OpenCV Experiments

Robotics Lab I Lab Work

B.Tech in Mechatronics

Submitted by

Ankit Aggarwal

(Reg. No. – 200929036)
Batch: A1

Department of Mechatronics

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

December 2021

Contents

Experiment 1		**Colour Detection using OpenCV**	**3-5**
	1.1	Aim	3
	1.2	Problem Statement	3
	1.3	Theory	3
	1.4	Procedure	3
	1.5	Code	3-5
	1.6	Results	5
	1.7	Conclusion	5
Experiment 2		**Shape Detection using OpenCV**	**6-8**
	2.1	Aim	6
	2.2	Problem Statement	6
	2.3	Theory	6
	2.4	Procedure	6
	2.5	Code	6-7
	2.6	Results	8
	2.7	Conclusion	8
Experiment 3		**Biscuit Quality Detection using OpenCV**	**9-11**
	3.1	Aim	9
	3.2	Problem Statement	9
	3.3	Theory	9
	3.4	Procedure	9
	3.5	Code	9-10
	3.6	Results	11
	3.7	Conclusion	11
REFERENCES			**12**

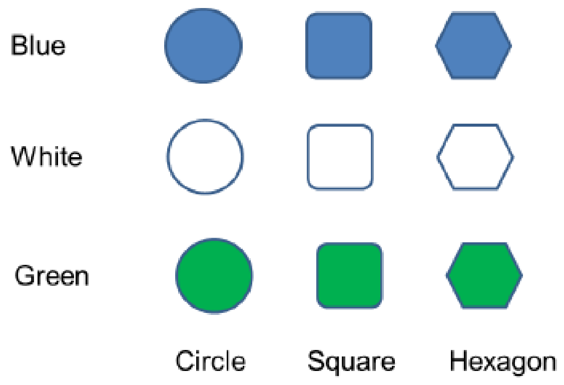
Experiment 1 - Colour Detection using OpenCV

Aim

To identify and classify objects based on colour using HSV values.

Problem Statement

Classify the following images based on colour



Theory

Colour detection is performed based on the HSV Colour Scale. The HSV (Hue Saturation Value) scale provides a numerical readout of an image that corresponds to the colour names contained therein.

The Computer Vision library has dedicated functions such as cv2.cvtColor for colour detection. These functions are used in conjunction with NumPy to accurately perform colour detection of the given images.

Procedure

- Read the given image using cv2.read and show it in a window titled 'image' using cv2.imshow.
- Scale the image as required and find its dimensions.
- Resize the image to its new dimensions using cv2.resize.
- Define a mouse event to return HSV values upon mouse click.
- Use the returned values to define ranges for the given colours.
- Set a reference area in each image to assess colour.
- Average out HSV values of all pixels in the given reference area using NumPy.
- Compare with the previously set ranges for each colour.
- Print the colour onto the image using cv2.putText.

