Course Name:	**Electronics Application Using Python Programming**	**Semester:**	**V**
Date of Performance:	**20 / 08 / 2024**	**Batch No:**	**APPA 2**
Faculty Name:	**Prof. Deepa Jain**	**Roll No:**	**16014022096**
Faculty Sign & Date:		**Grade/Marks:**	

Experiment No: 4

Title: Write a Python Program for perform Edge detection, Contours operation in Open CV

Aim and Objective of the Experiment:

1) To learn Concept of Canny edge detection.
2) To learn following contours in OpenCV
   - Find contours, draw contours
   - Matching different shapes

COs to be achieved:

CO2: Illustrate python libraries for image processing and its applications

Tools required:

Any python editor tool

Theory:

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.

1. Noise Reduction

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter.

2. Finding Intensity Gradient of the Image

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (Gx) and vertical direction (Gy). From these two images, we can find edge gradient and direction for each pixel as follows:

$$edge\ gradient = \sqrt{Gx^2 + Gy^2}$$

$$edge\ direction(Angle) = \tan^{-1}\left(\frac{Gx}{Gy}\right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3. Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient

4. Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded.

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.

• findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.

• 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.

there are three arguments in cv2.findContours() function, first one is source image, second is contour retrieval mode, third is contour approximation method. And it outputs the image, 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

To draw the contours, cv2.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.

Match Shapes: OpenCV comes with a function cv2.matchShapes() which enables us to compare two shapes, or two contours and returns a metric showing the similarity. The lower the result, the better match it is. It is calculated based on the hu-moment values.

Code:

1. Write a small application to find the Canny edge detection whose threshold values can be varied using two trackbars.
2. Implement AHE and CLAHE Histogram Equalization
3. Take an any suitable image and apply findcontour and drawcontour functions and analysis the any 5 properties of contours.
4. Compare images of digits or letters using cv2.matchShapes()

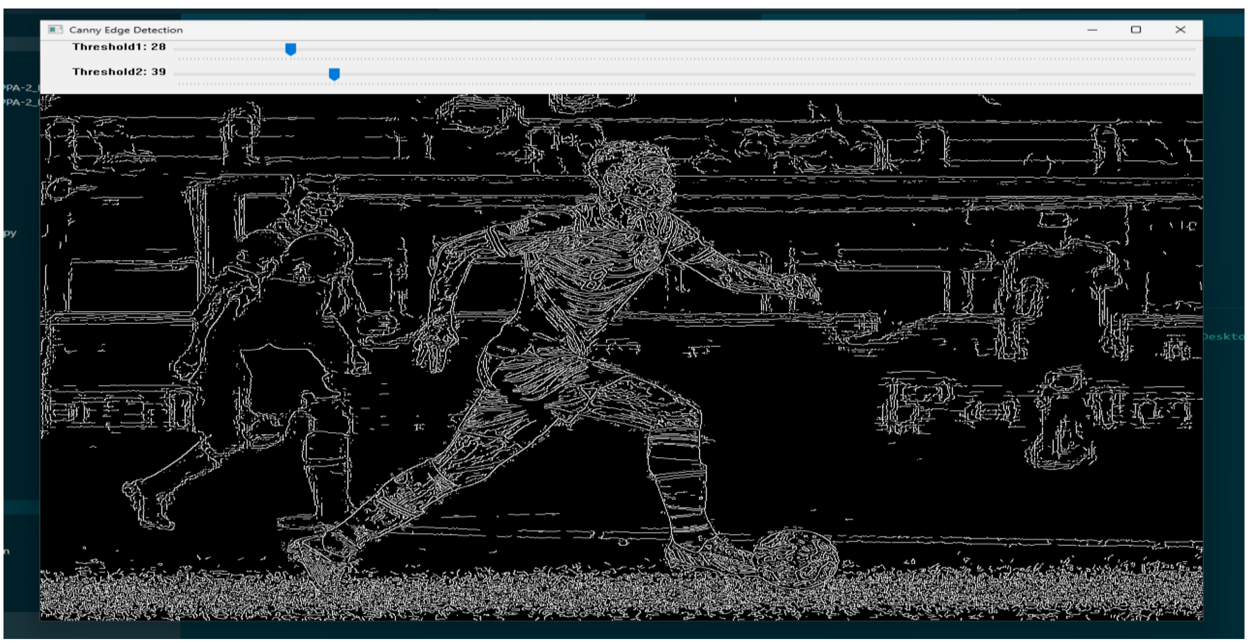
Output:

CODE 1:

```
import cv2
import numpy as np

def nothing(x):
    pass

cv2.namedWindow('Canny Edge Detection')
cv2.createTrackbar('Threshold1', 'Canny Edge Detection', 0, 255, nothing)
cv2.createTrackbar('Threshold2', 'Canny Edge Detection', 0, 255, nothing)

cv2.setTrackbarPos('Threshold1', 'Canny Edge Detection', 50)
cv2.setTrackbarPos('Threshold2', 'Canny Edge Detection', 150)

image = cv2.imread('EXPERIMENT 03//bellingam.jpg', cv2.IMREAD_GRAYSCALE)
while True:
    thresh1 = cv2.getTrackbarPos('Threshold1', 'Canny Edge Detection')
    thresh2 = cv2.getTrackbarPos('Threshold2', 'Canny Edge Detection')
    edges = cv2.Canny(image, thresh1, thresh2)
    cv2.imshow('Canny Edge Detection', edges)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


cv2.destroyAllWindows()
```

OUTPUT 1:

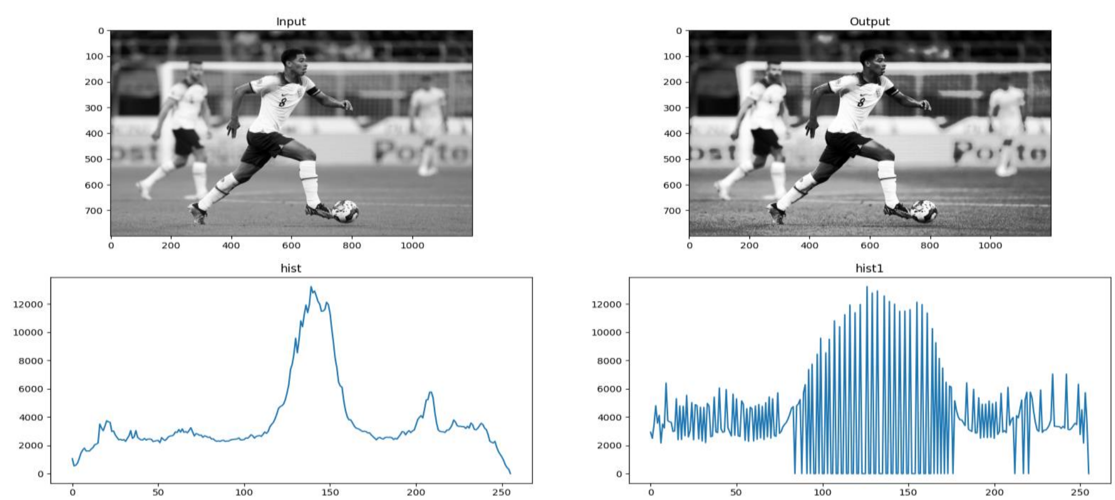

Canny Edge Detection
Threshold1: 50
Threshold2: 150


Canny Edge Detection
Threshold1: 28
Threshold2: 39

CODE 2a:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt


image = cv2.imread('EXPERIMENT 03//bellingam.jpg', 0)

hist = cv2.calcHist([image],[0],None,[256],[0,255])
img_hist = cv2.equalizeHist(image)
hist1 = cv2.calcHist([img_hist],[0],None,[256],[0,255])

plt.subplot(221)
plt.imshow(image, cmap='gray')
plt.title('Input')

plt.subplot(222)
plt.imshow(img_hist, cmap='gray')
plt.title('Output')

plt.subplot(223)
plt.plot(hist)
plt.title('hist')

plt.subplot(224)
plt.plot(hist1)
plt.title('hist1')

