DROWSINESS DETECTION SYSTEM USING OPEN-CV

MINOR PROJECT REPORT

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

Saransh Pandita	Tanishq Malhotra	Vedant Sahrawat	Deepika Singh
Enrollment No.:	Enrollment No.:	Enrollment No.:	Enrollment No.:
06815002716	07615002716	36215002716	00315002717

Guided and mentored by

Ms. Shalu

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)

DELHI – 110058

November 2019

CANDIDATE'S DECLARATION

It is hereby certified that the work which is being presented in the B. Tech. minor project report entitled **"DROWSINESS DETECTION SYSTEM USING OPEN-CV"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science and Engineering** of **MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY, Janakpuri, Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi)** is an authentic record of our own work carried out during a period of **August 2019 to November 2019** under the guidance of **Ms. Shalu, Assistant Professor**.

The matter presented in the B. Tech. minor project report has not been submitted by me for the award of any other degree of this or any other institute.

Saransh Pandita	**Tanishq Malhotra**	**Vedant Sahrawat**	**Deepika Singh**
Enrollment No.:	**Enrollment No.:**	**Enrollment No.:**	**Enrollment No.:**
06815002716	**07615002716**	**36215002716**	**00315002717**

CERTIFICATE

This is to certify that the above declaration made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

Ms. Shalu **Dr. Koyel Datta Gupta**

Assistant Professor **HOD, CSE Department**

The B. Tech. Minor Project Viva-Voce Examination of **Saransh Pandita (Enrollment No.: 06815002716), Tanishq Malhotra (Enrollment No.: 07615002716), Vedant Sahrawat (Enrollment No.: 36215002716), and Deepika Singh (Enrollment No.: 00315002716)**, has been held on . . . . . .

Project Coordinator **Project Coordinator** **Signature of External Examiner**

ACKNOWLEDGEMENT

We express our deep gratitude to **Ms. Shalu, Assistant Professor, Department of Computer Science and Technology** for her valuable guidance and suggestions throughout our project work. We would also like to extend our sincere thanks to our **Head of the Department, Dr. Koyel Dutta Gupta**, for her time to time suggestions to complete our project work.

Last but not the least, we would also like to thank our parents, friends, and colleagues who helped us a lot in finalizing this project within the limited time frame.

Saransh Pandita	**Tanishq Malhotra**	**Vedant Sahrawat**	**Deepika Singh**
Enrollment No.:	**Enrollment No.:**	**Enrollment No.:**	**Enrollment No.:**
06815002716	**07615002716**	**36215002716**	**00315002717**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

- EAR: Eye Aspect Ratio
- HOG: Histogram of Oriented Gradients
- PERCLOS: Percentage of Eye Closure
- Open-CV: Open Source Computer Vision

ABSTRACT

A real-time program to detect a vehicle driver's drowsiness, using a video sequence from a standard camera is built and implemented using Open-CV. The program uses the *dlib* library, or more specifically its face detection and facial landmark predictor. We use the detected landmarks to reliably estimate the level of eye openness, using the algorithm to extract a single quantity – eye aspect ratio, or EAR – which is used as measure of the eye openness in each frame. Finally, open or closed eye state is detected by applying standard logic checks i.e. comparisons with a minimum threshold value of the EAR calculated, and then identifying periods of drowsiness by comparing number of frames with low EAR. The program then issues a loud alert to wake the driver up. The program shows reliable and consistent results for real-time testing.

INTRODUCTION

After continuous driving for a long time, drivers easily get tired resulting into driver fatigue and drowsiness. Fatigue is defined as a gradual and cumulative process associated with 'a loss of efficiency, and a disinclination for any kind of effort'. Fatigue increases as time-on-task progresses. Research studies have stated that majority of accidents occur due to driver fatigue; it is estimated that nearly 20% of all accidents is caused by fatigue as per the Royal Society for Prevention of Accidents (RoSPA). Different countries have different statistics for accidents that occurred due to driver fatigue. According to the report by "Ministry of Road Transport & Highways", there were 4,552 accidents reported in India in 2016, that took lives of thousands of people because of sleepy drivers (as per "Road Accidents in India", 2016). Developing technology for detecting driver fatigue to reduce accident is the main challenge. Detecting drowsiness of drivers is still an ongoing research in order to reduce the number of such mishaps and accidents.

Our program aims to provide a solution to this problem by using currently popular technologies and resources, such as Open-CV and the *dlib* library, especially its facial feature recognition modules, to build a reliable and consistent driver alert system to ensure that the driver is able to drive to a safe place in case of fatigue. The program's main logic revolves around the scalar quantity known as Eye Aspect Ratio, and its direct relation with eye openness or closedness, to identify a drowsy state.

However, this program should not be treated as a permanent fix, but as a contingency; the alert system is built to ensure that the driver stays awake and alert while he/she drives to a place where they can recover from fatigue. Drivers should refrain from driving unless they are certain that they are no longer drowsy.

DROWSINESS AND ITS DETECTION

1.1. What is drowsiness?

Drowsiness or sleepiness can be described as a biological state where the body is in-transition from an awake state to a sleeping state. At this stage, a driver can lose concentration and be unable to take actions and/or have slower reflexes. There are obvious signs that suggest a driver is drowsy, such as:

- Frequently yawning.
- Inability to keep eyes open.
- Swaying the head forward.
- Face complexion changes due to blood flow.

A number of studies recommend countering drowsiness by taking naps between trips, consuming caffeine (coffee, energy drinks etc.), or driving with company.

1.2. How to determine level of driver drowsiness?

There are various measures to determine the level of driver drowsiness. These measures can be grouped into three major categories:

1. Physiological Measures
2. Vehicle-based Measures, and
3. Behavioural Measures

Physiological measures involve the usage of electronic monitoring devices to assess the driver's condition. These devices include Electroencephalography (EEG), Electrocardiography (ECG) and Electrooculogram (EOG). Although physiological measures are extremely accurate, they are not widely accepted due to practical implementations.[2] However, with the advancement in wearable smart devices, physiological measures may become more feasible in the future.

For vehicle-based measures, driver's drowsiness is analysed based on vehicle control systems, which could include steering wheel movements, braking patterns, and lane departure measurements, with steering wheel movement monitoring being more accurate than the other methods in this type of measure. While vehicle-based measures

are non-invasive and easier to implement, external factors such as road conditions and driving skills affect the accuracy of such measures to an extent where results may no longer be accurate or reliable.

Behavioural measures consist of behavioural or computer vision measures. These measures tend to be more reliable than vehicle-based measures since they focus on the driver rather than the vehicle. Furthermore, behavioural measures are non-invasive as well, and therefore more practical than physiological measures, and is one of the main reasons why they are becoming a popular way of drowsiness detection. Here, the information is obtained by using sensors and cameras to detect slight changes in the driver's facial expressions.

We will be using behavioural measures in our project, for the aforementioned reasons.

1.3. Behavioural Methods in Driver Drowsiness Detection

Behavioural methods measure levels of drowsiness using mounted cameras in the vehicle to observe the driver's facial features, such as eye state, blinking rate, yawning, and head movement. Most algorithms use a general method to identify and extract said features from the camera feed. Once these features have been obtained, further processing is carried out on these features, typically involving machine learning techniques such as Support Vector Machines (SVM), Convolutional Neural Networks (CNN) or Hidden Markov Models (HMM).[2] We will, however be using a different approach for further processing and drowsiness detection.

Facial features that are typically extracted from the driver's face include the following:

1. **Eye Closure Analysis:** The eye state is a very common and accurate feature used to determine driver drowsiness. The most common methods used to measure the level of drowsiness are the Percentage of Eye Closure (PERCLOS) and Eye Aspect Ratio (EAR). PERCLOS is the percentage of eye closure over a period of time. In contrast, EAR is the ratio between the height and width of the eye. The concept of EAR was introduced by Soukupova and Cech in 2016 [1], and it is the measure that we will implement in our project. The primary difference between PERCLOS and EAR is that PERCLOS has a more discrete value range, while EAR has a continuous value range. This allows for a more

accurate control over the threshold value which we use to define the difference between drowsy and not drowsy states.

