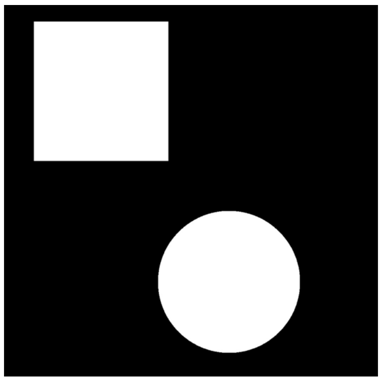
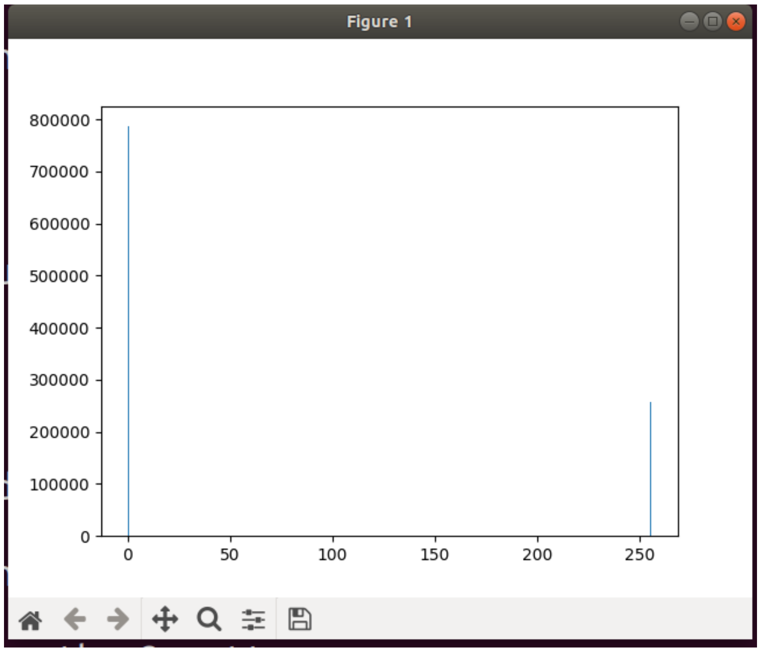
Goal is to have the computer recognize these shapes:



Pattern recognition process:

1. Noise reduction by blurring
    a. Noise reduction is done to distinguish the edge of each objects in the image
    b. There is no need to do noise reduction on this image since it is simple enough to distinguish two distinct colors. A noisy image in the thresholding step will show why.
2. Histogram
    a. Histogram is used to determine the threshold value that will be used for the next step
    b. Install OpenCV package
    ```
    sudo apt install python3-opencv
    ```
    c. Verify if installation is successful
    ```
    python3 -c "import cv2; print(cv2.__version__)"
    ```
    d. Download the image into the same folder. I named it as pattern.jpg
    e. Create pattern.py and create a code to plot histogram
    ```
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt

    img = cv2.imread('pattern.jpg',0)
    plt.hist(img.ravel(),256,[0,256])
    plt.show()
    ```
    f. Explanation
    plt.hist() finds the histogram and plot it
    ravel() changes 2D/multi-dimensional array to contiguous flattened array
    256 = number of pixels
    g. Output
    ```
    python3 pattern.py
    ```



