed transform |
| [gradientweight](https://in.mathworks.com/help/images/ref/gradientweight.html) | Calculate weights for image pixels based on image gradient |
| [graydiffweight](https://in.mathworks.com/help/images/ref/graydiffweight.html) | Calculate weights for image pixels based on grayscale intensity difference |
| [grayconnected](https://in.mathworks.com/help/images/ref/grayconnected.html) | Select contiguous image region with similar gray values |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [graythresh](https://in.mathworks.com/help/images/ref/graythresh.html) | | Global image threshold using Otsu's method | | |
| [multithresh](https://in.mathworks.com/help/images/ref/multithresh.html) | | Multilevel image thresholds using Otsu’s method | | |
| [otsuthresh](https://in.mathworks.com/help/images/ref/otsuthresh.html) | | Global histogram threshold using Otsu's method | | |
| [adaptthresh](https://in.mathworks.com/help/images/ref/adaptthresh.html) | | Adaptive image threshold using local first-order statistics | | |
| [boundarymask](https://in.mathworks.com/help/images/ref/boundarymask.html) | | Find region boundaries of segmentation | | |
| [superpixels](https://in.mathworks.com/help/images/ref/superpixels.html) | | 2-D superpixel oversegmentation of images | | |
| [lazysnapping](https://in.mathworks.com/help/images/ref/lazysnapping.html) | | Segment image into foreground and background using graph-based segmentation | | |
| [grabcut](https://in.mathworks.com/help/images/ref/grabcut.html) | | Segment image into foreground and background using iterative graph-based segmentation | | |
| [superpixels3](https://in.mathworks.com/help/images/ref/superpixels3.html) | | 3-D superpixel oversegmentation of 3-D image | | |
|  | | |
| [labeloverlay](https://in.mathworks.com/help/images/ref/labeloverlay.html) | | | Overlay label matrix regions on 2-D image |
| [label2idx](https://in.mathworks.com/help/images/ref/label2idx.html) | | | Convert label matrix to cell array of linear indices |
| [jaccard](https://in.mathworks.com/help/images/ref/jaccard.html) | Jaccard similarity coefficient for image segmentation | | |
| [dice](https://in.mathworks.com/help/images/ref/dice.html) | Sørensen-Dice similarity coefficient for image segmentation | | |
| [bfscore](https://in.mathworks.com/help/images/ref/bfscore.html) | Contour matching score for image segmentation | | |

## Texture Analysis

The toolbox includes several texture analysis functions that filter an image using standard statistical measures. These statistics can characterize the texture of an image because they provide information about the local variability of the intensity values of pixels in an image. For example, in areas with smooth texture, the range of values in the neighborhood around a pixel is a small value; in areas of rough texture, the range is larger. Similarly, calculating the standard deviation of pixels in a neighborhood can indicate the degree of variability of pixel values in that region. The table lists these functions.

| **Function** | **Description** |
| --- | --- |
| [rangefilt](https://in.mathworks.com/help/images/ref/rangefilt.html) | Calculates the local range of an image. |
| [stdfilt](https://in.mathworks.com/help/images/ref/stdfilt.html) | Calculates the local standard deviation of an image. |
| [entropyfilt](https://in.mathworks.com/help/images/ref/entropyfilt.html) | Calculates the local entropy of a grayscale image. Entropy is a statistical measure of randomness. |

The functions all operate in a similar way: they define a neighborhood around the pixel of interest, calculate the statistic for that neighborhood, and use that value as the value of the pixel of interest in the output image.

This example shows how the rangefilt function operates on a simple array.

A = [ 1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20 ]

A =

1 2 3 4 5

6 7 8 9 10

11 12 13 14 15

16 17 18 19 20

B = rangefilt(A)

B =

6 7 7 7 6

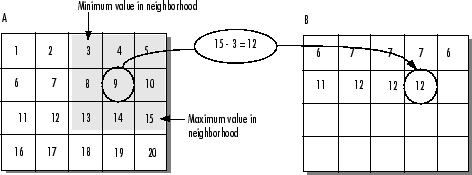
11 12 12 12 11

11 12 12 12 11

6 7 7 7 6

The following figure shows how the value of element B(2,4) was calculated from A(2,4). By default, the rangefilt function uses a 3-by-3 neighborhood but you can specify neighborhoods of different shapes and sizes.