2. **Eye Blink Analysis:** Methods that study the eye blinks use the frequency and duration of eye blinks to measure drowsiness. For example, the normal blinking rate per minute is roughly 10. A drowsy person would blink less i.e. the blinking rate would decrease for drowsy state.
3. **Yawning Analysis:** Yawning can be caused by fatigue or boredom. In drivers, yawning usually indicates that they might fall asleep while falling. Similar to eye closure analysis, methods can measure the widening of the mouth to detect yawning traits in the driver by tracking mouth shape and position of lip corners.
4. **Facial Expression Analysis:** This type of analysis makes use of a combination of multiple facial features to detect drowsiness. These features include wrinkling of the forehead, extreme head poses and involuntary jerking of the head.

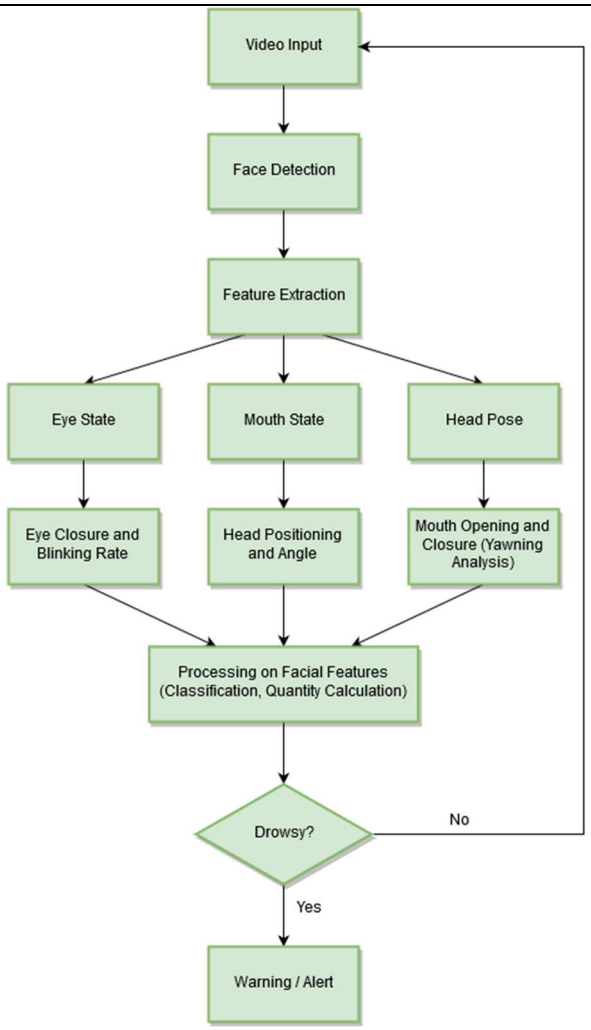


Fig. 1.1. Driver Drowsiness Detection Process

We will be using Eye Closure Analysis, or more specifically, Eye Aspect Ratio for our project. EAR calculation and implementation is comparatively easier, and is more accurate than other analysis methods across all processing methodologies.

1.4. Eye Aspect Ratio (EAR)

Eye Aspect Ratio, or EAR, is a metric devised by Soukupova and Cech in their 2016 paper on eye blink analysis. [1] In traditional image processing methods for eye closure analysis, the algorithm involves differentiating the iris and sclera of the eye, and then checking for frames where the sclera disappears, indicating that the eye is closed. EAR is a much more optimised and streamlined solution, which involves a simple calculation based on the ratio of distances between facial landmarks of the eyes.

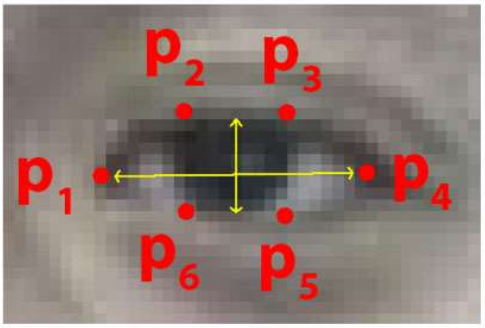


Fig. 1.2. The six facial landmarks associated with the eye.

The eye aspect ratio (EAR) is essentially the ratio of the distances between the height and width of the eye. Since there are four landmarks corresponding to the height of the eye and only two landmarks for the width of the eye, the scalar distance corresponding to the width of the eye is taken twice into consideration, while the two scalar distances corresponding to the height of the eye are added.

Mathematically, this is expressed as

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

where p1, p2, … , p6 are the 2D facial landmark locations for the human eye, as depicted in figure 1.2.

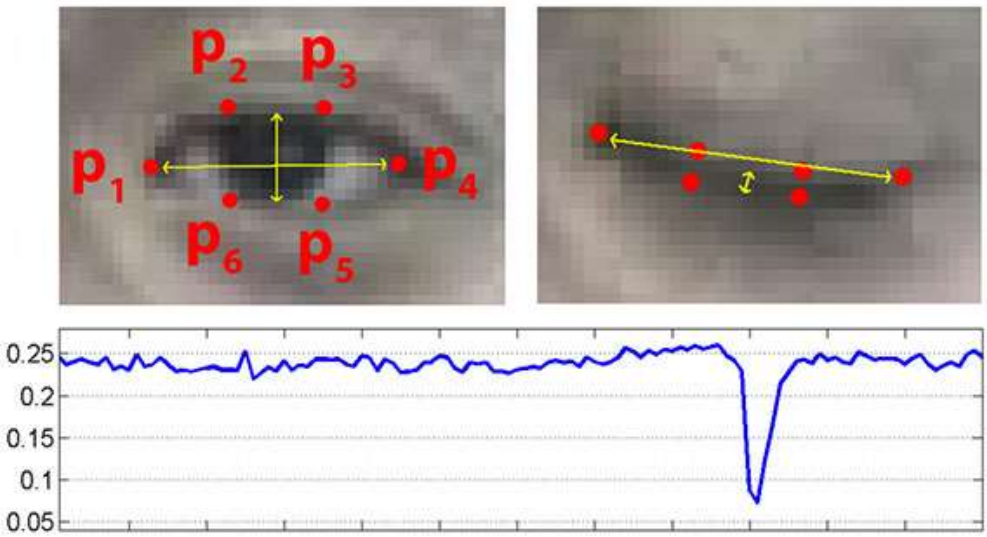


Fig. 1.3. *Top-left:* Visualisation of an open eye with landmarks. *Top-right:* Visualisation of a closed eye with landmarks. *Bottom:* Plotting EAR over time. The dip in the values indicates a blink. (*Courtesy: Soukupova and Cech)* [1]

As shown by figure 1.3., the dip in EAR value corresponds to the closing of the eye. We will use this property in our program; a long dip in EAR values below a specific threshold value would indicate that the eyes were closed for a longer interval of time, indicating drowsiness.

LITERATURE SURVEY

In order to detect drowsiness of drivers, numerous approaches have been proposed. This section summarises the existing approaches to detect drowsiness.

M. Ngxande et al. [2] presented a literature review of driver drowsiness detection based on behavioural measures using machine learning techniques. It starts by listing different types of behaviour that a person displays when he/she feels drowsy. These behaviours are generally observable on the face of the person or by the effect on the driving behaviour of the person. The paper then explains why facial features based behavioural measures are better than vehicular features for determining driver drowsiness.

They then proceed to explain how there were many facial features that could be extracted from the face to infer the level of drowsiness, including eye blinks, head movements and yawning.

They then talk about work on driver drowsiness detection in the past, and how the recent rise of deep learning and related technologies is causing researches to revisit their techniques in order to improve and re-evaluate the accuracy of their algorithms regarding drowsiness detection.

The paper then reviews machine learning techniques which include support vector machines, convolutional neural networks and hidden Markov models in the context of drowsiness detection. Furthermore, a further analysis was conducted on 25 papers that use machine learning techniques for drowsiness detection, comparing the different techniques employed in those papers.

The analysis revealed that support vector machine technique was the most commonly used technique to detect drowsiness, but convolutional neural networks performed better than the other two techniques. Finally, the paper listed some publicly available datasets at the time that could be used for benchmarking future drowsiness detection algorithms.

In her paper, T. Soukupova [1] presented a new measure for identification of eye blinks, which it referred to as Eye Aspect Ratio or EAR, and then proposed a real-time algorithm to detect eye blinks in a video sequence from a standard camera.

The paper begins by explaining eye blinks, including how and why they happened, some statistics related to eye blinks and their parameters, namely frequency and duration, and the effect of external factors, including drowsiness and fatigue, on these parameters.

