REPORT ON ASSIGNMENT 1

MCTE 4323 MACHINE VISION

SECTION 2

Name : Azizi Bin Mohamad Tambi

Matric No : 1919661

Session : Sem 1, 2022/2023

Date : 28 November 2022

Instructor : Dr. Hasan Firdaus Bin Mohd Zaki

1) By using google colab code to access your webcam, capture three different facial expressions of yourself.
A) Apply both Canny Edge Detection and Sobel Edge Detection to only the face area of the image using a suitable threshold values. (5 Marks)
B) Perform blurring of only faces in those images. (5 Marks)
C) Calculating the number of coins in an image using contours. (5 Marks)

2) Instructions for Submission

A) Your codes should be shared on Github. A separate word file should be prepared with the answers to these questions (output from codes, explanations where deemed necessary,etc) and also Github links for the codes. You will need a Github account for this purpose.

B) Submit a Github link and doc through google form. In the doc, you can write some descriptions of functions used and outputs

3) Opening a Github account

A) If you do not have a Github account yet, follow the Sign Up instructions on this link: https://github.com/

Submission Link:
https://docs.google.com/forms/d/e/1FAIpQLSfdzgrEleK_rZKpHF5r6qTCd9-OFwosuKGYR8BVr5xhiwtdgg/viewform

Assignment 1A

Firstly, in the assignment, we need to use a webcam to capture three different facial expressions.

```
[118] from IPython.display import display, Javascript
      from google.colab.output import eval_js
      from base64 import b64decode

def take_photo(filename='Expression_Angry.jpg', quality=1.0):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);
```

Figure 1.0: Naming a file

As shown inside the figure 1.0 above, we can change the highlighted string (which in my case is “Angry”) three times for each different expression. The picture will be saved automatically in our google drive as shown in figure 2.0 below.

Figure 2.0: Output picture from the webcam

```
#opening image using openCV2
#for happy expression
img_happy_bgr = cv2.imread(path+"Expression_Happy.jpg",cv2.IMREAD_COLOR)
img_happy_rgb = cv2.cvtColor(img_happy_bgr, cv2.COLOR_BGR2RGB)
happy_gray = cv2.cvtColor(img_happy_rgb, cv2.COLOR_RGB2GRAY)
```

Figure 3.0: Reading and converting image

For the codes shown in figure 3.0, we first use cv2.imread to obtain the picture from our google drive. Then we use cv2.IMREAD_COLOR to read the picture in BGR mode. As the picture is in BGR mode, we will need to convert the picture by using cv2.cvtColor and choose whether to convert the picture into RGB or Grayscale picture.

Since google colab cannot use cv2.imshow to show an image, we can use matplotlib to plot our picture by using plt.imshow as shown in the figure 4.0 and 5.0 below.

```
def plot_expression_rgb():
  plt.figure(figsize=[18,5])
  plt.subplot(131);plt.imshow(img_happy_rgb);plt.title("Happy");
  plt.subplot(132);plt.imshow(img_angry_rgb);plt.title("RGB Figure \n Angry");
  plt.subplot(133);plt.imshow(img_sad_rgb);plt.title("Sad");

def plot_expression_gray():
  plt.figure(figsize=[18,5])
  plt.subplot(131);plt.imshow(happy_gray, cmap='gray');plt.title("Happy");
  plt.subplot(132);plt.imshow(angry_gray, cmap='gray');plt.title("Grayscale Figure
  plt.subplot(133);plt.imshow(sad_gray, cmap='gray');plt.title("Sad");

plot_expression_rgb()
plot_expression_gray()
```

