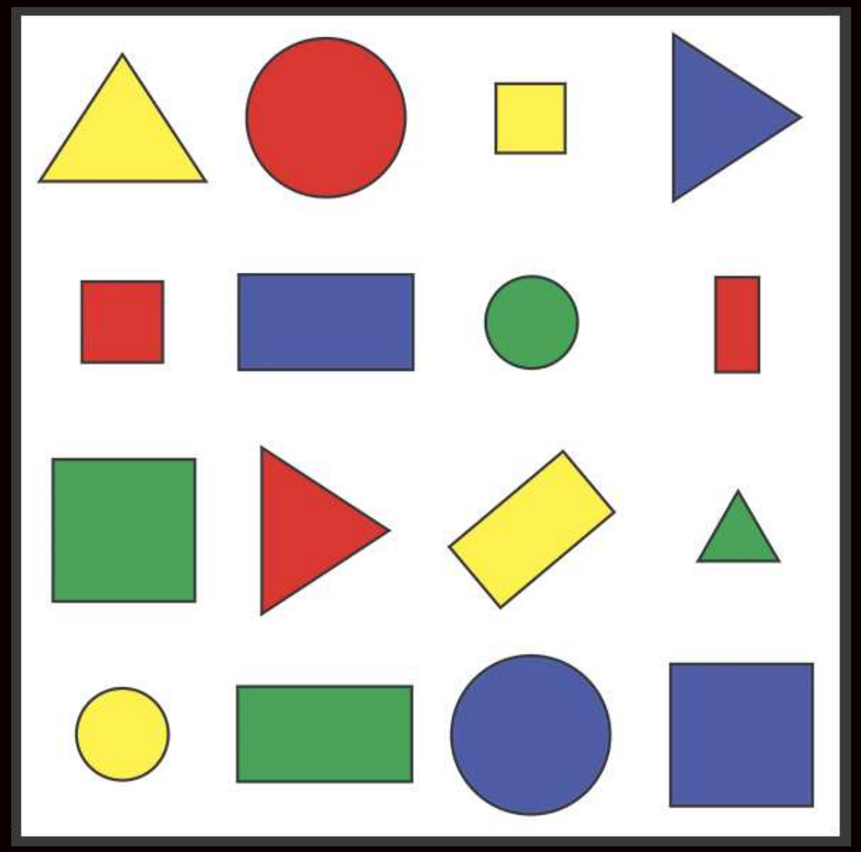
Image Processing and Computer Vision

Mid-term Assignment

Bhavesh Waghela - N01639685

Measuring the size of the objects on a given source using OpenCV

Find the size of the Geometrical objects of different shapes.



```
[ ]  import cv2
     import numpy as np
```