The paper then reviews existing methods for identifying eye blinks, including and not limited to intrusive sensor-based methods and non-intrusive static methods. They then explain the shortcomings of these methods, and therefore the need for a dynamic non-intrusive method for eye blink identification. The paper also briefly explains PERCLOS, one of the few dynamic methods which was close to their own method, and its use in existing drowsiness detection methods.

They then introduce us to a scalar quantity of their creation, the Eye Aspect Ratio. Soukupova explains how EAR works, and the two major approaches towards its implementation in eye blink detection, which were:

1. Supervised training process using a Support Vector Machine classifier trained on the training dataset

2. Unsupervised training process, modelled after the Hidden Markov Model.

The paper then explains both processes, before comparing the methodologies and their relative accuracy in different conditions and datasets. Based on this analysis, Soupukova builds a new algorithm using a simple state machine recognizing the blinks according to the eye closure lengths, which she then analyses in extreme detail, with explanation of all external factors on EAR and her algorithm based on its result on three standard datasets. This also includes comparison with previous eye blink detection methods in similar conditions. The testing reveals that the proposed algorithm has comparable results with the state-of-the-art methods mentioned previously in the paper.

The paper concludes with details on the algorithm's testing and future plans for its use in different applications, including drowsiness detection.

S. Sangle et al. [6] presented an approach and implementation that was more application – based; their paper explains the implementation of a Haar Cascade Classifier – based drowsiness detection system, and its deployment on the Raspberry Pi platform.

They used a camera fixed on the dashboard to capture and send images to a Raspberry Pi server installed in the vehicle. This server then used a Haar Cascade Classifier to detect faces and then isolated facial landmarks using the dlib library. Their implementation of the dlib library also uses EAR to identify drowsy and non-drowsy states of the driver.

The main work done by the paper was in the deployment of an algorithm like Soukupova's on a portable platform like the Raspberry Pi. The paper talks about the advantages and limitations of using a small-scale device for drowsiness detection, such as having to work on a low-FPS video feed, since the Pi could not handle high frames per second due to its computational limitations.

The work by Varghese et al. [7] can be treated as an extension of the previous paper in the sense that this paper is also about the implementation of an algorithm similar to Soukupova's on a Raspberry Pi. However, there are some key differences between the work done by Sangle et al. and Varghese et al.

The first difference is that Varghese used a Histogram of Oriented Gradients based classifier for face detection instead of a Haar Cascade Clasifier, which was used by Sangle. The second difference was that Varghese used some optimisations in order to avoid calculation of spatial distances that weren't required for EAR calculation. This was possible due to their use of HOG for facial landmark detection, since it returns an indexable list. Lastly, Varghese also worked on optimisation of EAR and drowsy frame thresholds in order to slightly improve the accuracy of their implementation over that of Sangle's.

However, in spite of all these optimisations, the performance of both implementations was almost indistinguishable, due to Raspberry Pi's hardware limitations overshadowing any software-based optimisations.

Jabbar et al. [5] in their paper, use an innovative deep learning based method that was deployed using an Android application. This paper was significantly important since not only was their implementation highly accurate, but they also managed to implement this drowsiness detection system onto a portable platform without any of the debilitating drawbacks of previous attempts made on similar platforms.

Their paper proposed a drowsiness detection system based on multilayers perceptron classifiers, built specifically for embedded systems, such as Android devices. Their Android application used a Convolutional Neural Network – based backend trained on the National Tsing Hua University (NTHU) Driver Drowsiness Detection Dataset. The facial feature extraction was again done using the dlib library. The computational limitations were also overcome by using a high end device for real-world testing of the applications.

The paper concluded with Jabbar et al. presenting statistics of real world testing showing an accuracy of 81%. It also discussed the limitations this implementation could face on lower end devices, or on devices with limited storage capacity due to the size of the CNN model.

METHODOLOGY

3.1. Objectives

Our main objective is **to build a program that can monitor driver drowsiness, and double as an alert system in case of drowsy state**. Based on our findings during the literature study, our project has the following sub - objectives:

1. To detect faces and extract relevant facial landmark features
2. To calculate EAR from the six facial landmarks corresponding to the eye
3. To monitor EAR values over a period of time and identifying periods of drowsiness
4. To issue an alert to wake up the driver and ensure his/her focus on the road

3.2. Rough Methodology

1. We use the *dlib* library's Histogram of Oriented Gradients (HOG)-based face detector to detect faces from the camera's video feed.
2. We then use *dlib*'s facial landmark detector to obtain an indexable list of facial landmarks.
3. From the list of facial landmarks, we extract the landmarks corresponding to the eye region.
4. We calculate EAR for both eyes, and then take an average of both EARs. This helps maintain consistency.
5. We then cross-check EAR obtained from the feed with a threshold value. When EAR is lower than threshold, a counter is incremented. When this counter's value crosses a particular value, the program issues an alert to the driver. The counter resets whenever EAR is above threshold.

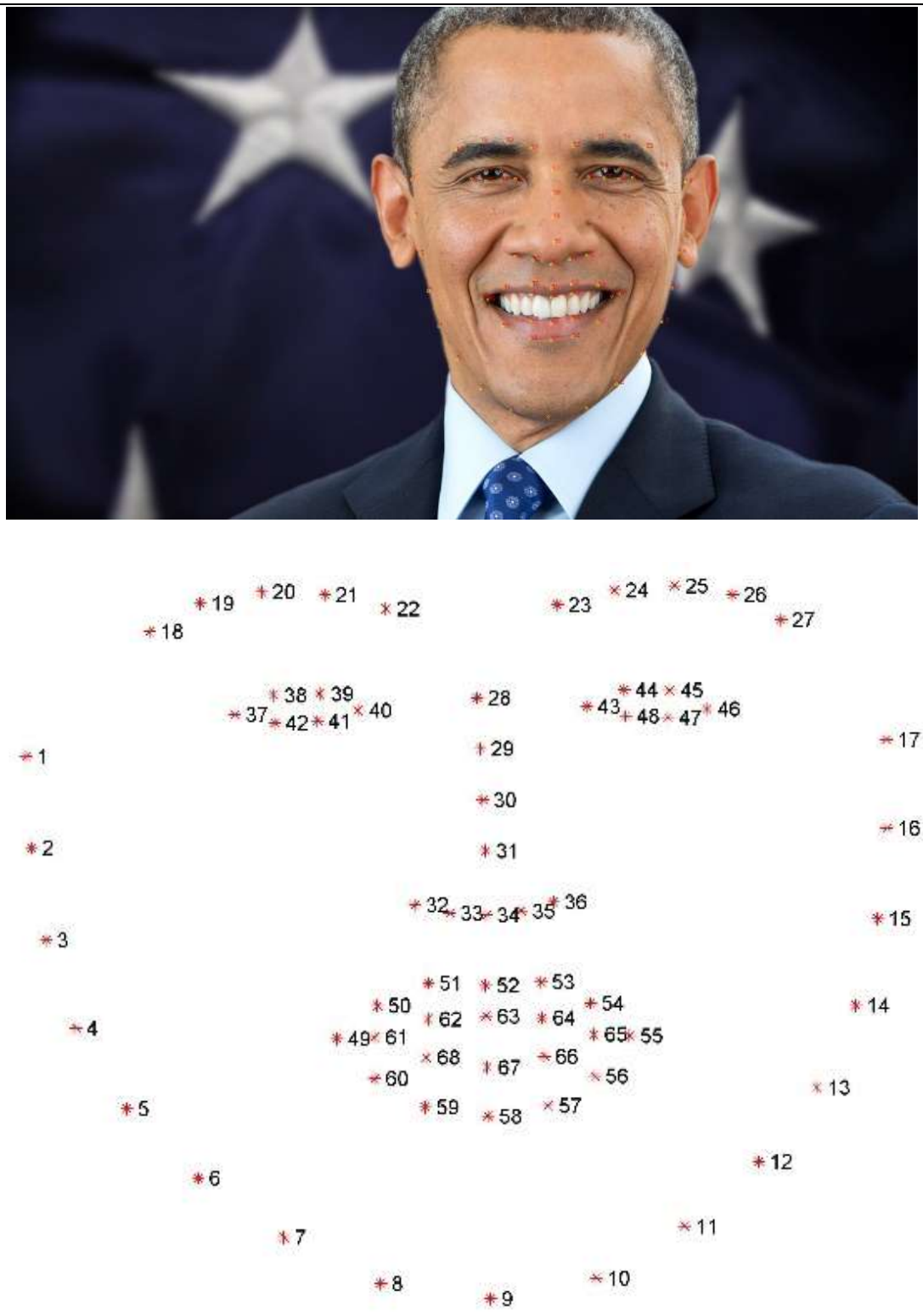


Fig. 3.1. *Top*: An example of facial landmark detection as done by *dlib*'s facial landmark detector. *Bottom*: Visualising the 68 facial landmarks coordinates from the iBug 300-W dataset.[3]