Figure 4.0 : Using matplot to plot picture

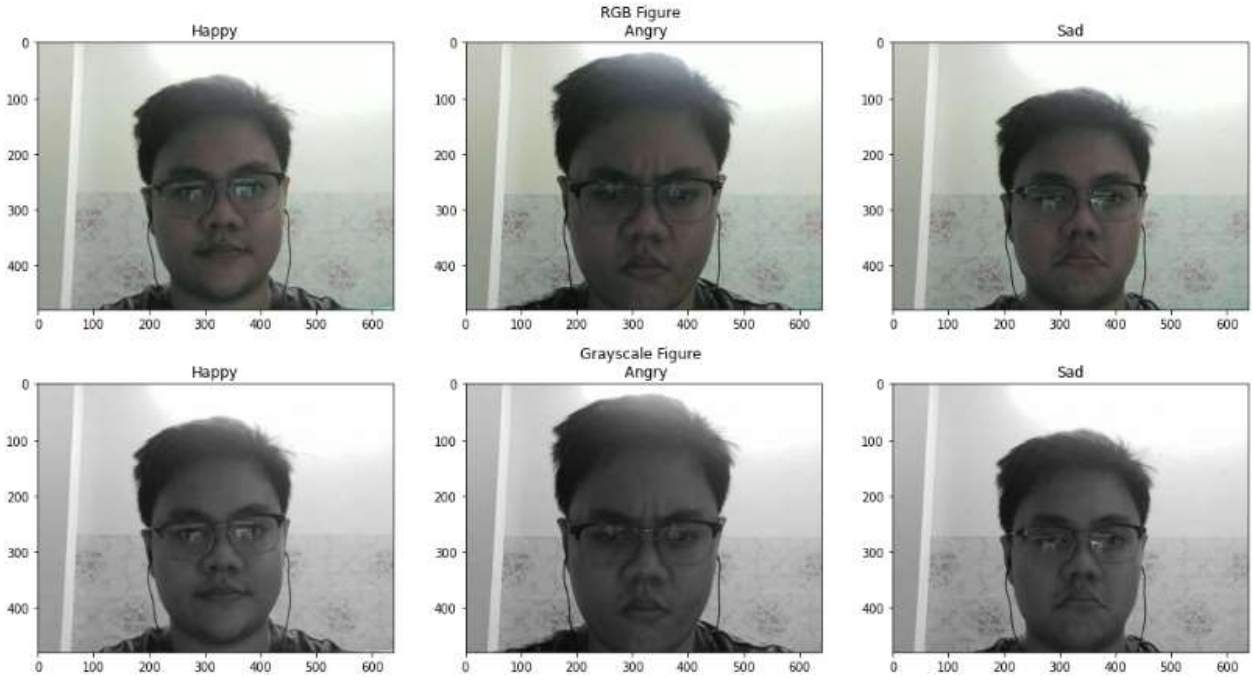

Figure 5.0 : Output from using plt.imshow

To detect a face, we can use the Haar Cascade method by first importing the Haar cascade library into our codes.

```
#def face_detection(fileName, expression):
face_cascade = cv2.CascadeClassifier(path +"haarcascade_frontalface_alt.xml")
face1 = face_cascade.detectMultiScale(img_happy_rgb, 1.1 , 4)
face2 = face_cascade.detectMultiScale(img_angry_rgb, 1.1 , 4)
face3 = face_cascade.detectMultiScale(img_sad_rgb, 1.1 , 4)
```

Figure 6.0: Using Cascade Classifier

From figure 6.0, we are initializing a Cascade Classifier. The “haarcascade_frontalface _alt.xml” is a specification for which pretrained model is used for this assignment. For face1, face2 and face3, this is where the detection happens and the parameters here are very important. For the first parameter which is scale factor with a value of 1.1, what it means here is that I am using a small step for resizing where I resize the picture by 10% so that the chance of a matching size with the model for detection will be increased. Then, the value 4 is the parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces where higher value results in fewer detections but with higher quality.

```
for(x, y, w, h) in face1:
  cv2.rectangle(img_happy_rgb, (x, y), (x + w, y + h), (255, 0, 0), 3)
  #crop image
  happy_rgb = img_happy_rgb[y:y + h, x:x +w]
  happy = cv2.cvtColor(happy_rgb, cv2.COLOR_RGB2GRAY)
```

Figure 7.0: Drawing rectangle and cropping image

Based on figure 7.0, x, y are pixel location of faces, w, h are width and height of faces. We use cv2.rectangle() function to draw a rectangle over the detected object image. (x,y),(x+w, y+h) are the location of the rectangle. Then we crop the image inside the rectangle and convert the cropped image to a grayscale picture. The output of the face detection and cropped picture are shown in figure 8.0.