**Determining Pixel Values in Range Filtered Output Image**



The stdfilt and entropyfilt functions operate similarly, defining a neighborhood around the pixel of interest and calculating the statistic for the neighborhood to determine the pixel value in the output image. The stdfilt function calculates the standard deviation of all the values in the neighborhood.

The entropyfilt function calculates the entropy of the neighborhood and assigns that value to the output pixel. By default, the entropyfilt function defines a 9-by-9 neighborhood around the pixel of interest. To calculate the entropy of an entire image, use the entropy function.

## Texture Analysis Using the Gray-Level Co-Occurrence Matrix (GLCM)

A statistical method of examining texture that considers the spatial relationship of pixels is the gray-level co-occurrence matrix (GLCM), also known as the gray-level spatial dependence matrix. The GLCM functions characterize the texture of an image by calculating how often pairs of pixel with specific values and in a specified spatial relationship occur in an image, creating a GLCM, and then extracting statistical measures from this matrix. (The texture filter functions, described in [Texture Analysis](https://in.mathworks.com/help/images/texture-analysis.html) cannot provide information about shape, that is, the spatial relationships of pixels in an image.)

After you create the GLCMs, using [graycomatrix](https://in.mathworks.com/help/images/ref/graycomatrix.html), you can derive several statistics from them using [graycoprops](https://in.mathworks.com/help/images/ref/graycoprops.html). These statistics provide information about the texture of an image. The following table lists the statistics.

| **Statistic** | **Description** |
| --- | --- |
| Contrast | Measures the local variations in the gray-level co-occurrence matrix. |
| Correlation | Measures the joint probability occurrence of the specified pixel pairs. |
| Energy | Provides the sum of squared elements in the GLCM. Also known as uniformity or the angular second moment. |
| Homogeneity | Measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal. |

## Create a Gray-Level Co-Occurrence Matrix

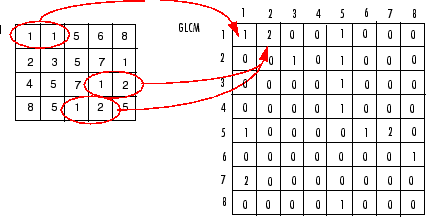
To create a GLCM, use the [graycomatrix](https://in.mathworks.com/help/images/ref/graycomatrix.html) function. The function creates a gray-level co-occurrence matrix (GLCM) by calculating how often a pixel with the intensity (gray-level) value i occurs in a specific spatial relationship to a pixel with the value j. By default, the spatial relationship is defined as the pixel of interest and the pixel to its immediate right (horizontally adjacent), but you can specify other spatial relationships between the two pixels. Each element (i,j) in the resultant glcm is simply the sum of the number of times that the pixel with value i occurred in the specified spatial relationship to a pixel with value j in the input image.

The number of gray levels in the image determines the size of the GLCM. By default, graycomatrix uses scaling to reduce the number of intensity values in an image to eight, but you can use the NumLevels and the GrayLimits parameters to control this scaling of gray levels. See the graycomatrix reference page for more information.

GStexture image. For example, if most of the entries in the GLCM are concentrated along the diagonal, the texture is coarse with respect to the specified offset. You can also derive several statistical measures from the GLCM. See [Derive Statistics from GLCM and Plot Correlation](https://in.mathworks.com/help/images/derive-statistics-from-glcm-and-plot-correlation.html) for more information.

To illustrate, the following figure shows how graycomatrix calculates the first three values in a GLCM. In the output GLCM, element (1,1) contains the value 1 because there is only one instance in the input image where two horizontally adjacent pixels have the values 1 and 1, respectively. glcm(1,2) contains the value 2 because there are two instances where two horizontally adjacent pixels have the values 1 and 2. Element (1,3) in the GLCM has the value 0 because there are no instances of two horizontally adjacent pixels with the values 1 and 3. graycomatrix continues processing the input image, scanning the image for other pixel pairs (i,j) and recording the sums in the corresponding elements of the GLCM.

