

Report OpenCV

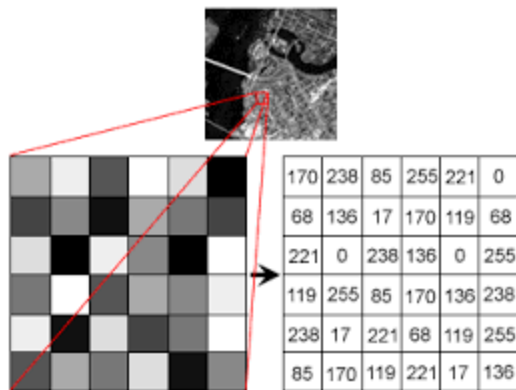
Vibhav Joshi

19 May,2020

Task that were performed

Reading, Writing and Displaying Images

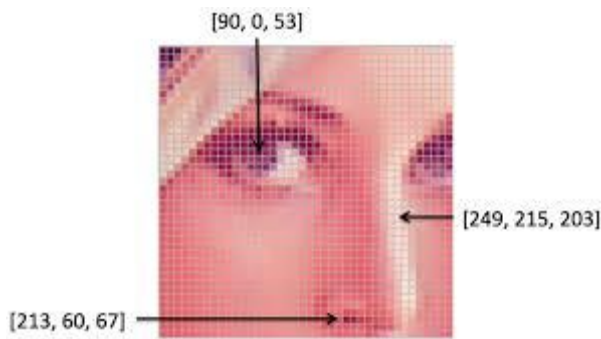
Machines see and process everything using numbers, including images and text. How do you convert images to numbers – I can hear you wondering. Two words – pixel values:



Every number represents the pixel intensity at that particular location. In the above image, I have shown the pixel values for a grayscale image where every pixel contains only one value i.e. the intensity of the black color at that location.

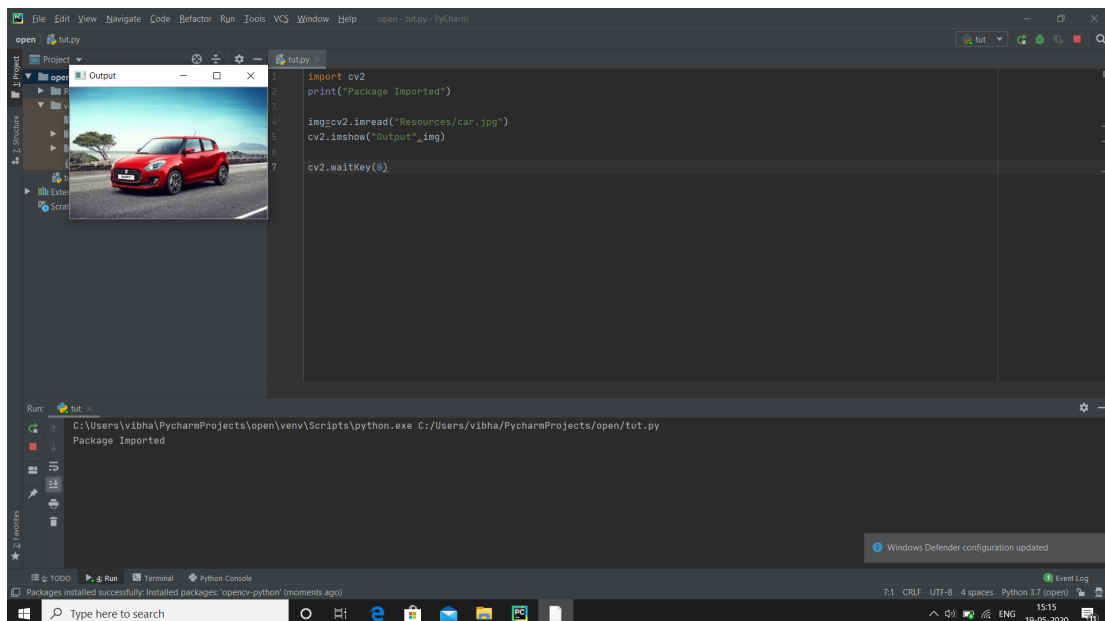
Note that color images will have multiple values for a single pixel. These values represent the intensity of respective channels – Red, Green and Blue channels for RGB images, for instance.