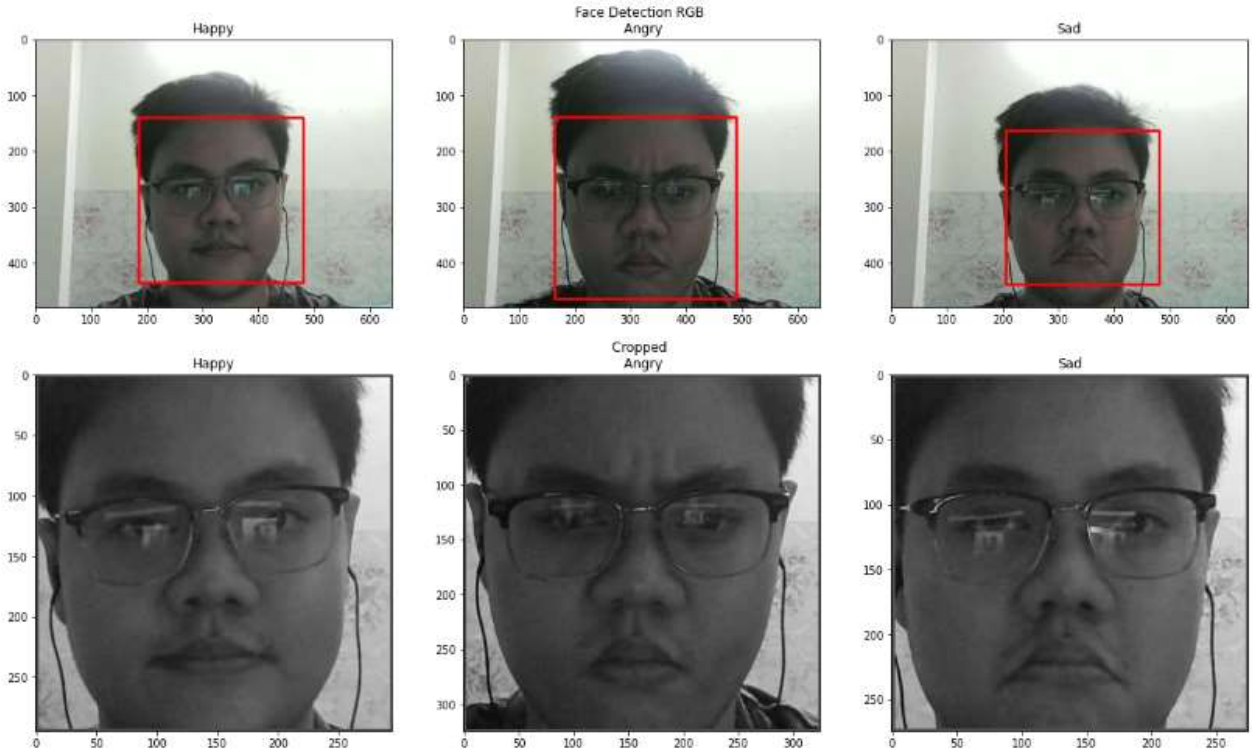

Figure 8.0: Face detection and Face detection cropping

After detecting a face, we can use a canny edge detection method to detect edges. Figure 9.0 below shows the function created for canny edge detection on 3 different images with different expressions.

```
def canny_expression(Gx, Gy):
  canny_happy = cv2.Canny(happy,Gx,Gy) #canny for happy expression
  canny_angry = cv2.Canny(angry,Gx,Gy)
  canny_sad = cv2.Canny(sad,Gx,Gy)
  plt.figure(figsize=[20,10])
```

