Script Documentation

VIVIA

Aymen Rebhi

Internship Technical Assessment

Contents

Introduction

This assignment aims to develop a system that analyzes video inputs to determine stress levels based on multiple factors including facial expressions

1 Objectives

The main goal of this test is to create a script that can predict stress levels in individuals by leveraging computer vision techniques. analyzing various facial and body features such as eye blinks, lip movements, eyebrow gestures, emotions ..etc .

2 Methodology

The methodology involves the utilization of existing libraries, pre-trained models, and computer vision techniques to extract and analyze relevant features from video inputs. Various factors including facial expressions emotional expressions will be considered to predict stress levels.

The system will involve the following steps:

- **Video Input and Processing**
- **Facial Feature Extraction and Tracking**
- **Emotion Recognition**
- **Eye Blink and Lip Movement Detection**
- **Predicting Stress ratio Based on considered factors**

Script Explanation

1 tools and libraries

MediaPipe is an open-source, cross-platform framework developed by Google that offers a comprehensive set of tools and solutions for building real-time perception applications. It enables developers to efficiently construct pipelines for processing multimedia data such as video streams or images, allowing tasks like object detection, facial recognition, hand tracking, and gesture recognition.

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of tools and functionalities for image and video processing

time: A Python module providing functionalities for working with time used for timing operations in code.

utils: A custom utility module containing methods to draw over display.

math: The built-in Python module offering mathematical functions .

numpy (np): A common library for numerical computations in Python, dealing with arrays, matrices, and mathematical operations on them

```
import cv2 as cv
import mediapipe as mp
import time
import utils, math
import numpy as np
```

Make sure to install the necessary packages and using the right python (3.9 – 3.11 64 bits) to use the mediapipe package
you can run the following commands

pip install opencv-python

pip install mediapipe

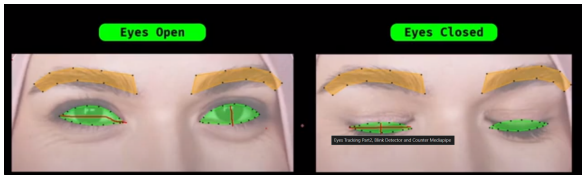
pip install numpy

cv.FONT HERSHEY COMPLEX is used to specify a font type for drawing text on images or video frames.

```
r_min_x = (min(right_eye_coords, key=lambda item: item[0]))[0]
r_max_y = (max(right_eye_coords, key=lambda item: item[1]))[1]
r_min_y = (min(right_eye_coords, key=lambda item: item[1]))[1]

# For LEFT Eye
l_max_x = (max(left_eye_coords, key=lambda item: item[0]))[0]
l_min_x = (min(left_eye_coords, key=lambda item: item[0]))[0]
l_max_y = (max(left_eye_coords, key=lambda item: item[1]))[1]
l_min_y = (min(left_eye_coords, key=lambda item: item[1]))[1]

# croping the eyes from mask
cropped_right = eyes[r_min_y: r_max_y, r_min_x: r_max_x]
cropped_left = eyes[l_min_y: l_max_y, l_min_x: l_max_x]

# returning the cropped eyes
return cropped_right, cropped_left
```

The official Mediapipe documentation has an array number view of the face mesh mapped onto the image.
In mediapipe.solutions.face_mesh, if refine_landmarks=True, a total of 478 landmark points can be obtained. If that option is False, the number of landmarks is 468.

```
10 usages
def euclaideanDistance(point, point1):
    x, y = point
    x1, y1 = point1
    distance = math.sqrt((x1 - x) ** 2 + (y1 - y) ** 2)
    return distance
```



initialize video capture using the default camera

```
# camera object
camera = cv.VideoCapture(0)
```

the provided function's code take an image as input, the results of landmark detection and whether to draw the coordinates or not parameter. the return will be the facial land<mark coordinates in each frame of the video.