Code

```
import cv2
print(cv2.__version__)

import cv2
import numpy as np

image=cv2.imread('C:/Users/MAHE/Documents/OpenCV_200929036/GS.png')
cv2.imshow("image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

scale_percent = 100
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
dim = (width, height)
print(dim)
```

```
# resize image
img = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
cv2.imshow("image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

def mouse(event,x,y,flags,param):
    if event==cv2.EVENT_LBUTTONDOWN:
        h=hsv[y,x,0]
        s=hsv[y,x,1]
        v=hsv[y,x,2]
        print("H:",h)
        print("S:",s)
        print("V:",v)

cv2.namedWindow('mouse')
cv2.setMouseCallback('mouse',mouse)
hsv=cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
cv2.imshow("mouse", hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()

blueLower=(100,180,245)
blueUpper=(110,220,255)
greenLower=(64,60,190)
greenUpper=(74,70,210)
whiteLower=(0,0,245)
whiteUpper=(10,10,255)

refPt=[(width//2-25, height//2-25),(width//2+25, height//2+25)]
roi = img[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
cv2.imshow("ROI", roi)
cv2.waitKey(0)
cv2.destroyAllWindows()

hsv=cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
havg= np.average(hsv, axis=0)
h_avg = np.average(havg, axis=0)
print(h_avg)

x = img.ravel()[0]
y = img.ravel()[1]

if (h_avg[0]>=np.array(blueLower[0])) and (h_avg[0]<=np.array(blueUpper[0])) and
(h_avg[1]>=np.array(blueLower[1])) and (h_avg[1]<=np.array(blueUpper[1])) and
(h_avg[2]>=np.array(blueLower[2])) and (h_avg[2]<=np.array(blueUpper[2])):
    print("Color is Blue")
    cv2.putText(img, "Blue", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))

elif (h_avg[0]>=np.array(greenLower[0])) and (h_avg[0]<=np.array(greenUpper[0])) and
(h_avg[1]>=np.array(greenLower[1])) and (h_avg[1]<=np.array(greenUpper[1])) and
(h_avg[2]>=np.array(greenLower[2])) and (h_avg[2]<=np.array(greenUpper[2])):
    print("Color is Green")
    cv2.putText(img, "Green", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))

elif (h_avg[0]>=np.array(whiteLower[0])) and (h_avg[0]<=np.array(whiteUpper[0])) and
(h_avg[1]>=np.array(whiteLower[1])) and (h_avg[1]<=np.array(whiteUpper[1])) and
(h_avg[2]>=np.array(whiteLower[2])) and (h_avg[2]<=np.array(whiteUpper[2])):
    print("Color is White")
    cv2.putText(img, "White", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
else:
    print("Color is not Blue, Green or White.")

cv2.imshow('colours', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results

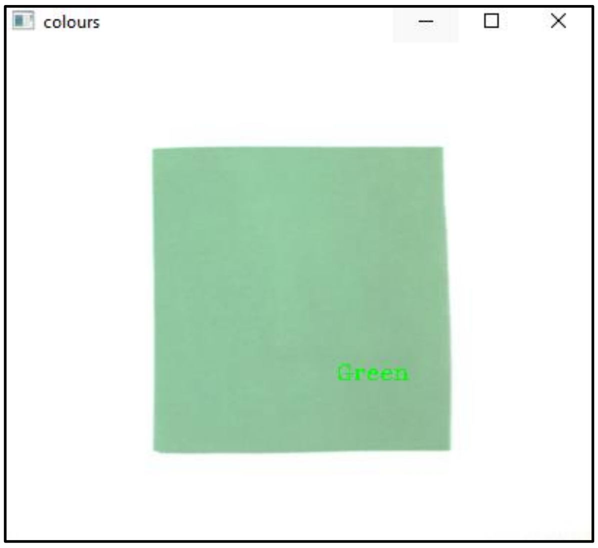


Figure 1 - Image detected as Green

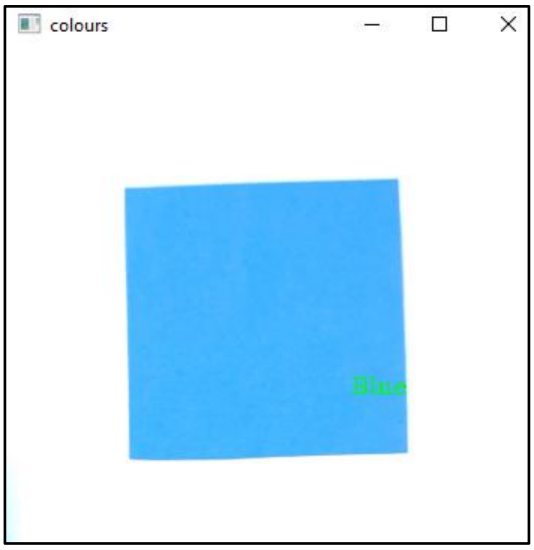


Figure 2 - Image detected as Blue

Conclusion

The given images were classified based on their colour. The name of the colour was also printed on the image.

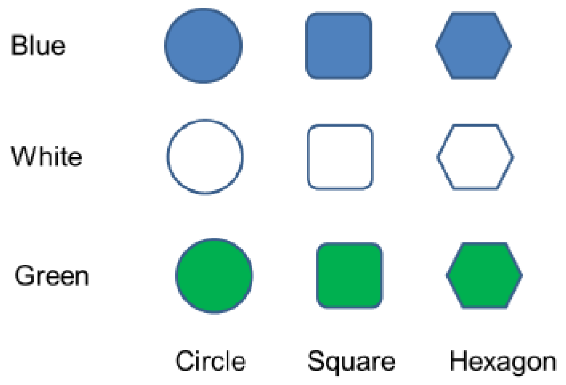
Experiment 2 - Shape Detection using OpenCV

Aim

To identify and classify objects based on shape using suitable algorithms.

Problem Statement

Classify the following images based on shape



Theory

Shape Detection is performed based on edge and contour detection. The image is read as a matrix and edges are detected based on points of high magnitude change in saturation values. This process is known as thresholding. The edges are then checked for continuity, to detect edges and corners of a shape.

The Computer Vision Library has dedicated functions such as cv2.thresholding for shape detection. These functions are used in conjunction with other functions like cv2.GaussianBlur and cv2.cvtColor for maximizing accuracy in shape detection.

Procedure

- Read the given image using cv2.read and show it in a window titled ‘image’ using cv2.imshow.
- To improve accuracy of edge detection the following steps are taken:
    - Blur the image using Gaussian Blur [cv2.GaussianBlur]
    - Convert the image into grayscale [cv2.cvtColor]
- Perform edge detection using thresholding [cv2.threshold]
- Find contours from the detected edges [cv2.findContours]
- For each of the contours determined in the image, determine the shape of the polygons present in the image [cv2.approxPolyDP]
- Draw the determined image onto the input image [cv2.drawContours]
- Based on the detected contours, count number of edges and classify into a shape.
- Print the name of the shape onto the image [cv2.putText]

Code