Reading and writing images is essential to any computer vision project. And the OpenCV library makes this function a whole lot easier.



By default, the *imread* function reads images in the BGR (Blue-Green-Red) format. We can read images in different formats using extra flags in the *imread* function:

- **cv2.IMREAD_COLOR:** Default flag for loading a color image
- **cv2.IMREAD_GRAYSCALE:** Loads images in grayscale format
- **cv2.IMREAD_UNCHANGED:** Loads images in their given format, including the alpha channel. Alpha channel stores the transparency information – the higher the value of alpha channel, the more opaque is the pixel

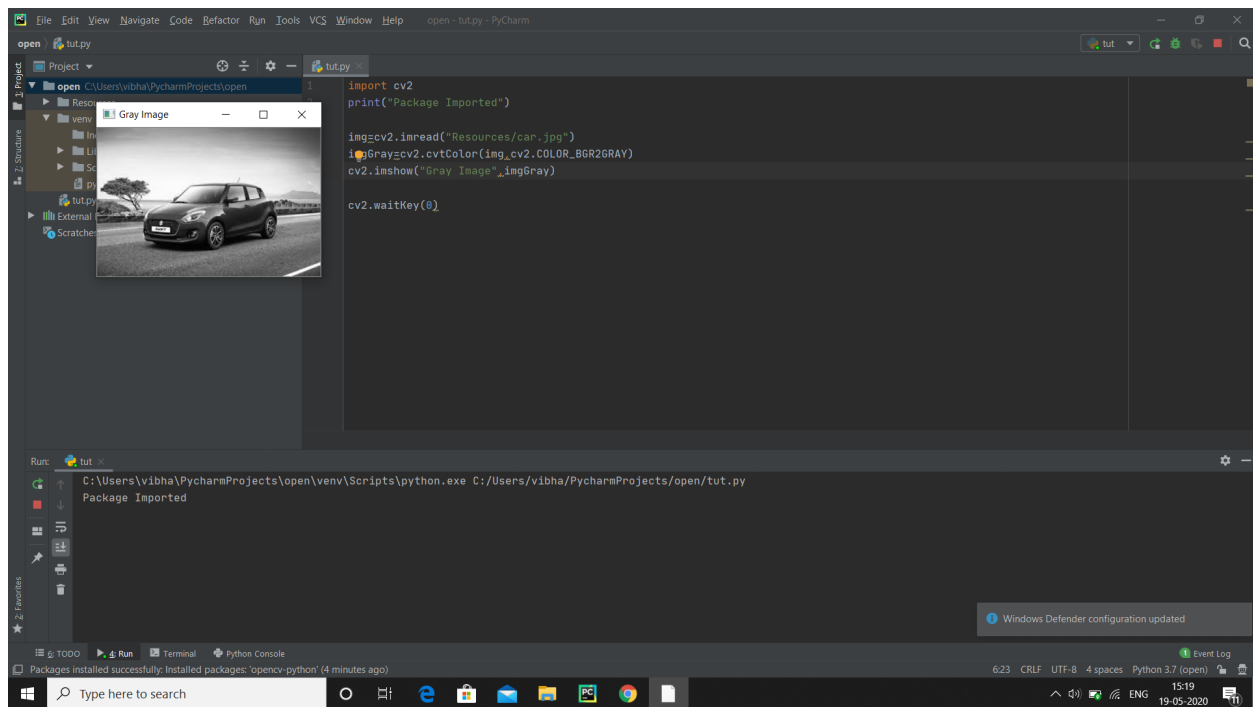


Changing Color Spaces

A color space is a protocol for representing colors in a way that makes them easily reproducible. We know that grayscale images have single pixel values and color images contain 3 values for each pixel – the intensities of the Red, Green and Blue channels.