**Process Used to Create the GLCM**



GABOR FILTERS

### Object Detection Using Features

|  |  |
| --- | --- |
| [acfObjectDetector](https://in.mathworks.com/help/vision/ref/acfobjectdetector.html) | Detect objects using aggregate channel features |
| [peopleDetectorACF](https://in.mathworks.com/help/vision/ref/peopledetectoracf.html) | Detect people using aggregate channel features |
| [vision.CascadeObjectDetector](https://in.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html) | Detect objects using the Viola-Jones algorithm |
| [vision.ForegroundDetector](https://in.mathworks.com/help/vision/ref/vision.foregrounddetector-system-object.html) | Foreground detection using Gaussian mixture models |
| [vision.PeopleDetector](https://in.mathworks.com/help/vision/ref/vision.peopledetector-system-object.html) | Detect upright people using HOG features |
| [vision.BlobAnalysis](https://in.mathworks.com/help/vision/ref/vision.blobanalysis-system-object.html) | Properties of connected regions |
| [trainACFObjectDetector](https://in.mathworks.com/help/vision/ref/trainacfobjectdetector.html) | Train ACF object detector |
| [trainCascadeObjectDetector](https://in.mathworks.com/help/vision/ref/traincascadeobjectdetector.html) | Train cascade object detector model |
| [trainImageCategoryClassifier](https://in.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html) | Train an image category classifier |
| [detectBRISKFeatures](https://in.mathworks.com/help/vision/ref/detectbriskfeatures.html) | Detect BRISK features and return BRISKPoints object |
| [detectFASTFeatures](https://in.mathworks.com/help/vision/ref/detectfastfeatures.html) | Detect corners using FAST algorithm and return cornerPoints object |
| [detectHarrisFeatures](https://in.mathworks.com/help/vision/ref/detectharrisfeatures.html) | Detect corners using Harris–Stephens algorithm and return cornerPoints object |
| [detectKAZEFeatures](https://in.mathworks.com/help/vision/ref/detectkazefeatures.html) | Detect KAZE features |
| [detectMinEigenFeatures](https://in.mathworks.com/help/vision/ref/detectmineigenfeatures.html) | Detect corners using minimum eigenvalue algorithm and return cornerPoints object |
| [detectMSERFeatures](https://in.mathworks.com/help/vision/ref/detectmserfeatures.html) | Detect MSER features and return MSERRegions object |
| [detectSURFFeatures](https://in.mathworks.com/help/vision/ref/detectsurffeatures.html) | Detect SURF features and return SURFPoints object |
| [extractFeatures](https://in.mathworks.com/help/vision/ref/extractfeatures.html) | Extract interest point descriptors |
| [matchFeatures](https://in.mathworks.com/help/vision/ref/matchfeatures.html) | Find matching features |
| [evaluateDetectionMissRate](https://in.mathworks.com/help/vision/ref/evaluatedetectionmissrate.html) | Evaluate miss rate metric for object detection |
| [evaluateDetectionPrecision](https://in.mathworks.com/help/vision/ref/evaluatedetectionprecision.html) | Evaluate precision metric for object detection |
| [bbox2points](https://in.mathworks.com/help/vision/ref/bbox2points.html) | Convert rectangle to corner points list |
| [bboxOverlapRatio](https://in.mathworks.com/help/vision/ref/bboxoverlapratio.html) | Compute bounding box overlap ratio |
| [bboxPrecisionRecall](https://in.mathworks.com/help/vision/ref/bboxprecisionrecall.html) | Compute bounding box precision and recall against ground truth |
| [selectStrongestBbox](https://in.mathworks.com/help/vision/ref/selectstrongestbbox.html) | Select strongest bounding boxes from overlapping clusters |
| [selectStrongestBboxMulticlass](https://in.mathworks.com/help/vision/ref/selectstrongestbboxmulticlass.html) | Select strongest multiclass bounding boxes from overlapping clusters |

# Feature Detection and Extraction

Image registration, interest point detection, extracting feature descriptors, and point feature matching

Local features and their descriptors are the building blocks of many computer vision algorithms. Their applications include image registration, object detection and classification, tracking, and motion estimation. These algorithms use local features to better handle scale changes, rotation, and occlusion. Computer Vision Toolbox™ algorithms include the FAST, Harris, and Shi & Tomasi corner detectors, and the SURF, KAZE, and MSER blob detectors. The toolbox includes the SURF, FREAK, BRISK, LBP, ORB, and HOG descriptors. You can mix and match the detectors and the descriptors depending on the requirements of your application. You can also extract features using a pretrained convolutional neural network which applies techniques from the field of deep learning.