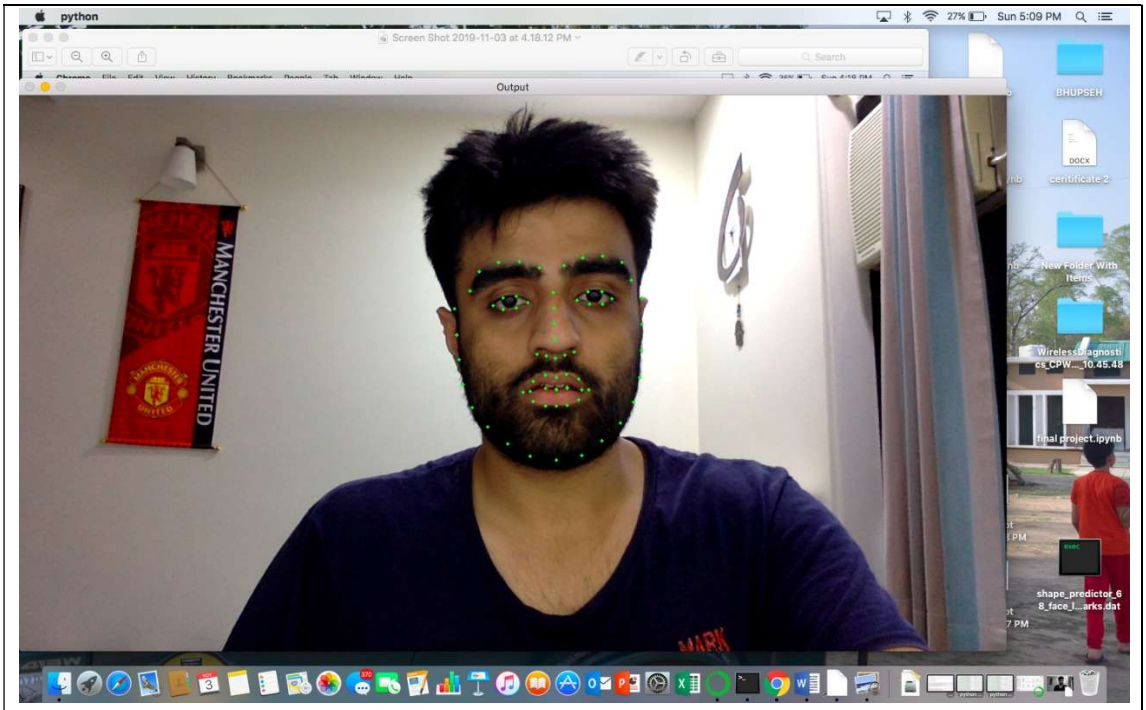


Fig. 3.2. Facial landmark detection implemented in our program

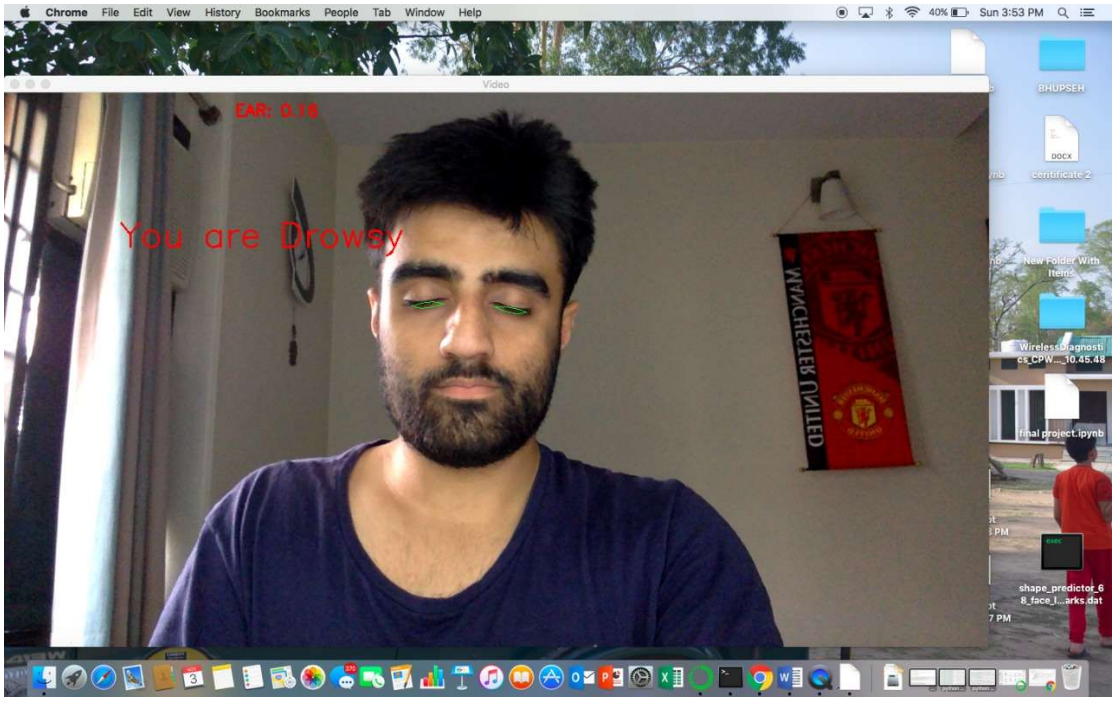


Fig. 3.3. *Top*: An example of the program detecting person in non-drowsy state. Note the EAR value on top in green, indicating the value being above threshold. *Bottom*: An example of the program detecting person in drowsy state. Note the EAR value on top in red, indicating the value being below threshold, and the text alert. The program also issues a loud audio alert.

3.3. Technologies and Modules Used

- **Python:** Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

  Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

  Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality.

- **Open-CV:** OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to

provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

Open-CV is an integral part of the drowsiness detection system, with multiple methods from Open-CV being implemented in the code. Some notable examples are:

- cv2.VideoCapture is used to connect the program to the camera and access its video feed. This feed is then stored as a collection of still images or frames, which can be used for further processing.
- cv2.flip is used to flip the image captured by the camera. It is used for mirroring, transformation and pixel swapping. In this case, cv2.flip is used for flipping the frames from the video stream. cv2.flip has multiple arguments in order to specify axis about which we need to flip the image. We have used the argument '1', meaning the frame flips corresponding to the y-axis.
- cv2.cvtColor is used to change the image coloration scheme. Here, we turn the image from coloured (or RGB) to grayscale, for better detection of facial landmarks.
- cv2.drawContours is used to detect the edges on a face and mark them accordingly using the facial landmarks. In other words, it detects the facial features like eyes, nose, lips, jaw line etc. This is used to highlight the eyes in the output screen, indicating that the program is detecting the eyes.

- cv2.putText is used to display text on our output screen. In our case, this is used to print EAR value on the output screen, as well as show alert text as and when required.
- cv2.imshow is used to generate the output screen.
- cv2.waitKey(1) waits indefinitely for a key to be pressed to close the output window and cv2.DestroyWindows is invoked to shut down the output screen and the webcam. We are using 'q' as our quit button i.e. pressing 'q' on the keyboard will shut the program down.

- ***dlib*:** *dlib* is a modern toolkit containing machine learning algorithms and tools for creating complex software to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high-performance computing environments. *dlib*'s open source licensing allows you to use it in any application, free of charge. Since development began in 2002, *dlib* has grown to include a wide variety of tools. As of 2018, it contains software components for dealing with networking, threads, graphical user interfaces, data structures, linear algebra, machine learning, image processing, data mining, XML and text parsing, numerical optimization, Bayesian networks, and many other tasks.

  The *dlib* library is essential for the drowsiness detection system, as it is this library that gives it the ability to distinguish facial features.