Most computer vision use cases process images in RGB format. However, applications like video compression and device independent storage – these are heavily dependent on other color spaces, like the Hue-Saturation-Value or HSV color space.

As you understand a RGB image consists of the color intensity of different color channels, i.e. the intensity and color information are mixed in RGB color space but in HSV color space the color and intensity information are separated from each other. This makes HSV color space more robust to lighting changes.



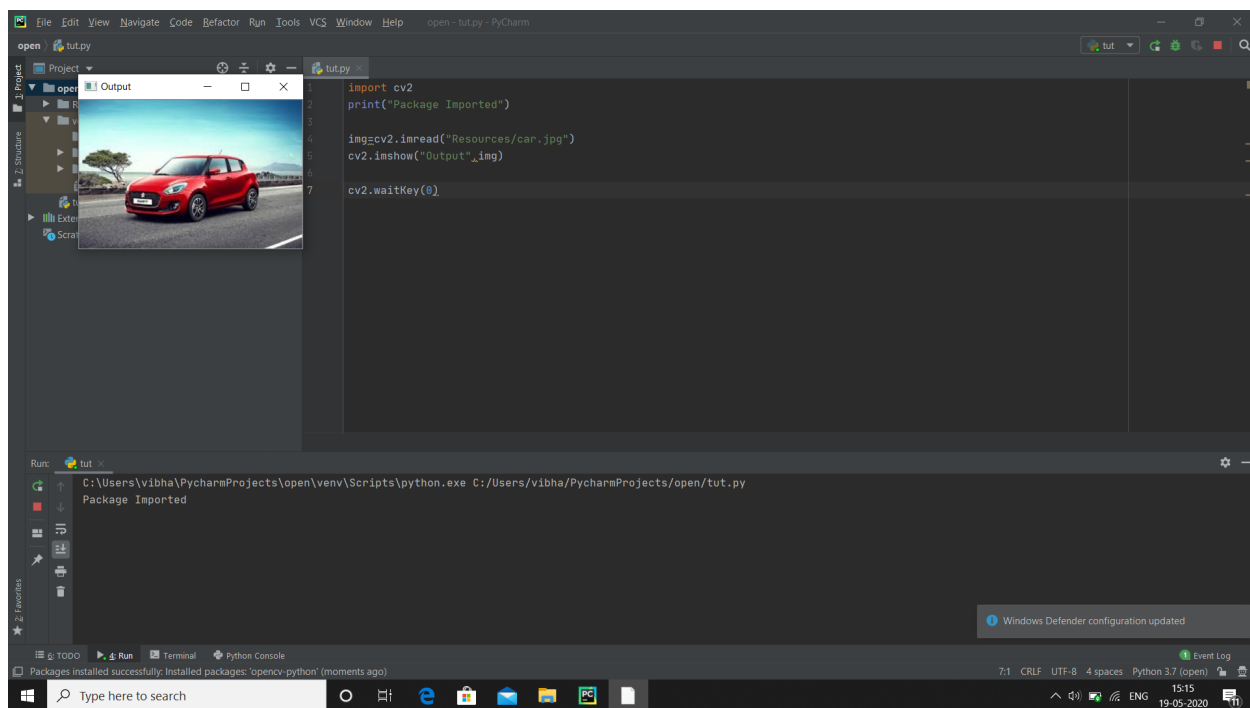
Edge Detection

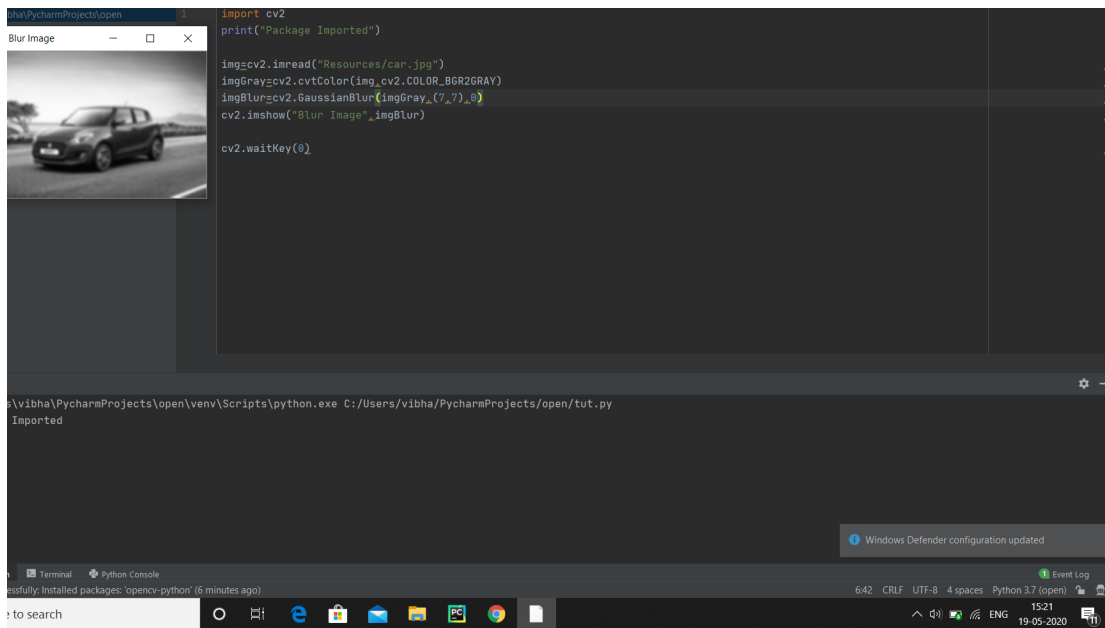
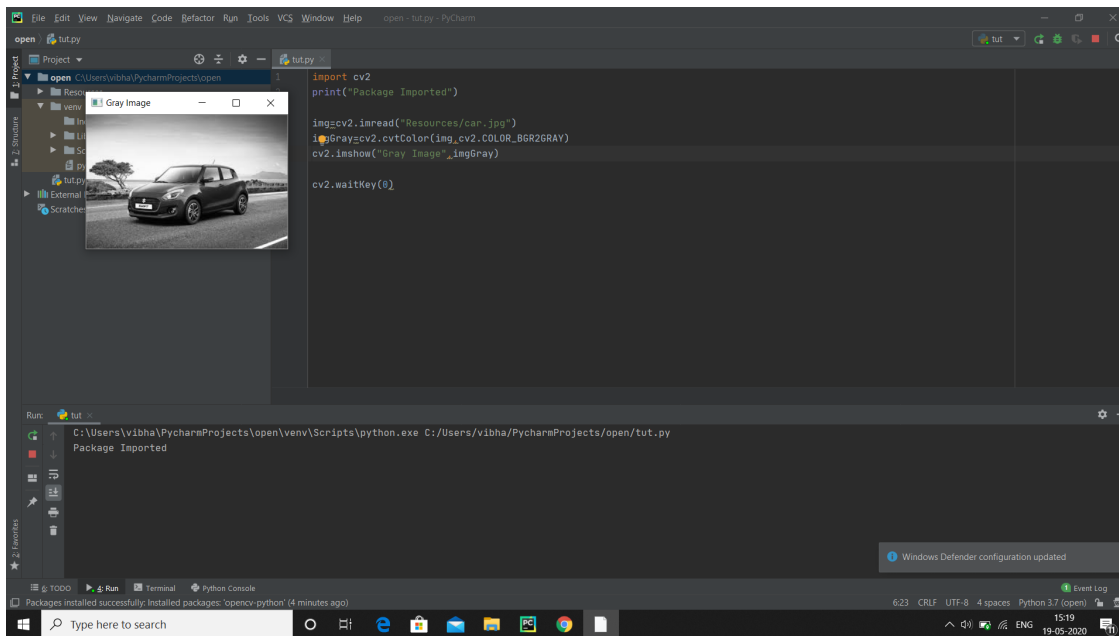
Edges are the points in an image where the image brightness changes sharply or has discontinuities. Such discontinuities generally correspond to:

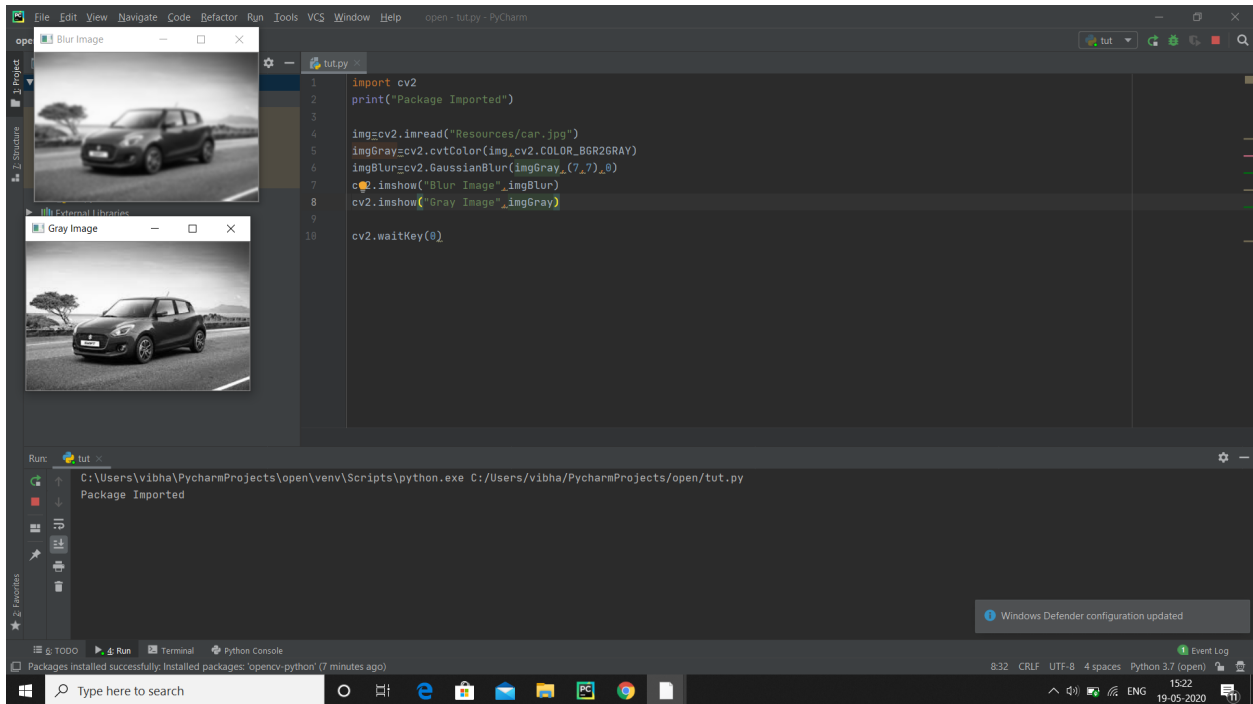
- Discontinuities in depth
- Discontinuities in surface orientation
- Changes in material properties
- Variations in scene illumination