3. Thresholding
   a. In Otsu's thresholding, it iterates through all possible threshold values and calculates the spread measurement for the pixel levels for each threshold. Its goal is to find the threshold value where the sum of the foreground and background spreads are at the minimum.
   b. Added lines of code

```
# Global thresholding
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding
(v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

c. Explanation

threshold() applies fixed-level thresholding to a multiple-channel array. It is typically used to get a binary image out of a grayscale image or for removing noise by filtering out pixels with too small or too large values.

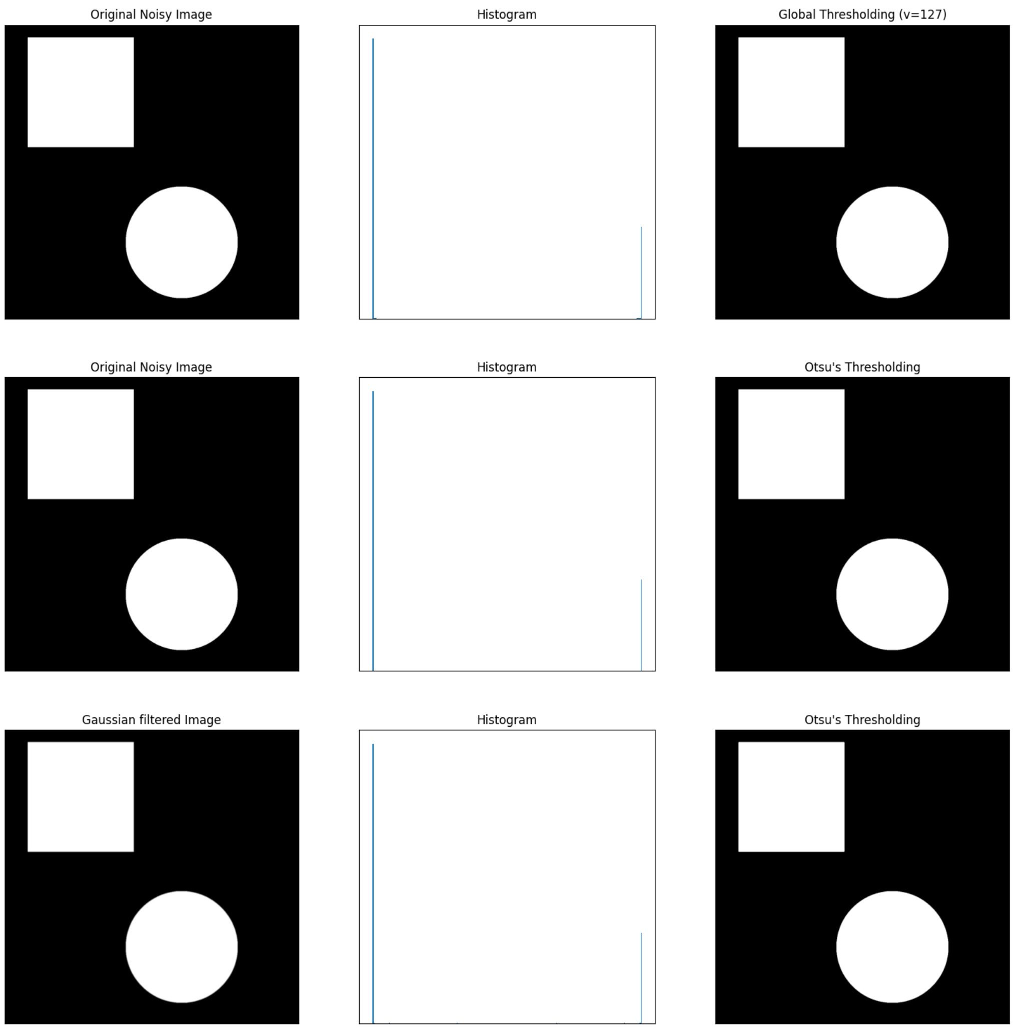
THRESH_BINARY

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

THRESH_OTSU uses the Otsu algorithm to choose the optimal threshold value. It is implemented only for 8-bit single-channel images.

Gaussianblur() blurs an image using a Gaussian filter

d. Output



4. Connectivity analysis

a. This step distinguishes separate objects in an image by using either a 4-pixel or 8-pixel connectivity. 4-pixel connects pixels with the same value along the edges. 8-pixel does the same with the addition of edges and corners.

b. Added lines of code

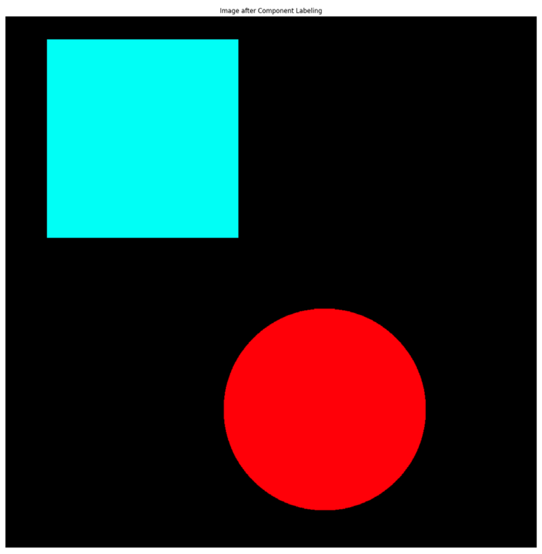
```
def connected_component_label(path):
    # Getting the input image
    img = cv2.imread('pattern.jpg', 0)
    # Converting those pixels with values 1-127 to 0 and others to 1
    img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]
    # Applying cv2.connectedComponents()
    num_labels, labels = cv2.connectedComponents(img)
    # Map component labels to hue val, 0-179 is the hue range in OpenCV
    label_hue = np.uint8(179*labels/np.max(labels))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
    # Converting cvt to BGR
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
    # set bg label to black
    labeled_img[label_hue==0] = 0
    #Showing Image after Component Labeling
    plt.imshow(cv2.cvtColor(labeled_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Image after Component Labeling")
    plt.show()
connected_component_label(img)
```

c. Explanation

The above uses a classical algorithm with makes two passes. First to record equivalences and assign temporary labels. Second to replace the temporary labels with the label of its equivalent class.

d. Output

Different objects are assigned different colors



5. Pattern recognition
    a. This step is to recognize what kind of objects are in the picture
    b. Install

