GEOMETRICAL SHAPE DETECTION AND RECOGNITION USING PYTHON IN IMAGE PROCESSING

An ELC activity report submitted in partial fulfilment of the requirements for the degree of

Bachelor of Engineering

in

Electronics and Communication Engineering

Submitted by

Saksham Sohal

1024060114

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY, PATIALA

February-April 2024

Table of Contents

Objective

The objective of this project is to develop a Python-based system for detecting and recognizing various geometrical shapes in digital images using image processing techniques. This involves preprocessing images, applying edge detection, contour detection, and shape classification methods to identify shapes such as circles, squares, triangles, and rectangles. The project aims to enhance understanding of computer vision techniques and their applications in real-world scenarios, such as object recognition, robotics, and industrial automation.

Introduction

Image processing is a crucial field in computer vision, enabling machines to analyse and interpret visual data. One of its key applications is **geometrical shape detection and recognition**, which helps in identifying objects based on their structural properties.

The process involves several steps, including **image preprocessing, edge detection, contour detection, and shape classification**. These techniques allow us to detect fundamental shapes like triangles, squares, circles, and polygons in an image.

Python, with libraries like OpenCV, NumPy, PIL (Pillow), and Matplotlib, provides efficient tools for image processing:

- **OpenCV** handles image manipulation, edge detection, and contour detection.
- **NumPy** facilitates numerical operations for image arrays.
- **PIL (Pillow)** is used for opening, processing, and converting images.
- **Matplotlib** helps visualize images and output results at different stages.

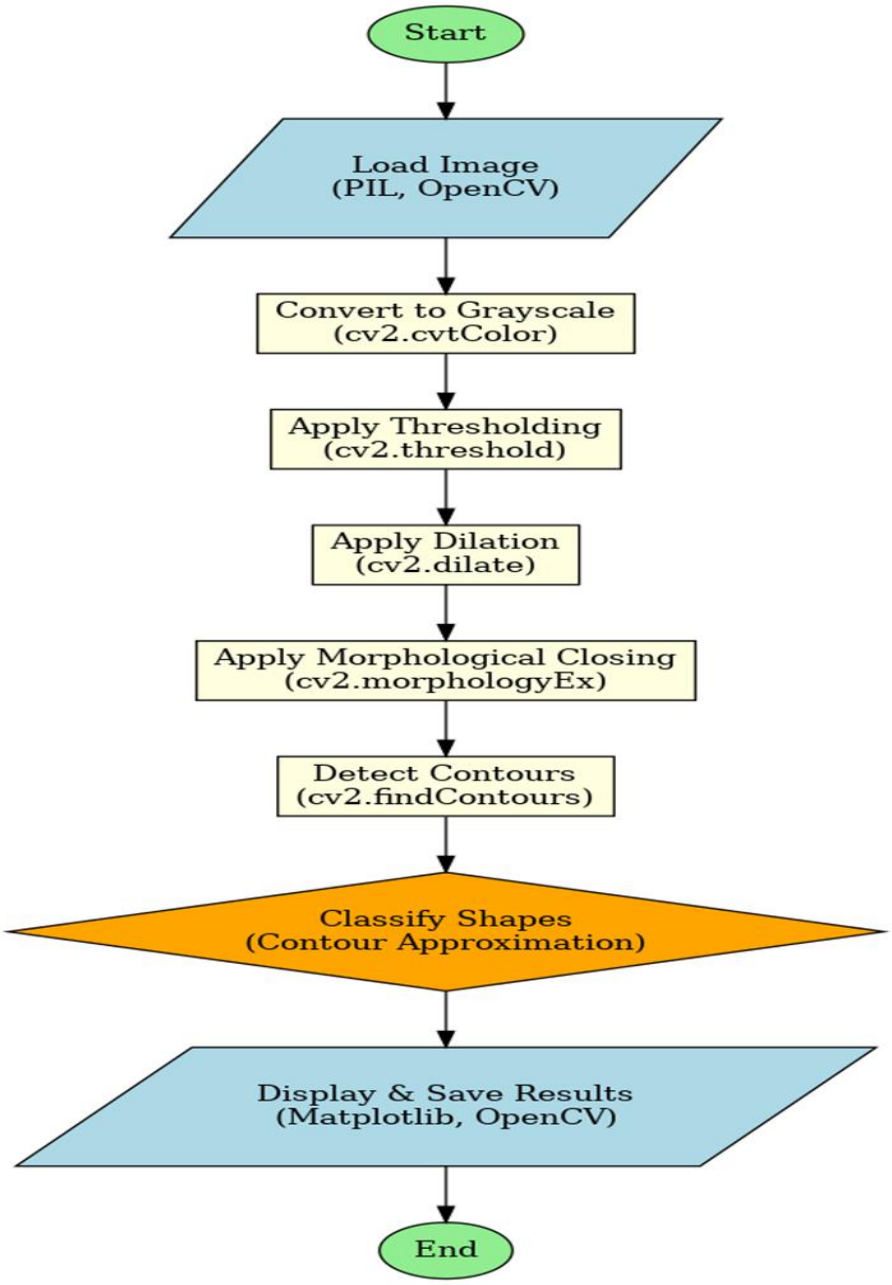
The detection process typically follows these steps:

- **Image Preprocessing** – Convert the image to grayscale and apply filters to reduce noise.
- **Edge Detection** – Use methods like Canny edge detection to highlight object boundaries.
- **Contour Detection** – Identify closed shapes by finding their contours.
- **Shape Classification** – Approximate contours and determine shape type using properties like number of edges and aspect ratio.

Applications of shape detection include **robotics, industrial automation, object tracking, medical imaging and traffic sign recognition**.

This project demonstrates a step-by-step approach to detecting and recognizing geometrical shapes using Python and OpenCV, along with NumPy, PIL, and Matplotlib, providing insights into computer vision and image analysis techniques.

Flowchart

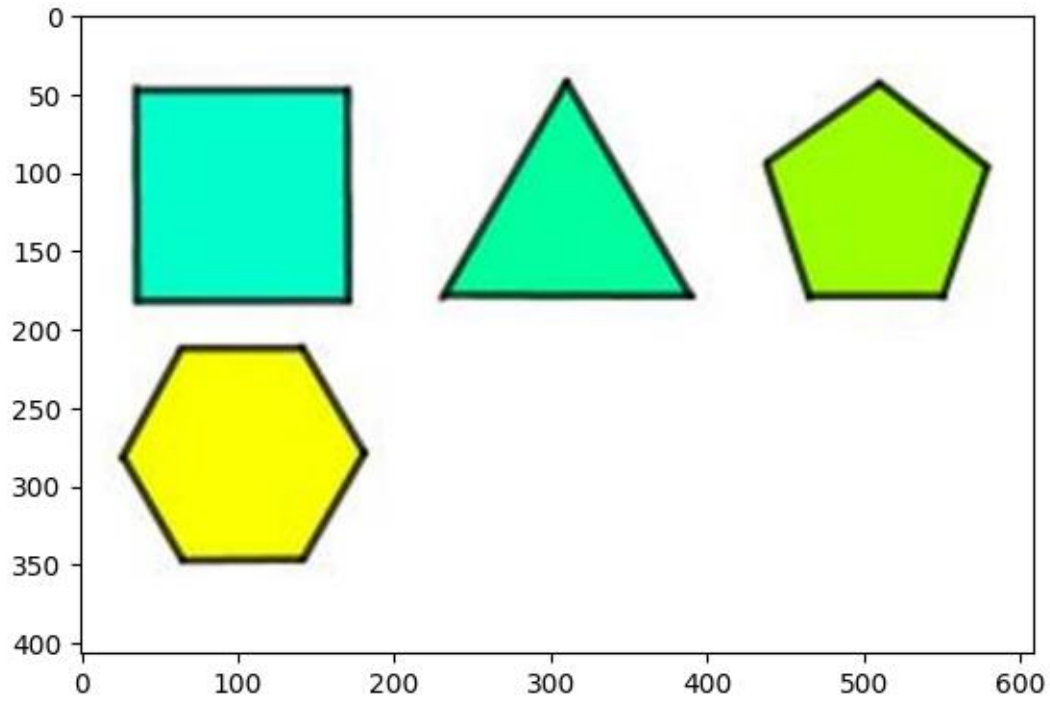


Flowchart created using Lucidchart

Output

```
1. import cv2 #computer vision library
2. import PIL #python imaging library
3. import numpy as np
4. from matplotlib import pyplot as plt
5. from PIL import Image
6. from matplotlib import image as mpimg
```