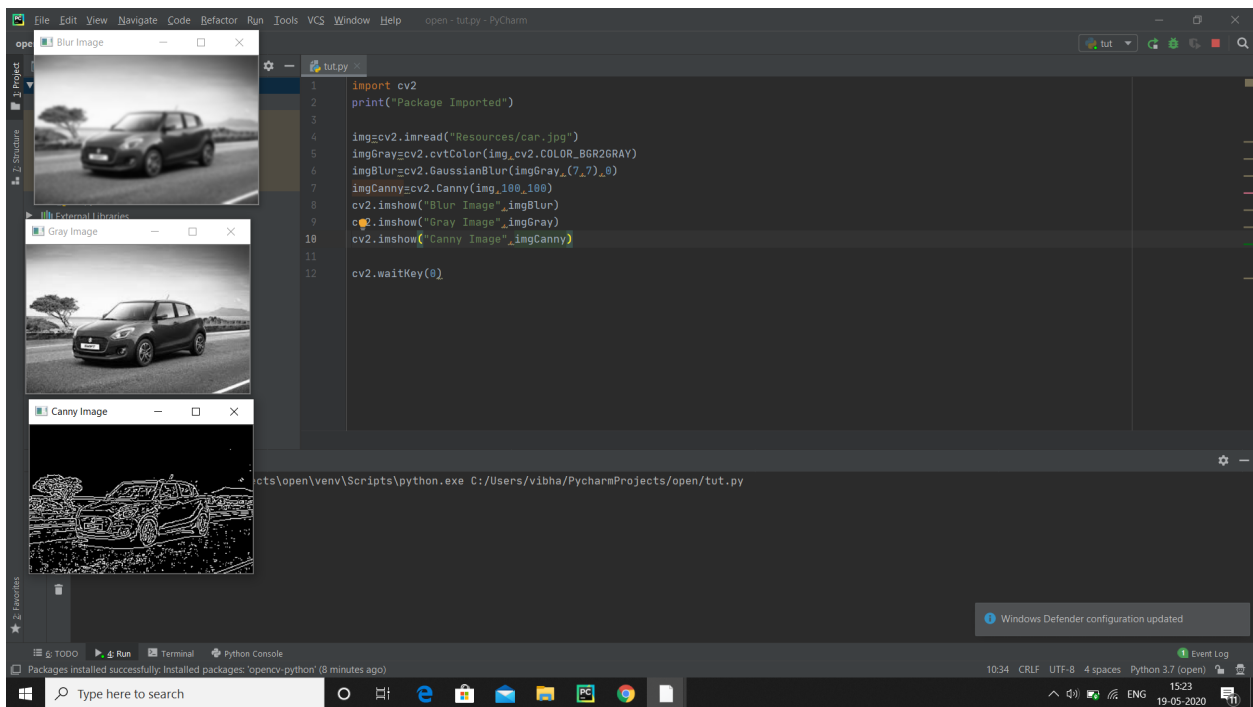
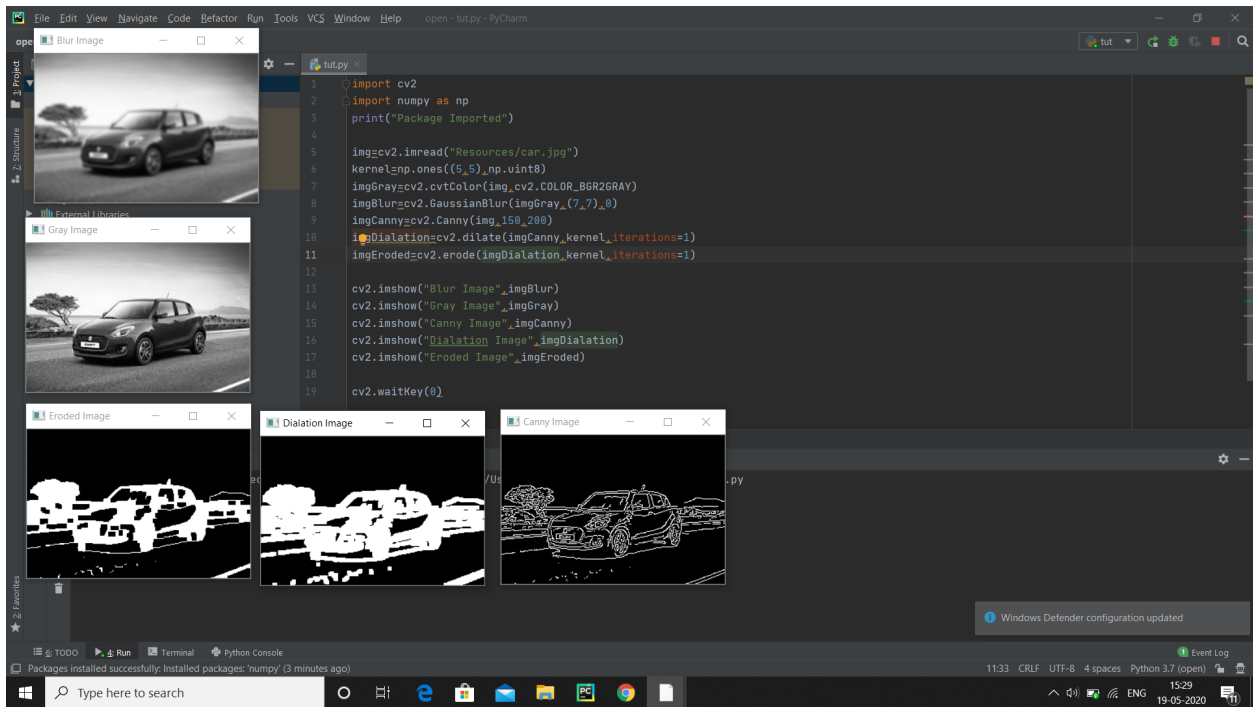
Edges are very useful features of an image that can be used for different applications like classification of objects in the image and localization. Even deep learning models calculate edge features to extract information about the objects present in image.

