Assignment #1

(LO 1)

Name: Mohamad Amin bin Rosli **Matric No:** 1917197

1) By using Google Colab code to access your webcam, capture three different facials expressions of yourself.

A new code box is added that allows to choose which functions to execute during the program.

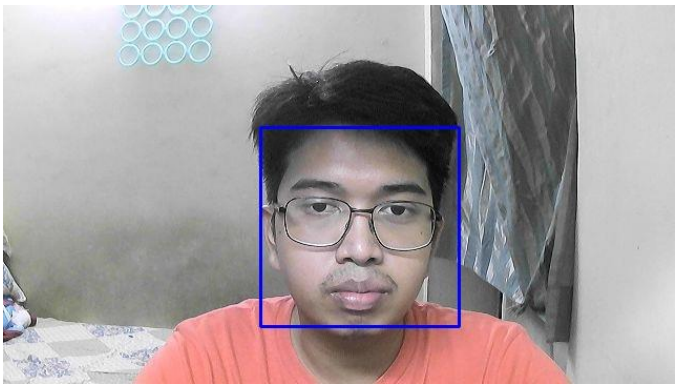
```
[ ]  # modify to change function of the program
     # q1 : Apply both Canny Edge Detection and Sobel Edge Detection
     # q2 : Perform blurring of only faces
     # q3 : Calculating the number of coins in an image using contours
     # default : bounding box for face

     mode="q3"
```

Modify the variable mode according to comments provided to change the function of the program.

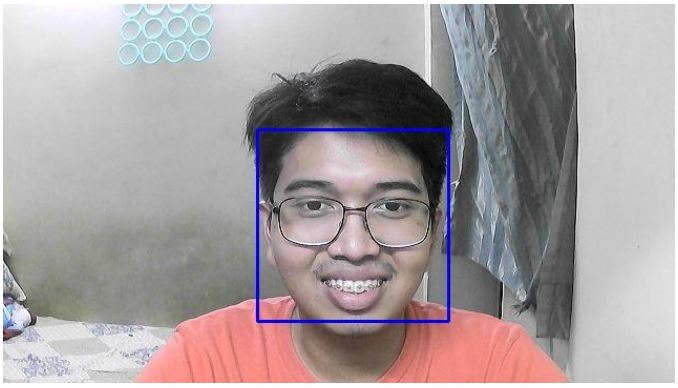
```
if mode=="q1":
  # passes only_face to Canny edge detection with a (low,high) threshold of (120,150)
  img_canny = cv2.Canny(only_face,120,150)
  # passes img_canny to Sobel edge detection with a kernel size of 1
  sobelx = cv2.Sobel(img_canny, cv2.CV_8U,1,0,ksize=-1)
  sobely = cv2.Sobel(img_canny, cv2.CV_8U,0,1,ksize=-1)
  # merge both sobel edge detection for x and y into img_sob
  img_sob = sobelx + sobely
  # convert the img sobel edge detection to RGB to make it the same dimension as img
  img_sobel = cv2.cvtColor(img_sob, cv2.COLOR_GRAY2RGB)
  # merge img_sobel on top of img obtained from captured camera
  img[y:y+img_sobel.shape[0], x:x+img_sobel.shape[1]] = img_sobel

elif mode=="q2":
  # passes only_face to Gaussian blur with a kernel size of 15x15 and sigmaX of 30
  img_blur = cv2.GaussianBlur(only_face,(15,15),30)
  # merge img_blur on top of img obtained from captured camera
  img[y:y+img_blur.shape[0], x:x+img_blur.shape[1]] = img_blur

elif mode=="q3":
  # post processing is done to enable easier detection of contours
  # passes gray to Gaussian blur with a kernel size of 11x11
  img_blur = cv2.GaussianBlur(gray,(11,11), 0)
  # passes img_blur for canny edge detection with a (low,high) threshold of (30,60)
  img_canny = cv2.Canny(img_blur,30, 60)
  # dilate img_canny to make the edges produced by the canny edge detection to be thicker
  dilate = cv2.dilate(img_canny , (1, 1), iterations=2)
  # find contours with the image of dilate
  (cnt, heirarchy) = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

  # draw the contours to img
  img=cv2.drawContours(img, cnt, -1, (0,255,0), 2)
  # prints the amount of countours found
  print('Coins in the image: ', len(cnt))

else :
  # if mode is anything else than q1,q2 or q3, run bounding box command
  img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
```

Output:

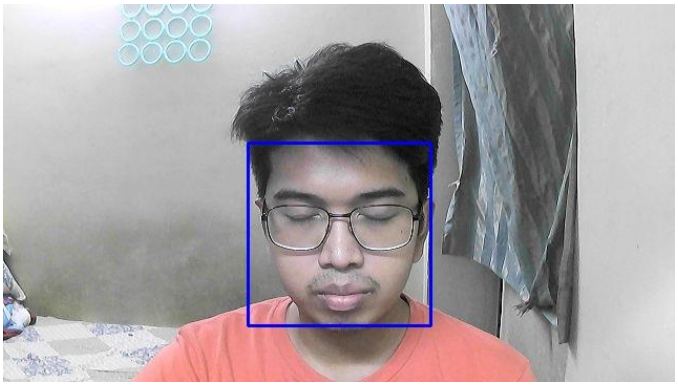
i) Normal expression



ii) Happy expression



iii) Sleeping



A) Apply both Canny Edge Detection and Sobel Edge Detection to only the face area of the image using a suitable threshold value. (5 Marks)

```
# initialize only_face to face area that is detected by Haar Cascade
  only_face = img[y:y+h,x:x+w]

  if mode=="q1":
    # passes only_face to Canny edge detection with a (low,high) threshold of (120,150)
    img_canny = cv2.Canny(only_face,120,150)
    # passes img_canny to Sobel edge detection with a kernel size of 1
    sobelx = cv2.Sobel(img_canny, cv2.CV_8U,1,0,ksize=1)
    sobely = cv2.Sobel(img_canny, cv2.CV_8U,0,1,ksize=1)
    # merge both sobel edge detection for x and y into img_sob
    img_sob = sobelx + sobely
    # convert the img sobel edge detection to RGB to make it the same dimension as img
    img_sobel = cv2.cvtColor(img_sob, cv2.COLOR_GRAY2RGB)
    # merge img_sobel on top of img obtained from captured camera
    img[y:y+img_sobel.shape[0], x:x+img_sobel.shape[1]] = img_sobel
```

Code flow:

1) Obtains face image detected by initializing only_face.
2) Checks mode variable, if mode='q1' runs the code for question 1.
3) Passes only_face to Canny edge detection with low and high thresholds of 120 and 150 as img_canny.
4) Passes img_canny to Sobel edge detection with a kernel size of 1 as sobelx and sobely.
5) Adds sobelx and sobely as img_sob.
6) Convert img_sob from GRAY to RGB to avoid conflicts with the dimensions of img as img_sobel.
7) Merge img_sobel with img from the camera.

Output:

i) Normal expression

ii) Happy expression

iii) Sleeping

B) Perform blurring of only faces in those images. (5 Marks)

```
elif mode=="q2":
  # passes only_face to Gaussian blur with a kernel size of 15x15 and sigmaX of 30
  img_blur = cv2.GaussianBlur(only_face,(15,15),30)
  # merge img_blur on top of img obtained from captured camera
  img[y:y+img_blur.shape[0], x:x+img_blur.shape[1]] = img_blur
```

Code flow:

1) Obtains face image detected by initializing only_face.
2) Checks mode variable, if mode='q2' runs the code for question 2.
3) Passes only_face to Gaussian blur with a kernel size of 15x15 and sigmaX of 30 as img_blur.
4) Merges img_blur with img obtained at camera with the position of the bounding box obtained by Haars Cascade.

Output:

i) Normal expression

ii) Happy expression

iii) Sleeping

C) Calculating the number of coins in an image using contours. (5 Marks)

```
elif mode=="q3":
  # post processing is done to enable easier detection of contours
  # passes gray to Gaussian blur with a kernel size of 11x11
  img_blur = cv2.GaussianBlur(gray,(11,11), 0)
  # passes img_blur for canny edge detection with a (low,high) threshold of (30,60)
  img_canny = cv2.Canny(img_blur,30, 60)
  # dilate img_canny to make the edges produced by the canny edge detection to be thicker
  dilate = cv2.dilate(img_canny , (1, 1), iterations=2)
  # find contours with the image of dilate
  (cnt, heirarchy) = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
  # draw the contours to img
  img=cv2.drawContours(img, cnt, -1, (0,255,0), 2)
  # prints the amount of countours found
  print('Coins in the image: ', len(cnt))
```

Code flow:

1) Runs some image processing to make it easier to detect the contours of the coins.
2) Gaussian blur with a kernel size of 11x11 is done to reduce noise in the image.
3) Canny edge detection with a low and high threshold of 30 and 60 is used to detect edges of the coins.
4) The img_canny is dilated to largen the edges detected by Canny edge detection.
5) The variable cnt is used to store the number of contours found by cv2.findCountours.
6) The variable cnt is drawn on the image by using cv2.drawCountours on the original image captured by the webcam.
7) Print ‘Coins in the image:’ and proceed to print the length of the cnt variable which corresponds to the number of contours found in the processes image.

Output:

i) Input

ii) Output

```
try:
  filename = take_photo('photo.jpg')
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))

  #1917197
```

```
Coins in the image:  5
Saved to photo.jpg
```