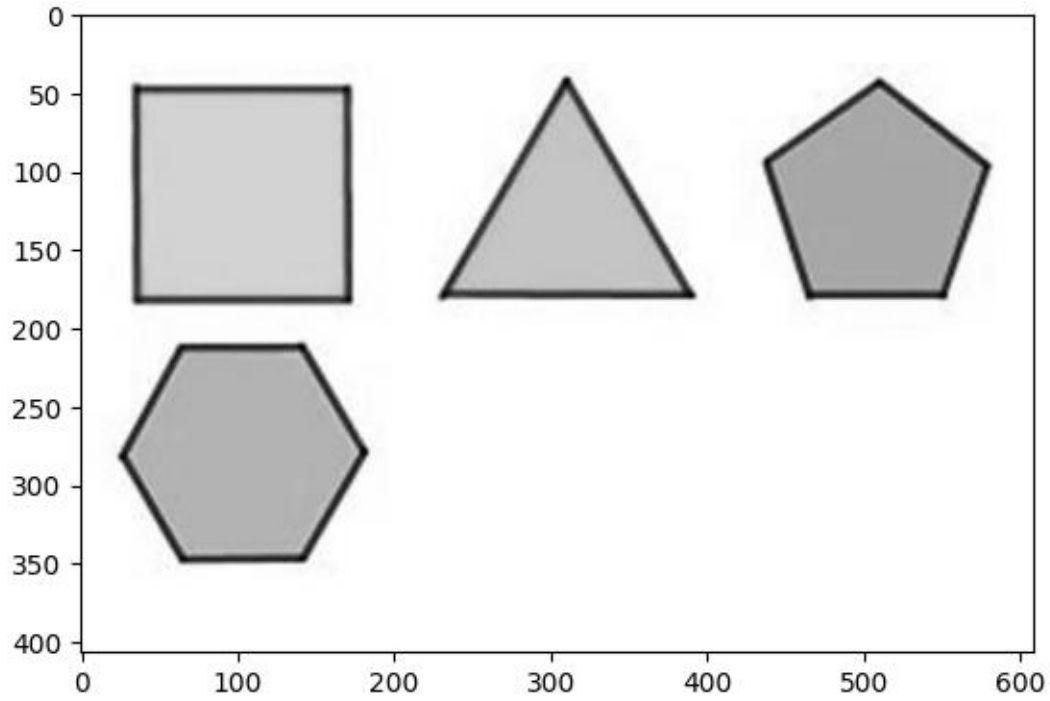
```
1. img=Image.open("4 shapes3.jpeg")
2. plt.imshow(img)
```



```
1. print(img.format)
2. print(img.size)
3. print(img.mode)
```

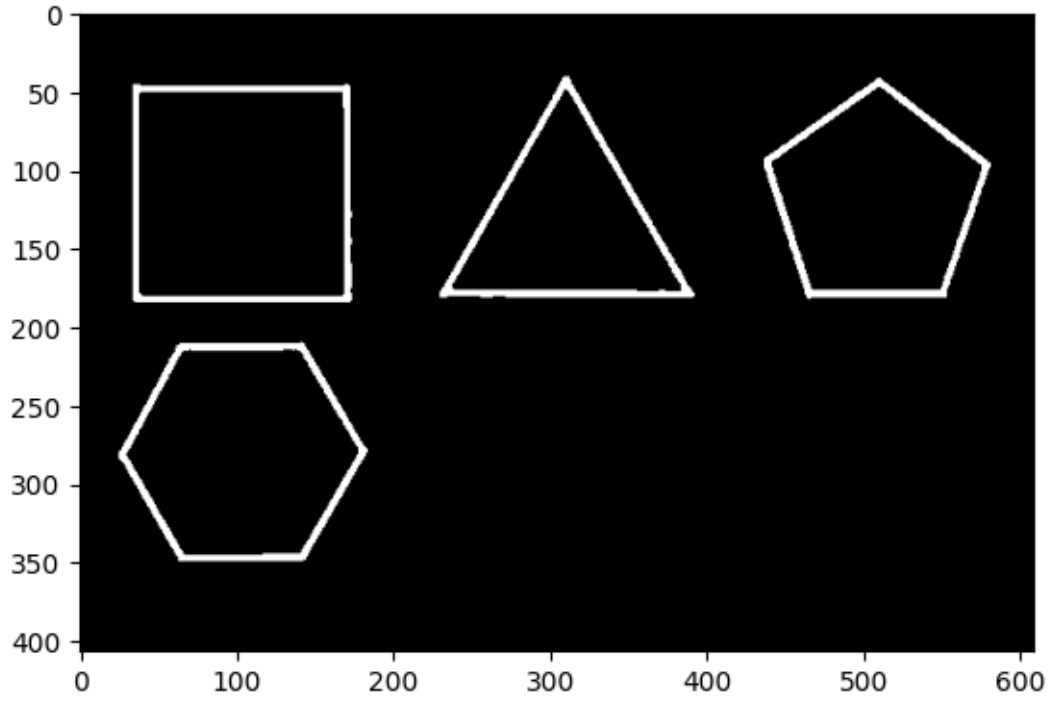
```
JPEG
(609, 407)
RGB
```

```
1. img = np.array(img)
2. gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
3. plt.imshow(gray,cmap ='gray')
```