Edges are different from contours as they are not related to objects rather they signify the changes in pixel values of an image. Edge detection can be used for image segmentation and even for image sharpening.











Project on labelling shapes using contours

```
import cv2

import numpy as np

def getContours(img):

    contours,hierarchy = cv2.findContours(img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)

    for cnt in contours:

        area = cv2.contourArea(cnt)

        print(area)

        if area>500:

            cv2.drawContours(img,cnt, -1, (255, 0, 0), 3)

            peri = cv2.arcLength(cnt,True)

            #print(peri)

            approx = cv2.approxPolyDP(cnt,0.02*peri,True)

            print(len(approx))

            objCor = len(approx)

            x, y, w, h = cv2.boundingRect(approx)

            if objCor ==3: objectType ="Tri"

            elif objCor == 4:

                aspRatio = w/float(h)
```



```
if aspRatio >0.98 and aspRatio <1.03: objectType= "Squ"
else:objectType="Rect"
elif objCor>4: objectType= "Circl"
else:objectType="None"
```

```
cv2.rectangle(imgContour,(x,y),(x+w,y+h),(0,255,0),2)
cv2.putText(imgContour,objectType,
            (x+(w//2)-10,y+(h//2)-10),cv2.FONT_HERSHEY_COMPLEX,0.6,
            (0,0,0),2)
```

```
path = 'Resources/shapes.jpg '
```

```
img = cv2.imread(path)
```


```
imgContour = img.copy()
```

```
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
imgBlur = cv2.GaussianBlur(imgGray,(7,7),1)
```

```
imgCanny = cv2.Canny(imgBlur,50,50)
```

```
getContours(imgCanny)
```



```
imgBlank = np.zeros_like(img)

imgStack = stackImages(0.8,([img,imgGray,imgBlur],
                             [imgCanny,imgContour,imgBlank]))

cv2.imshow("Stack", imgStack)

cv2.waitKey(0)
```

What are contours?

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.

Since OpenCV 3.2, `findContours()` no longer modifies the source image.

In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

See, there are three arguments in `cv.findContours()` function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the contours and hierarchy. Contours is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x,y) coordinates of boundary points of the object.

How to draw the contours?

To draw the contours, `cv.drawContours` function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

Python: `cv.FindContours(image, storage, mode=CV_RETR_LIST, method=CV_CHAIN_APPROX_SIMPLE, offset=(0, 0)) → contours`

Parameters:

`image` – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary. You can use `compare()`, `inRange()`, `threshold()`, `adaptiveThreshold()`, `Canny()`, and others to create a binary image out of a grayscale or color one. The function modifies the image while extracting the contours. If mode equals to `CV_RETR_CCOMP` or `CV_RETR_FLOODFILL`, the input can also be a 32-bit integer image of labels (`CV_32SC1`).