```
import cv2
print(cv2.__version__)

import numpy as np
import cv2
img = cv2.imread('C:/Users/MAHE/Documents/OpenCV_200929036/YC.png')
cv2.imshow("img",img)
height, width = img.shape[:2]
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Blur=cv2.GaussianBlur(img, (1,1), 0)
cv2.imshow("Blur",Blur)
cv2.waitKey(0)
cv2.destroyAllWindows()

Gray=cv2.cvtColor(Blur,cv2.COLOR_BGR2GRAY)
cv2.imshow("Gray",Gray)
cv2.waitKey(0)
cv2.destroyAllWindows()

ret,thresh=cv2.threshold(Gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
cv2.imshow('thresh', thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()

contours , hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
print(len(contours))

for contour in contours:
    approx = cv2.approxPolyDP(contour, 0.01* cv2.arcLength(contour, True), True)
    cv2.drawContours(img, [approx], 0, (255, 0, 0), 2)
    x = approx.ravel()[0]
    y = approx.ravel()[1]
    edge = str(len(approx))
    if len(approx) == 3:
        cv2.putText(img, "Triangle", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
        cv2.putText(img, edge, (x-60, y-60), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
    elif len(approx) == 4:
        x, y , w, h = cv2.boundingRect(approx)
        aspectRatio = float(w)/h
        print(aspectRatio)
        if aspectRatio >= 0.95 and aspectRatio <= 1.05:
            cv2.putText(img, "square", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
            cv2.putText(img, edge, (x-60, y-60), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
        else:
            cv2.putText(img, "rectangle", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
            cv2.putText(img, edge, (x-60, y-60), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
    elif len(approx) == 5:
        cv2.putText(img, "pentagon", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
        cv2.putText(img, edge, (x-60, y-60), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
    else:
        cv2.putText(img, "circle", (x-25, y-25), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))
        cv2.putText(img, edge, (x-60, y-60), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0))

