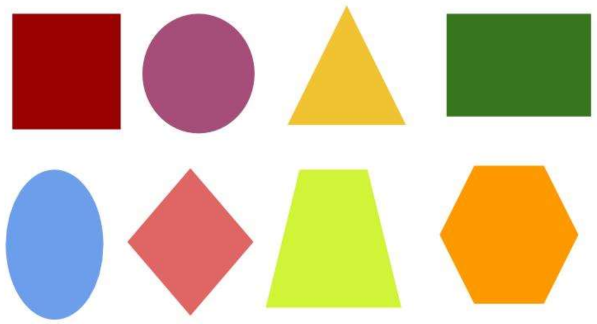
Indian Institute of Technology
Indore

Shape Detection Project for Computer Vision

(CS419 – Autumn Semester 2020)



Submitted by:

Dharmesh Kumar - 170001021
Rahul Anand Yadav - 170001038

Submitted to:

Dr. Surya Prakash Sir – Associate Professor
(Discipline of Computer Science and Engineering)

Introduction to shape detection

There is a good development in the human-machine interface and effective information processing algorithms and powerful microprocessors. It makes it possible to achieve the extra perception of the environment and detect objects with new technologies. The general image process consists of Digitization, Pre-Processing, Image segmentation, Feature Extraction, Classification, and image recognition and interpretation.

Shape detection is an important part of Image Processing, referring to modules that deal with identifying and detecting parts of an image that differ in brightness, colour, or texture. There are several reasons we need to study shape detection, the most important being identifying shapes in images for object recognition or feature detection/extraction.

After blob detection, we perform shape detection, i.e., once we get the blob details, and it is processed to find the shape of the blob for further classification of blobs. Various advanced means of shape detection and research are still going on to find even better algorithms for the same. However, OpenCV has already used one of the most widely used Hough Transform algorithms for shapes like lines, circles, ellipse, and squares.

Algorithm Used – Hough Transform

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The technique's purpose is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in the parameter space. Object candidates are obtained as local maxima in a so-called accumulator space explicitly constructed by the algorithm for computing the Hough transform.

The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described $r = x\cos\theta + y\sin\theta$. The accumulator's dimension equals the number of unknown parameters, i.e., two, considering quantized values of r and θ in the pair (r,θ). For each pixel at (x,y) and its neighbourhood, the Hough transform algorithm determines if there is enough evidence of a straight line at that pixel. If so, it will calculate the parameters (r,θ) of that line, then look for the accumulator's bin that the parameters fall into, and increment that bin's value. By finding the bins with the highest values, typically by looking for local maxima in the accumulator space, the most likely lines can be extracted. Their (approximate) geometric definitions read off. The simplest way of finding these peaks is by applying some form of the threshold, but other techniques may yield better results in different circumstances – determining which lines are found and how many. Since the lines returned do not contain any length information, it is often necessary, in the next step, to find which parts of the image match up with which lines. Moreover, due to imperfection errors in the edge detection step, there will usually be errors in the accumulator space, making it non-trivial to find the appropriate peaks, and thus the appropriate lines.

The linear Hough transform's final result is a two-dimensional array (matrix) similar to the accumulator—one dimension of this matrix is the quantized angle θ, and the other dimension is the quantized distance r. Each element of the matrix has a value equal to the sum of the points or pixels positioned on the line represented by quantized parameters (r, θ). So, the element with the highest value indicates the straight line that is most represented in the input image.

Following is the code implementation (Python)

(Also attached in the zip file)

```
import numpy as np
import cv2

img = cv2.imread('image1.webp')
# img = cv2.resize(img , (720 , 480))
imgGrey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, thrash = cv2.threshold(imgGrey, 240, 255, cv2.THRESH_BINA
RY)
contours, _ = cv2.findContours(thrash, cv2.RETR_TREE, cv2.CH
AIN_APPROX_NONE)


cv2.imshow("img", thrash)
for contour in contours:
    if cv2.contourArea(contour) < 5:
        continue
    approx = cv2.approxPolyDP(
        contour, 0.01 * cv2.arcLength(contour, True), True)
    cv2.drawContours(img, [approx], -1, (0, 0, 0), 2)
    x = approx.ravel()[0]
    y = approx.ravel()[1] - 5
    if len(approx) == 3:
        cv2.putText(img, "Triangle", (x-30, y),
                    cv2.FONT_HERSHEY_COMPLEX, 0.4, (0, 0, 0)
)
    elif len(approx) == 4:
        x1, y1, w, h = cv2.boundingRect(approx)
        aspectRatio = float(w)/h
        # print(aspectRatio)
        if aspectRatio >= 0.94 and aspectRatio <= 1.05:
```

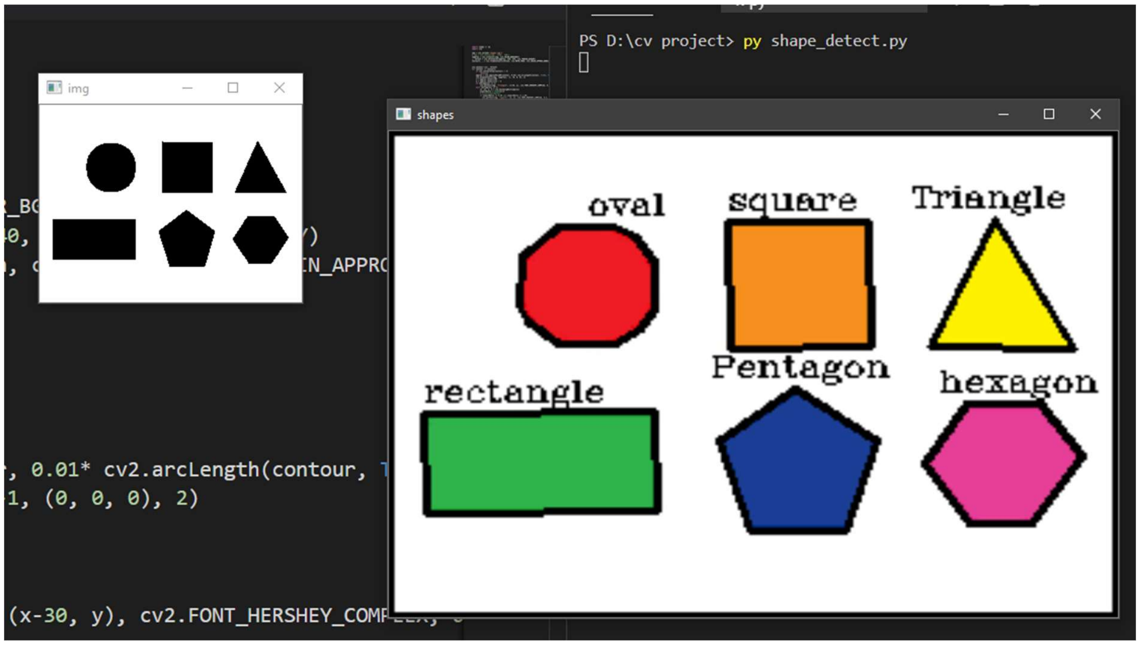
```
      cv2.putText(img, "square", (x, y),
            cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
    else:
      cv2.putText(img, "rectangle", (x, y),
            cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
  elif len(approx) == 5:
    cv2.putText(img, "Pentagon", (x-30, y),
          cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
  elif len(approx) == 6:
    cv2.putText(img, "hexagon", (x-10, y),
          cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
  elif len(approx) == 7:
    cv2.putText(img, "heptagon", (x-10, y),
          cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
  elif 7 < len(approx) < 15:
    cv2.putText(img, "oval", (x+10, y), cv2.FONT_HERSHEY
_COMPLEX, 0.4, (0))
  else:
    cv2.putText(img, "circle", (x-5, y),
          cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))

imgS = cv2.resize(img, (720, 480))
cv2.imshow("shapes", imgS)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

No dataset is used for geometrical shape detection. Hough Transform Algorithm does it.

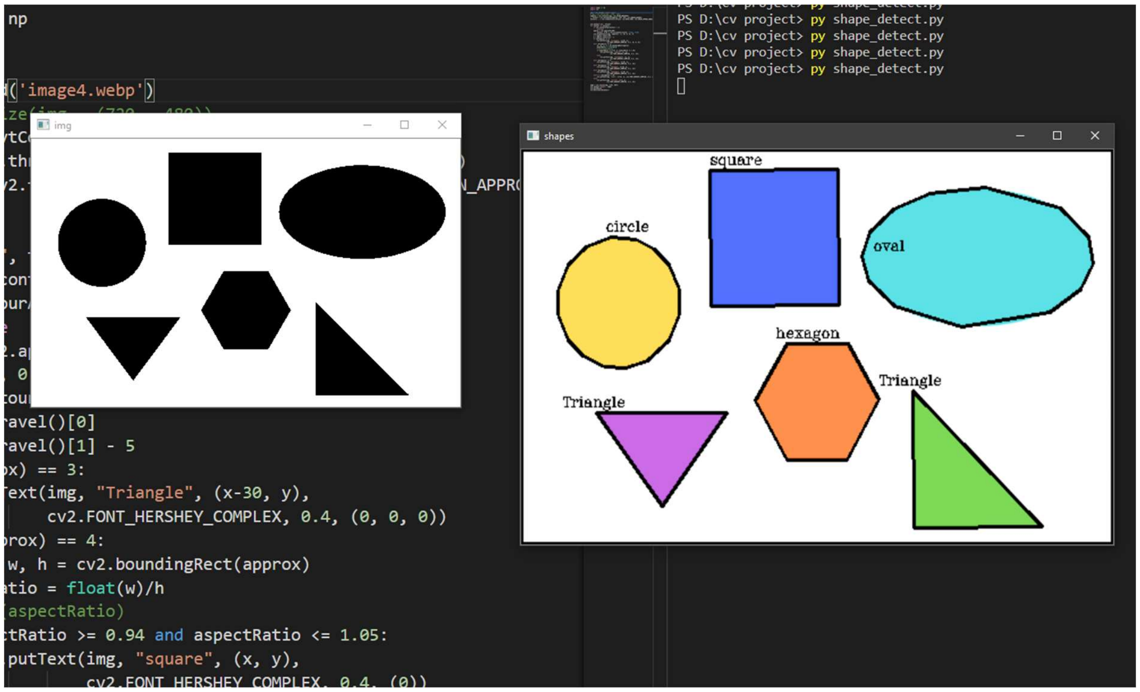
Result, Analysis and Conclusion

The Code executes successfully and detects geometrical shape in images and appropriately labels them.



```
PS D:\cv project> py shape_detect.py
```

```
_BO
0,                                  )
, c                                 N_APPRO
, 0.01* cv2.arcLength(contour, T
1, (0, 0, 0), 2)
(x-30, y), cv2.FONT_HERSHEY_COMP
```



```
PS D:\cv project> py shape_detect.py
PS D:\cv project> py shape_detect.py
PS D:\cv project> py shape_detect.py
PS D:\cv project> py shape_detect.py
```

```
np
('image4.webp')
avel()[0]
avel()[1] - 5
x) == 3:
ext(img, "Triangle", (x-30, y),
    cv2.FONT_HERSHEY_COMPLEX, 0.4, (0, 0, 0))
rox) == 4:
w, h = cv2.boundingRect(approx)
tio = float(w)/h
aspectRatio)
tRatio >= 0.94 and aspectRatio <= 1.05:
putText(img, "square", (x, y),
    cv2.FONT_HERSHEY_COMPLEX, 0.4, (0))
```

References:

1. Computer Vision Lectures - CS419
2. Robotix Tutorial
3. Wikipedia