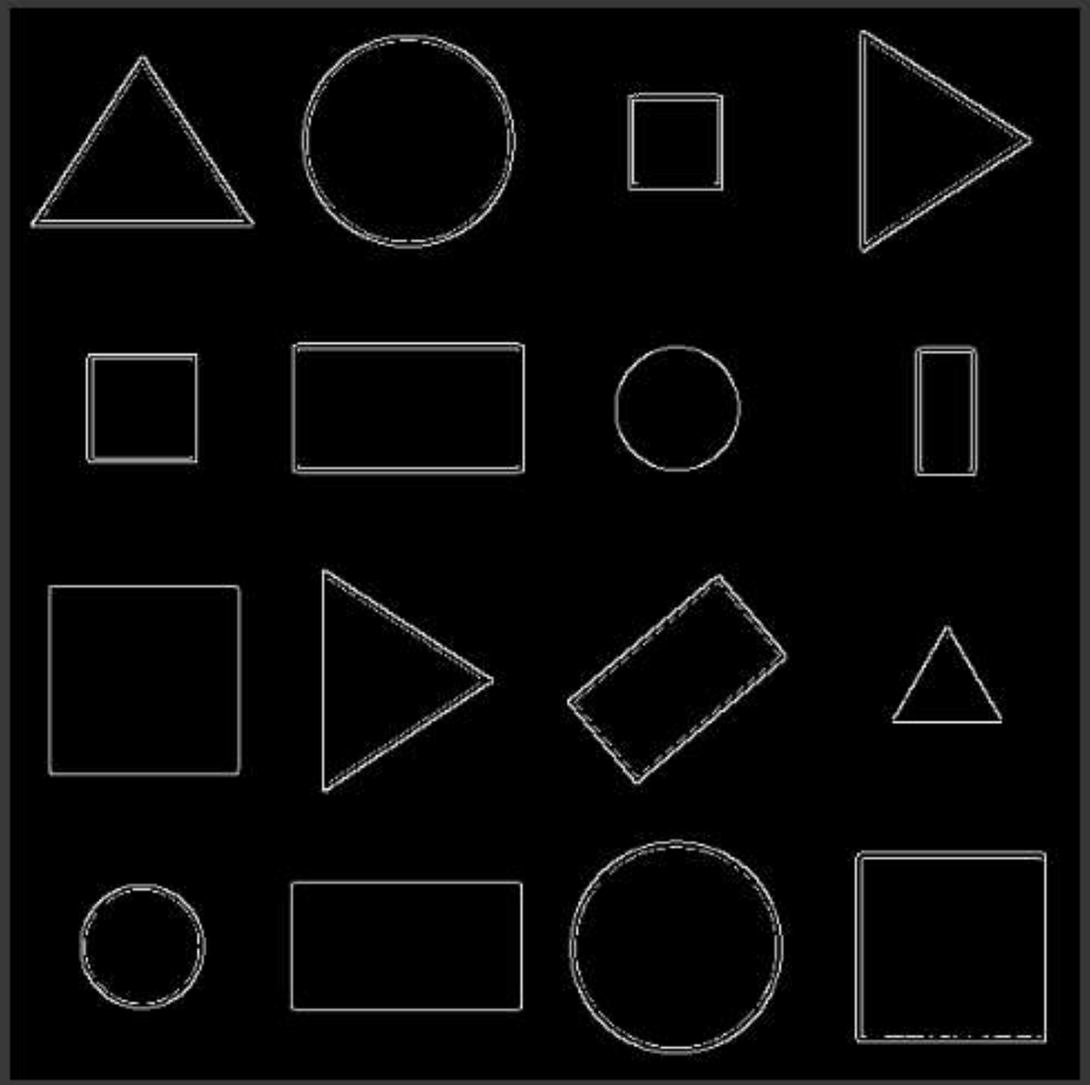
```
[ ]  image = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Sem 2 - IP and CV/Geometrical_Shapes.png')
```

```
[ ]  from google.colab.patches import cv2_imshow
     # Show the result
     cv2_imshow(image)
```

```python
# Apply GaussianBlur to reduce noise and help Canny edge detection
blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Apply Canny edge detection
edges = cv2.Canny(blurred, 50, 150)

# Show the original image and the result
#cv2_imshow(image)
cv2_imshow(edges)
```



Apply Gaussian Blur to reduce the noise.

Apply Canny edge detection.