cv2.imshow('shapes', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results

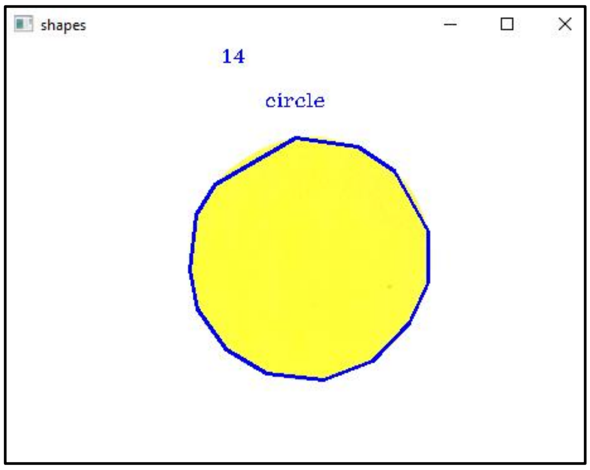


Figure 3 - Image detected as a Circle

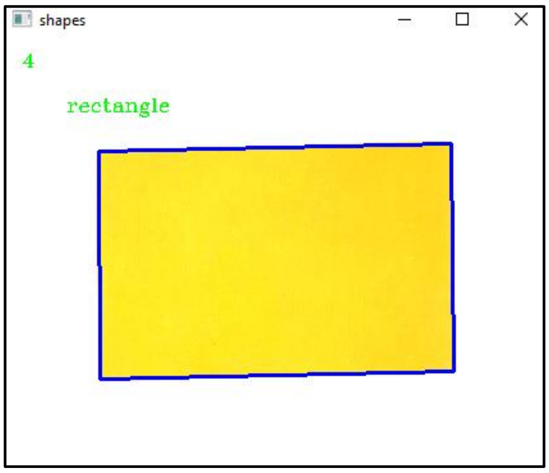


Figure 4 - Image detected as a Rectangle

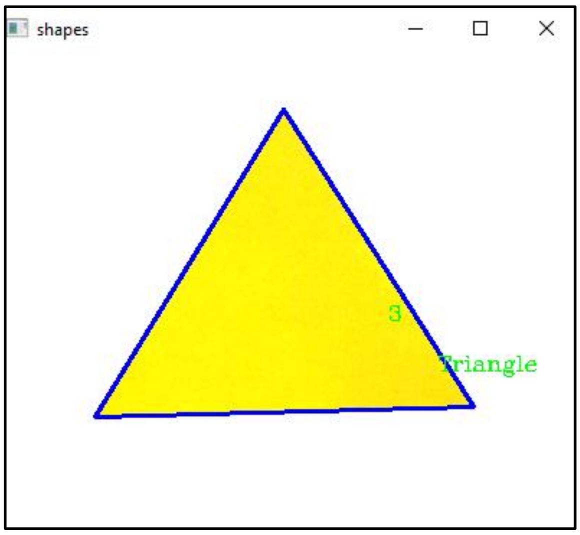


Figure 5 - Image detected as a Triangle

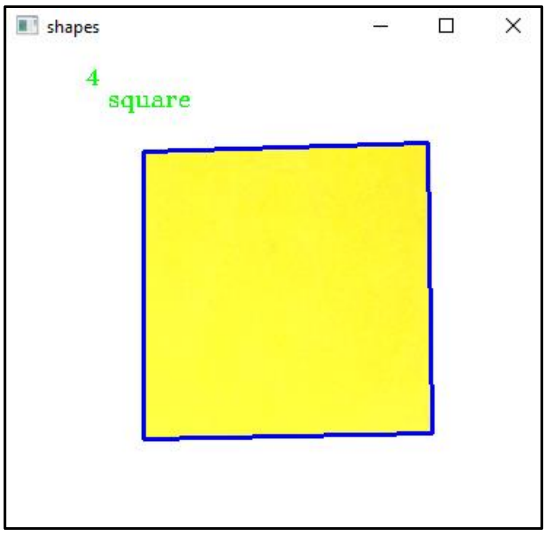


Figure 6 - Image detected as a Square

Conclusion

The given images were classified based on their shape. The name of the shape and number of edges detected were also printed on the image.

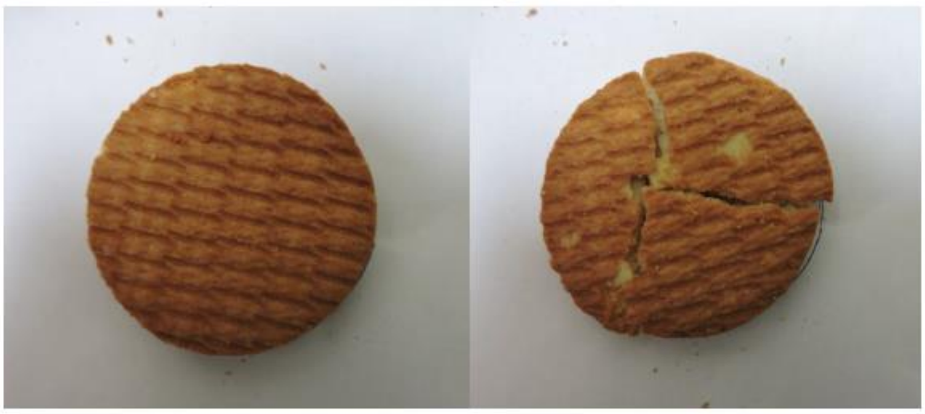
Experiment 3 - Biscuit Quality Detection using OpenCV

Aim

To determine the quality of biscuits from images based on suitable algorithms.

Problem Statement

Determine the quality of the biscuits using the following images.



Theory

Industries would require several quality inspection applications. One such application has been demonstrated for food industries. The quality of the biscuits must be checked and classified as broken or not broken.

Quality Detection will be performed as a combination of colour, edge and shape detection. The area encompassed by the detected contours will be evaluated to determine the size of the shape detected. If the size is large enough it will be considered as a piece of the biscuit. If multiple pieces are detected, the biscuit will be classified as Broken. If only one long, continuous edge is detected, the biscuit will be classified as Good Quality.

