**cv2.cvtColor()**

cv2.cvtColor() method is used to convert an image from one colour space to another.

**Syntax:** cv2.cvtColor(src, code[, dst[, dstCn]])

Return Value: It returns an image.

dst, dstCn are optional.

code is colour conversion code like cv2.COLOR\_BGR2GRAY or cv2.COLOR\_BGR2HSV

**Thresholding**

Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided.

Thresholding is the binarization of an image. In general, we seek to convert a grayscale image to a binary image, where the pixels are either 0 or 255.

In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value

**Syntax**: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)

**The different Simple Thresholding Techniques are:**

* **cv2.THRESH\_BINARY**: If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).
* **cv2.THRESH\_BINARY\_INV:** Inverted or Opposite case of cv2.THRESH\_BINARY.
* **cv.THRESH\_TRUNC:** If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
* **cv.THRESH\_TOZERO:** Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
* **cv.THRESH\_TOZERO\_INV:** Inverted or Opposite case of cv2.THRESH\_TOZERO.
* **Adaptive Thresholding**

the previous section, we used one global value as a threshold. But this might not be good in all cases, e.g. if an image has different lighting conditions in different areas. In that case, adaptive thresholding can help. Here, the algorithm determines the threshold for a pixel based on a small region around it. So we get different thresholds for different regions of the same image which gives better results for images with varying illumination

The adaptiveMethod decides how the threshold value is calculated:

* **cv.ADAPTIVE\_THRESH\_MEAN\_C:** The threshold value is the mean of the neighbourhood area minus the constant C.
* **cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C:** The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant C.
* **Otsu's Binarization**

In global thresholding, we used an arbitrary chosen value as a threshold. In contrast, Otsu's method avoids having to choose a value and determines it automatically.

**cv2.dilate**

To apply a morphological filter to images in Python using OpenCV, use the cv2 dilate() method. The dilate() method takes two inputs in which one is our input image; the second is called the structuring element or kernel, which decides the nature of the operation. Image dilation Increases the object area. See the following syntax of cv2.dilate() method.

**Syntax :**cv2.dilate(src, kernel[, anchor[, iterations[, borderType[, borderValue]]]])

***Parameters***

The dilate() function takes the following parameters.

image: It is a required parameter and an original image on which we need to perform dilation.

kernel: It is the required parameter is the matrix with which the image is convolved.

dst: It is the output image of the same size and type as image src.

anchor: It is a variable of type integer representing the anchor point and its default value Point is (-1, -1) which means that the anchor is at the kernel center.

borderType: It depicts what kind of border to be added. It is defined by flags like cv2.BORDER\_CONSTANT, cv2.BORDER\_REFLECT, etc.

iterations: It is an optional parameter that takes several iterations.

borderValue: It is border value in case of a constant border.

**cv2.findContours**

Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. Contours come handy in shape analysis, finding the size of the object of interest, and object detection.

• To draw all the contours in an image:

**cv.drawContours(img, contours, -1, (0,255,0), 3)**

• To draw an individual contour, say 4th contour:

**cv.drawContours(img, contours, 3, (0,255,0), 3)**

• But most of the time, below method will be useful:

**cnt = contours[4]**

**cv.drawContours(img, [cnt], 0, (0,255,0), 3)**

**Note**

Last two methods are same, but when you go forward, you will see last one is more useful.

**cv.FindContours(image, mode=CV\_RETR\_LIST, method=CV\_CHAIN\_APPROX\_SIMPLE)**

Where

image is the name of the image

*Mode is Contour retrieval mode*

*Method is Contour approximation method*

**Contour Retrieval Mode**

**1. RETR\_LIST**

This is the simplest of the four flags (from explanation point of view). It simply retrieves all the contours, but doesn't create any parent-child relationship. Parents and kids are equal under this rule, and they are just contours. ie they all belongs to same hierarchy level.

**2. RETR\_EXTERNAL**

If you use this flag, it returns only extreme outer flags. All child contours are left behind. We can say, under this law, Only the eldest in every family is taken care of. It doesn't care about other members of the family :).

**3. RETR\_CCOMP**

This flag retrieves all the contours and arranges them to a 2-level hierarchy. ie external contours of the object (ie its boundary) are placed in hierarchy-1. And the contours of holes inside object (if any) is placed in hierarchy-2. If any object inside it, its contour is placed again in hierarchy-1 only. And its hole in hierarchy-2 and so on.

Just consider the image of a "big white zero" on a black background. Outer circle of zero belongs to first hierarchy, and inner circle of zero belongs to second hierarchy.

**4. RETR\_TREE**

And this is the final guy, Mr. Perfect. It retrieves all the contours and creates a full family hierarchy list. It even tells, who is the grandpa, father, son, grandson and even beyond... :).

**Contour Approximation Method**

Above, we told that contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates? That is specified by this contour approximation method.

If you pass **cv.CHAIN\_APPROX\_NONE**, all the boundary points are stored. But actually, do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what **cv.CHAIN\_APPROX\_SIMPLE** does. It removes all redundant points and compresses the contour, thereby saving memory.

**Bounding Rectangle**

There are two types of bounding rectangles.

***Straight Bounding Rectangle***

It is a straight rectangle; it doesn't consider the rotation of the object. So area of the bounding rectangle won't be minimum. It is found by the function cv.boundingRect().

***Rotated Rectangle***

Here, bounding rectangle is drawn with minimum area, so it considers the rotation also.

***cv2.rectangle()***

This method is used to draw a rectangle on any image.

**Syntax: cv2.rectangle(image, start\_point, end\_point, color, thickness)**

Parameters:

image: It is the image on which rectangle is to be drawn.

start\_point: It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

end\_point: It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

color: It is the color of border line of rectangle to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue color.

thickness: It is the thickness of the rectangle border line in px. Thickness of -1 px will fill the rectangle shape by the specified color.

Return Value: It returns an image.