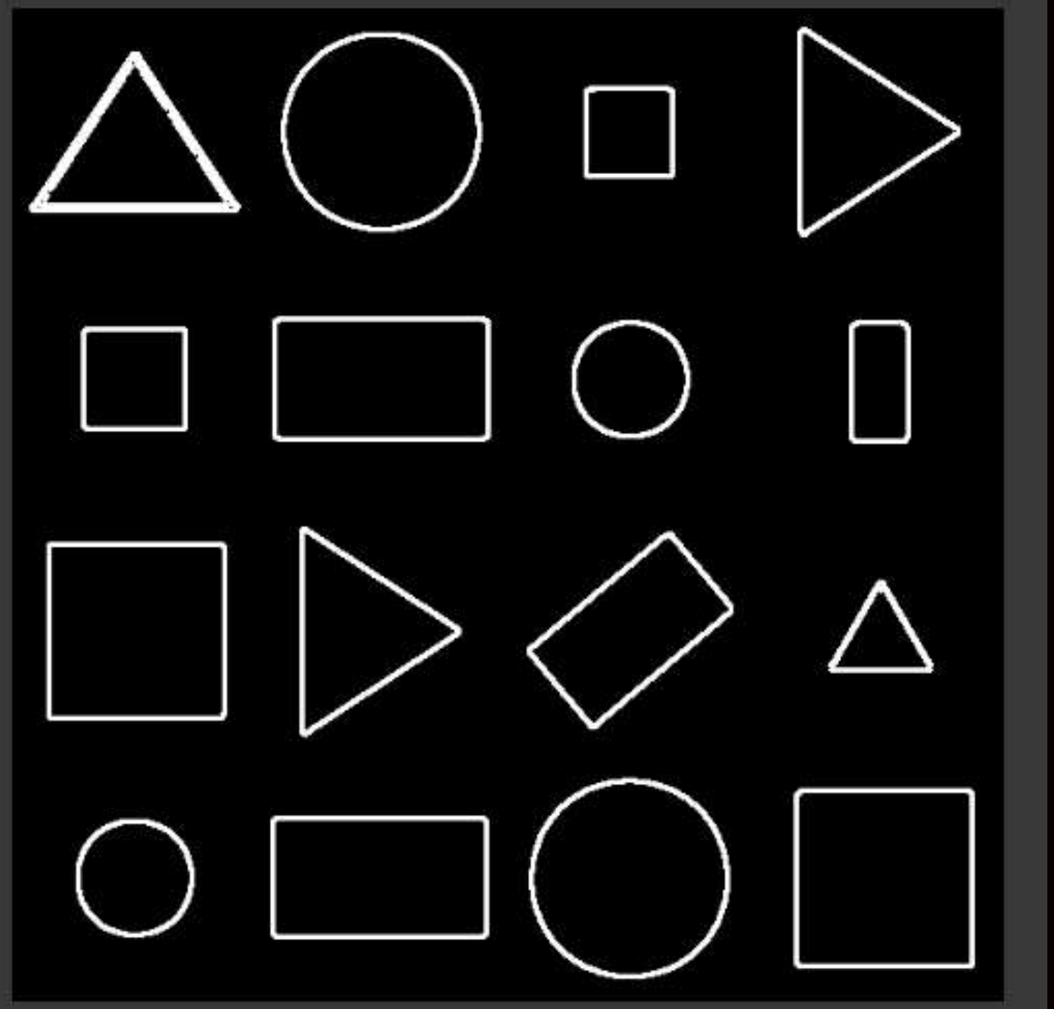
```
[ ] # Find contours in the edge image
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Create a blank image to draw contours on
    contour_image = np.zeros_like(image)

    # Draw contours on the blank image
    cv2.drawContours(contour_image, contours, -1, (255, 255, 255), 2)

    # Show the original image, edges, and contours

    #cv2_imshow(edges)
    cv2_imshow(contour_image)
```



Finding contours in the edge image.