- **Histogram of Oriented Gradients (HOG):** The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

  HOG features were first introduced by Dalal and Triggs in their CVPR 2005 paper, Histogram of Oriented Gradients for Human Detection. In their work, Dalal and Triggs proposed HOG and a 5-stage descriptor to classify humans in still images. [8] The 5 stages include:

1. Normalizing the image prior to description.
2. Computing gradients in both the x and y directions.
3. Obtaining weighted votes in spatial and orientation cells.
4. Contrast normalizing overlapping spatial cells.
5. Collecting all Histograms of Oriented gradients to form the final feature vector.

The *dlib* library uses a HOG-based face detector. The reason HOG is so popular is because of its accuracy and relative ease of use in applications regarding object recognition.

The file "shape_predictor_68_face_landmarks.dat" file that is a part of the program's source code is *dlib*'s pre-trained facial landmark detector, which uses the concept of HOG. The path to this file is given to the program as an argument at run-time.

- **imutils Package:** The imutils package is a series of OpenCV + convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, and detecting edges, much easier to use with OpenCV and both Python 2.7 and Python 3**.**

  The package is developed and maintained by Adrian Rosebrock, also known for his website PyImageSearch.

  imutils includes the VideoStream sub-package, which we use for capturing the video stream from the camera. This sub-package also allows us to grayscale the input, which helps improve the facial landmark detector's accuracy.

  A subpackage/class of imutils is face_utils, which is used in our program for all the detection functions. This includes isolating faces from the video feed, identifying the facial landmarks, indexing the facial landmarks corresponding to the eyes, and conversion of the indexed landmarks to a Numpy array for EAR calculations.
- **Scipy:** SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and

engineering. SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.

We need the SciPy package in order to compute the Euclidean distance between facial landmarks points in the EAR calculation. This is done using scipy.distance.spatial, which is used to calculate the spatial distance between any two points in a plane. In our case, these points were the 6 facial landmarks corresponding to each eye.

- **NumPy:** NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

  It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

  - A powerful N-dimensional array object
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities

  Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

  Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

  In our program, a two-dimensional Numpy array is used for storing the x and y coordinates of the facial landmarks identified by OpenCV. This Numpy array and the stored coordinates stored in it are then used for calculation of EAR values.

- **PyGame:** Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language.

  In our program, PyGame's sound library-based functions are used to import the alert sound file, and then play it whenever an alert is to be issued.

3.4. Working of the Project

We begin our implementation of the project code by opening a new Python file labelled *detect_drowsiness.py*, and import the necessary libraries.

```
from scipy.spatial import distance

from imutils import face_utils

import numpy as np

import pygame

import time

import dlib

import cv2
```

The SciPy package is required in order to compute the Euclidean distance between facial landmarks points in the EAR calculation.

The imutils package is a user-made package consisting of CV and image processing functions used to make it easier to work with OpenCV in our code.

```
pygame.mixer.init()

pygame.mixer.music.load('audio/alert.wav')
```

The PyGame package is required in order to load the alert sound into the program. This alert sound is in the audio subdirectory of the folder where we created the Python program.

```
EYE_ASPECT_RATIO_THRESHOLD = 0.3

EYE_ASPECT_RATIO_CONSEC_FRAMES = 50

COUNTER = 0
```

As the name indicates, these variables are used to store the values essential for the working of the code. The first value specifies the minimum EAR threshold i.e. the value below which the program will identify the person as drowsy. The second value specifies the number of consecutive frames for which EAR value must be below the threshold value for the program to issue an alert. The counter for this is stored by the third variable.

```
face_cascade =
cv2.CascadeClassifier("haarcascades/haarcascade_frontalface_default.xml")
```

This statement is used to simply highlight the face detected by the program. This is a Haar Cascade classifier which is inbuilt in OpenCV.

```
def eye_aspect_ratio(eye):

    A = distance.euclidean(eye[1], eye[5])

    B = distance.euclidean(eye[2], eye[4])

    C = distance.euclidean(eye[0], eye[3])

    ear = (A+B) / (2*C)

    return ear
```

This eye_aspect_ratio function is essential for our program's functionality. This function is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmarks. The eye array used for calculation of these distances is from *dlib*'s facial landmark detector, which is in the file *shape_predictor_68_face_landmarks.dat*. The 68 major facial landmarks are stored in the form of an indexable list. Of these 68, the eye subarray is the 37^{th} - 42^{nd} index for the left eye, and 43^{rd} – 48^{th} for the right eye i.e. there are 6 facial landmark values for each eye.

The return value of this function will more or less remain constant throughout the operation of the program, with short, periodic dips towards zero, which correspond

to blinks. When the eye is closed, this EAR value will again stay constant. However, this value will be much lower than the value when the eye was open.

```
detector = dlib.get_frontal_face_detector()

predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS['left_eye']

(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS['right_eye']
```

As we previously mentioned, *dlib* is the library being used for facial landmark detection. Here, we call on this library to access its frontal face detector, which uses the concept of Histogram of Oriented Gradients for facial detection, and then use the facial landmark detector present to extract the indexable list of all 68 major facial landmarks. The face_utils class is a part of the *imutils* library, which basically is used to iterate the list based on index. The FACIAL_LANDMARKS_IDXS function is an inbuilt function of the face_utils class, which already knows which indexes to access based on the argument, which are the left and right eye in this case. The index for both eyes is then stored in a tuple for ease of future access.

```
video_capture = cv2.VideoCapture(0)

time.sleep(2)
```

cv2.VideoCapture is a function of OpenCV used to access the camera and getting the video feed from the camera. This video feed is used as input for the program.

```
while(True):

    #Read each frame and flip it, and convert to grayscale

    ret, frame = video_capture.read()

    frame = cv2.flip(frame,1)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Here, we capture each frame, flip it for normalisation since the camera is already receiving a mirrored input, and then switch the colour scheme of the frame to grayscale. Grayscaling the frame has been noted to facilitate better facial landmark detection.

```
#Detect facial points through detector function

  faces = detector(gray, 0)

  #Detect faces through haarcascade_frontalface_default.xml

  face_rectangle = face_cascade.detectMultiScale(gray, 1.3, 5)

  #Draw rectangle around each face detected

  for (x,y,w,h) in face_rectangle:

    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
```

This part of the program is for highlighting the face being actively monitored by the program via the camera. This is shown on the screen by plotting a red box around the face. The colour is indicated by the RGB argument specified for the cv2.rectangle command. The actual detection is being done by *dlib*'s facial detection algorithm, which was referenced to the detector variable earlier in the code.

```
  for face in faces:

    shape = predictor(gray, face)

    shape = face_utils.shape_to_np(shape)
```

Here, we run the predictor on the captured the frame and then convert the detected face into a NumPy array which basically is a two-dimensional array that lists intensity in its particular grid coordinate. To explain further, in a grayscale image, each pixel can be on a scale of black to white consisting of 256 divisions, with 0 indicating black and 255 indicating white. Similarly, the shape_to_np function converts the captured face to a grid, and then identifies the most prevalent shade in

the sub division. The corresponding grayscale value is then stored in a NumPy array. This allows for quantification of the image to terms that can be understood by the system.

```
leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

leftEyeAspectRatio = eye_aspect_ratio(leftEye)

rightEyeAspectRatio = eye_aspect_ratio(rightEye)

eyeAspectRatio = (leftEyeAspectRatio + rightEyeAspectRatio) / 2
```

Now we use the tuples containing the index for the left and right eyes to create two lists; one for each eye. This list basically contains the coordinate values corresponding to the eye region as identified using the Numpy array we talked about earlier.

These values are then used for calculating EAR independently for both eyes. Then an average of both values is used for further calculations. This is simply a step used to make the calculations more consistent.

```
leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)

cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

This part of the code is basically used to highlight the eyes on the face captured by the HOG facial detector. This is done using a green contour around the eyes, as specified by the RGB argument of the cv2.drawContours command. The cv2.convexHull is to remove any convex contour discrepancies caused due to the convex shape of the eye, which may cause incomplete or incorrect highlighting.

```
    if(eyeAspectRatio < EYE_ASPECT_RATIO_THRESHOLD):

        COUNTER += 1

        if COUNTER >= EYE_ASPECT_RATIO_CONSEC_FRAMES:

            pygame.mixer.music.play(-1)

            cv2.putText(frame, "You are Drowsy", (150,200),
            cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,0,255), 2)

    else:

        pygame.mixer.music.stop()

        COUNTER = 0
```

This is the part where the main comparison takes place. If the EAR calculated by the program is lower than the specified threshold value, the program will increment the counter by 1 for each consecutive frame where the EAR is below the threshold. The moment the counter passes the frame threshold i.e. the limit for number of consecutive frames where EAR was below threshold, PyGame will play the alert sound, while OpenCV will display the alert text on the screen of the platform on which the program is running. If and when the next frame has an EAR value above the threshold, which will usually happen once the driver wakes up, the counter is reset and the program stops playing the alert and displaying the text on screen.

```
    cv2.imshow('Video', frame)

 if(cv2.waitKey(1) & 0xFF == ord('q')):

        break

video_capture.release()

cv2.destroyAllWindows()
```

This is the implementation of the frontend i.e. the window visible to the driver. cv2.imshow basically shows the camera feed on the screen with all augmentations as done by the program beforehand (highlighting the eyes and displaying the real time EAR value). Once the quit button is pressed (q in this case), OpenCV stops capturing video feed, releases the concerned resources and closes the window.