Functions such as cv2.contourArea and cv2.arclength along with dedicated functions for colour and edge detection will be used from the Computer Vision Library, in conjunction with NumPy to perform accurate biscuit quality detection.

Procedure

- Read the given image using cv2.read and show it in a window titled 'image' using cv2.imshow.
- Define a mouse event to return HSV values upon mouse click.
- Use the returned values to define ranges for the colour Red.
- Create a 3x3 matrix of ones and dilate it. [np.ones, cv2.dilate]
- Perform thresholding operation to detect edges. [cv2.inRange]
- Find contours based on detected edges [cv2.findContours]
- Analyse the area values encompassed by contours [cv2.contourArea]
- Analyse the length of contours detected [cv2.arcLength]
- Based on above parameters, classify biscuit as Broken or Good Quality.
- Print the result and contours found onto the input image. [cv2.putText]

Code

```
import cv2
import numpy as np

image=cv2.imread('C:/Users/MAHE/Documents/OpenCV_200929036/Biscuit/2.jpg')
cv2.imshow("image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
def mouse(event,x,y,flags,param):
    if event==cv2.EVENT_LBUTTONDOWN:
        h=hsv[y,x,0]
        s=hsv[y,x,1]
        v=hsv[y,x,2]
        print("H:",h)
        print("S:",s)
        print("V:",v)

cv2.namedWindow('mouse')
cv2.setMouseCallback('mouse',mouse)
hsv=cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
cv2.imshow("mouse", hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()

redLower=(0,150,50)
redUpper=(20,200,200)

mask=cv2.inRange(hsv,redLower,redUpper)
cv2.imshow("mask",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
kernel=np.ones((3,3),np.uint8)
mask=cv2.dilate(mask,kernel,iterations=1)
contours,heirarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
cv2.imshow("dilate",mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
print("Number of contours found="+str(len(contours)))

for c in contours:
    area = cv2.contourArea(c)
    perimeter = cv2.arcLength(c, True)
    ((x,y),radius)= cv2.minEnclosingCircle(c)
    if (area>500):
        cv2.drawContours(image,[c],-1,(255,0,0),5)
        print("Area.{},Perimeter.{}".format(area,perimeter))
        print("number of contours:{}".format(len(contours)))
        if(perimeter>2100):
            string="broken"
        else:
            string="good quality"
        cv2.putText(image,string, (50,50), cv2.FONT_HERSHEY_COMPLEX,2,(0,0,255), 2)
cv2.imshow("contour",image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results

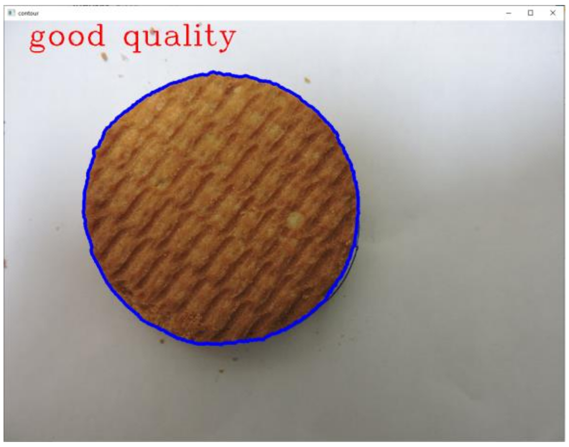


Figure 7 - Biscuit detected as Good Quality

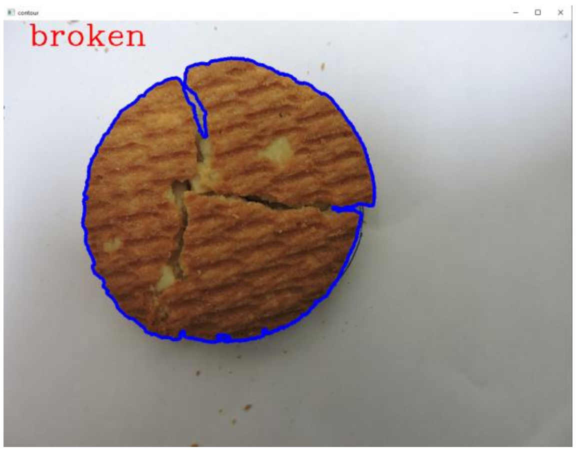


Figure 8 - Biscuit detected as Broken

Conclusion

The biscuits in the given image were classified into Broken and Good Quality. The results were also printed onto the images along with the detected contours.

References

- OpenCV: OpenCV modules
- First Principles of Computer Vision - YouTube
- Robotics I Lab Manual – 3rd Semester