## Functions

[collapse all](javascript:void(0);)

### Detect Features

|  |  |
| --- | --- |
| [detectBRISKFeatures](https://localhost:31515/static/help/vision/ref/detectbriskfeatures.html) | Detect BRISK features and return BRISKPoints object |
| [detectFASTFeatures](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html) | Detect corners using FAST algorithm and return cornerPoints object |
| [detectHarrisFeatures](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html) | Detect corners using Harris–Stephens algorithm and return cornerPoints object |
| [detectMinEigenFeatures](https://localhost:31515/static/help/vision/ref/detectmineigenfeatures.html) | Detect corners using minimum eigenvalue algorithm and return cornerPoints object |
| [detectMSERFeatures](https://localhost:31515/static/help/vision/ref/detectmserfeatures.html) | Detect MSER features and return MSERRegions object |
| [detectORBFeatures](https://localhost:31515/static/help/vision/ref/detectorbfeatures.html) | Detect and store ORB keypoints |
| [detectSURFFeatures](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html) | Detect SURF features and return SURFPoints object |
| [detectKAZEFeatures](https://localhost:31515/static/help/vision/ref/detectkazefeatures.html) | Detect KAZE features |

### Extract Features

|  |  |
| --- | --- |
| [extractFeatures](https://localhost:31515/static/help/vision/ref/extractfeatures.html) | Extract interest point descriptors |
| [extractLBPFeatures](https://localhost:31515/static/help/vision/ref/extractlbpfeatures.html) | Extract local binary pattern (LBP) features |
| [extractHOGFeatures](https://localhost:31515/static/help/vision/ref/extracthogfeatures.html) | Extract histogram of oriented gradients (HOG) features |

### Match Features

|  |  |
| --- | --- |
| [matchFeatures](https://localhost:31515/static/help/vision/ref/matchfeatures.html) | Find matching features |
| [showMatchedFeatures](https://localhost:31515/static/help/vision/ref/showmatchedfeatures.html) | Display corresponding feature points |

### Image Registration

|  |  |
| --- | --- |
| [imwarp](https://localhost:31515/static/help/images/ref/imwarp.html) | Apply geometric transformation to image |
| [estimateGeometricTransform](https://localhost:31515/static/help/vision/ref/estimategeometrictransform.html) | Estimate geometric transform from matching point pairs |
| [vision.BlockMatcher](https://localhost:31515/static/help/vision/ref/vision.blockmatcher-system-object.html) | Estimate motion between images or video frames |
| [vision.LocalMaximaFinder](https://localhost:31515/static/help/vision/ref/vision.localmaximafinder-system-object.html) | Find local maxima in matrices |
| [vision.TemplateMatcher](https://localhost:31515/static/help/vision/ref/vision.templatematcher-system-object.html) | Locate template in image |

### Visualization and Display

|  |  |
| --- | --- |
| [insertMarker](https://localhost:31515/static/help/vision/ref/insertmarker.html) | Insert markers in image or video |
| [insertShape](https://localhost:31515/static/help/vision/ref/insertshape.html) | Insert shapes in image or video |
| [insertObjectAnnotation](https://localhost:31515/static/help/vision/ref/insertobjectannotation.html) | Annotate truecolor or grayscale image or video stream |
| [insertText](https://localhost:31515/static/help/vision/ref/inserttext.html) | Insert text in image or video |
| [imshow](https://localhost:31515/static/help/images/ref/imshow.html) | Display image |
| [imshowpair](https://localhost:31515/static/help/images/ref/imshowpair.html) | Compare differences between images |

### Store Features

|  |  |
| --- | --- |
| [binaryFeatures](https://localhost:31515/static/help/vision/ref/binaryfeatures.html) | Object for storing binary feature vectors |
| [BRISKPoints](https://localhost:31515/static/help/vision/ref/briskpoints.html) | Object for storing BRISK interest points |
| [KAZEPoints](https://localhost:31515/static/help/vision/ref/kazepoints.html) | Object for storing KAZE interest points |
| [cornerPoints](https://localhost:31515/static/help/vision/ref/cornerpoints.html) | Object for storing corner points |
| [SURFPoints](https://localhost:31515/static/help/vision/ref/surfpoints.html) | Object for storing SURF interest points |
| [MSERRegions](https://localhost:31515/static/help/vision/ref/mserregions.html) | Object for storing MSER regions |
| [ORBPoints](https://localhost:31515/static/help/vision/ref/orbpoints.html) | Object for storing ORB keypoints |

