CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Problem Set 03

Lucas Santos Zatta

oriented by

Flávio Luis Cardeal Pádua

Belo Horizonte

2021

Introduction

The goal of this work is to try to apply concepts seen in class as well as formulas in order to perform image analyzes. To solve this problem set, the codes implemented integrate functions created by the author with library functions(mostly, due to performance and time reasons).

For this problem set(problem set 3) we will be using Python3 with the libraries **Numpy**, **OpenCV** and **Matplotlib**. It's important to make sure that these libs are properly installed and fully working so that we can achieve the desired end results.

Questions

1. Identification and Analysis of Components in Binary Images

1.1 Threshold the image

First step is loading the desired case study image using openCV and then threshold the image. In order to threshold the image we will convert it to grayscale and apply (as the name suggests) a threshold function. This can be done with the following code snippet:

```
image = cv2.imread(archive_name)
grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

tret,thresh = cv2.threshold(grey,127,255,cv2.THRESH_BINARY)
```

Listing 1: Image thresholding

These are the results compared to the original images:

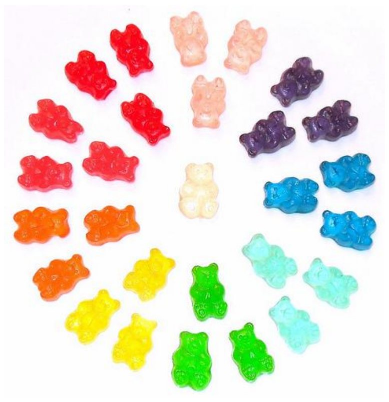


Figure 1: Original Gummy Input

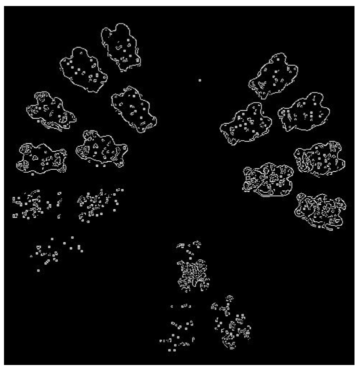


Figure 2: Thresholded Gummy

1.2 Counting Components and calculating measures

Now we want to count the elements in our image.

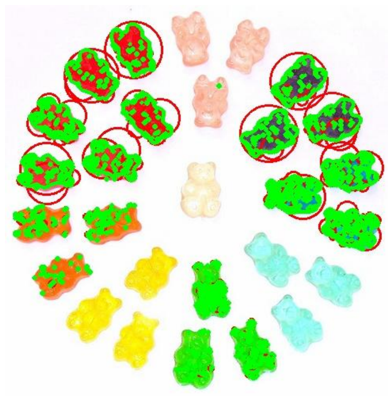


Figure 3: Counting all components

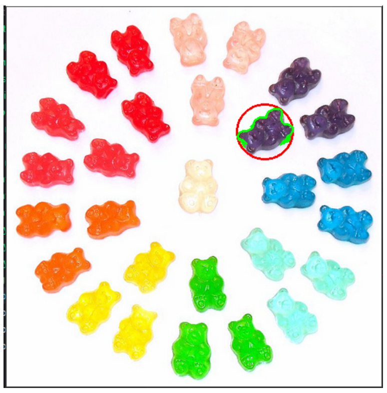
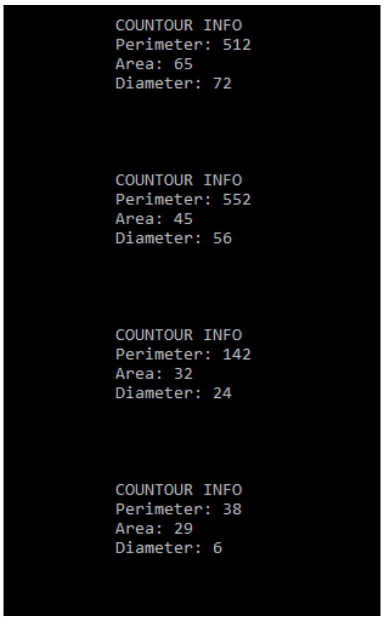


Figure 4: Counting components one at a time


```
COUNTOUR INFO
Perimeter: 512
Area: 65
Diameter: 72


COUNTOUR INFO
Perimeter: 552
Area: 45
Diameter: 56


COUNTOUR INFO
Perimeter: 142
Area: 32
Diameter: 24


COUNTOUR INFO
Perimeter: 38
Area: 29
Diameter: 6
```

Figure 5: Component measure info output

Observations

Other input images were used as means of output comparison. In images containing components with white surfaces or objects that weren't 100% opaque, the edges and contour became harder to detect. Sutil edges seen by the human eye were harder and harder to detect with the script as the transparency rose.