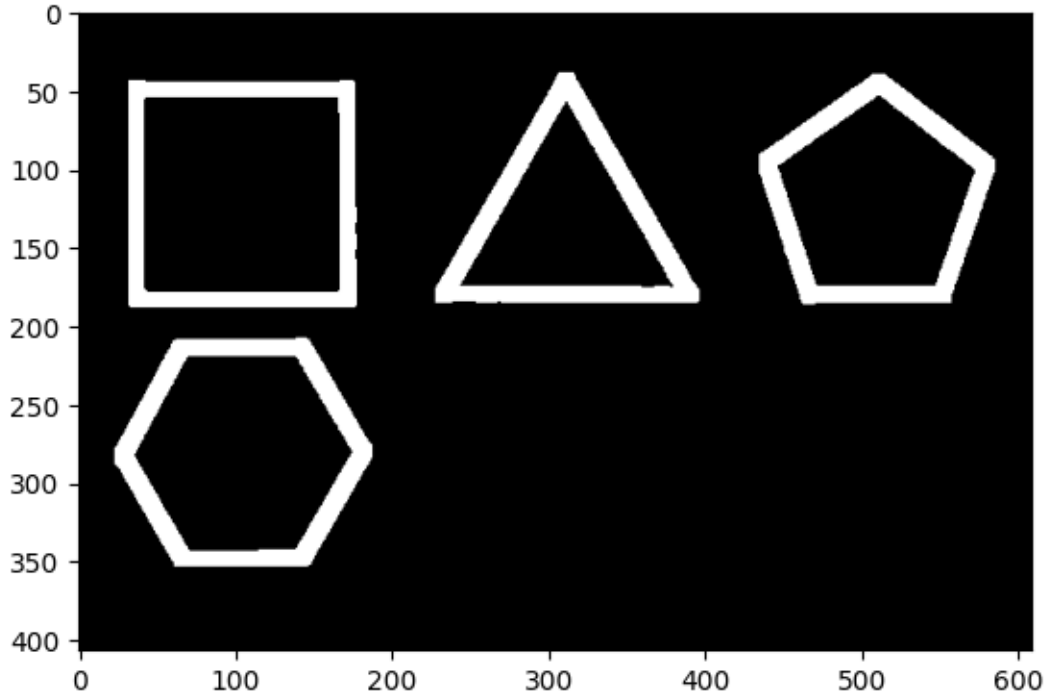

```
1. _,threshold=cv2.threshold(gray,110,255,cv2.THRESH_BINARY_INV)
```

```
1. plt.imshow(threshold, cmap='gray')
```