plt.show()
```

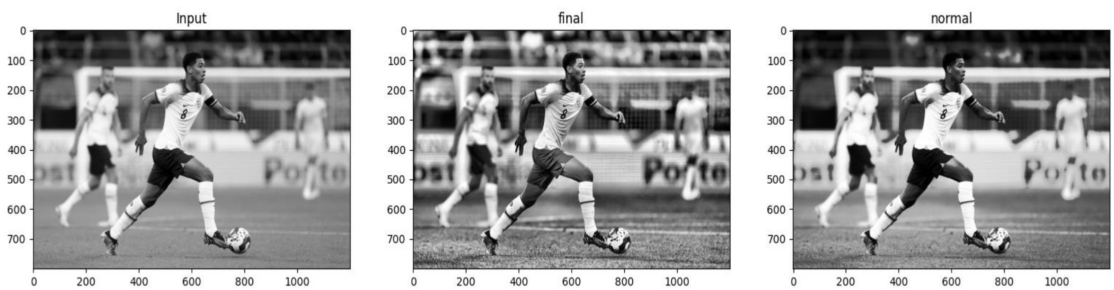
OUTPUT 2a:



CODE 2b:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt


image = cv2.imread('EXPERIMENT 03//bellingam.jpg', 0)

clahe = cv2.createCLAHE(clipLimit=5)
final = clahe.apply(image)
normal_hist = cv2.equalizeHist(image)

plt.subplot(131)
plt.imshow(image, cmap='gray')
plt.title('Input')

plt.subplot(132)
plt.imshow(final, cmap='gray')
plt.title('final')

plt.subplot(133)
plt.imshow(normal_hist, cmap='gray')
plt.title('normal')


plt.show()
```

OUTPUT 2b:



CODE 2c:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt


image = cv2.imread('EXPERIMENT 03//bellingam.jpg', 1)

b,g,r = cv2.split(image)

img_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
h,s,v = cv2.split(img_hsv)
s = cv2.equalizeHist(s)
merged_hsv = cv2.merge((h,s,v))
bgr_enhanced = cv2.cvtColor(merged_hsv, cv2.COLOR_HSV2BGR)

cv2.imshow('a', image)
cv2.imshow('a', bgr_enhanced)

hist = cv2.calcHist([b],[0],None,[256],[0,255])
plt.plot(hist)
hist = cv2.calcHist([g],[0],None,[256],[0,255])
plt.plot(hist)
hist = cv2.calcHist([r],[0],None,[256],[0,255])
plt.plot(hist)

plt.show()
cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT 2c:

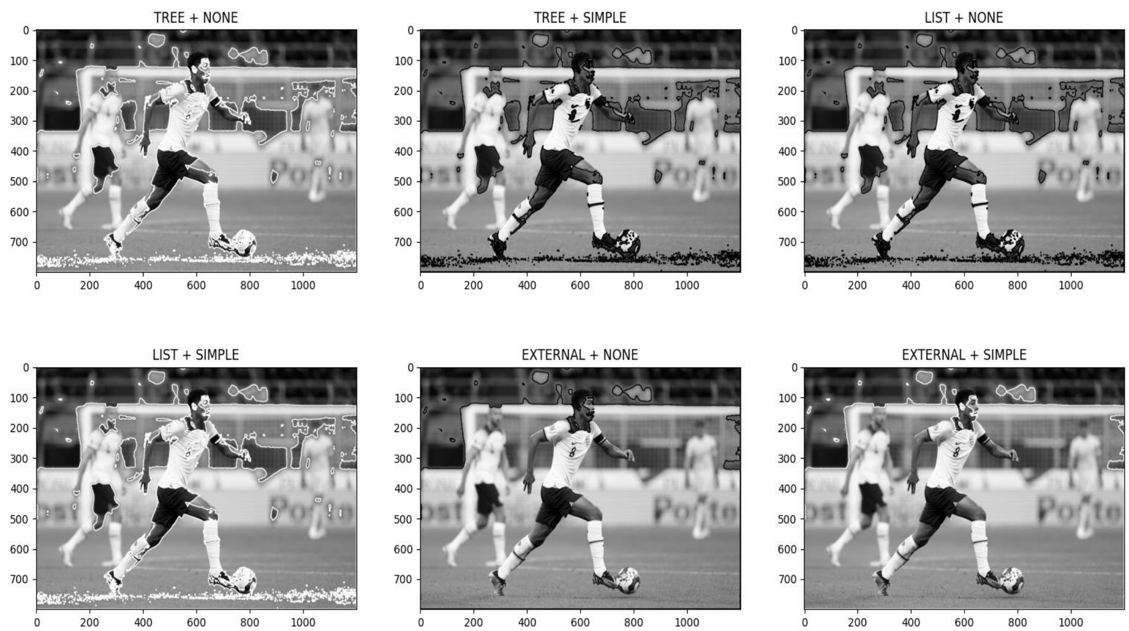

Explorer (Ctrl+Shift+E)
EXPERIMENT 01
EXPERIMENT 02
EXPERIMENT 03 > demo4.py > ...
Figure 1
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
demo5.py
demo6.py
demo7.py
demo8.py
demo9.py
ea.png
opencv.jpg
tempCodeRunnerFile.python
OUTLINE

CODE 3:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('EXPERIMENT 03//bellingam.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, binary = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

contours1, _ = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
contours2, _ = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours3, _ = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
contours4, _ = cv2.findContours(binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
contours5, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
contours6, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

image1 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
image2 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
image3 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
image4 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
image5 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
image6 = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)

cv2.drawContours(image1, contours1, -1, (255, 0, 0), 2)
cv2.drawContours(image2, contours2, -1, (0, 255, 0), 2)
cv2.drawContours(image3, contours3, -1, (0, 0, 255), 2)
cv2.drawContours(image4, contours4, -1, (255, 255, 0), 2)
cv2.drawContours(image5, contours5, -1, (0, 255, 255), 2)
cv2.drawContours(image6, contours6, -1, (255, 0, 255), 2)

plt.subplot(231)
plt.imshow(image1, cmap='gray')
plt.title('TREE + NONE')

plt.subplot(232)
plt.imshow(image2, cmap='gray')
plt.title('TREE + SIMPLE')

plt.subplot(233)
plt.imshow(image3, cmap='gray')
plt.title('LIST + NONE')

plt.subplot(234)
```

```python
plt.imshow(image4, cmap='gray')
plt.title('LIST + SIMPLE')

plt.subplot(235)
plt.imshow(image5, cmap='gray')
plt.title('EXTERNAL + NONE')

plt.subplot(236)
plt.imshow(image6, cmap='gray')
plt.title('EXTERNAL + SIMPLE')

