Marker based localization

Shape Detection

Team members

Niharika Jayanthi Dheeraj Kamath

Under the guidance of
Sanam Shakya

Goal

In this chapter we will see,
* how to find various properties of contours
* how to apply these properties
* we will learn about : **cv2.contourArea()**, **cv2.arcLength()**, **cv2.approxPolyDP()**

Theory

Contours have various features like area, perimeter, moments etc which can be used for various applications.

Working principle

1.**Contour Area** - The area of the object is found with the help of moments. Area is calculated by zero order moment. For more details on moments refer: Moments

```
cv2.contourArea(contour, oriented)
```

Parameters:
* **contour** – Input vector of 2D points (contour vertices)
* **oriented** – Oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.

2.**Contour Perimeter** - Used to find the arc length of the contour.
`cv2.arcLength(curve, closed)`

Parameters:
* **curve** - Input vector of 2D points
* **closed** - Flag indicating whether the curve is closed or not

3.**ApproxPolyDP()** - This is used to obtain the number of points found in the figure. `epsilon` is an accuracy parameter. A wise selection of epsilon is needed to get the correct output.

```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(curve, epsilon, closed, approxCurve)
```

Parameters:
* **curve** – Input vector of a 2D point.
* **approxCurve** – Result of the approximation. The type should match the type of the input curve. In case of C interface the approximated curve is stored in the memory storage and pointer to it is returned. epsilon – Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation.
* **closed** – If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

Shape Detection

Using all these properties we will see how to detect shapes and display the name of the shape at its center. All the shapes can be first identified by the number of vertices and then for shapes with the shapes with the same number of vertices we can use Hu moments based matchShapes() function. We will be trying to detect the following shapes:
`Triangle, Square, Rectangle, Parallelogram, Pentagon, Hexagon, Star and Circle.`
Since there are three shapes having four points we will compare the return value of each of the shapes with the square. The output of `cv2.matchShapes()` is a return value indicating the amount of deviation from sample image. If ret $= 0$ its a square, if $0 <$ ret < 0.5 its a rectangle and if ret > 0.5 its a parallelogram. These values were obtained by trial and error basis.