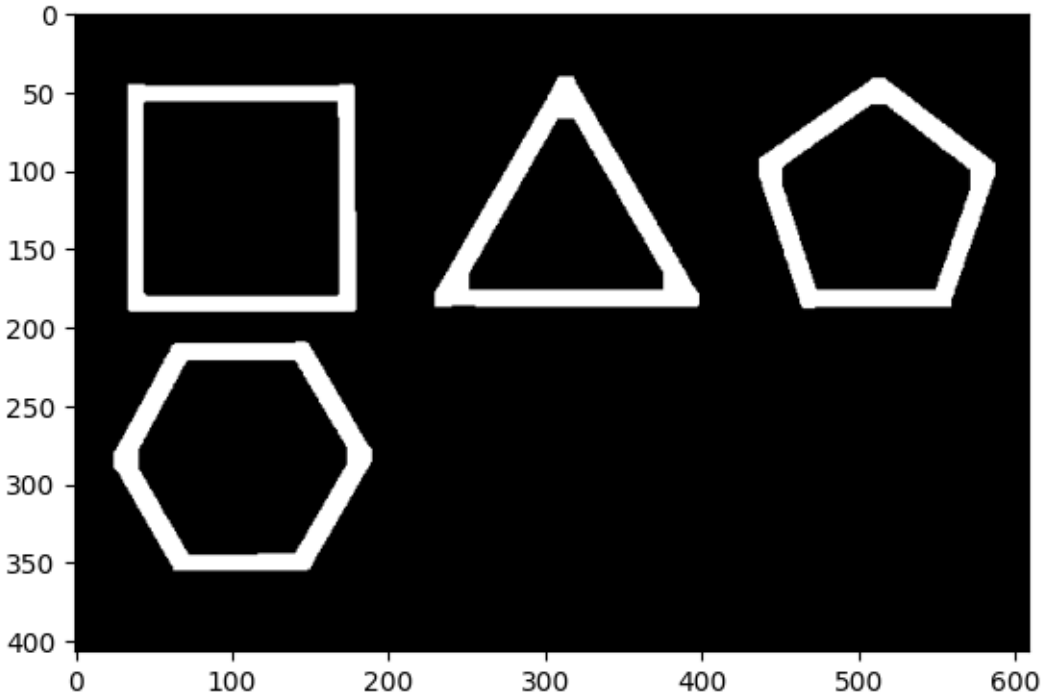

```
 #Different kernel matrices for some morphological operations.
2. kernel=cv2.getStructuringElement(cv2.MORPH_RECT,(4,4))
3. kernelc=cv2.getStructuringElement(cv2.MORPH_RECT,(6,6))
4. kernelo=cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
```

```
1. #Dilation on threshold image output
2. dilated=cv2.dilate(threshold,kernel, iterations=2)
3. plt.imshow(dilated,cmap='gray')
```



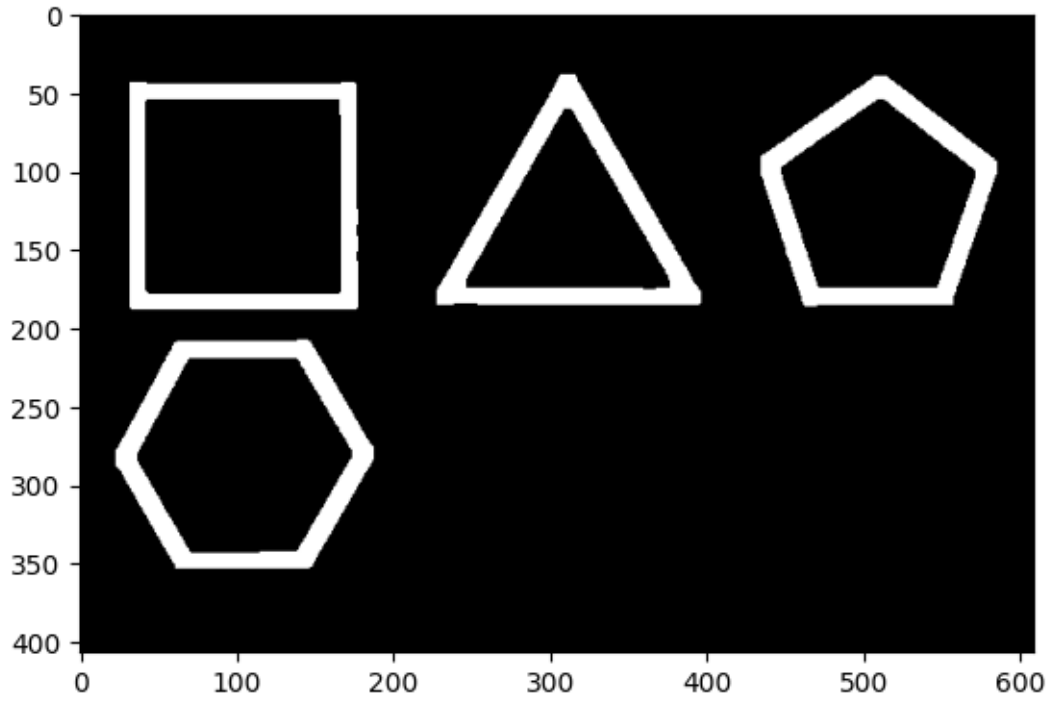

```
1. #connection of disjoining edges in the shape.
2. closed=cv2.morphologyEx(dilated,cv2.MORPH_CLOSE,kernelc, iterations=2)
```

```
1. plt.imshow(closed,cmap='gray')
```

