Parameter list:

1. myLine(img, pt1(first), pt2(second), color, thickness, lineType, shift)
2. myCircle(img, center, radius, color, thickness, lineType, shift)
3. myEllipse(image, center, axes, angle, startAngle, endAngle, color, thickness, lineType, shift)
4. myRect(img, pt1(start), pt2(end), color, thickness, lineType, shift)
5. myPolyLine(img, pts(array of polygon curves), isClosed, color, thickness, lineType, shift)
6. myputText(img, text, org(bottom\_left\_corner), fontface, fontScale, color, thickness, lineType, **bottomLeftOrigin**(if true, otherwise top-left corner))
7. cv2.**cornerHarris**(src, blockSize, ksize, k[, dst[, borderType]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/feature_detection.html" \l "cv2.cornerHarris" \o "Permalink to this definition)

|  |  |
| --- | --- |
|  | * **src** – Input single-channel 8-bit or floating-point image. * **dst** – Image to store the Harris detector responses. It has the type CV\_32FC1 and the same size as src . * **blockSize** – Neighborhood size (see the details on **[cornerEigenValsAndVecs()](http://docs.opencv.org/modules/imgproc/doc/feature_detection.html" \l "void cornerEigenValsAndVecs(InputArray src, OutputArray dst, int blockSize, int ksize, int borderType)" \o "void cornerEigenValsAndVecs(InputArray src, OutputArray dst, int blockSize, int ksize, int borderType))** ). * **ksize** – Aperture parameter for the **[Sobel()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize, double scale, double delta, int borderType)" \o "void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize, double scale, double delta, int borderType))** operator. * **k** – Harris detector free parameter. See the formula below. * **borderType** – Pixel extrapolation method. See **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))** . |

1. cv2.**erode**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

|  |  |
| --- | --- |
|  | * **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV\_8U, CV\_16U, CV\_16S,CV\_32F` or ``CV\_64F. * **dst** – output image of the same size and type as src. * **element** – structuring element used for erosion; if element=Mat() , a 3 x 3 rectangular structuring element is used. * **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center. * **iterations** – number of times erosion is applied. * **borderType** – pixel extrapolation method (see **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details). * **borderValue** – border value in case of a constant border (see **[createMorphologyFilter()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))** for details). * 9. cv2.**dilate**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "cv2.dilate" \o "Permalink to this definition) * **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV\_8U, CV\_16U, CV\_16S,CV\_32F` or ``CV\_64F. * **dst** – output image of the same size and type as src. * **element** – structuring element used for dilation; if element=Mat() , a 3 x 3 rectangular structuring element is used. * **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center. * **iterations** – number of times dilation is applied. * **borderType** – pixel extrapolation method (see **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details). * **borderValue** – border value in case of a constant border (see **[createMorphologyFilter()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))** for details). * **10**. cv2.**morphologyEx**(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst * **src** – Source image. The number of channels can be arbitrary. The depth should be one of CV\_8U, CV\_16U, CV\_16S,CV\_32F` or ``CV\_64F. * **dst** – Destination image of the same size and type as src . * **element** – Structuring element. * **op** –   Type of a morphological operation that can be one of the following:   * + **MORPH\_OPEN** - an opening operation   + **MORPH\_CLOSE** - a closing operation   + **MORPH\_GRADIENT** - a morphological gradient   + **MORPH\_TOPHAT** - “top hat”   + **MORPH\_BLACKHAT** - “black hat” * **iterations** – Number of times erosion and dilation are applied. * **borderType** – Pixel extrapolation method. See **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details. * **borderValue** – Border value in case of a constant border. The default value has a special meaning. See[**createMorphologyFilter()**](http://docs.opencv.org/modules/imgproc/doc/filtering.html#Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue)) for details.   11. cv2.**adaptiveThreshold**(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html" \l "cv2.adaptiveThreshold" \o "Permalink to this definition)   * **src** – Source 8-bit single-channel image. * **dst** – Destination image of the same size and the same type as src . * **maxValue** – Non-zero value assigned to the pixels for which the condition is satisfied. See the details below. * **adaptiveMethod** – Adaptive thresholding algorithm to use, ADAPTIVE\_THRESH\_MEAN\_C or ADAPTIVE\_THRESH\_GAUSSIAN\_C . See the details below. * **thresholdType** – Thresholding type that must be either THRESH\_BINARY or THRESH\_BINARY\_INV . * **blockSize** – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on. * **C** – Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.   12. cv2.**threshold**(src, thresh, maxval, type[, dst]) → retval, dst[¶](http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html" \l "cv2.threshold" \o "Permalink to this definition) |
|  | * **src** – input array (single-channel, 8-bit or 32-bit floating point). * **dst** – output array of the same size and type as src. * **thresh** – threshold value. * **maxval** – maximum value to use with the THRESH\_BINARY and THRESH\_BINARY\_INV thresholding types. * **type** – thresholding type (see the details below). |