Code

```
#Importing modules
import cv2
import numpy
import matplotlib.pyplot as plt

#Sample image for shapes with 4 vertices
img = cv2.imread('square.png')

#Reading the image from the user
inp_img = cv2.imread(raw_input("Enter the name of the unknown image:"))

#Converting to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray1 = cv2.cvtColor(inp_img, cv2.COLOR_BGR2GRAY)

'''
OpenCV represents RGB images as multi-dimensional NumPy arrays...but in reverse
order!This means that images are actually represented in BGR order rather than
RGB!
'''
#Convert to RGB
inp_img = cv2.cvtColor(inp_img, cv2.COLOR_BGR2RGB)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


#Thresholding
ret, thresh = cv2.threshold(gray, 0,255,0)
ret, thresh1 = cv2.threshold(gray1, 0,255,0)

#Using Canny for perfect edge detection
canny = cv2.Canny(img,100,200)

#Finding contours
contours,heirarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnt1 = contours[0]
contours1,heirarchy = cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnt2 = contours1[0]


#Finding the centroid
for s in contours1:
    M = cv2.moments(s)
    cx = int(M['m10']/M['m00'])
    cy = int(M['m01']/M['m00'])


#####################Detection of shape####################################
font = cv2.FONT_HERSHEY_SIMPLEX

#Creating lists
a = []
b = []

#Finding vertices in input image
for i in contours1:
    approx = cv2.approxPolyDP(i,0.01*cv2.arcLength(i,True),True)
    print len(approx)
    x = len(approx)
    a.append(x)
print a

#Finding vertices in sample image
for i in contours:
    approx = cv2.approxPolyDP(i,0.01*cv2.arcLength(i,True),True)
    print len(approx)
    x = len(approx)
    b.append(x)
print b

 #Detection and display of name of shape
 for i in contours1:
    for m in a:
        for n in b:
             if m == 4 & n ==4:
                 ret = cv2.matchShapes(cnt1,cnt2,1,0.0)
                 print ret

                 if ret> 0.5:
                    print "Parallelogram"
                    cv2.drawContours(img,[i],0,(0,255,0),2)
                    cv2.putText(img,"Slight match",(50,50),font,1,(0,255,0),2)
                    cv2.putText(inp_img,"Parallelogram",(cx-20,cy),font,0.5,(0,0,255),2)
                    cv2.imshow('Compare',img)
                elif 0.3< ret < 0.5:
                    print "Rectangle"
                    cv2.drawContours(img,[i],0,(0,255,0),2)
                    cv2.putText(img,"Slight Match",(50,50),font,0.5,(0,0,255),2)
                    cv2.putText(inp_img,"Rectangle",(cx-20,cy),font,0.5,(0,0,255),2)
                    cv2.imshow('Compare',img)
```

```
            elif 0 < ret < 0.3:
                print "Rhombus"
                cv2.drawContours(img,[i],0,(0,255,0),2)
                cv2.putText(img,"Slight Match",(50,50),font,0.5,(0,0,255),2)
                cv2.putText(inp_img,"Rhombus",(cx-20,cy),font,0.5,(0,0,255),2)
                cv2.imshow('Compare',img)
            else:
                print "Square"
                cv2.drawContours(img,[i],0,(0,255,0),2)
                cv2.putText(img,"Perfect Match",(50,50),font,2,(0,0,255),2)
                cv2.putText(inp_img,"Square",(cx-25,cy),font,1,(0,0,255),2)
                cv2.imshow('Compare',img)




         elif m ==3:
            cv2.drawContours(inp_img,[i],0,(0,255,0),2)
            print "Input image is a triangle"
            cv2.putText(inp_img,"Triangle",(cx-20,cy),font,0.5,(0,0,255),2)


        elif m ==5:
            print "pentagon"
            cv2.drawContours(inp_img,[i],0,(0,255,0),2)
            cv2.putText(inp_img,"Pentagon",(cx-20,cy),font,0.5,(0,0,255),2)

        elif m == 6:
            print "hexagon"
            cv2.drawContours(inp_img,[i],0,(0,255,0),2)
            cv2.putText(inp_img,"Hexagon",(cx-20,cy),font,0.5,(0,0,255),2)


        elif m == 7:
            print "Arrow"
            cv2.drawContours(inp_img,[i],0,(0,255,0),2)
            cv2.putText(img,"Arrow",(cx-20,cy),font,0.5,(0,0,255),2)

        elif m > 7:
            print "Circle"
            cv2.drawContours(inp_img,[i],0,(0,255,0),2)
            cv2.putText(inp_img,"Circle",(cx-20,cy),font,0.5,(0,255,0),2)


#Plotting the image
plt.subplot(121),plt.imshow(img), plt.title('Original image'),plt.set_cmap('bone')
```

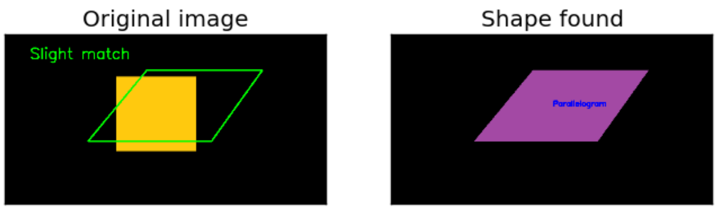
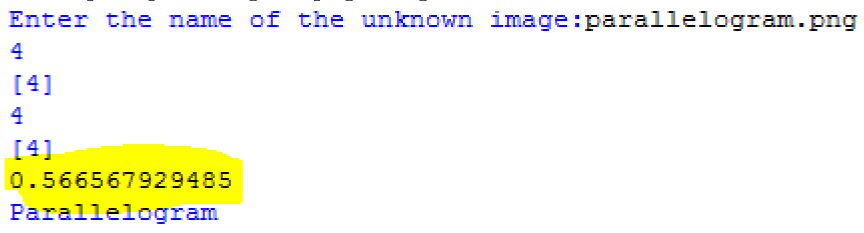


Figure 1: Shape detection

```
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(inp_img),plt.title('Shape found'),plt.set_cmap('bone')
plt.xticks([]), plt.yticks([])
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

If we input “parallelogram.png” we get:



```
Enter the name of the unknown image:parallelogram.png
4
[4]
4
[4]
0.566567929485
Parallelogram
```

The highlighted portion represents the return value.

We get the following as output:

Resources

1. Contour features
2. Contour Properties