### Affine and Projective Transformations

|  |  |
| --- | --- |
| [affine2d](https://localhost:31515/static/help/images/ref/affine2d.html) | 2-D affine geometric transformation |
| [affine3d](https://localhost:31515/static/help/images/ref/affine3d.html) | 3-D affine geometric transformation |
| [projective2d](https://localhost:31515/static/help/images/ref/projective2d.html) | 2-D projective geometric transformation |

Object Detection using features

* 1. SURF:

# detectSURFFeatures

Detect SURF features and return SURFPoints object

[collapse all in page](javascript:void(0);)

## Syntax

[points = detectSURFFeatures(I)](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#d117e145315)

[points = detectSURFFeatures(I,Name,Value)](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#d117e145344)

## Description

[example](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#btack4t)

[points](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-points) = detectSURFFeatures([I](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-I)) returns a [SURFPoints](https://localhost:31515/static/help/vision/ref/surfpoints.html) object, points, containing information about SURF features detected in the 2-D grayscale input image I. The detectSURFFeaturesfunction implements the Speeded-Up Robust Features (SURF) algorithm to find blob features.

[points](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-points) = detectSURFFeatures([I](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html" \l "bs39_qf-1-I),[Name,Value](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#namevaluepairarguments)) specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

## Examples

[collapse all](javascript:void(0);)

### Detect SURF Interest Points in a Grayscale Image

[Open Live Script](matlab:openExample('vision/DetectSURFInterestPointsInAGrayscaleImageExample'))

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));

# detectSURFFeatures

Detect SURF features and return SURFPoints object

[collapse all in page](javascript:void(0);)

## Syntax

[points = detectSURFFeatures(I)](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#d117e145315)

[points = detectSURFFeatures(I,Name,Value)](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#d117e145344)

## Description

[example](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#btack4t)

[points](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-points) = detectSURFFeatures([I](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-I)) returns a [SURFPoints](https://localhost:31515/static/help/vision/ref/surfpoints.html) object, points, containing information about SURF features detected in the 2-D grayscale input image I. The detectSURFFeaturesfunction implements the Speeded-Up Robust Features (SURF) algorithm to find blob features.

[points](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#bs39_qf-1-points) = detectSURFFeatures([I](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html" \l "bs39_qf-1-I),[Name,Value](https://localhost:31515/static/help/vision/ref/detectsurffeatures.html#namevaluepairarguments)) specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

## Examples

[collapse all](javascript:void(0);)

### Detect SURF Interest Points in a Grayscale Image

[Open Live Script](matlab:openExample('vision/DetectSURFInterestPointsInAGrayscaleImageExample'))

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));



# detectHarrisFeatures

Detect corners using Harris–Stephens algorithm and return cornerPoints object

[collapse all in page](javascript:void(0);)

## Syntax

[points = detectHarrisFeatures(I)](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#d117e142860)

[points = detectHarrisFeatures(I,Name,Value)](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#d117e142889)

## Description

[example](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#btolyhy-7)

[points](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#btolyhy-1-points) = detectHarrisFeatures([I](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#btolyhy-1-I)) returns a [cornerPoints](https://localhost:31515/static/help/vision/ref/cornerpoints.html) object, points. The object contains information about the feature points detected in a 2-D input image, I. The detectHarrisFeatures function uses the Harris–Stephens algorithm to find these feature points.

[points](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#btolyhy-1-points) = detectHarrisFeatures([I](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html" \l "btolyhy-1-I),[Name,Value](https://localhost:31515/static/help/vision/ref/detectharrisfeatures.html#namevaluepairarguments)) uses additional options specified by one or more Name,Value pair arguments.