`contours` – Detected contours. Each contour is stored as a vector of points.

`hierarchy` – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each *i*-th contour `contours[i]`, the elements `hierarchy[i][0]`, `hierarchy[i][1]`, `hierarchy[i][2]`, and `hierarchy[i][3]` are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour *i* there are no next, previous, parent, or nested contours, the corresponding elements of `hierarchy[i]` will be negative.

`mode` –

Contour retrieval mode (if you use Python see also a note below).

`CV_RETR_EXTERNAL` retrieves only the extreme outer contours. It sets `hierarchy[i][2]=hierarchy[i][3]=-1` for all the contours.

`CV_RETR_LIST` retrieves all of the contours without establishing any hierarchical relationships.

`CV_RETR_CCOMP` retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are

boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.

CV_RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.c demo.

method –

Contour approximation method (if you use Python see also a note below).

CV_CHAIN_APPROX_NONE stores absolutely all the contour points. That is, any 2 subsequent points (x_1, y_1) and (x_2, y_2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, $\max(\text{abs}(x_1 - x_2), \text{abs}(y_1 - y_2)) = 1$.

CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS applies one of the flavors of the Teh-Chin chain approximation algorithm. See [TehChin89] for details.

C: `CvSeq* cvApproxPoly(const void* src_seq, int header_size, CvMemStorage* storage, int method, double eps, int recursive=0)`

Parameters:

curve –

Input vector of a 2D point stored in:

std::vector or Mat (C++ interface)

Nx2 numpy array (Python interface)

CvSeq or `CvMat` (C interface)

approxCurve – Result of the approximation. The type should match the type of the input curve. In case of C interface the approximated curve is stored in the memory storage and pointer to it is returned.

epsilon – Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.

closed – If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

header_size – Header size of the approximated curve. Normally, sizeof(CvContour) is used.

storage – Memory storage where the approximated curve is stored.

method – Contour approximation algorithm. Only CV_POLY_APPROX_DP is supported.

recursive – Recursion flag. If it is non-zero and curve is CvSeq*, the function cvApproxPoly approximates all the contours accessible from curve by h_next and v_next links.

Python: cv.ArcLength(curve, slice=CV_WHOLE_SEQ, isClosed=-1) → float

Parameters:

curve – Input vector of 2D points, stored in std::vector or Mat.

closed – Flag indicating whether the curve is closed or not.

Python: cv.ContourArea(contour, slice=CV_WHOLE_SEQ) → float

Parameters:

contour – Input vector of 2D points (contour vertices), stored in std::vector or Mat.

oriented – Oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.

Python: cv.BoundingRect(points, update=0) → CvRect

Parameters: points – Input 2D point set, stored in std::vector or Mat.

Python: cv.DrawContours(img, contour, external_color, hole_color, max_level, thickness=1, lineType=8, offset=(0, 0)) → None

Parameters:

image – Destination image.

contours – All the input contours. Each contour is stored as a point vector.

contourIdx – Parameter indicating a contour to draw. If it is negative, all the contours are drawn.

color – Color of the contours.

thickness – Thickness of lines the contours are drawn with. If it is negative (for example, `thickness=CV_FILLED`), the contour interiors are drawn.

lineType – Line connectivity. See `line()` for details.

hierarchy – Optional information about hierarchy. It is only needed if you want to draw only some of the contours (see `maxLevel`).

maxLevel – Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. This parameter is only taken into account when there is hierarchy available.

offset – Optional contour shift parameter. Shift all the drawn contours by the specified `\texttt{offset}=(dx,dy)`.

contour – Pointer to the first contour.

externalColor – Color of external contours.

holeColor – Color of internal contours (holes).