Figure 9.0: Using Canny Edge Detection

For cv2.Canny(), Gx represents the first derivative in horizontal direction while Gy is the first derivative for vertical direction. For this function, the value for Gx is 15 while Gy is 70. We use this value because this value when added up will produce an edge gradient that results in a suitable output. Other values will start to erase the expression features that are present on the image.

```
def sobel_expression(x, y):
  sobel_x_happy = cv2.Sobel(happy, cv2.CV_8U,x,y,ksize=3)
  sobel_y_happy = cv2.Sobel(happy, cv2.CV_8U,x,y,ksize=3)
  sobel_happy = sobel_x_happy + sobel_y_happy
```

Figure 10.0: Using Sobel Edge Detection

For Sobel edge detection, we need to use cv2.Sobel() function with x and y parameters. The way sobel edge detection works is by convolving I with a kernel x and y with odd size where for this picture, we use x equal to 1 and y equal to 0. The reason we use this value is because other values will remove the edges from the picture and thus make the picture blank. The final result of the picture is the convolution of both horizontal changes,x and vertical changes, y.

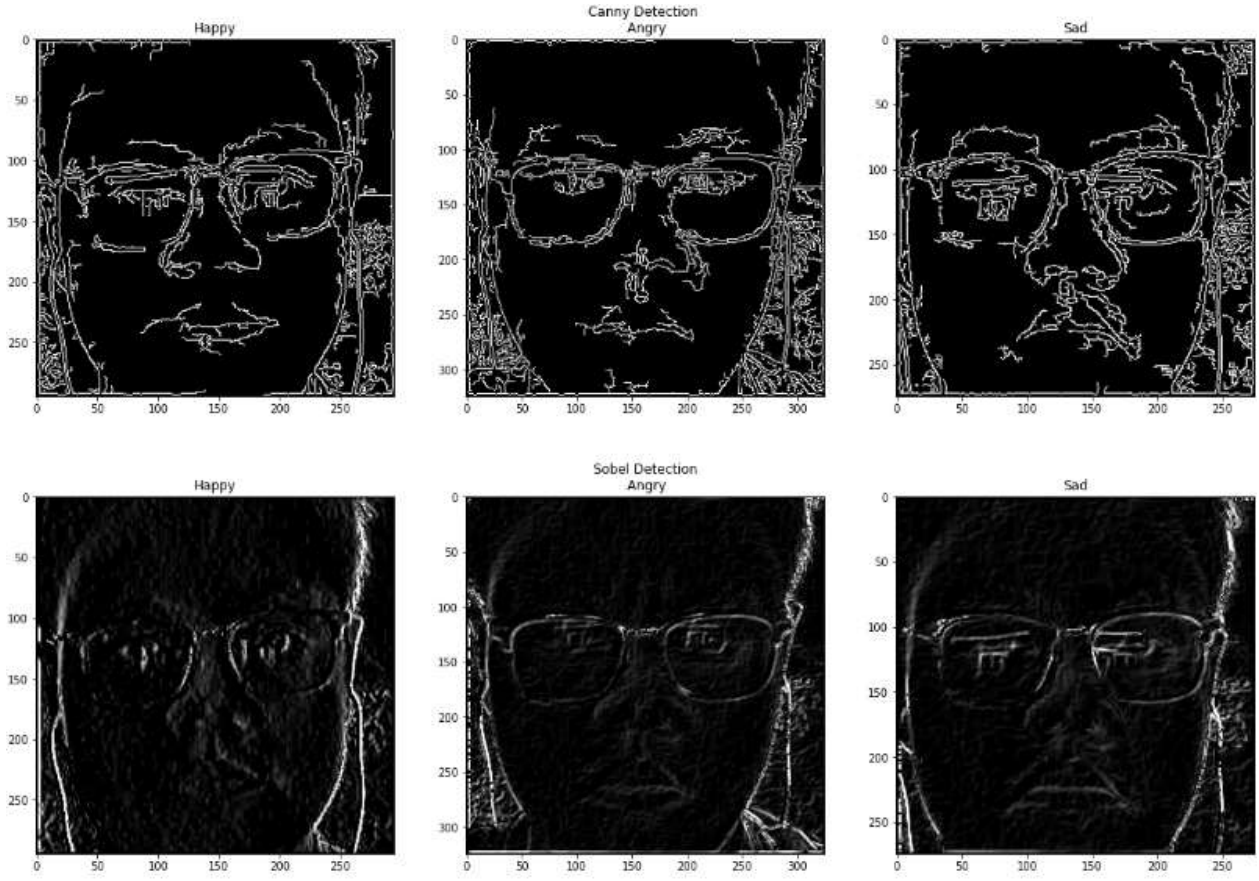

Figure 11.0: Output of Canny Edge Detection vs Sobel Edge Detection

Assignment 1B