## Examples

[collapse all](javascript:void(0);)

### Find Corner Points Using the Harris-Stephens Algorithm

[Open Live Script](matlab:openExample('vision/FindCornerPointsUsingTheHarrisStephensAlgorithmExample'))

**Read the image.**

I = checkerboard;

**Find the corners.**

corners = detectHarrisFeatures(I);

**Display the results.**

imshow(I); hold on;

plot(corners.selectStrongest(50));

# detectFASTFeatures

Detect corners using FAST algorithm and return cornerPoints object

[collapse all in page](javascript:void(0);)

## Syntax

[points = detectFASTFeatures(I)](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#d117e142333)

[points = detectFASTFeatures(I,Name,Value)](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#d117e142362)

## Description

[example](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#btoe5z2-2)

[points](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#btoe1zy-points) = detectFASTFeatures([I](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#btoe1zy-I)) returns a [cornerPoints](https://localhost:31515/static/help/vision/ref/cornerpoints.html) object, points. The object contains information about the feature points detected in a 2-D grayscale input image, I. The detectFASTFeatures function uses the Features from Accelerated Segment Test (FAST) algorithm to find feature points.

[points](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#btoe1zy-points) = detectFASTFeatures([I](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html" \l "btoe1zy-I),[Name,Value](https://localhost:31515/static/help/vision/ref/detectfastfeatures.html#namevaluepairarguments)) uses additional options specified by one or more Name,Value pair arguments.

## Examples

[collapse all](javascript:void(0);)

### Find Corner Points in an Image Using the FAST Algorithm

[Open Live Script](matlab:openExample('vision/FindCornerPointsInAnImageUsingTheFASTAlgorithmExample'))

**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

[collapse all](javascript:void(0);)

### I — Input image M-by-N 2-D grayscale image

Input image, specified in 2-D grayscale. The input image must be real and nonsparse.

**Data Types:**single | double | int16 | uint8 | uint16 | logical

### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**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.

Syntax:

Points/regions=detect\_\_\_\_\_Features(I);

figure;imshow(I);hold on;plot(\_.selectStrongest(\_));

The table describes how the function implements the descriptor extraction methods.

| **Method** | **Feature Vector (Descriptor)** |
| --- | --- |
| BRISK | Binary Robust Invariant Scalable Keypoints (BRISK).  The function sets the Orientation property of the [validPoints](https://localhost:31515/static/help/vision/ref/extractfeatures.html" \l "bsxmas0-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](https://localhost:31515/static/help/vision/ref/extractfeatures.html#bsxmas0-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. |
| Block | Simple square neighbhorhood.   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](https://localhost:31515/static/help/vision/ref/cornerpoints.html) input object. | | The SURF method for a [SURFPoints](https://localhost:31515/static/help/vision/ref/surfpoints.html) or [MSERRegions](https://localhost:31515/static/help/vision/ref/mserregions.html) input object. | | The BRISK method for a [BRISKPoints](https://localhost:31515/static/help/vision/ref/briskpoints.html) input object. | | The ORB method for a [ORBPoints](https://localhost:31515/static/help/vision/ref/orbpoints.html) input object. |   For an *M*-by-2 input matrix of [*x* *y*] coordinates, the function implements the Block method. |

FIGURE EXTRACTION

[features,valid\_points]=extractFeatures(I,points);

>> [features1,valid\_points1]=extractFeatures(I1,points1);

>> indexpairs=matchFeatures(features,features1);

>> matchedPoints = valid\_points(indexPairs(:,1),:);

figure;imshow(I1);hold on;plot(points1.selectStrongest(70));

>> figure;imshow(I);hold on;plot(points.selectStrongest(70));

>> [features,valid\_points]=extractFeatures(I,points);

>> [features1,valid\_points1]=extractFeatures(I1,points1);

>> indexpairs=matchFeatures(features,features1);

>> matchedPoints = valid\_points(indexpairs(:,1),:);

>> matchedPoints1 = valid\_points1(indexpairs(:,2),:);

>> figure;showMatchedFeatures(I,I1,matchedPoints,matchedPoints1);

>> figure; ax = axes;

>> showMatchedFeatures(I1,I,matchedPoints,matchedPoints,'montage','Parent',ax);

>> [tform, inlierBoxPoints, inlierScenePoints] = ...

estimateGeometricTransform(matchedPoints, matchedPoints1, 'affine');