- Create a blank image
- Draw contours on the blank image
- Display the contours image

```python
from tabulate import tabulate

image = cv2.imread('/content/drive/MyDrive/Colab Notebooks/Sem 2 - IP and CV/Geometrical_Shapes.png', cv2.IMREAD_GRAYSCALE)

# Find contours in the edge image
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Create a blank image to draw contours on
contour_image = np.zeros_like(image)

# Draw contours on the blank image
cv2.drawContours(contour_image, contours, -1, (255, 255, 255), 2)

# List to store details for contours with area > 0
details_report = []
```

```python
# Iterate through each contour
for idx, contour in enumerate(contours):
    # Calculate the area of the contour
    area = cv2.contourArea(contour)

    # Check if area is greater than 0
    if area > 0:
        # Calculate the bounding box
        x, y, w, h = cv2.boundingRect(contour)

        # Calculate the minimum enclosing circle
        (center, radius) = cv2.minEnclosingCircle(contour)
        radius = int(radius)

        # Get the number of vertices in the contour
        num_vertices = len(cv2.approxPolyDP(contour, 0.02 * cv2.arcLength(contour, True), True))

        # Calculate circularity for detecting circles
        circularity = 4 * np.pi * area / (cv2.arcLength(contour, True) ** 2)

        # Guess the shape based on the number of vertices and circularity
        if num_vertices == 3:
            shape = "Triangle"
        elif num_vertices == 4 and circularity > 0.7:
            shape = "Square/Rectangle"
        elif num_vertices == 5:
            shape = "Pentagon"
        elif num_vertices == 6:
            shape = "Hexagon"
        elif circularity > 0.8:
            shape = "Circle"
        else:
            shape = "Unknown"

        # Add details to the report
        details_report.append({
            "Contour": idx + 1,
            "Area": area,
            "Bounding Box": {"X": x, "Y": y, "Width": w, "Height": h},
            "Enclosing Circle": {"Center": (int(center[0]), int(center[1])), "Radius": radius},
            "Num Vertices": num_vertices,
            "Shape": shape
        })
```

Final Output Calculating the area of the shape.

Contour	Area	Bounding Box	Enclosing Circle	Num Vertices	Shape
1	3481.5	{'X': 38, 'Y': 472, 'Width': 67, 'Height': 67}	{'Center': (70, 504), 'Radius': 33}	8	Circle
2	8550.0	{'X': 151, 'Y': 470, 'Width': 125, 'Height': 70}	{'Center': (213, 504), 'Radius': 70}	4	Square/Rectangle
3	10397.0	{'X': 455, 'Y': 454, 'Width': 103, 'Height': 103}	{'Center': (506, 505), 'Radius': 71}	4	Square/Rectangle
4	10220.0	{'X': 301, 'Y': 448, 'Width': 115, 'Height': 115}	{'Center': (358, 505), 'Radius': 57}	8	Circle
7	56.0	{'X': 475, 'Y': 333, 'Width': 59, 'Height': 50}	{'Center': (504, 366), 'Radius': 33}	4	Unknown
8	10297.5	{'X': 21, 'Y': 311, 'Width': 103, 'Height': 102}	{'Center': (71, 361), 'Radius': 71}	4	Square/Rectangle
9	6222.5	{'X': 300, 'Y': 305, 'Width': 118, 'Height': 113}	{'Center': (359, 361), 'Radius': 60}	4	Unknown
10	5587.0	{'X': 168, 'Y': 302, 'Width': 92, 'Height': 121}	{'Center': (194, 361), 'Radius': 65}	3	Triangle
11	3419.0	{'X': 41, 'Y': 186, 'Width': 60, 'Height': 59}	{'Center': (71, 215), 'Radius': 41}	4	Square/Rectangle
12	2262.0	{'X': 487, 'Y': 182, 'Width': 34, 'Height': 70}	{'Center': (503, 217), 'Radius': 37}	4	Square/Rectangle
13	3471.5	{'X': 326, 'Y': 182, 'Width': 67, 'Height': 67}	{'Center': (358, 215), 'Radius': 33}	8	Circle
14	8670.0	{'X': 152, 'Y': 180, 'Width': 125, 'Height': 71}	{'Center': (214, 215), 'Radius': 70}	4	Square/Rectangle
58	2544.0	{'X': 333, 'Y': 46, 'Width': 51, 'Height': 52}	{'Center': (358, 72), 'Radius': 35}	4	Square/Rectangle
70	118.5	{'X': 11, 'Y': 26, 'Width': 121, 'Height': 92}	{'Center': (70, 90), 'Radius': 65}	6	Hexagon
71	10154.0	{'X': 157, 'Y': 15, 'Width': 115, 'Height': 114}	{'Center': (214, 71), 'Radius': 57}	8	Circle
72	5699.0	{'X': 457, 'Y': 12, 'Width': 93, 'Height': 120}	{'Center': (484, 71), 'Radius': 64}	3	Triangle