Firstly, we need to define the path to our picture and import it into our codes as shown in figure 1.0.

```
#for happy expression
img_happy_bgr = cv2.imread(path+"Expression_Happy.jpg",cv2.IMREAD_COLOR)
#for angry expression
img_angry_bgr = cv2.imread(path+"Expression_Angry.jpg",cv2.IMREAD_COLOR)
#for sad expression
img_sad_bgr = cv2.imread(path+"Expression_Sad.jpg",cv2.IMREAD_COLOR)
```

Figure 1.0: Importing Pictures

We will use cv2.IMREAD_COLOR to read the pictures in BGR mode. Then we plot the pictures by using plt.imshow() and the output is shown in figure 2.0 as below.

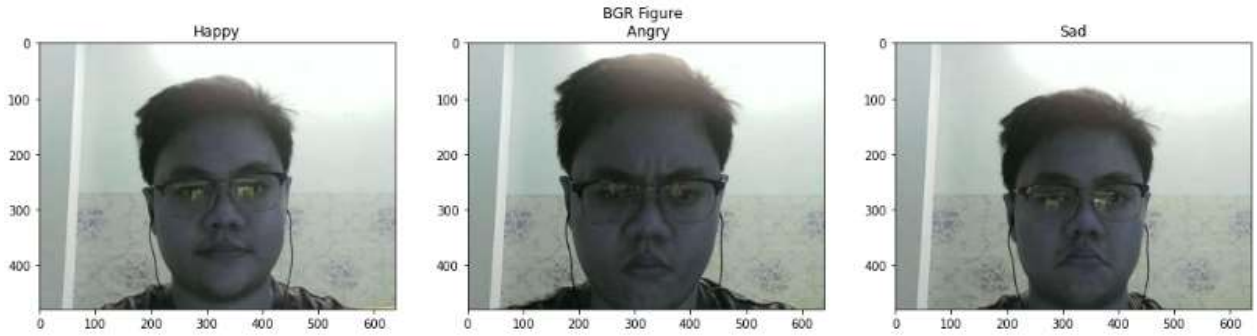

Figure 2.0: Pictures imported from Google Drive

Notice that the colour of the pictures are distorted. This is due to matplotlib capability to initially read any images in RGB mode while our pictures are imported in BGR mode. This explains why the colour for the images are distorted.

```
def face_detection(filename):
  face_cascade = cv2.CascadeClassifier(path +"haarcascade_frontalface_alt.xml")
  face = face_cascade.detectMultiScale(filename, 1.1 , 4)

  for(x, y, w, h) in face:
    cv2.rectangle(filename, (x, y), (x+w, y+h), (0,255,0), 3)
    face_region = filename[y:y + h, x:x + w]
    #blurring image
    blur = cv2.GaussianBlur(face_region, (85,85),0)

    filename[y:y+h, x:x+w] = blur
```