13. cv2.**filter2D**(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "cv2.filter2D" \o "Permalink to this definition)

* **src** – input image.
* **dst** – output image of the same size and the same number of channels as src.
* **ddepth** –

desired depth of the destination image; if it is negative, it will be the same as src.depth(); the following combinations ofsrc.depth() and ddepth are supported:

* + src.depth() = CV\_8U, ddepth = -1/CV\_16S/CV\_32F/CV\_64F
  + src.depth() = CV\_16U/CV\_16S, ddepth = -1/CV\_32F/CV\_64F
  + src.depth() = CV\_32F, ddepth = -1/CV\_32F/CV\_64F
  + src.depth() = CV\_64F, ddepth = -1/CV\_64F

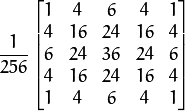
when ddepth=-1, the output image will have the same depth as the source.

* **kernel** – convolution kernel (or rather a correlation kernel), a single-channel floating point matrix; if you want to apply different kernels to different channels, split the image into separate color planes using [**split()**](http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#void split(const Mat& src, Mat* mvbegin)) and process them individually.
* **anchor** – anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor is at the kernel center.
* **delta** – optional value added to the filtered pixels before storing them in dst.
* **borderType** – pixel extrapolation method (see **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details).

14. cv2.**pyrDown**(src[, dst[, dstsize[, borderType]]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "cv2.pyrDown" \o "Permalink to this definition)

|  |
| --- |
| * **src** – input image. * **dst** – output image; it has the specified size and the same type as src. * **dstsize** –   size of the output image; by default, it is computed as Size((src.cols+1)/2, (src.rows+1)/2), but in any case, the following conditions should be satisfied:  \begin{array}{l} | \texttt{dstsize.width} *2-src.cols| \leq  2  \\ | \texttt{dstsize.height} *2-src.rows| \leq  2 \end{array} |

The function performs the downsampling step of the Gaussian pyramid construction. First, it convolves the source image with the kernel:



15. cv2.**pyrUp**(src[, dst[, dstsize[, borderType]]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "cv2.pyrUp" \o "Permalink to this definition)

* **src** – input image.
* **dst** – output image. It has the specified size and the same type as src .
* **dstsize** –

size of the output image; by default, it is computed as Size(src.cols\*2, (src.rows\*2), but in any case, the following conditions should be satisfied:

\begin{array}{l}
| \texttt{dstsize.width} -src.cols*2| \leq  ( \texttt{dstsize.width}   \mod  2)  \\ | \texttt{dstsize.height} -src.rows*2| \leq  ( \texttt{dstsize.height}   \mod  2) \end{array}

16. cv2.**getRotationMatrix2D**(center, angle, scale) → retval[¶](http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html" \l "cv2.getRotationMatrix2D" \o "Permalink to this definition)

|  |  |
| --- | --- |
| **Parameters:** | * **center** – Center of the rotation in the source image. * **angle** – Rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner). * **scale** – Isotropic scale factor. * **map\_matrix** – The output affine transformation, 2x3 floating-point matrix. |

17. cv2.**warpAffine**(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst[¶](http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html" \l "cv2.warpAffine" \o "Permalink to this definition)

* **src** – input image.
* **dst** – output image that has the size dsize and the same type as src .
* **M** – 2\times 3 transformation matrix.
* **dsize** – size of the output image.
* **flags** – combination of interpolation methods (see [**resize()**](http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#void resize(InputArray src, OutputArray dst, Size dsize, double fx, double fy, int interpolation)) ) and the optional flag WARP\_INVERSE\_MAP that means that Mis the inverse transformation ( \texttt{dst}\rightarrow\texttt{src} ).
* **borderMode** – pixel extrapolation method (see **[borderInterpolate()](http://docs.opencv.org/modules/imgproc/doc/filtering.html" \l "int borderInterpolate(int p, int len, int borderType)" \o "int borderInterpolate(int p, int len, int borderType))**); when borderMode=BORDER\_TRANSPARENT , it means that the pixels in the destination image corresponding to the “outliers” in the source image are not modified by the function.
* **borderValue** – value used in case of a constant border; by default, it is 0.