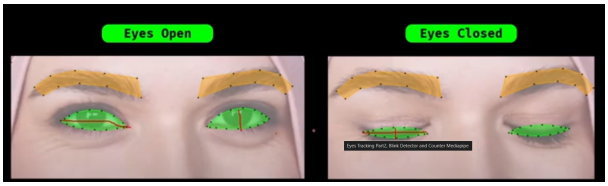
```
r_min_x = (min(right_eye_coords, key=lambda item: item[0]))[0]
r_max_y = (max(right_eye_coords, key=lambda item: item[1]))[1]
r_min_y = (min(right_eye_coords, key=lambda item: item[1]))[1]

# For LEFT Eye
l_max_x = (max(left_eye_coords, key=lambda item: item[0]))[0]
l_min_x = (min(left_eye_coords, key=lambda item: item[0]))[0]
l_max_y = (max(left_eye_coords, key=lambda item: item[1]))[1]
l_min_y = (min(left_eye_coords, key=lambda item: item[1]))[1]

# croping the eyes from mask
cropped_right = eyes[r_min_y: r_max_y, r_min_x: r_max_x]
cropped_left = eyes[l_min_y: l_max_y, l_min_x: l_max_x]

# returning the cropped eyes
return cropped_right, cropped_left
```

this function will draw the eyebrows based on landmarks results and right eyebrow indices

```
# Eyebrow detector
2 usages
def draw_eyebrows(img, landmarks, eyebrow_indices):
    for i in range(len(eyebrow_indices) - 1):
        cv.line(img, landmarks[eyebrow_indices[i]], landmarks[eyebrow_indices[i + 1]], utils.YELLOW, thickness: 2)
```

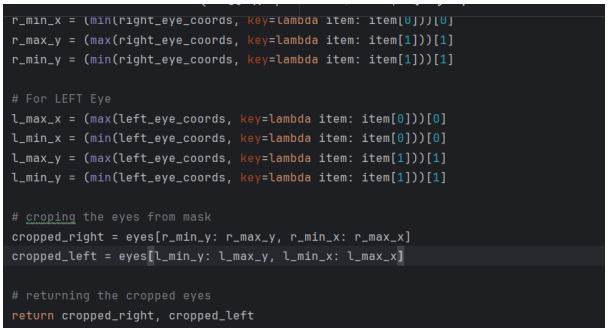
this function is used to calculate the euclidean distance between two points and it will be used to calculate the distance between the vertical indices and horizontal in single eye so if a ratio is over 3.5 this means that the eye is blinking .

```
10 usages
def euclaideanDistance(point, point1):
    x, y = point
    x1, y1 = point1
    distance = math.sqrt((x1 - x) ** 2 + (y1 - y) ** 2)
    return distance
```

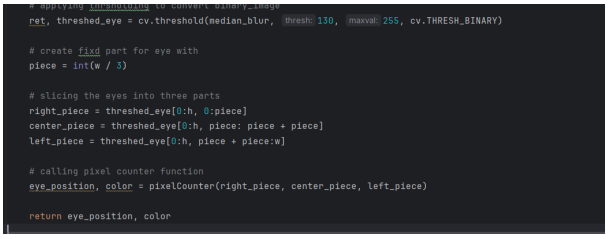



```
# vertical line
lv_top = landmarks[left_indices[12]]
lv_bottom = landmarks[left_indices[4]]

rhDistance = euclaideanDistance(rh_right, rh_left)
rvDistance = euclaideanDistance(rv_top, rv_bottom)

lvDistance = euclaideanDistance(lv_top, lv_bottom)
lhDistance = euclaideanDistance(lh_right, lh_left)

reRatio = rhDistance / rvDistance
leRatio = lhDistance / lvDistance

ratio = (reRatio + leRatio) / 2
return ratio
```

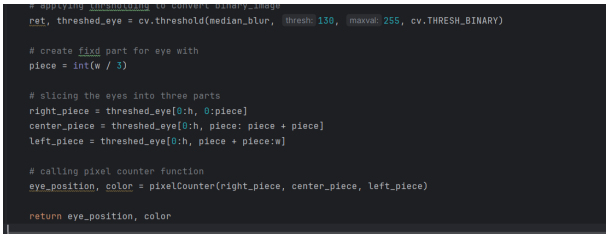
The eyesExtractor function takes an image (img) and preproceesed to grey scale and coordinates of the right and left eye regions, creates masks based on these coordinates, extracts the eyes' regions from image using bitwise operations, and returns the cropped images of the right and left eyes and this will be used to estimate the eyes position based on these unified coordinates .



The positionEstimator function takes a cropped eye image, performs noise reduction using Gaussian and median blurring, applies thresholding to convert the image into a binary format, divides the eye into three parts horizontally, and calculates the eye position based on the distribution of pixels among these sections. Finally, it returns the estimated eye position.



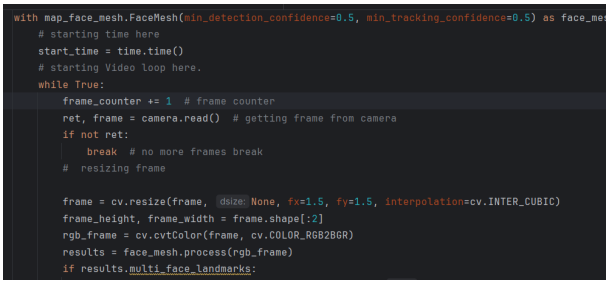
so now we will divide the eye position width into three parts left which is the first part of the width the center which is the middle and the last one is the right position 0 and these results will be used in pixelcounter Function to siplay the eyes position.



This function leverages facial landmarks' coordinates for eyes, eyebrows, and lips to estimate emotions based on characteristic patterns observed in these facial regions. for example if the eyebrows coordinates are very high this means that this person is surprised

i implemented this simple logic to indicate the stress level . So if someone is blinking than much more than once in a second this means that he might be stressed and if he is surprised or angry or sad the stress level is a bit high and lastly if he is looking right or left he might be stressed.

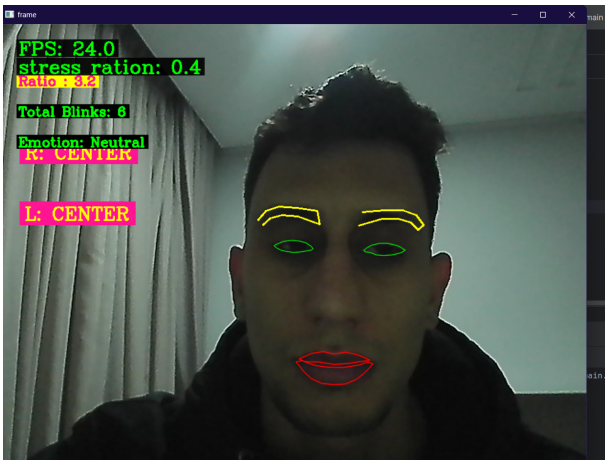
This code creates a loop that captures frames from the camera, resizes them, converts them to the required format, and processes them through the FaceMesh model to detect facial landmarks in real-time.



now we will get the coordinates of facial landmarks and extract the blink ratio in each frame.then if the person the eyes are closed for more than three frames it doesn't he is blinking his eyes are just closed.

Now we will estimate facial landmarks positions and coordinates and we will try to display them. and after getting the coordinates and the eyes position we will estimate emotions and stress ratio based on provided simple formula stress .

this is the how the video will be processed in real time. you can quit pressing q.



Conclusion

In conclusion, while the current code provides a simple basic solution for stress detection through facial expressions and eye behaviors, I recommend leveraging domain expertise from psychologists to accurately interpret specific facial expressions. Stress determination is and might not rely on a few parameters like blinks, considering scenarios where excessive blinking may not necessarily indicate stress.
Moreover, this solution might have limitations in low-light conditions or with different camera setups, affecting its accuracy. I propose developing a comprehensive machine learning model which can take months or maybe years Utilizing sequence models combined with domain expert insights could enhance stress detection accuracy by effectively labeling and extracting information from facial expressions. Collaborating with psychologists could help leading to a more robust stress detection system.