Figure 3.0: Face detection and Blurring

Same like in assignment 1A, we will use Haar Cascade for the face detection and here, we will add cv2.GaussianBlur to blur the region inside the rectangle. The first value of 85 in the cv2.GaussianBlur represents kernel standard deviation along X-axis while the second value of 85 represents kernel standard deviation along Y-axis. By making this value higher, we will be able to make the picture even blurry. After that, we initialise the function for each different image and plot each image as shown in figure 4.0.

```
face_detection(img_happy_bgr)
face_detection(img_angry_bgr)
face_detection(img_sad_bgr)

plt.figure(figsize=[18,5])
plt.subplot(131);plt.imshow(cv2.cvtColor(img_happy_bgr, cv2.COLOR_BGR2RGB));plt.title("Happy");
plt.subplot(132);plt.imshow(cv2.cvtColor(img_angry_bgr, cv2.COLOR_BGR2RGB));plt.title("Image Blur \n Angry");
plt.subplot(133);plt.imshow(cv2.cvtColor(img_sad_bgr, cv2.COLOR_BGR2RGB));plt.title("Sad");
```

Figure 4.0: Initializing and plotting the images

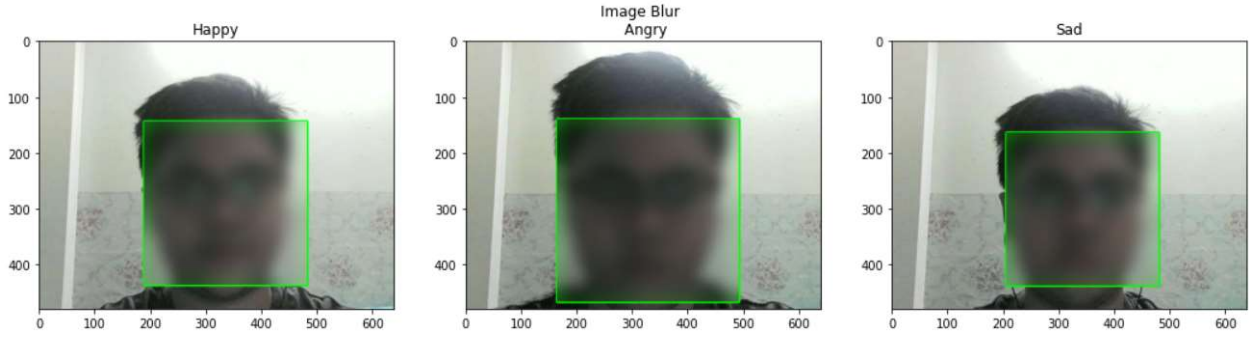

Figure 5.0: Final results for face detection and blurring

Notice that in figure 5.0, we are able to have the colour for the picture in order. This is because as shown in figure 4.0, we use cv2.cvtColor() to convert the colour of the image from BGR to RGB.

Assignment 1C

Firstly, we need to import the picture of the coin just like assignment 1A and 1B but for this assignment, we will import it in a grayscale picture.

```
coin = cv2.imread(path+"coin_detection3.jpg", cv2.IMREAD_GRAYSCALE)
coin_blur = cv2.GaussianBlur(coin, (11,11), 0)
coin_blur_canny = cv2.Canny(coin_blur, 20, 160, 3)

plt.imshow(coin_blur, cmap = 'gray')
```

Figure 1.0: Importing pictures, Blurring and using Canny Edge Detection

To import the picture in a grayscale mode, we can use cv2.IMREAD_GRAYSCALE or simply replace this with a value of 0. After that, we will need to blur the picture by using GaussianBlur and here, we want to just blur the picture slightly so we use a smaller value which is 11 compared to assignment 1B. Then, we will get the result as shown in figure 2.0.

