DRIVER DROWSINESS DETECTION USING OPENCV

By Julius Asante Kyere and Isaac Ferkah

1. Introduction

Drowsy driving is the dangerous combination of driving and sleepiness or fatigue. This usually happens when a driver has not slept enough, but it can also happen due to untreated sleep disorders, medications, drinking alcohol, or shift work. Makes drivers less able to pay attention to the road. Over 100,000 deaths are caused yearly due to drivers sleeping behind the wheels [1].

This project focuses on detecting drowsiness in drivers using OpennCV and dlib

2. Related Work

Researchers have attempted to determine driver drowsiness using the following measures: (1) vehicle-based measures; (2) behavioral measures and (3) physiological measures [2]. A detailed review on these measures will provide insight on the present systems, issues associated with them and the enhancements that need to be done to make a robust system [2]. The various ways through which drowsiness has been experimentally manipulated is also discussed [2]. It is concluded that by designing a hybrid drowsiness detection system that combines non-intrusive physiological measures with other measures one would accurately determine the drowsiness level of a driver. A number of road accidents might then be avoided if an alert is sent to a driver that is deemed drowsy [2].

3. Methodology

We need the SciPy package so we can compute the Euclidean distance between facial landmarks points in the eye aspect ratio calculation. We also imported the Thread class so we can play the alarm in a separate thread from the main thread to ensure our script doesn't pause execution while the alarm sounds. In order to actually play our WAV/MP3 alarm, we installed the playsound library, a pure Python, cross-platform implementation for playing simple sounds.

To detect and localize facial landmarks we imported the dlib library. Next, we defined our sound_alarm function which accepts a path to an audio file residing on disk and then plays the file. We also defined the eye_aspect_ratio function which is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmarks.

The returned value of the eye aspect ratio will be approximately constant when the eye is open. The value will then rapid decrease towards zero during a blink. If the eye is closed, the eye aspect ratio will again remain approximately constant, but will be much smaller than the ratio when the eye is open.

We defined the EYE_AR_THRESH . If the eye aspect ratio falls below this threshold, we'll start counting the number of frames the person has closed their eyes for. If the number of frames the person has closed their eyes, exceeds EYE_AR_CONSEC_FRAMES, which is 48 consecutive frames, the alarm will sound.

We defined COUNTER , the total number of consecutive frames where the eye aspect ratio is below EYE_AR_THRESH. If COUNTER exceeds EYE_AR_CONSEC_FRAMES , then wel update the boolean ALARM_ON.

We then instantiate our VideoStream using the supplied --webcam index. We then pause for a second to allow the camera sensor to warm up. We then start looping over frames in our video stream.

We read the next frame, which we then preprocess by resizing it to have a width of 450 pixels and converting it to grayscale. We apply dlib's face detector to find and locate the face(s) in the image. The next step is to apply facial landmark detection to localize each of the important regions of the face.

We loop over each of the detected faces on Line 85 — in our implementation (specifically related to driver drowsiness), we assume there is only one face — the driver — but I left this for loop in here just in case you want to apply the technique to videos with more than one face.

For each of the detected faces, we apply dlib's facial landmark detector and convert the result to a NumPy array. Using NumPy array slicing, we can extract the (x, y)-coordinates of the left and right eye, respectively.

Given the (x, y)-coordinates for both eyes, we then compute the eye aspect ratios. We can then visualize each of the eye regions on our frame by using the cv2.drawContours function — this is often helpful when we are trying to debug our script and want to ensure that the eyes are being correctly detected and localized.

We then make a check to see if the eye aspect ratio is below the "blink/closed" eye threshold, EYE_AR_THRESH. If it is, we increment COUNTER , the total number of consecutive frames where the person has had their eyes closed. If COUNTER exceeds EYE_AR_CONSEC_FRAMES, then we assume the person is starting to doze off. Another check is made, to see if the alarm is on — if it's not, we turn it on. We take special care to create a separate thread responsible for calling sound_alarm to ensure that our main program isn't blocked until the sound finishes playing.