2. Evaluation of Co-Occurrence Measures

For this section we will use the following image as an example:

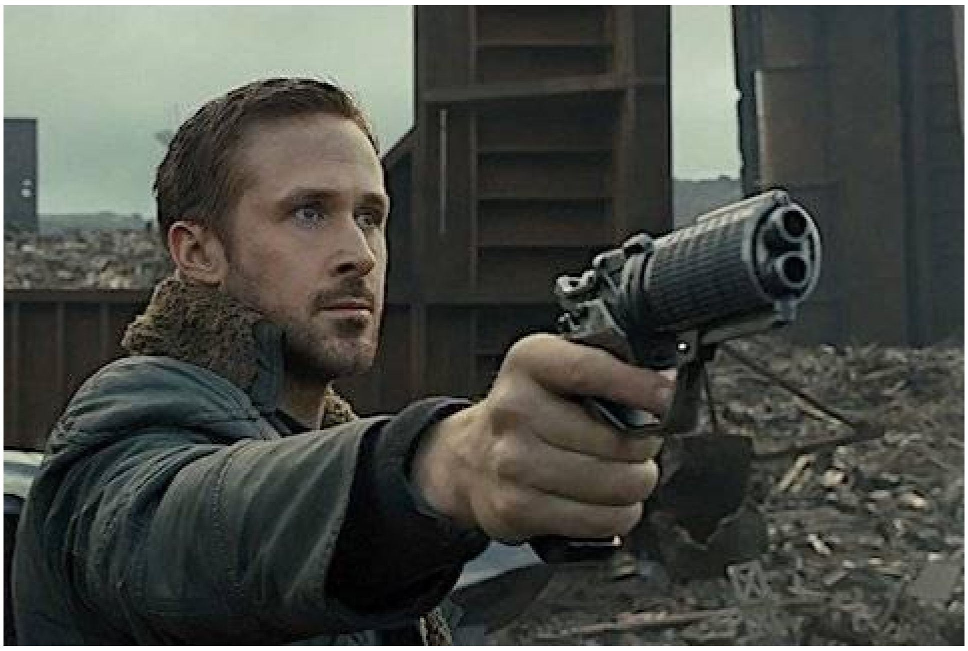


Figure 6: Image for Evaluation of Co-Occurrence Measures section

In order to compute the co-occurrence matrix we will now convert the images to gray

scale and perform the following operation:

```
for x in range(0,width-1):
      for y in range(0,height-1):
          #coocurrence matrix of the filtered image
          cm_filtered[filtered[x,y], filtered[x+1,y]] += 1
          cm_filtered[filtered[x,y], filtered[x-1,y]] += 1
          cm_filtered[filtered[x,y], filtered[x,y+1]] += 1
          cm_filtered[filtered[x,y], filtered[x,y-1]] += 1

          #Coocurrence matrix of the residual image
          cm_residual[residual[x,y], residual[x+1,y]] += 1
          cm_residual[residual[x,y], residual[x-1,y]] += 1
          cm_residual[residual[x,y], residual[x,y+1]] += 1
          cm_residual[residual[x,y], residual[x,y-1]] += 1
```

Listing 2: Calculation of co-occurrence and residual matrix

Considering I(p) our image and S(p) the 3x3 local median smoothing function. We dene
the residual R(p) = I(p) - S(p). S will be applied recursively as S^n = S(S^(n-1)), and the residual
again as R^n = I - S^n. After computing this recursively, for S30 and R30 we have:

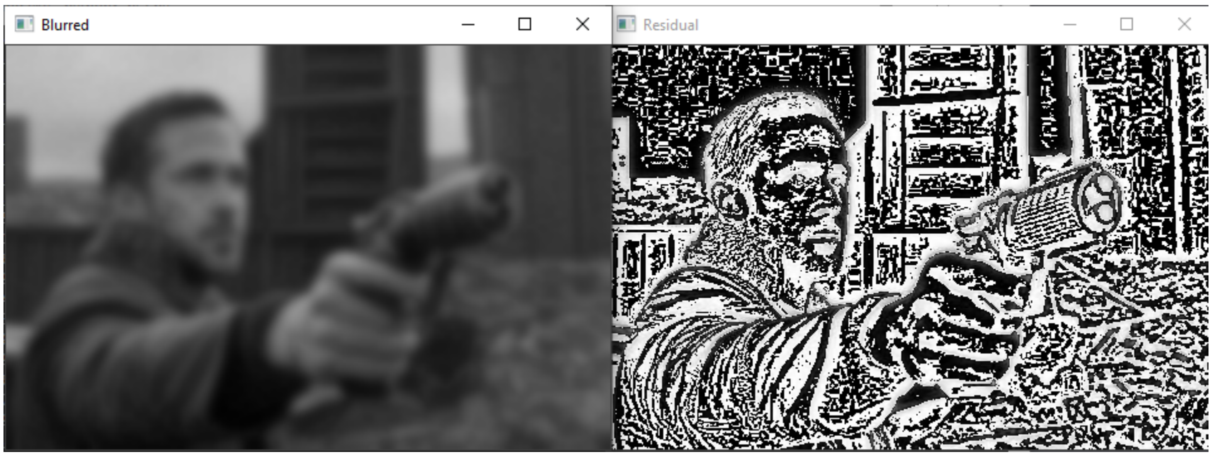


Figure 7: Smoothed and Residual

3. Features of Components Using Moments

In this problem, we will use a similar approach to Problem 1. In order to calculate centroid and a mainx axis we first need to detect and establish the components axis.

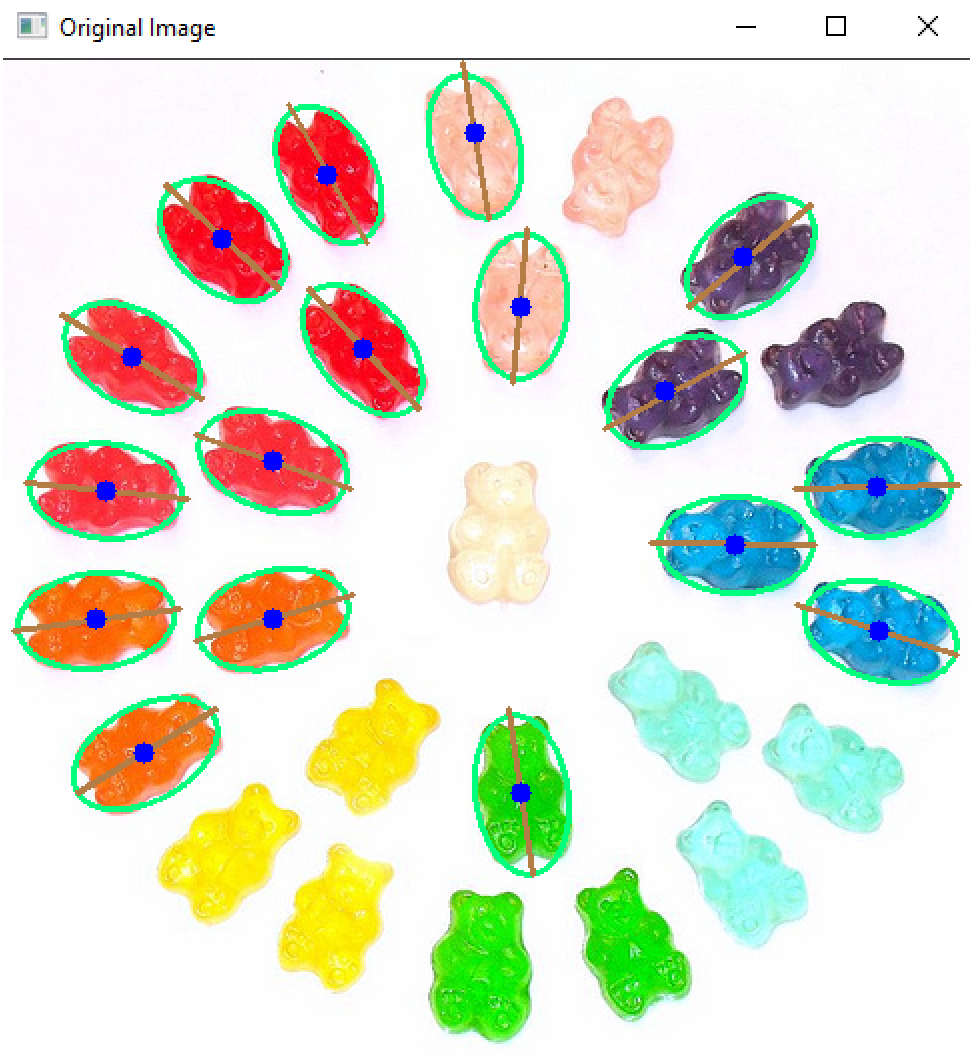


Figure 8: Gummy Components with main axis and centroid

This can be accomplished as shown below:

```python
for c in cnts:
    (x,y),radius = cv2.minEnclosingCircle(c)
    center = (int(x),int(y))
    radius = int(radius)
    #If the radius of the minimum enclosing circles is too small
    #we asume its an artifact
    if radius > 20:
        M = cv2.moments(c)

        if(M['m00']) != 0:
            #Find the center of mass
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])

            #Find the direction
            rows,cols = image.shape[:2]
            [vx,vy,x,y] = cv2.fitLine(c, cv2.DIST_L2,0,0.01,0.01)

            u,v,w,h = cv2.boundingRect(c)

            #Define the line cordinates
            line_x1 = int(x + radius*(vx)*1.1)
            line_y1 = int(y + radius*(vy)*1.1)

            line_x2 = int(x - radius*(vx)*1.1)
            line_y2 = int(y - radius*(vy)*1.1)

            #Find the ellipse (eccentricity)
            ellipse = cv2.fitEllipse(c)

            #Draw it all in the image
            cv2.ellipse(image,ellipse,(125,255,0),2)
            cv2.line(image,(line_x1,line_y1),(line_x2,line_y2),(70,125,180),2)
            cv2.circle(image,(cx, cy), 5, (255,0,0), -1)
```

Listing 3: Code snippet showing main axes and centroid calculated and drawn

4. Generation of Noisy Line Segments and Hough Transforms

In this problem we will generate noisy line segments and use a line-detection algorithm.
The lines were generated using the following algorithm:

```
def sp_noise_line(prob, m,b,x_interval,y_interval):
    global image
    if x_interval[0] <= x_interval[1]:
        x1 = x_interval[0]
        x2 = x_interval[1]
    else:
        x1 = x_interval[1]
        x2 = x_interval[0]
    if y_interval[0] <= y_interval[1]:
        y1 = y_interval[0]
        y2 = y_interval[1]
    else:
        y1 = y_interval[1]
        y2 = y_interval[0]

    if((x2-x1)>(y2-y1)):
        for x in range(x1,x2):
                rdn = random.random()
                if rdn < prob:
                    y = int(m*x + b)
                    image[y+random.randint(1,
5)][x++random.randint(1, 3)] = 0
                    image[y+random.randint(1,
5)][x++random.randint(1, 3)] = 0
    else:
        for y in range(y1,y2):
                rdn = random.random()
                if rdn < prob:
                    x = int((y - b)/m)
                    image[y+random.randint(1,
5)][x++random.randint(1, 3)] = 0
                    image[y+random.randint(1,
5)][x++random.randint(1, 3)] = 0
```

Listing 4: Noisy Line Drawer

After generating, we invert the scale and apply the Hough transform to detect lines we have the following:

Noisy Lines output:

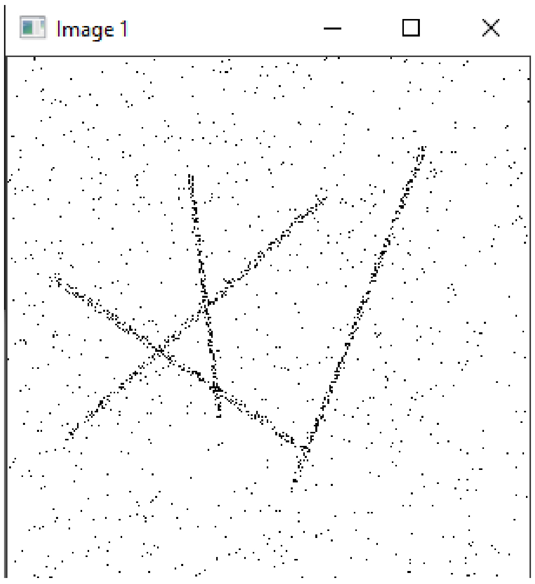


Figure 9: Noisy lines displayed

Line detection on Noisy Lines

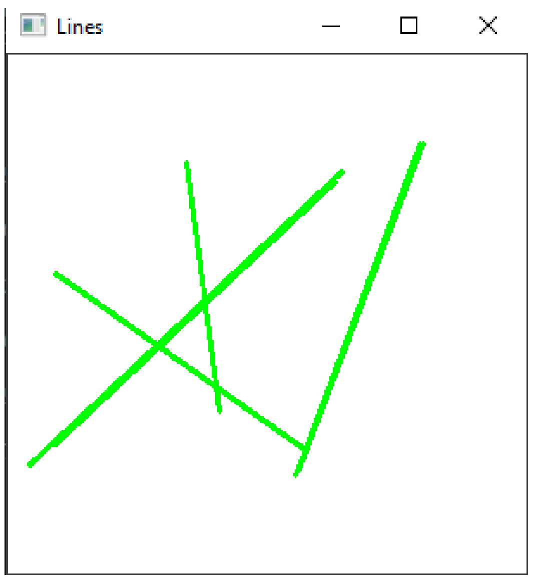


Figure 10: Highlighted lines