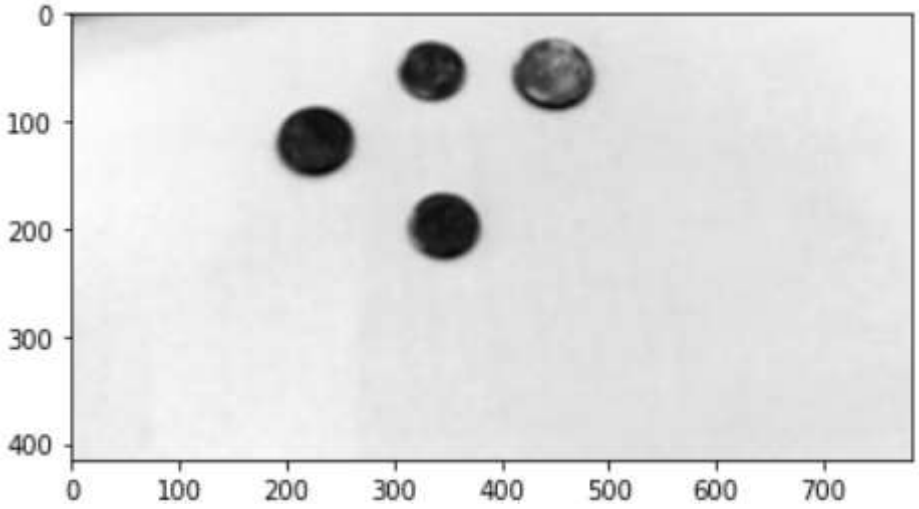

Figure 2.0: Grayscale and blurred picture of the coins

Then we use canny edge detection with the parameter of x equal to 20 and y equal to 120 to produce the best edge detection for this picture. Note that the value of x and y will vary depending on the pictures itself. We will need to vary the value until we find it suitable to our needs. For this assignment, using canny edge detection will result in the picture as shown in figure 3.0.

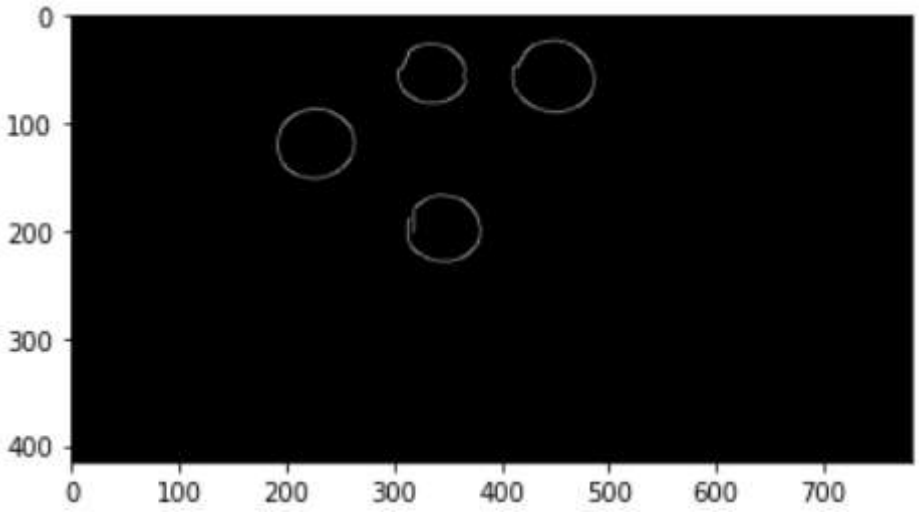

Figure 3.0: Using Canny Edge Detection on the picture

```
dilated = cv2.dilate(coin_blur_canny, (1,2), iterations = 2)
(cnt, _) = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

text = "There are {} coins".format(len(cnt))
font = cv2.FONT_HERSHEY_PLAIN
cv2.putText(dilated, text, (25,20), font, 2, (255, 0, 0), 1, cv2.LINE_AA)

plt.imshow(dilated, cmap = 'gray')
```

Figure 4.0: Codes to detect number of coins

Based on figure 4.0, we use cv2.dilate() function with an iterations of 2 to do erosion and dilation of the pictures. What this function does is that it increases the object area and we use it to accentuate features so that we can detect the coin. Then, we use the cv2.findContours() function to find the region of the coin so that we can deduce the number of coins.