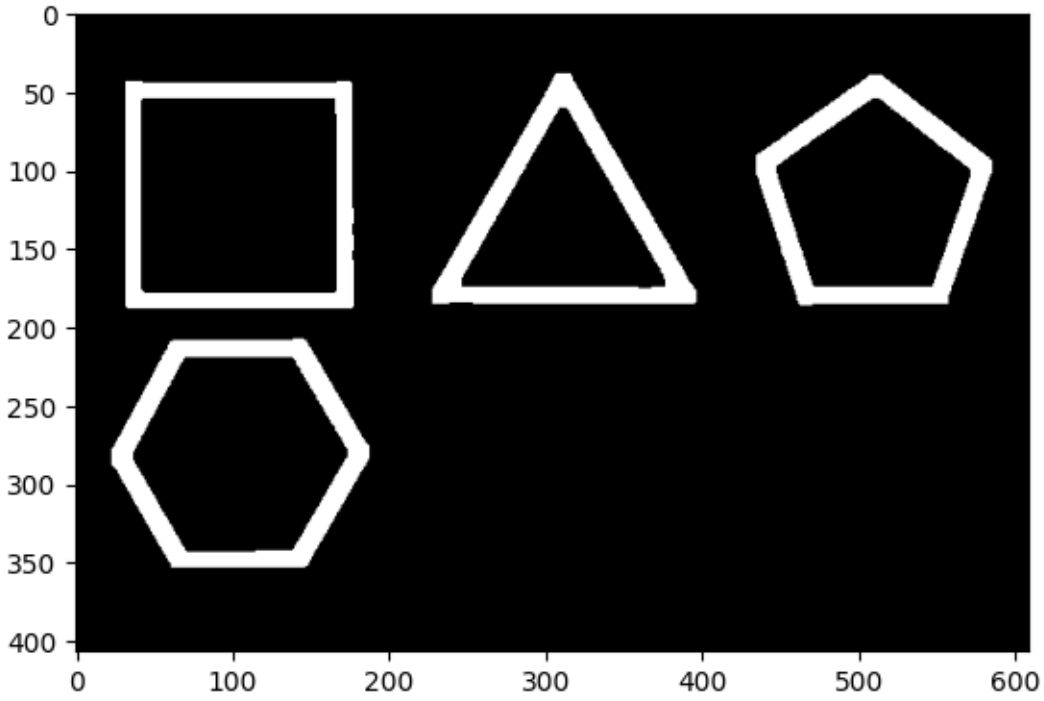


```
1. closed=cv2.morphologyEx(dilated,cv2.MORPH_CLOSE,kernelo, iterations=2)
```

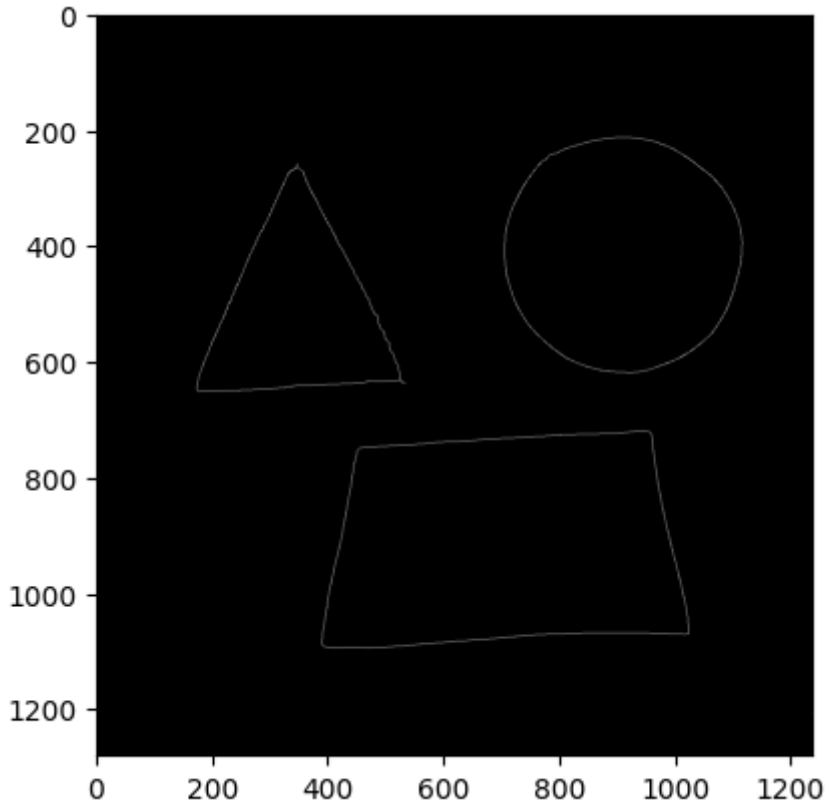
```
1. plt.imshow(closed,cmap='gray')
```



