Contours

# Finding contours via thresholding:

The first step is preprocessing the image as it can significantly improve the results. You can preprocess via blurring the image (e.g GaussianBlur) or by converting a BGR(Blue Green Red) image into a grayscale image.

*# Read the BGR image*

*img = cv.imread(‘Photos/cats.jpg’)*

*# Convert the image into grayscale*

*gray = cv.cvtCOLOR(img, cv.COLOR\_BGR2GRAY)*

*# Apply Thresholding, threshold binarizes the image*

*ret, thresh = cv.threshold(gray, 125, 255, cv.THRESH\_BINARY)*

Let’s discuss the above line in detail before moving ahead:

The cv.threshold() function is used to apply a threshold to an image. It converts the image from grayscale to binary by comparing each pixel value to a specified threshold value.

General syntax:

*ret, thresholded\_image = cv.threshold(src, thresh, maxval, type)*

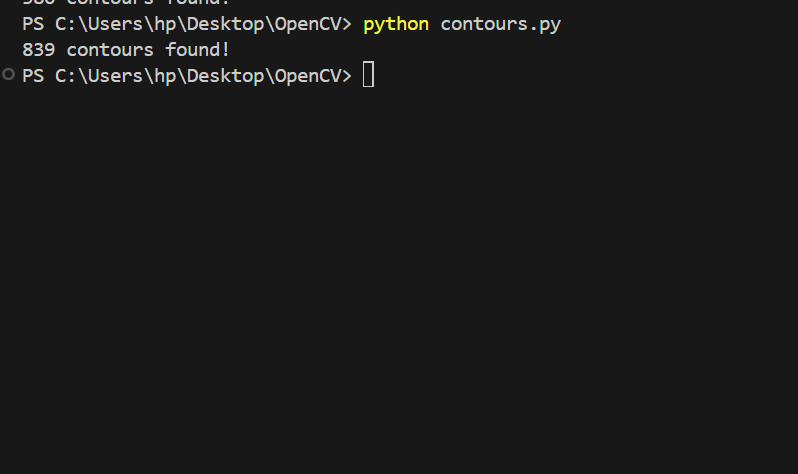
where:

* src: The source image (must be in grayscale).
* thresh: The threshold value to compare each pixel to.
* maxval: The value to assign to pixels that exceed the threshold (for binary thresholding).
* type: The type of thresholding to apply (e.g., binary, binary inverse, etc.).
* ret: This is the threshold value used (in this case, 125).
* thresh: This is the output binary image where:
  + Pixels with intensity greater than 125 are set to 255 (white).
  + Pixels with intensity less than or equal to 125 are set to 0 (black).

# Find no. of contours

*contours, heirarchies = cv.findContours(thresh,cv.RETR\_LIST,cv.CHAIN\_APPROX\_NONE)*

*print(f'{len(contours)} contours found!')*

**

The cv.findContours() function retrieves the contours from a binary image. The contours are stored as a Python list of points, where each contour is a list of points representing its boundary.

General syntax:

*contours, hierarchy = cv.findContours(image, mode, method)*

where:

image: This is the binary image obtained after applying thresholding or canny edge detection.

Contour Retrieval Mode: This parameter defines the contour retrieval mode, which dictates how contours are retrieved from the image.

* cv.RETR\_LIST retrieves all the contours in the image without establishing any hierarchical relationships between them (i.e., no parent-child relationships between contours). It gives you a flat list of all contours.
* Other retrieval modes you could use include:
* cv.RETR\_EXTERNAL: Retrieves only the outermost contours (i.e., contours with no parents). This ignores any nested contours inside objects.
* cv.RETR\_CCOMP: Retrieves all the contours and organizes them into a two-level hierarchy (outer and inner contours).
* cv.RETR\_TREE: Retrieves all contours and reconstructs the full hierarchy of nested contours (contours within contours).