After we find the region by using cv2.findContours(), we can customize the output image by using cv2.putText() function to put text in the picture. We can use many types of font however for this assignment, we used cv2.FONT_HERSHEY_PLAIN where we are using a font named Hershey Plain. Then we position the text to be at (25,20) where 25 is the horizontal coordinate and 20 is the vertical coordinate. The number 2 in the cv2.putText() is the parameter for the size of the font. The output of the overall codes are shown in figure 5.0.

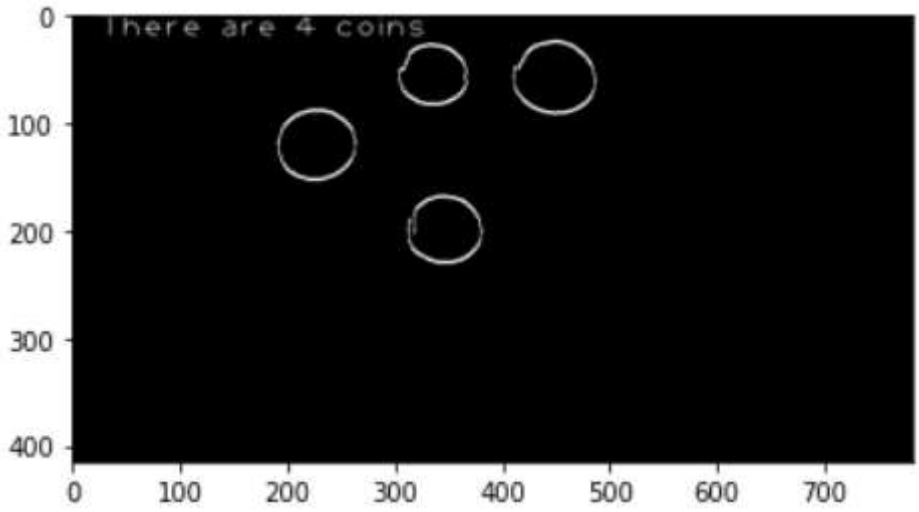

Figure 5.0: Output images from code in figure 4.0

```
coin_rgb = cv2.cvtColor(coin, cv2.COLOR_GRAY2RGB)
cv2.drawContours(coin_rgb, cnt,-1, (0,255,0), 2)

plt.imshow(coin_rgb)
print("Number of coins in the image:", len(cnt), "coins")
```

Figure 6.0: Codes for drawing a contours line

By using cv2.drawContours() in figure 6.0, we can draw the contours for the picture of the coins in RGB mode to make it look better. Then we can print the value of the coins detected in the picture as shown in figure 7.0.

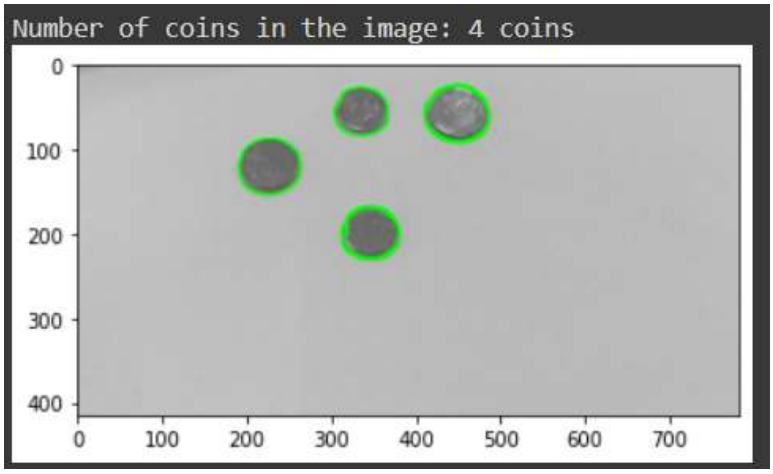

Figure 7.0: Final result of coin detection

Github Link: https://github.com/xxazizixx1/MCTE4323_Machine-Vision