```
pip3 install cvlib
```

    c. Added lines of code

```
font = cv2.FONT_HERSHEY_COMPLEX
_, threshold =
  cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
contours, _ = cv2.findContours(threshold, cv2.RETR_TREE,
  cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
  approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt, True), True)
  cv2.drawContours(img, [approx], 0, (0), 5)
  x = approx.ravel()[0]
  y = approx.ravel()[1]
  if len(approx) == 4:
       cv2.putText(img, "Square", (x,y), font, 1, (255))
  else:
       cv2.putText(img, "Circle", (x,y), font, 1, (255))
img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.drawContours(img, contours, -1, (255,0,0), 2)
cv2.imshow("Pattern recognition", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

    d. Explanation

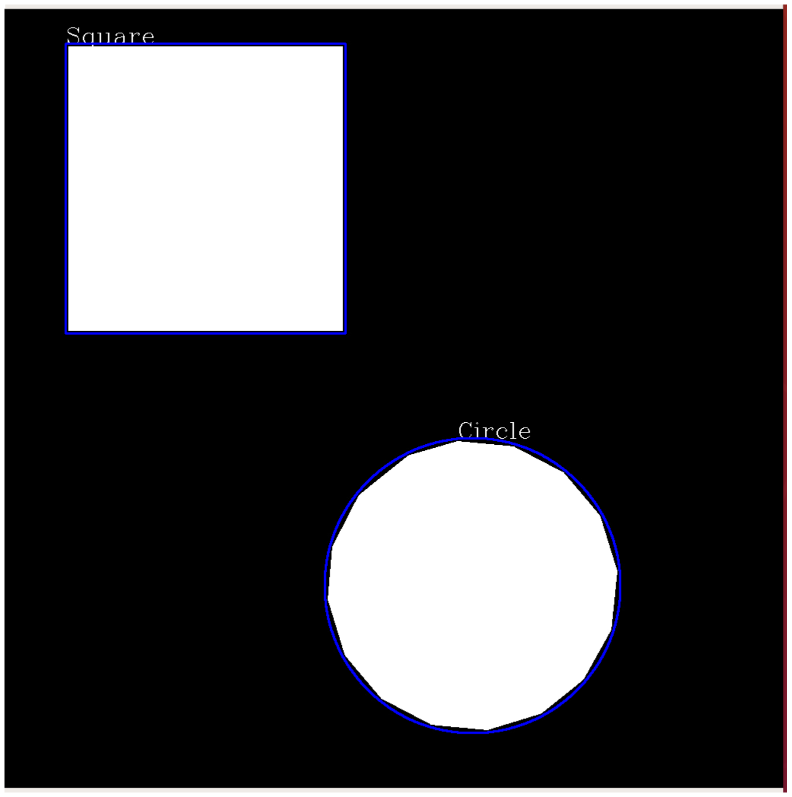
Threshold was used to get a black and white image, then the contours or boundaries of the shapes were found. Using the contours, the coordinates for each shape is known. By approximating the contours, we can find the name of the shape of the object.

len(approx) == ?? can be changed into different numbers (e.g. 3 for Triangle)

6 < len(approx) < 15 is for ellipse or oval

img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR) is to convert the black and white image and contour back to colored

    e. Output



Reference:

Histogram

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_begins/py_histogram_begins.html

Threshold

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a147222a96556ebc1d948b372bcd7ac59

Connectivity analysis

https://iq.opengenus.org/connected-component-labeling/

https://github.com/yashml/OpenGenus_Articles_Code/tree/master/Connected%20Component%20Labeling

Pattern recognition

https://pysource.com/2018/09/25/simple-shape-detection-opencv-with-python-3/