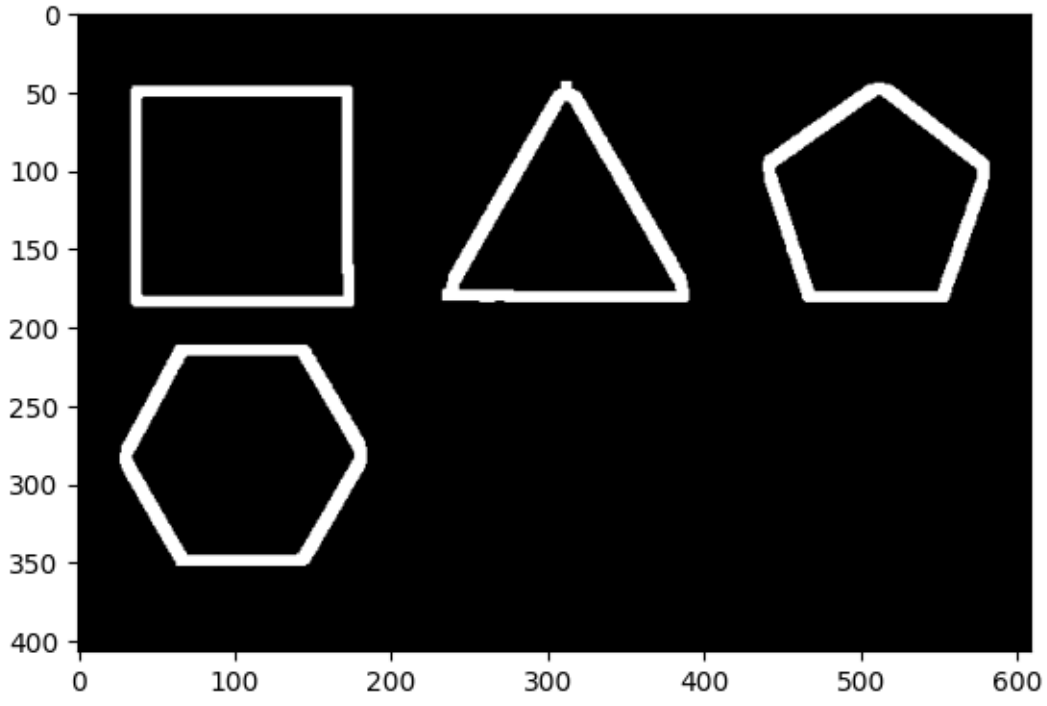

```
1. opening=cv2.morphologyEx(dilated,cv2.MORPH_OPEN,kernelo, iterations=2)
2. plt.imshow(closed,cmap='gray')
```



```
1. thinned = cv2.ximgproc.thinning(opening)
2. plt.imshow(thinned,cmap='gray')
```