IMPLEMENTATION

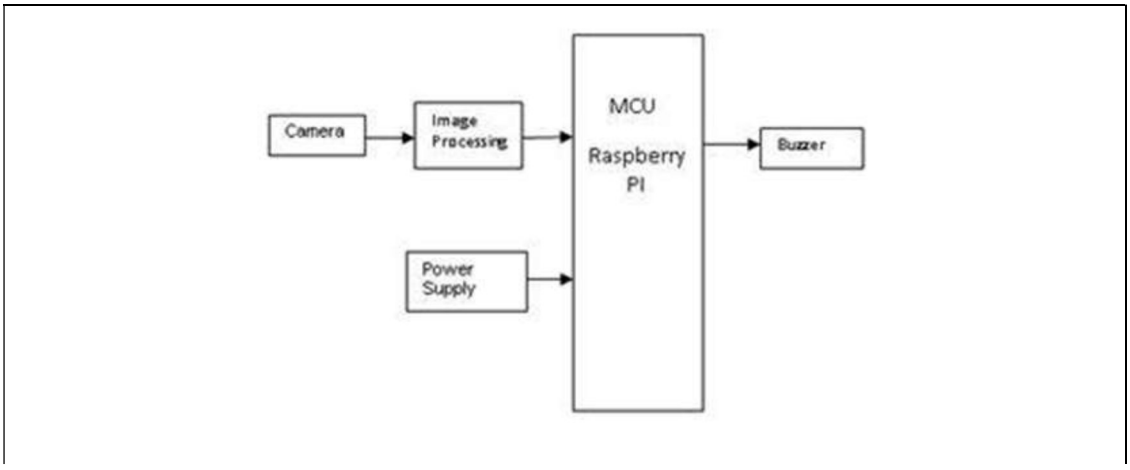
4.1. Prototype Implementation Using Laptop

The program was successfully tested by running it on a laptop with a built-in webcam for camera feed. This implementation is ideal for testing purposes since the developer is able to debug and make changes on the spot. However, this type of implementation is not ideal for real-world applications, since it is not feasible to carry a laptop along while driving at all times due to possible space constraints in case of multiple passengers.

The real world testing for our program was done on an Apple Macbook Pro running Python 3.7. This installation of Python was done using the Anaconda Distribution version 2019.09, which came with Numpy 1.16.2. OpenCV 4.1.1 and Dlib 19.18 were installed for use in the program. The Python program was compiled on Jupyter Notebook 5.7.7.

4.2. Possible Implementation on a Raspberry Pi

In such cases, miniature computing platforms, such as the Raspberry Pi, would work perfectly. The Raspberry Pi is not only affordable, making the real-world implementation cheaper, its form factor also allows for easy physical installation in a vehicle. The camera would be provided separately, allowing for better servicing and maintenance.



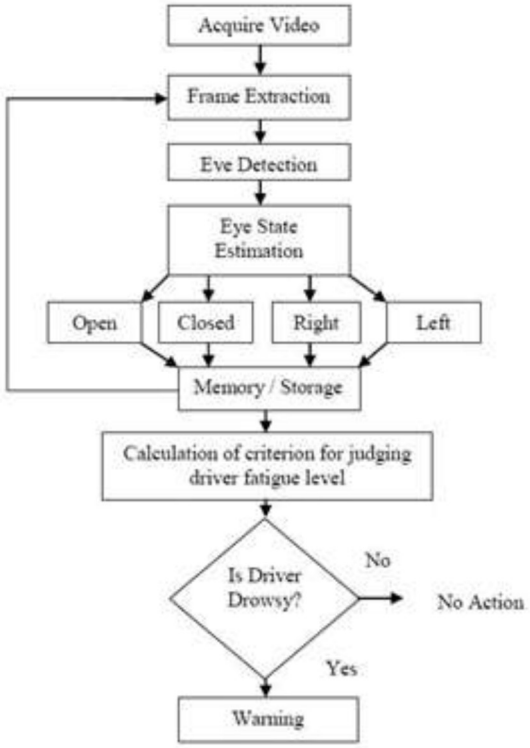


Fig. 4.1. *Top:* Rough Block Diagram of Raspberry Pi based implementation. *Bottom:* Working of algorithm within MCU. The warning block at the end leads to the buzzer.[6]

However, as was concluded during the work by Sangle et al [6] and Varghese et al [7], a huge limitation for implementation of any real time input based system on such platforms is the low-end specifications of the hardware. For example, the Raspberry Pi 3 ships with only 1 GB of RAM, which is used by the OS and other devices as well. This severely impacts the *dlib* library's performance on this platform.[3] Getting a platform with better specifications may sound simple, but doing so will impact its cost factor. One can use some software optimisations to run *dlib* better, such as increasing swap file size, but the impact on the hardware due to such optimisations may break the platform itself.

One of the best implementations of the driver drowsiness detection system on a Raspberry Pi was done by Adrian Rosebrock.[9] He did this by swapping out the HOG + Linear SVM-based face detector for a Haar cascade. Haar cascades, while less accurate, are significantly faster than HOG + Linear SVM detectors, which meant that not only was facial feature detection faster, it also consumed less resources. The alert system was a Raspberry Pi TrafficHAT attachment connected to the system. The

TrafficHAT comes with 3 bright LEDs and a loud buzzer, and is highly optimised for such systems, meaning the alert itself wouldn't cost the Pi any significant resources.

4.3. Possible Implementation on Android Platform

This is the ideal implementation for the system. In this, the program is integrated into an application for the Android platform, and the user can simply use his/her device as a platform for running the system using the application. This also allows for better integration with other globally used applications for further improving the user experience.

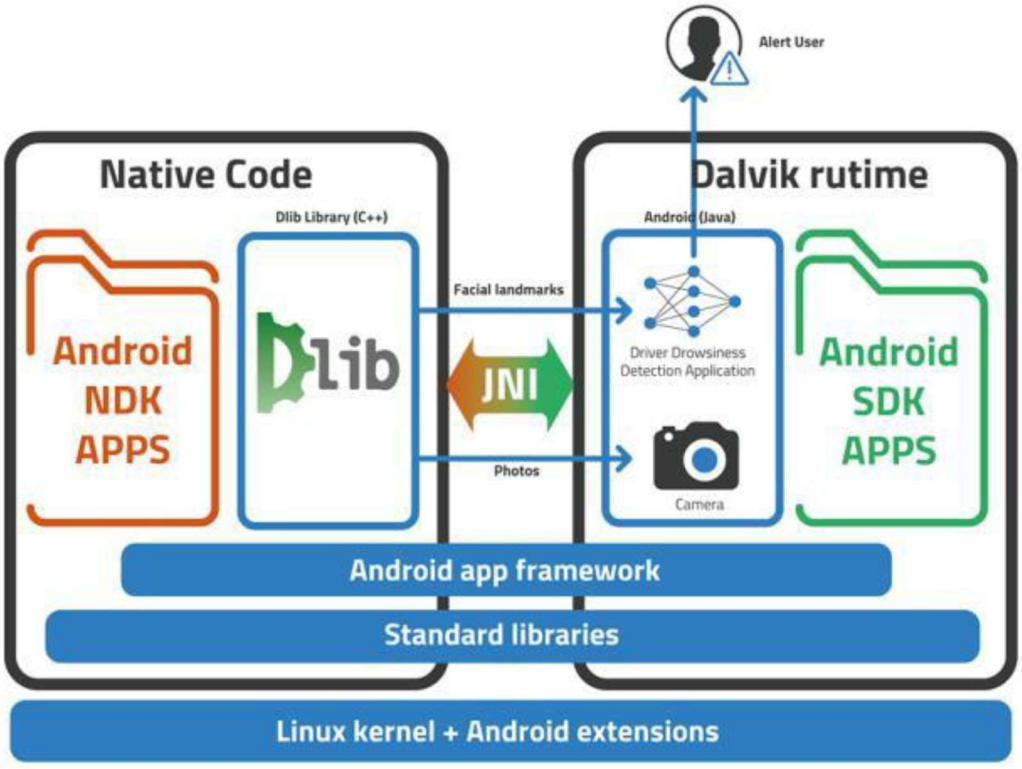


Fig. 4.2. Proposed architecture for deployment on Android Devices [5]

A notable example of possible deployment of our program onto Android devices is the implementation by Jabbar et al. Here, the Android mobile camera is given permission to take facial pictures of the driver. After taking the picture, it will be transferred to the Dlib Library. Here, the Java Native Interface (JNI) framework serves to relate and pass information between the Android application, which uses Java language, and Dlib Library written in C++ languages. [5] The Dalvik runtime interface also allows for a modular design, facilitating future expansion and integration with other applications.