plt.show()
```

OUTPUT 3:

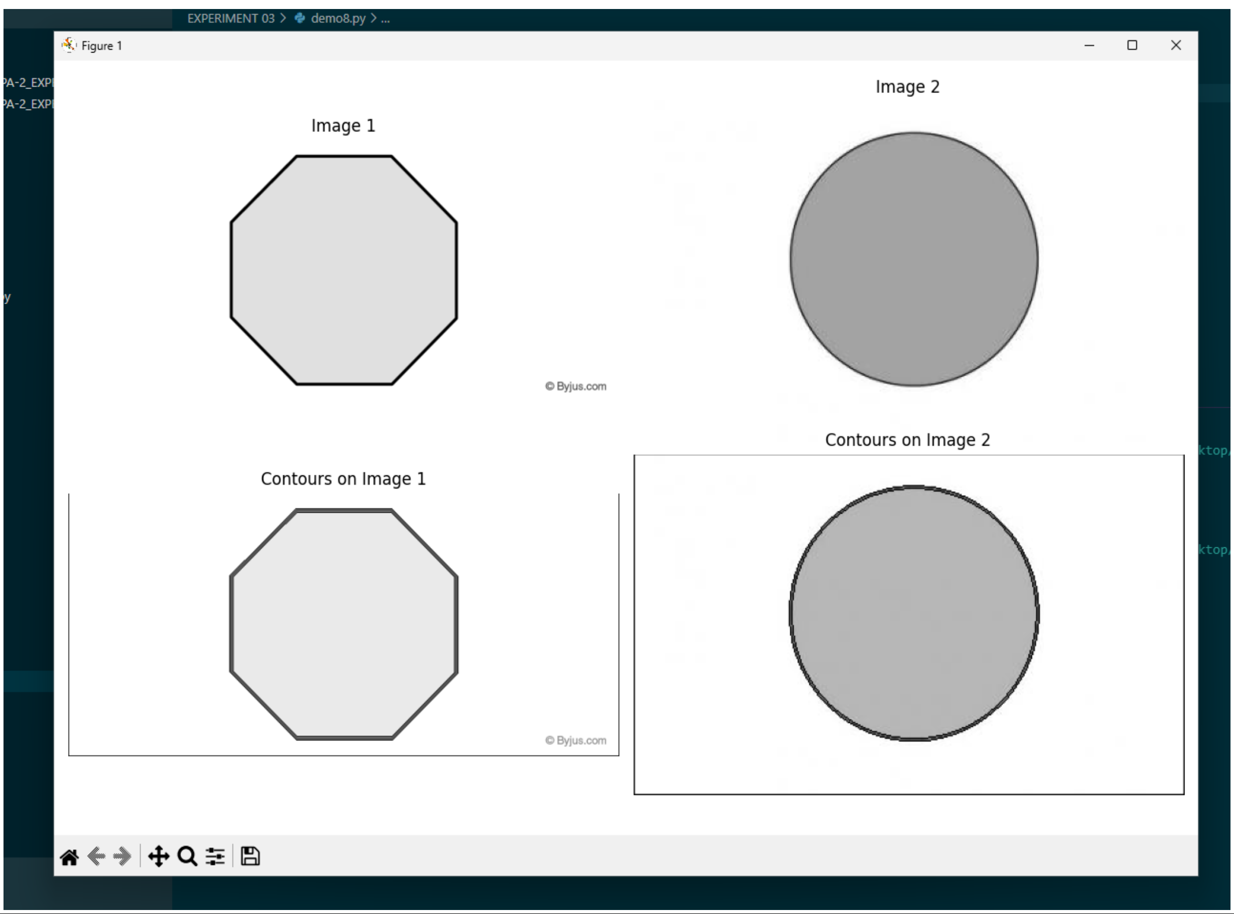


CODE 4:

```
import cv2
import matplotlib.pyplot as plt

# Read two images as grayscale images
img1 = cv2.imread('EXPERIMENT 03//shape1.png', 0)
img2 = cv2.imread('EXPERIMENT 03//shape2.jpg', 0)

# Apply thresholding on the images to convert to binary images
ret, thresh1 = cv2.threshold(img1, 127, 255, 0)
ret, thresh2 = cv2.threshold(img2, 127, 255, 0)

# Find the contours in the binary images
contours1, hierarchy1 = cv2.findContours(thresh1, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
contours2, hierarchy2 = cv2.findContours(thresh2, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

# Print the number of shapes detected
print("Number of Shapes detected in Image 1:", len(contours1))
print("Number of Shapes detected in Image 2:", len(contours2))

# Create a figure to display the results
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

# Display the original images
axs[0, 0].imshow(img1, cmap='gray')
axs[0, 0].set_title('Image 1')
axs[0, 0].axis('off')
axs[0, 1].imshow(img2, cmap='gray')
axs[0, 1].set_title('Image 2')
axs[0, 1].axis('off')

# Draw contours on the images
contour_img1 = cv2.drawContours(img1.copy(), contours1, -1, (0, 255, 0), 1)
contour_img2 = cv2.drawContours(img2.copy(), contours2, -1, (0, 255, 0), 1)

axs[1, 0].imshow(contour_img1, cmap='gray')
axs[1, 0].set_title('Contours on Image 1')
axs[1, 0].axis('off')
axs[1, 1].imshow(contour_img2, cmap='gray')
axs[1, 1].set_title('Contours on Image 2')
axs[1, 1].axis('off')
```

```
plt.tight_layout()
plt.show()

# Check if contours are available and then compute match scores
if len(contours1) > 0 and len(contours2) > 0:
    cnt1 = contours1[0]
    cnt2 = contours2[0]

    # Compute the match scores
    ret11 = cv2.matchShapes(cnt1, cnt1, 1, 0.0)
    ret22 = cv2.matchShapes(cnt2, cnt2, 1, 0.0)
    ret12 = cv2.matchShapes(cnt1, cnt2, 1, 0.0)