We then draw the text DROWSINESS ALERT! on our frame. Finally, we handle the case where the eye aspect ratio is larger than EYE_AR_THRESH, indicating the eyes are open. If the eyes are open, we reset COUNTER and ensure the alarm is off.

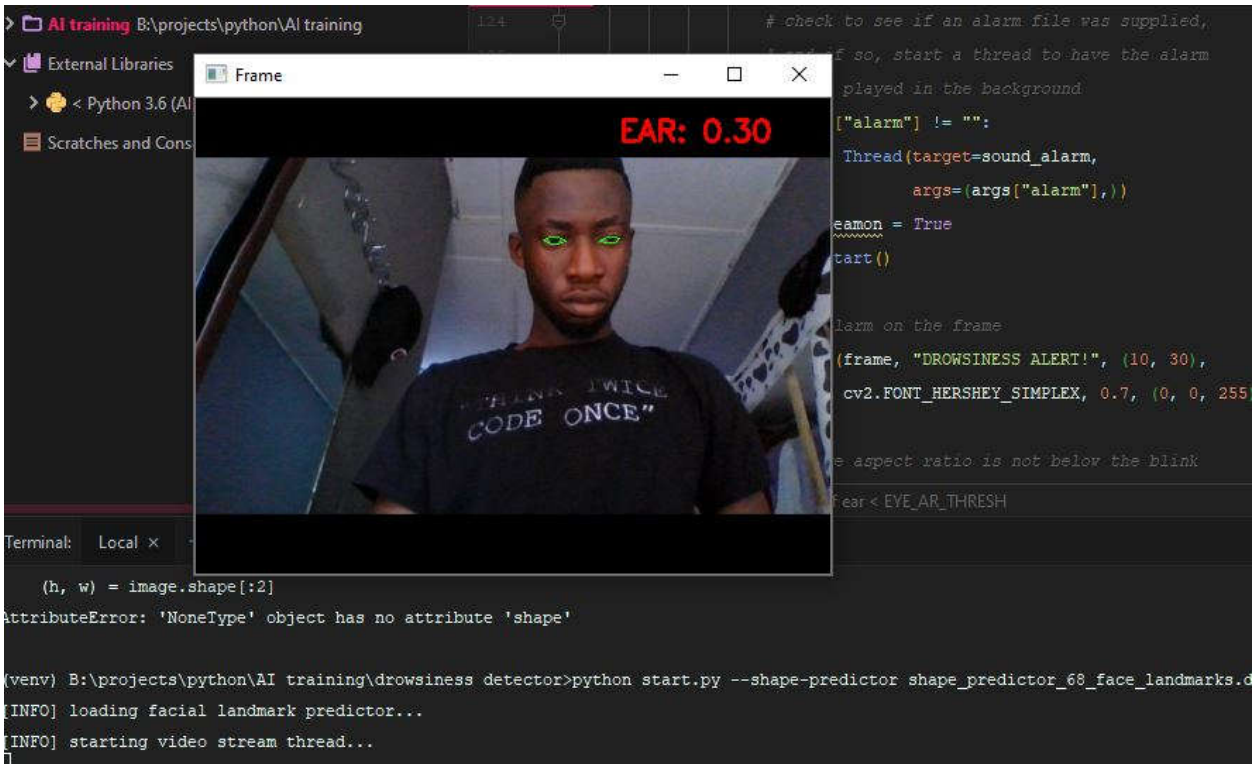

Fig 1 Drowsiness detection system

4. Conclusion

In conclusion, we successfully implemented a drowsiness detection algorithm which can be used to save a lot of life. This system is biased to people with small or sleepy eyes so for future works, we will train our model so it don't become biased. Also the model can be trained to also detect driver destruction.

REFERENCES

1. Accidents caused by drowsiness in 2018, available at https://medicalxpress.com/news/2018-11-sleepy-drivers-involved-year.html on 14th September, 2019
2. Arun Sahayadhas, Kenneth Sundaraj & Murugappan Murugappan “Detecting Driver Drowsiness Based on Sensors”, Sensors, 7 December 2012