RESULT

As previously mentioned, the project was tested by deploying the program on a laptop computer, in this case an Apple Macbook Pro. Its webcam was used for obtaining video input. During testing, the program's accuracy was fairly accurate provided ideal conditions i.e. an adequate resolution of the video feed with proper lighting and placement of target face, and results were comparable to other similar implementations done by other parties.

In order to check the impact of external factors on the system, we manually tweaked these aforementioned factors, and checked difference between control readings for the system, and the readings obtained under test conditions. The differences measured were in accuracy of the program (whether the program is able to distinguish drowsy and non-drowsy states during testing) and response time (time between driver feeling drowsy and the issuing of the alert).

All tests returned results that were within scope. Low or excessive brightness/illumination of the driver's face impacted the accuracy of the program, since the facial landmark detector was finding it difficult to differentiate facial contours due to bad contrast of the frames, which was further heightened during quantisation due to grayscaling and the subsequent conversion of the frame to an array of color intensity. One of these tests were the effects of different frame resolutions on the functioning of the program.

This test was carried out by using a static implementation of our program in conjunction with the ZJU Eyeblink dataset. This dataset consists of 80 short videos of 20 subjects with frame resolution of 320x240 px. This dataset was used since similar tests have been done on the same dataset in the past, allowing for easier comparison of results. In the static implementation, the program was given the frames from the dataset as input, as a replacement for the frames that the dynamic implementation processes as input. The program then calculated the EAR for that particular frame. Then the result was made from a continuous EAR range to a discrete result i.e. closed eye or open, depending upon the EAR threshold value.

For the test, we iteratively decreased face resolution to 80% of size of previous image and measured the variations in EAR value compared to those for the original image, or the ground truth values. The face resolution can be described by an indicator called inter-ocular distance, or IOD, which is a distance in pixels between centers of eyes. The average IOD of the original frames was 57.4 px, meaning that the tests were carried out for frames with IOD $\in \{57.4, 45.9, 36.7, \dots, 1.3, 1.0\}$ pixels.

We found that the accuracy for the program fell as IOD decreased, with the facial detector failing to register any faces when IOD was approximately 10 px or lower (most frames of 9.6 px IOD were detected, while there were only a few frames of the next iteration i.e. 7.7 px that were detected).

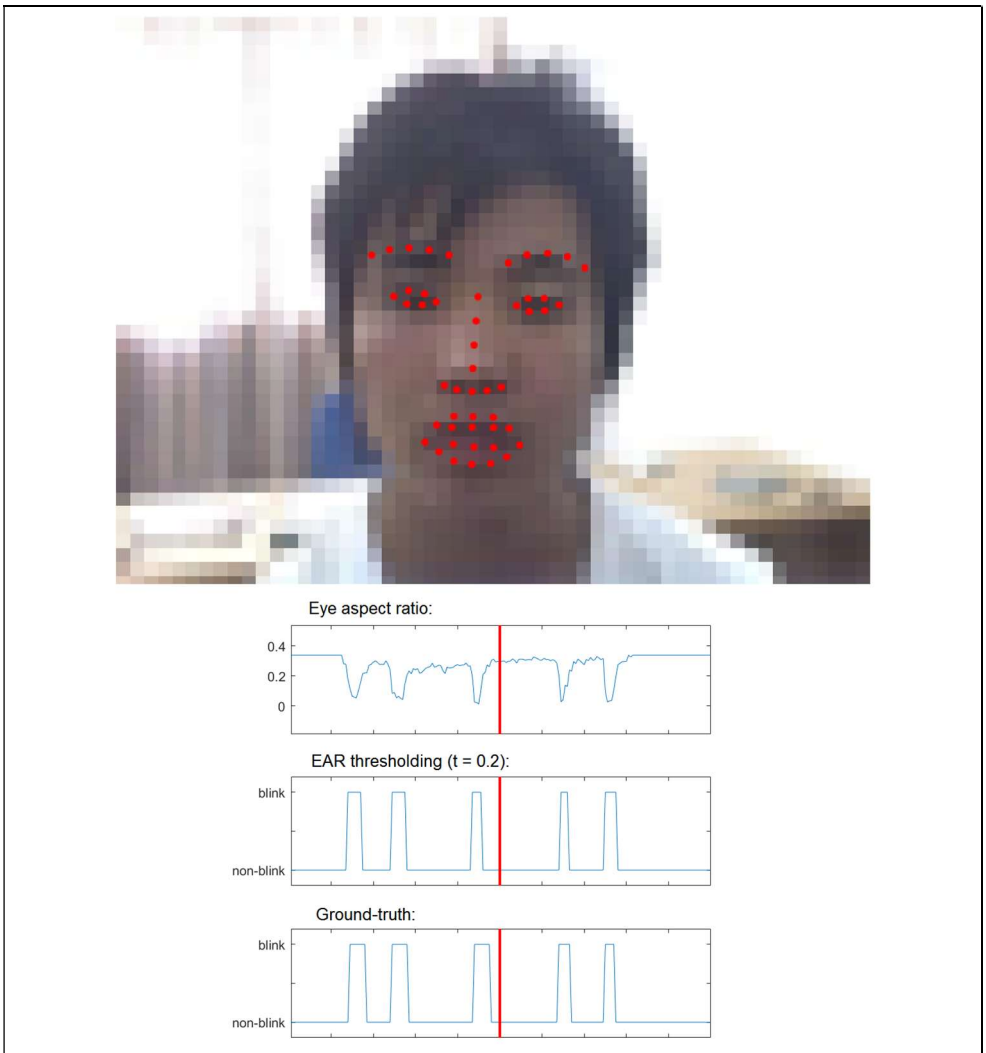


Fig. 5.1. Example of the test comparisons for frames with an IOD of 9.6 px. The plots respectively show the EAR as detected, then the discrete result as per detected EAR and threshold value, and lastly the actual result as per manual review by the authors of the dataset. Note that the discrete result and ground truth are fairly similar.