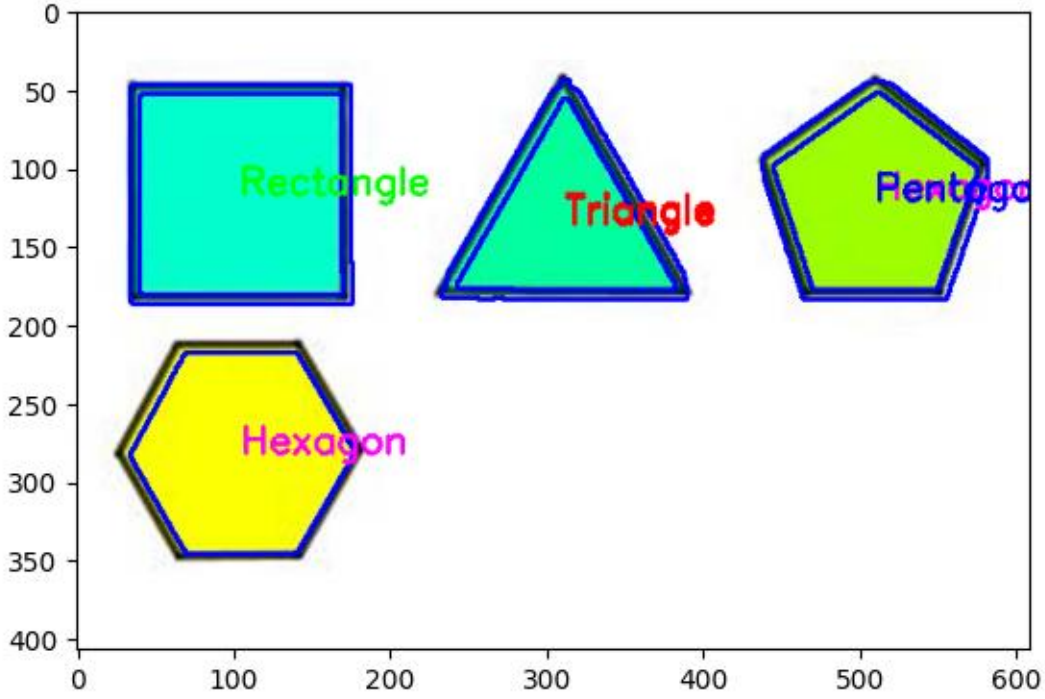

```
1. dilated_t = cv2.dilate(thinned, kernel, iterations=2)
2. plt.imshow(dilated_t, cmap='gray')
```



```
contours, _ = cv2.findContours(dilated_t, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
i=0
#list for storing names of shapes
for contour in contours:
  #here we are ignoring first counter because
  #findcontour function detects whole image as shape
  if i==0:
    i=1
    continue
  approx = cv2.approxPolyDP(contour, 0.02*cv2.arcLength(contour,True),True)
  #to draw contours over the image
  cv2.drawContours(img, [contour] , 0 ,(0 , 0,255), 2)
  #centroid of each of the shape
  M= cv2.moments(contour)
  if M['m00'] != 0.0:
    x = int(M['m10']/M['m00'])
    y = int(M['m01']/M['m00'])
  #put name of shape
  if len(approx) == 3:
    cv2.putText(img,"Triangle",(x,y),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,0,0),2)
  elif len(approx) == 4:
    cv2.putText(img,"Rectangle",(x,y),cv2.FONT_HERSHEY_SIMPLEX,0.8,(0,255,0),2)
  elif len(approx) == 5:
    cv2.putText(img,"Pentagon",(x,y),cv2.FONT_HERSHEY_SIMPLEX,0.8,(0,0,255),2)
  elif len(approx) == 6:
    cv2.putText(img,"Hexagon",(x,y),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,0,255),2)
  else :
    cv2.putText(img,"Circle",(x,y),cv2.FONT_HERSHEY_SIMPLEX,0.8,(0,0,255),2)
```

```
1.  plt.imshow(img)
```



Learnings

Through this project, I gained valuable insights into image processing and computer vision using Python. The key learnings are:

1. **Understanding Image Processing Basics**
- Learned how images are represented in grayscale and color spaces (RGB, BGR, HSV).
- Explored different techniques like thresholding, morphological operations, and edge detection.

2. **Working with OpenCV & Pillow**
- Used OpenCV for image manipulation, contour detection, and shape recognition.
- Learned how Pillow (PIL) can be used for loading and basic image processing.

3. **Contour Detection and Shape Approximation**
- Understood how cv2.findContours() works to detect object boundaries.
- Used contour approximation with cv2.approxPolyDP() to classify shapes based on their edges.

4. **Preprocessing for Better Detection**
- Applied Gaussian Blurring and thresholding to remove noise from images.
- Used morphological transformations (dilation & closing) to enhance shape visibility.

5. **Practical Applications of Shape Recognition**
- Realized how shape detection is used in object recognition, autonomous vehicles, industrial inspection, and medical imaging.

6. **Visualization and Debugging**
- Used matplotlib to visualize different stages of image processing.
- Learned how to overlay detected shapes on images for clear output representation.
- This project provided hands-on experience in implementing computer vision algorithms, and the knowledge gained can be extended to real-world AI and automation projects.

References

- Google Colab
  https://colab.research.google.com/drive/1w4uLbC8TdhmeY8DTFThRy_IPKVgyLit3#scrollTo=dJ5cyOr2ARN8
- OpenCV Documentation
  https://docs.opencv.org/
- NumPy Documentation
  https://numpy.org/doc/
- Pillow (PIL) Documentation
  https://pillow.readthedocs.io/
- Matplotlib Documentation
  https://matplotlib.org/stable/contents.html