    # Print the matching scores
    print("Matching Image 1 with itself:", ret11)
    print("Matching Image 2 with itself:", ret22)
    print("Matching Image 1 with Image 2:", ret12)
else:
    print("No contours found in one or both images.")
```

OUTPUT 4:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\vrish\OneDrive\Desktop\APPA> & C:/Users/vrish/Ap
Number of Shapes detected in Image 1: 3
Number of Shapes detected in Image 2: 84
Matching Image 1 with itself: 0.0
Matching Image 2 with itself: 0.0
Matching Image 1 with Image 2: 0.24338431205492295
PS C:\Users\vrish\OneDrive\Desktop\APPA>
```

Post Lab Subjective/Objective type Questions:

1. State and explain any five Contour Properties.

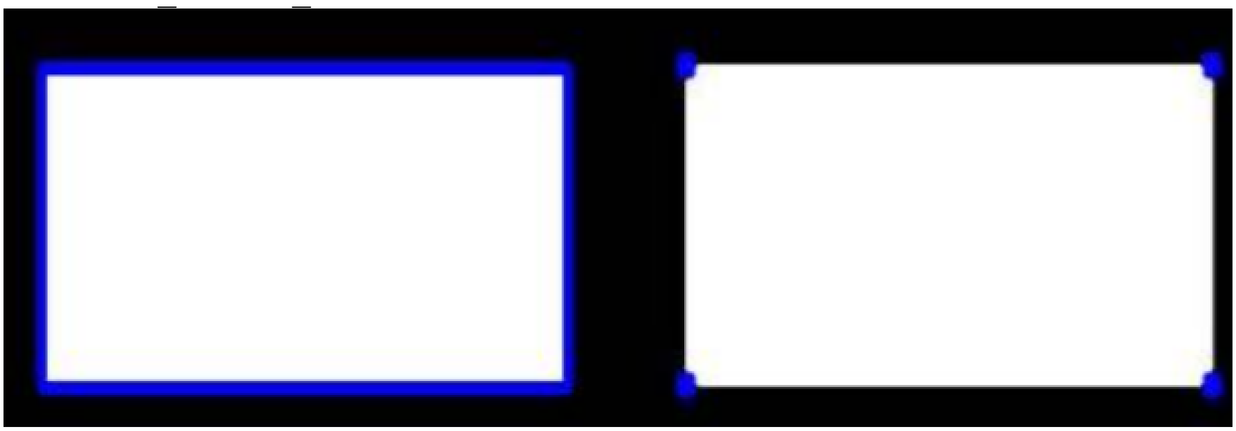
ANS:

- **Area:** Measures the number of pixels inside the contour. (cv2.contourArea())
- **Perimeter:** The length of the contour's boundary. (cv2.arcLength())
- **Bounding Rectangle:** The smallest rectangle enclosing the contour. (cv2.boundingRect())
- **Centroid:** The central point of the contour, calculated from moments.
- **Moment:** Provides shape information, including area and centroid, from the contour. (cv2.moments())

2. Explain and compare various Contour Approximation Method

ANS:

When using cv.CHAIN_APPROX_NONE, all boundary points of a contour are stored, which can be excessive. For example, if you have a contour of a straight line, you don't need all the points along that line to represent it—just the two endpoints are sufficient. This is where cv.CHAIN_APPROX_SIMPLE comes in. It reduces the number of points by removing redundant ones, thus compressing the contour and saving memory. For instance, in the case of a rectangle, cv.CHAIN_APPROX_SIMPLE would capture just 4 points (the corners), while cv.CHAIN_APPROX_NONE would store all the boundary points This clearly illustrates the memory savings achieved by using cv.CHAIN_APPROX_SIMPLE.



3. Explain the importance of LPF and HPF with any application.

ANS:

Low Pass Filter (LPF): Smooths signals by allowing low frequencies to pass through and blocking high frequencies. **Application:** Reduces noise and blurs images.

High Pass Filter (HPF): Highlights rapid changes and edges by allowing high frequencies to pass through and blocking low frequencies. **Application:** Enhances edges and fine details in images

Conclusion:

We have successfully learnt the cconcept of Canny edge detection. and also, about contours like to find contours, draw contours and matching different shapes using python libraries.

Signature of faculty in-charge with Date: