Identification of shape and size of a figure in a video

By:
Ahmad Nadeem Saigol

Contents

Recognition of Shape of the Figure

The goal of the project is to identify the different shapes placed on the reference paper with different sizes in a video. Furthermore, the size of the shape also needs to be determined. The size of the reference paper was limited to A3, A4 and Letter size papers while the figures were limited to square, circle, triangle, and rectangle. This problem can be solved using number of different approaches but the one I used, is described below:

Working of the Code:

The code has a single function which performs the recognition and determines the sizes of the shapes in the video. It takes three arguments:

- 'ref_paper': This defines the size of the reference paper. It can take one of the three values which maps to the corresponding height and width:
  - 'A3' (420 x 297 mm)
  - 'A4' (297 x 210 mm)
  - 'Letter' (279.4 x 215.9 mm)
- 'video': This defines the path to the input video, on which detection will be performed.
- 'resize': This takes a tuple of ints (width, height). The input frames are resized to these values. Further, the size of the output frames is also defined by these values.

The video that was used for testing had a reference paper of A4 size. It had dimension of (1080 x 1816), format of '.mov' and frame rate of 30fps. Each input frame was resized to (540 x 908) for faster computation. Further these frames, after processing were saved in '.avi' format at 30fps.

- The code reads each frame one by one in uint8 format.
- Each frame is resized to specified shape
- The frame is converted to grayscale image using the formula:

$$Y = 0.299 * R + 0.587 * G + 0.144 * B$$

- Noise is removed using Gaussian Filter of shape (7 x 7)
- Canny Edge detector is applied to obtain edges in the frame by setting the two thresholds as 50 and 100.
- the image is then dilated and followed by erosion using matrix of shape (3 x 3)
- Contours are found from the resulting images using the retrieval mode 'RETR_COMP' and contour approximation method 'CHAIN_APPROX_SIMPLE'. In this mode, the contours are organized into two levels: first level corresponds to external boundaries and second level corresponds to boundaries of the holes. In this method, only the end points are approximated and returned.
- For each contour retrieved, four points of the rectangle with minimum area, which is required to enclose it, are obtained.
- The perimeter of this box is calculated. And also, the length of each side is calculated using the distance formula:

$$c = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

  - If the perimeter is less than 150, the contour is ignored.

- If it is more than 2000, this contour is considered as the reference paper. It is represented in the video with green color and its dim (in mm) and heading in red color. The box coordinates obtained earlier starts from lowest value in x direction i.e. it could either be bottom left or bottom right. Compensating for these two cases and calculating pixels per mm value and storing it for future use. (Since we get two slightly different values from length and width, we use the average value). A flag is also set to true. This makes sure that reference paper has been detected. In case, it is not, the code will exit with an error.
- If it is less than 2000, it estimates a polygonal curve for the contour. Then it calculates the centroid of the contour using moments. Just like the previous case, it adjusts for both cases and calculates the height and width of the box. Afterwards, it identifies and determine the size and shape of the figures:
  - It calculates the perimeter of the box and of the approximated polygon and compares the two (takes difference between the two and compares with threshold). If its value is less than 20, then it must be a square or rectangle:
    - If the difference between calculated length and width is small (threshold = 10), then it is a square else it is rectangle.
  - If its value is more than 20, then it checks whether it is a triangle or circle:
    - The difference between the ratio (between contour area and perimeter of approximated polygon) and half of the radius (average of length and width) is calculated. If it is between -5 and 5, it is circle otherwise it is a triangle.

$$\frac{A}{P} = \frac{\pi r^2}{2\pi \mathrm{r}} = \frac{r}{2}$$

For each case, area is calculated, and corresponding lengths/widths/radius are calculated. These are shown in frame in red while contours are shown in blue.

Few Important Points Regarding Code:

- The code was written in Google Colab in python environment using OpenCV. (.ipynb is provided).
- As the shapes in the frame were not perfect, more than one value was obtained for a dimension. For example, a square has four equal sides but in case of the image, all sides were not equal. In such cases, average values were used for the calculation.
- As mentioned before, shapes were not perfect, thresholds were used to approximate the shape. These were obtained by hit and trial error.
- The code is highly dependent on the illumination in the video and sharp contrast between reference paper and background and between figures and reference paper. Thus, it is recommended to use a video in which most of the frame is covered by the reference paper and outside it, must have non-textual/plain contrasting background. Further, the shapes must have a sharp contrast with the reference paper and also definite edges. Moreover, the video must be made in a way that camera/mobile is completely parallel to the reference paper (on the top of it) and there are no shadows and reflections on the paper.
- The code creates a video file which can be used for viewing the results.

Results:

A4 white paper with four different shapes on it, was placed on a black background and The video was recorded while rotating the camera just enough, so that anything apart from black background isn't captured.

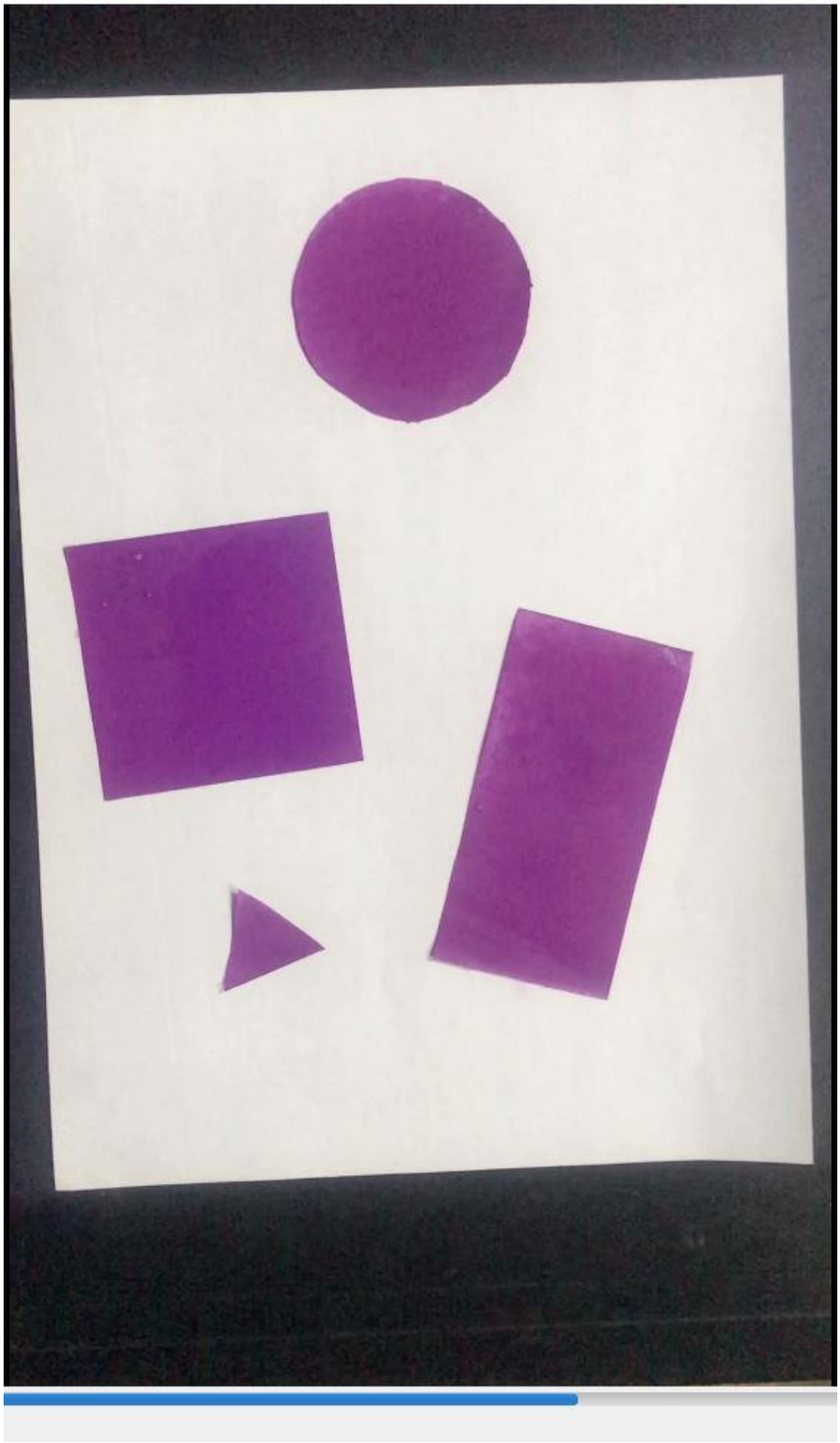
The following table shows the comparison between original and calculated values.

Shape	***Area (mm^2)***		***Length (mm)***		***Width (mm)***		***Radius (mm)***	
	Original	Calculated	Original	Calculated	Original	Calculated	Original	Calculated
Circle	3318.31	3268.8					32.5	32.3
Rectangle	5000	4983.1	100	99.1	50	50.3		
Square	4900	4938.7	70	70.3	70	70.3		
Equilateral Triangle	389.71	369.5	30					

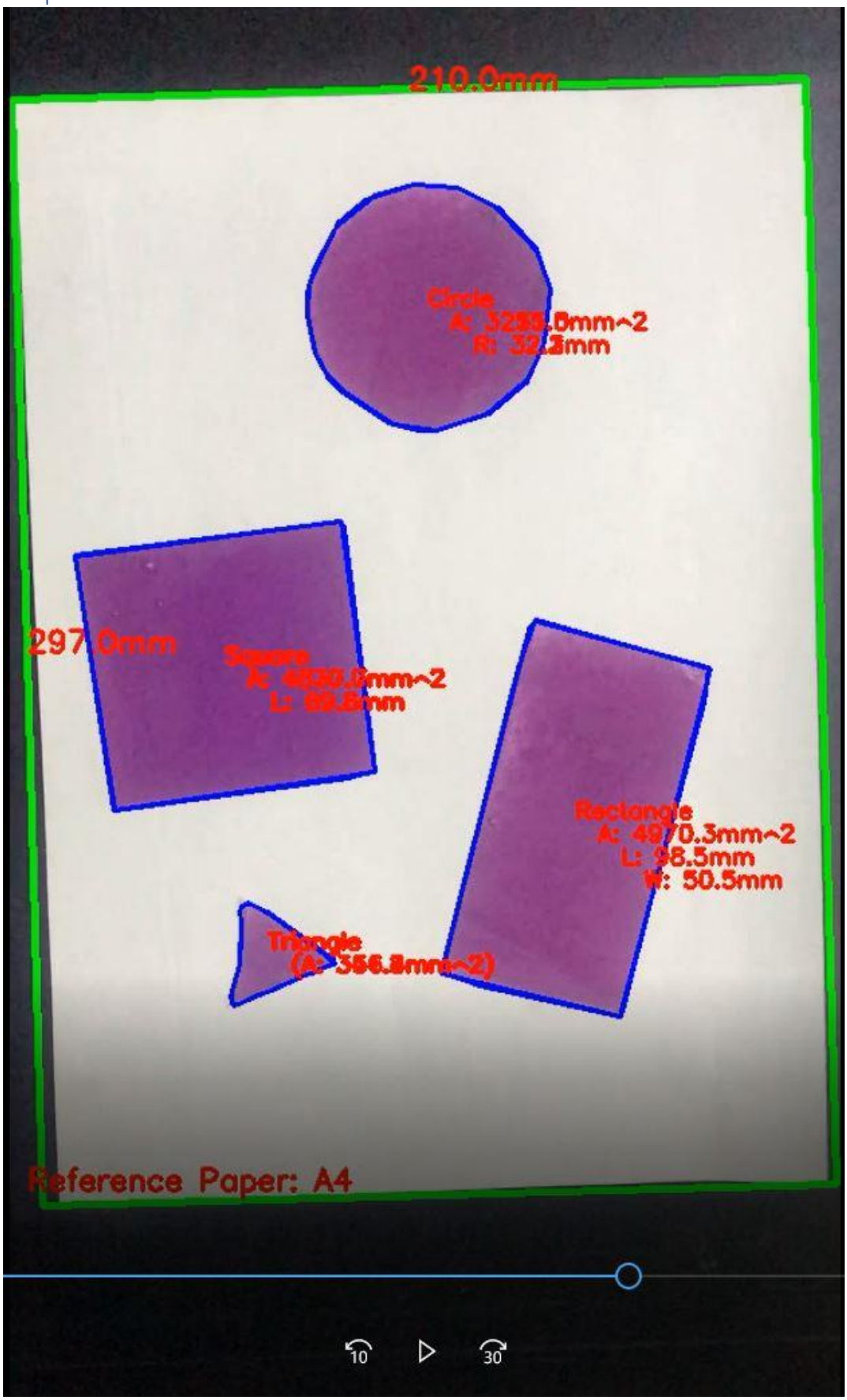
It can be seen that the difference between the two values is very small, suggesting the correctness of the code.

Snippets at time 5s is shown below:

Input:



Output:


210.0mm
Circle
297.0mm
Square
Rectangle
A: 4970.3mm^2
L: 98.5mm
W: 50.5mm
Triangle
Reference Paper: A4
10
30

Code:

```
import cv2
from google.colab.patches import cv2_imshow
import sys
import numpy as np

def shapeDetector (ref_paper, video, resize):
  '''
  Detects and determines the size and shape of the object in the video.
  Also returns the modified image

  Parameters:
    'ref_paper' : 'A3', 'A4' or 'Letter'
      size of the reference paper
    'video': str
      path to the video
    'resize': tuple of integers (Width, Height)
      resizes each frame to specified values. In case, resizing is not
required then set this paramter to None

  '''
  #set dim of the reference paper
  if ref_paper == 'A3':
    width = 297
    height = 420
  elif ref_paper == 'A4':
    width = 210
    height = 297
  elif ref_paper == 'Letter':
    width = 215.9
    height = 279.4
  else:
    sys.exit("Invalid Value provided for reference paper")

  print('The set dim of the reference paper is (WxH): (', width, 'mm X
', height, 'mm )')

  flag = True

  #for reading frames in video
  cap = cv2.VideoCapture(video)

  if (cap.isOpened()== False):
    sys.exit("Error opening video stream or file")


  #for storing video
```

```
  out = cv2.VideoWriter('output.avi', cv2.VideoWriter_fourcc('M','J','P
','G'), 30, resize)

  while(True):

    ret, frame = cap.read()

    if ret == True:

      #Resizing
      frame =cv2.resize(frame, resize)

      #convert to grayscale
      img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

      #Apply gaussian blur
      img = cv2.GaussianBlur(img, (7, 7), 0)

      #Find edges using edge detector
      img = cv2.Canny(img, 50, 100)

      #Perform erosion and dilataion
      img = cv2.dilate(img, None, iterations=1)
      img = cv2.erode(img, None, iterations=1)

      #Find Contours

      contour,_ = cv2.findContours(img, cv2.RETR_CCOMP, cv2.CHAIN_APPRO
X_SIMPLE)

      orig = frame.copy()

      ppm = 0

      for c in reversed(contour):

        #Find min Rectangle coordinates around the contour
        rect = cv2.minAreaRect(c)
        box = cv2.boxPoints(rect)
        box = np.array(box, dtype="int")

        #Calculate perimeter of bounding box
        peri = cv2.arcLength(box,True)

        #ignore conoturs with perimeter less than threshold
        if peri < 150:
          continue
```

```
        #Calculate length of each side of bounding box of shape
        temp = box[[3,0,1,2]]
        dist = np.sqrt(np.sum(np.square(box -
 temp), axis=1, keepdims=True))

        #Bounding Box for Reference Paper
        if (peri > 2000):

          #For mapping between mm and pixels

          #if vertex of paper is right-bottom
          if abs(box [0][0]-box[0][1]) < (box[0,1]/2):

            w= ((dist[1]+dist[3])/2) / width
            l= ((dist[0]+dist[2])/2) / height

            midpoint = (temp + box)//2
            midpoint [[1,2]] = midpoint [[2,1]]

            po = box[1]

          #if vertex of paper  is left bottom
          else:
            w= ((dist[0]+dist[2])/2) / width
            l= ((dist[1]+dist[3])/2) / height

            midpoint = (temp + box)//2
            po = box[0]

          ppm = (l + w) / 2 #using average values for pixels per mm

          #Draw Contours and display associated properties
          cv2.drawContours(orig, [box], -1, (0, 255, 0), 3)
          cv2.putText(orig, "{:.1f}mm".format(height), tuple(midpoint[1
]), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0,0, 255), 2)
          cv2.putText(orig, "{:.1f}mm".format(width), tuple(midpoint[2]
), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)
          cv2.putText(orig, "Reference Paper: {}".format(ref_paper), tu
ple(po-10), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)

          flag =True

        else:

          #Check whether reference page is detected
          if not flag:
            sys.exit("Reference Page not detected")
```

```
            #Find approx figure
            epsilon = 0.003 * peri
            approx = cv2.approxPolyDP(c, epsilon, True)

            #Calculate moments
            mc =cv2.moments(box)
            mc2=cv2.moments(approx)

            #Calculate centroids
            mt = np.array((mc['m10'] / (mc['m00'] + 1e-
5), mc['m01'] / (mc['m00'] + 1e-5)))
            mt2 = np.array((mc2['m10'] / (mc2['m00'] + 1e-
5), mc2['m01'] / (mc2['m00'] + 1e-5)))

            center = ((mt + mt2)//2).astype("int")

            cv2.drawContours(orig, [approx], -1, (255,0,0), 2)

            #if vertex of paper is right-bottom
            if abs(box [0][0]-box[0][1]) < (box[0,1]/2):

              w= ((dist[1]+dist[3])/2)
              l= ((dist[0]+dist[2])/2)

            #if vertex of paper  is left bottom
            else:
              w= ((dist[0]+dist[2])/2)
              l= ((dist[1]+dist[3])/2)


            #Check For Square and Rectangle
            if abs(cv2.arcLength(box, True) -
 cv2.arcLength(approx, True)) <20:

              #For Square
              if abs(l-w) < 10:

                cv2.putText(orig, "Square", tuple(center), cv2.FONT_HERSH
EY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "A: {:.1f}mm^2".format(float((l*l)/((pp
m*ppm)+ 1e-
5))), tuple(center+15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "L: {:.1f}mm".format(float(l/(ppm + 1e-
5))), tuple(center+30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

              #For Rectangle
              else:
```

```
                cv2.putText(orig, "Rectangle", tuple(center), cv2.FONT_HE
RSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "A: {:.1f}mm^2".format(float((l*w)/((pp
m*ppm)+ 1e-
5))), tuple(center+15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "L: {:.1f}mm".format(float(l/(ppm+ 1e-
5))), tuple(center+30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "W: {:.1f}mm".format(float(w/(ppm+ 1e-
5))), tuple(center+45), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

            #Else Circle and Triangle
            else:
              r = ((l+w)/2)/2 #using average value for radius

              ratio =abs(cv2.contourArea(approx, True))/(cv2.arcLength(ap
prox, True)+ 1e-5)
              if ((ratio - (r/2)) < 5) and ((ratio - (r/2)) > -5):

                cv2.putText(orig, "Circle",tuple(center), cv2.FONT_HERSHE
Y_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "A: {:.1f}mm^2".format(float(np.pi*np.s
quare(r)/((ppm*ppm)+ 1e-
5))),tuple(center+15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "R: {:.1f}mm".format(float(r/(ppm + 1e-
5))), tuple(center+30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

              else:
                cv2.putText(orig, "Triangle", tuple(center), cv2.FONT_HER
SHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                cv2.putText(orig, "(A: {:.1f}mm^2)".format(float((l*w)/((
2*ppm*ppm)+ 1e-
5))), tuple(center+15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)


        # Write the frame into the file 'output.avi'
        out.write(orig)

      else:
        break

  # release the video capture and video write objects
  cap.release()
  out.release()
  return orig

_ = shapeDetector('A4', '/content/input.mov', (540,908))
```