Contour Approximation Method: This defines the contour approximation method, which determines how the contour points are stored.

* cv.CHAIN\_APPROX\_NONE stores all the points of the contour, without any approximation. This means every pixel along the contour is stored, which can result in a large number of points for a simple shape.
* Other approximation methods include: cv.CHAIN\_APPROX\_SIMPLE which compresses horizontal, vertical, and diagonal segments, and leaves only their endpoints. This results in fewer points and reduces memory usage.

contours: This is a list of all the detected contours. Each contour is a NumPy array of shape (n, 1, 2), where n is the number of points in the contour. Each point is represented as a pair of (x, y) coordinates. Each contour in the list represents the boundary of a connected white region in the binary image.

hierarchies: This is an array that contains information about the hierarchy of the contours. It helps describe the parent-child relationships between contours (if any). For cv.RETR\_LIST, the hierarchies array will contain None values or -1 values because this mode doesn’t create any parent-child relationships.

However, in other retrieval modes (like cv.RETR\_TREE), hierarchies would contain information about which contour is the parent or child of another. The hierarchy array has shape (number\_of\_contours, 4), where each contour is described by four values:

* Next: The index of the next contour at the same hierarchy level.
* Previous: The index of the previous contour at the same hierarchy level.
* First Child: The index of the first child contour.
* Parent: The index of the parent contour.

**For clean images: Go with thresholding.**

# Finding contours with canny edge detection:

# Preprocess the image further by blurring, you can also do this when using thresholding

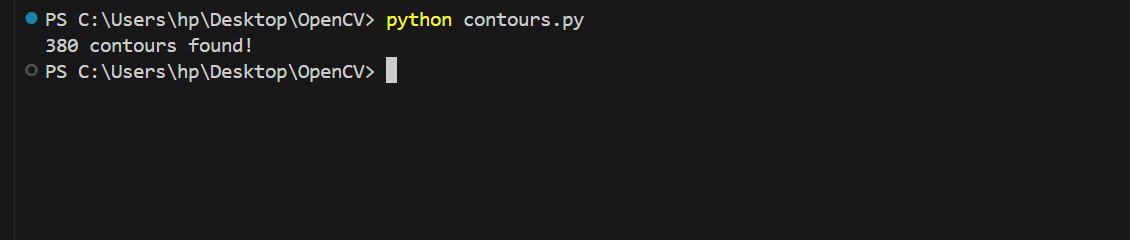
*blur = cv.GaussianBlur(gray,(5,5),cv.BORDER\_DEFAULT)*

*# Apply canny edge detection*

*canny = cv.Canny(blur,125,175)*

*contours, heirarchies = cv.findContours(canny,cv.RETR\_LIST,cv.CHAIN\_APPROX\_NONE)*

*print(f'{len(contours)} contours found!')*

**

**For images with more complex structure or lighting issues: Consider Canny edge detection for better accuracy.**

# Drawing contours:

# Draw blank image

*blank = np.zeros(img.shape,dtype='uint8')*

*# draw contours*

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

The above line is used to **draw contours** on an image (blank) using the contours found previously (e.g., with cv.findContours()), with a specified color and thickness.

General syntax:

*cv.drawContours(image, contours, contourIdx, color, thickness)*

where:

image: image on which contours are drawn

contours: This is the list of contours returned by the cv.findContours() function. Each contour is a list of points representing the boundary of an object in the image.

Contour Index: This parameter specifies which contour to draw.

* If you pass -1, it tells OpenCV to draw all the contours in the contours list.
* If you want to draw only a specific contour, you can pass the index of that contour (e.g., 0 for the first contour, 1 for the second contour, and so on).

Color: This is the color used to draw the contours. OpenCV uses BGR (Blue, Green, Red) format instead of the more common RGB format. (0,0,255) specifies red.

Thickness: This specifies the thickness of pixels of contour lines.

# Display the image

*cv.imshow('Contours drawn', blank)*