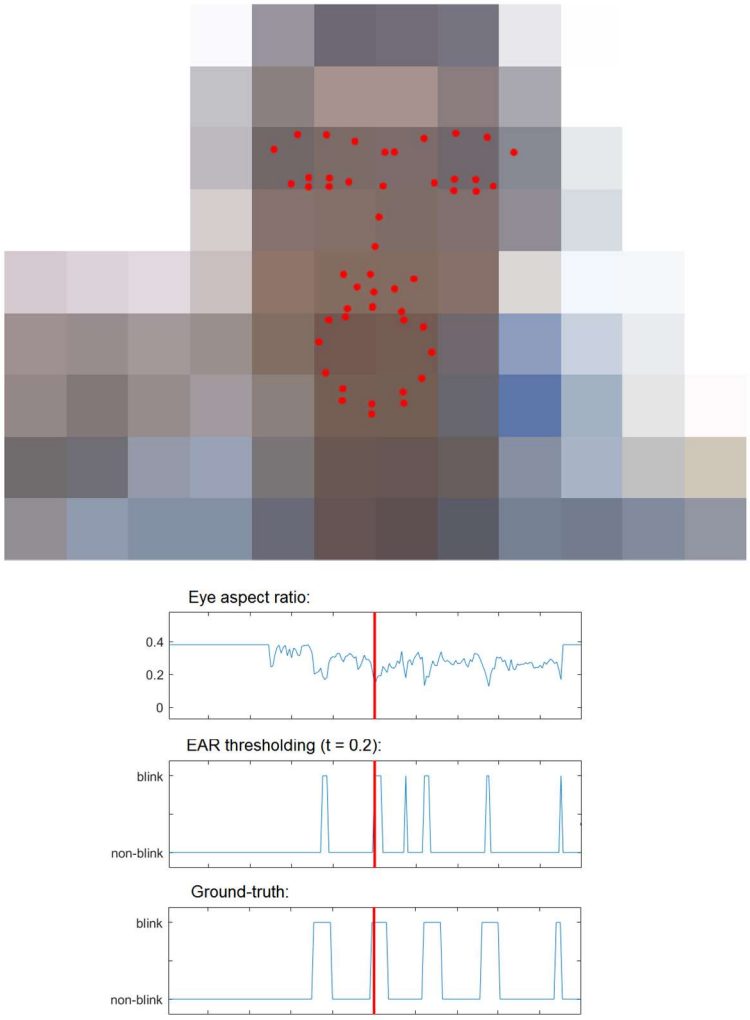


Fig. 5.2. Example of the test comparisons for frames with an IOD of 2 px. The plots respectively show the EAR as detected, then the discrete result as per detected EAR and threshold value, and lastly the actual result as per manual review by the authors of the dataset. Note that the discrete result is considerably inaccurate when compared to the ground truth, as shown by the detection of non-existent blinks.

CONCLUSION AND FUTURE WORK

In this work, a real time system that monitors and detects the loss of attention of drivers of vehicles is proposed. The face of the driver has been detected by capturing facial landmarks and warning is given to the driver to avoid real time crashes. Non-intrusive methods have been preferred over intrusive methods to prevent the driver from being distracted due to the sensors attached on his body. The proposed approach uses EAR to detect driver's drowsiness in real-time. This is useful in situations when the drivers are used to strenuous workload and have to drive continuously for long distances.

The future work can include integration of the proposed system with globally used applications such as Google Maps. For example, in case of repeated alerts, the program may guide the driver to the nearest coffee shop, since caffeinated drinks help in reducing drowsiness temporarily. With the advent of AI-assisted cars, such as the Tesla Model 3, the system can also be integrated in the manner that the vehicle would take over controls and drive itself to the driver's home or to his/her destination, depending on other factors.

In this manner, the system can reduce the number of casualties and injuries that happen regularly due to these drowsy states of the drivers. Accuracy of our program improves with the improvement in lighting of the main subject i.e. the driver, and proper illumination of the environment. The work can be extended with almost no changes to other types of users as well.

REFERENCES

[1] T. Soukupova and J. Cech, “Eye Blink Detection Using Facial Landmarks” Computer Vision Winter Workshop (CVWW), 2016.

[2] M. Ngxande, J.-R. Tapamo, and M. Burke, “Driver drowsiness detection using behavioural measures and machine learning techniques: A review of state-of-art techniques,” *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, 2017.

[3] A. Rosebrock, “Drowsiness detection with OpenCV,” PyImageSearch, 28-Jun-2019. [Online]. Available: https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/. [Accessed: 01-Nov-2019].

[4] S. Mallick, “Histogram of Oriented Gradients,” Learn OpenCV, 06-Dec-2016. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/. [Accessed: 01-Nov-2019].

[5] R. Jabbar, K. Al-Khalifa, M. Kharbeche, W. Alhajyaseen, M. Jafari, and S. Jiang, “Real-time Driver Drowsiness Detection for Android Application Using Deep Neural Networks Techniques,” Procedia Computer Science, vol. 130, pp. 400–407, 2018

[6] S. Sangle, B. Rathore, R. Rathod, A. Yadav, and A. Yadav, “Real Time Drowsiness Detection System,” pp. 87–92, 2018

[7] V. Varghese, A. Shenoy, S. Ks, and K. P. Remya, “EAR Based Driver Drowsiness Detection System,” vol. 2018, pp. 93–96, 2018

[8] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05).

[9] A. Rosebrock, “Raspberry Pi: Facial landmarks + drowsiness detection with OpenCV and dlib”, PyImageSearch, 23-Oct-2017. [Online]. Available: https://www.pyimagesearch.com/2017/10/23/raspberry-pi-facial-landmarks-drowsiness-detection-with-opencv-and-dlib/. [Accessed: 07-Nov-2019].

APPENDIX

Source Code

```
# import the necessary packages
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import argparse
import imutils
import time
import dlib
import cv2

def sound_alarm(path):
	# play an alarm sound
	playsound.playsound(path)

def eye_aspect_ratio(eye):
	# compute the euclidean distances between the two sets of
	# vertical eye landmarks (x, y)-coordinates
	A = dist.euclidean(eye[1], eye[5])
	B = dist.euclidean(eye[2], eye[4])

	# compute the euclidean distance between the horizontal
	# eye landmark (x, y)-coordinates
	C = dist.euclidean(eye[0], eye[3])

	# compute the eye aspect ratio
	ear = (A + B) / (2.0 * C)
```

```
        # return the eye aspect ratio
        return ear

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--shape-predictor", required=True,
        help="path to facial landmark predictor")
ap.add_argument("-a", "--alarm", type=str, default="",
        help="path alarm .WAV file")
ap.add_argument("-w", "--webcam", type=int, default=0,
        help="index of webcam on system")
args = vars(ap.parse_args())

# define two constants, one for the eye aspect ratio to indicate
# blink and then a second constant for the number of consecutive
# frames the eye must be below the threshold for to set off the
# alarm
EYE_AR_THRESH = 0.3
EYE_AR_CONSEC_FRAMES = 48

# initialize the frame counter as well as a boolean used to
# indicate if the alarm is going off
COUNTER = 0
ALARM_ON = False

# initialize dlib's face detector (HOG-based) and then create
# the facial landmark predictor
print("[INFO] loading facial landmark predictor...")
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(args["shape_predictor"])

# grab the indexes of the facial landmarks for the left and
# right eye, respectively
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
```

```
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

# start the video stream thread
print("[INFO] starting video stream thread...")
vs = VideoStream(src=args["webcam"]).start()
time.sleep(1.0)

# loop over frames from the video stream
while True:
	# grab the frame from the threaded video file stream, resize
	# it, and convert it to grayscale
	# channels)
	frame = vs.read()
	frame = imutils.resize(frame, width=450)
	gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

	# detect faces in the grayscale frame
	rects = detector(gray, 0)

	# loop over the face detections
	for rect in rects:
		# determine the facial landmarks for the face region, then
		# convert the facial landmark (x, y)-coordinates to a NumPy
		# array
		shape = predictor(gray, rect)
		shape = face_utils.shape_to_np(shape)

		# extract the left and right eye coordinates, then use the
		# coordinates to compute the eye aspect ratio for both eyes
		leftEye = shape[lStart:lEnd]
		rightEye = shape[rStart:rEnd]
		leftEAR = eye_aspect_ratio(leftEye)
		rightEAR = eye_aspect_ratio(rightEye)
```

```
# average the eye aspect ratio together for both eyes
ear = (leftEAR + rightEAR) / 2.0

# compute the convex hull for the left and right eye, then
# visualize each of the eyes
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

# check to see if the eye aspect ratio is below the blink
# threshold, and if so, increment the blink frame counter
if ear < EYE_AR_THRESH:
        COUNTER += 1

        # if the eyes were closed for a sufficient number of
        # then sound the alarm
        if COUNTER >= EYE_AR_CONSEC_FRAMES:
                # if the alarm is not on, turn it on
                if not ALARM_ON:
                        ALARM_ON = True

                        # check to see if an alarm file was supplied,
                        # and if so, start a thread to have the alarm
                        # sound played in the background
                        if args["alarm"] != "":
                                t = Thread(target=sound_alarm,
                                        args=(args["alarm"],))
                                t.deamon = True
                                t.start()

                # draw an alarm on the frame
                cv2.putText(frame, "DROWSINESS ALERT!", (10,
30),
```

```
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
255), 2)

                # otherwise, the eye aspect ratio is not below the blink
                # threshold, so reset the counter and alarm
                else:
                        COUNTER = 0
                        ALARM_ON = False

                # draw the computed eye aspect ratio on the frame to help
                # with debugging and setting the correct eye aspect ratio
                # thresholds and frame counters
                cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # show the frame
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
                break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```