DESIGN AND IMPLEMENTATION OF DRIVER DROWSINESS DETECTION SYSTEM

Pranjali Hiwale[1], Anand Kalsait[2], Kishori Choukade[3], Aishwarya Puri[4], Prof. Dhiraj Shirbhate[5]

[1,2,3,4]Students, [5]Assistant Professor
Department of Computer Engineering
Government College of Engineering Yavatmal, Maharashtra, India.
Dr. Babasaheb Ambedkar Technological University, Lonere

Abstract: **Drowsiness of drivers is amongst the significant causes of road accidents. Every year, it increases the amounts of deaths and fatalities injuries globally. In this project, a module for Advanced Driver Assistance System (ADAS) is presented to reduce the number of accidents due to drivers' fatigue and hence increase the transportation safety; this system deals with automatic driver drowsiness detection based on visual information and Artificial Intelligence. We proposed an algorithm to locate, track, and analyse both the drivers face and eyes to measure PERCLOS (percentage of eye closure), a scientifically supported measure of drowsiness associated with slow eye closure**

Keywords: **Face Detection, Eye Detection, Image processing, Driver Drowsiness Detection**.

1. INTRODUCTION

In this high-tech world, it's almost impossible to imagine life without vehicles. The invention of vehicles is one of the greatest human kind's inventions. vehicles have become an essential part of almost every day use for individuals of every age. In daily life, we interact and use many times with different vehicles to make our work easier. Thus, for the safety of the driver or owner the "Driver Drowsiness Detection System" has become a hot topic for research. Drowsiness detection is a safety technology that can prevent accidents that are caused by drivers who fell asleep while driving. This system will alert the driver when drowsiness is detected. Through its driver Availability Detection System, sensors will scan the head and face to ensure that the eyes are open and the driver is alert before the cars turns over the steering wheel. If drowsiness is detected, the driver is alerted to the nearest rest stop. In this project, drowsiness detection application will be designed and implemented using a regular webcam. To implement this, we will be using OpenCV module of Python.

The interest in equipping vehicles with driver drowsiness detection systems has been motivated by alarming statistics, such as the 2013 World Health Organization report (World Health Organization, 2013) stating that: 1.24 million people die on the road every year; approximately 6% of all the accidents are caused by drivers driving in a drowsy state; and most of the accidents of this type result in fatalities. Drowsiness (also referred to as sleepiness) can be defined as "the need to fall asleep". This process is a result of normal human biological rhythm and its sleep-wake cycles. The longer the period of wakefulness, the more pressure builds for sleep and the more difficult it is to resist it (Akertedt et al., 2002).

2. OUR WORK

This section describes the requirements, constraints, basic architecture, and selected algorithms associated with our driver drowsiness detection system. The hallmarks of the proposed system are its robustness, accuracy, and overall simplicity.

2.1 REQUIREMENT AND CONSTRAINTS

The driver drowsiness detection system described in this paper must comply with the following main requirements:

- **Algorithmically simple and easy to implement:** We chose to rely on off-the-shelf solutions for most stages, based on the popularity and success of the associated algorithms (e.g., Viola-Jones face detector, Support Vector Machine classifier).

- **Easily portable to different platforms:** The application must run on a mobile device (e.g., Android-based smartphone) mounted on the vehicle's dashboard. Ideally, it should be easily portable to other (e.g., iOS-based) mobile devices of comparable size and computational capabilities.

- **Computationally non-intensive**: Since (near) realtime performance is required, algorithms must be optimized to ensure continuous monitoring of driver's state without excessive burdening of the device's main processor. As a side benefit, battery consumption is reduced as well.

- **Accuracy:** One of the main challenges of designing such a system is related to the fact that both type I and type II errors are highly undesirable, for different reasons: type I errors (false positives) will annoy the driver and reduce their willingness to use the system (due to excessive false alarms), whereas type II errors (false negatives) can have literally catastrophic consequences and defeat the purpose of the entire system.

- **Robustness:** The system must be tolerant to modest amounts of lighting variations, relative camera motion (e.g. due to poor road conditions), changes to the driver's visual appearance (even in the course of a session, e.g., by wearing/removing a hat or sunglasses), camera resolution and frame rates, and different computational capabilities of the device's processors. Some of the anticipated constraints and limitations faced by the proposed system include:

- **Lighting conditions:** Frequent and drastic change in darkness or brightness of a scene (or part of it), which may happen even during the shortest driving intervals, have been proven to be a significant challenge for many computer vision algorithms.

- **Camera motion:** Poor road conditions as well as a more aggressive style of driving can introduce significant amount of vibrations and discomfort to the driving experience. Those vibrations can be passed onto the camera and cause distortion in the images which can significantly skew the results and decrease the overall performance of the system.

- **Relative positioning of device:** The camera must be positioned within a certain range from the driver and within a certain viewing angle. Every computer vision algorithm has a "comfort zone" in which it performs the best and most reliably. If that comfort zone is left, performance can be dropped significantly.

- **Hardware and software limitations:** Typical mobile devices have one or two processor cores, reduced working memory and tend to work on lower clock frequencies, compared to their desktop counterparts. The reason for all of this is to reduce the energy consumption but it creates a significant obstacle in designing this type of system.

- **Driver cooperation:** Last, but certainly not least, all driver drowsiness detection systems assume a cooperative driver, who is willing to assist in the setup steps, keep the monitoring system on at all times, and take proper action when warned by the system of potential risks due to detected drowsiness.

2.2 SYSTEM ARCHITECTURE

Our driver drowsiness detection system consists of four main stages (Figure 1):

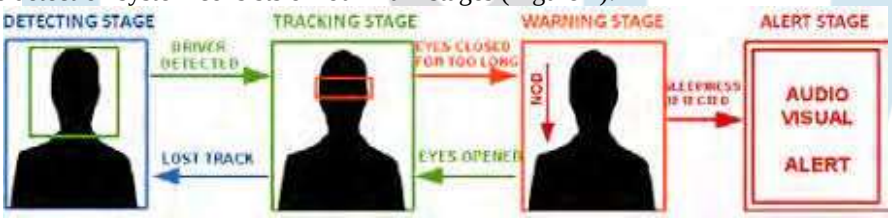


Figure 1: Four stages of Drowsiness Detection System

2.2.1 Detection stage:

This is the initialization stage of the system. Every time the system is started it needs to be set up and optimized for current user and conditions. The key step in this stage is successful head detection (Figure 2). If the driver's head is correctly located, we can proceed to extract the features necessary for setting up the system. Setup steps include: (i) extracting driver's skin color and using that information to create custom skin color model and (ii) collecting a set of open/closed eyes samples, along with driver's normal head position. To help achieve these goals, user interaction might be required. The driver might be asked to sit comfortably in its normal driving position so that system can determine upper and lower thresholds needed for detecting potential nodding. The driver might also be asked to hold their eyes closed and then open for a matter of few seconds each time. This is enough to get the system started. Over time, the system will expand the dataset of obtained images and will become more error resistant and overall, more robust.

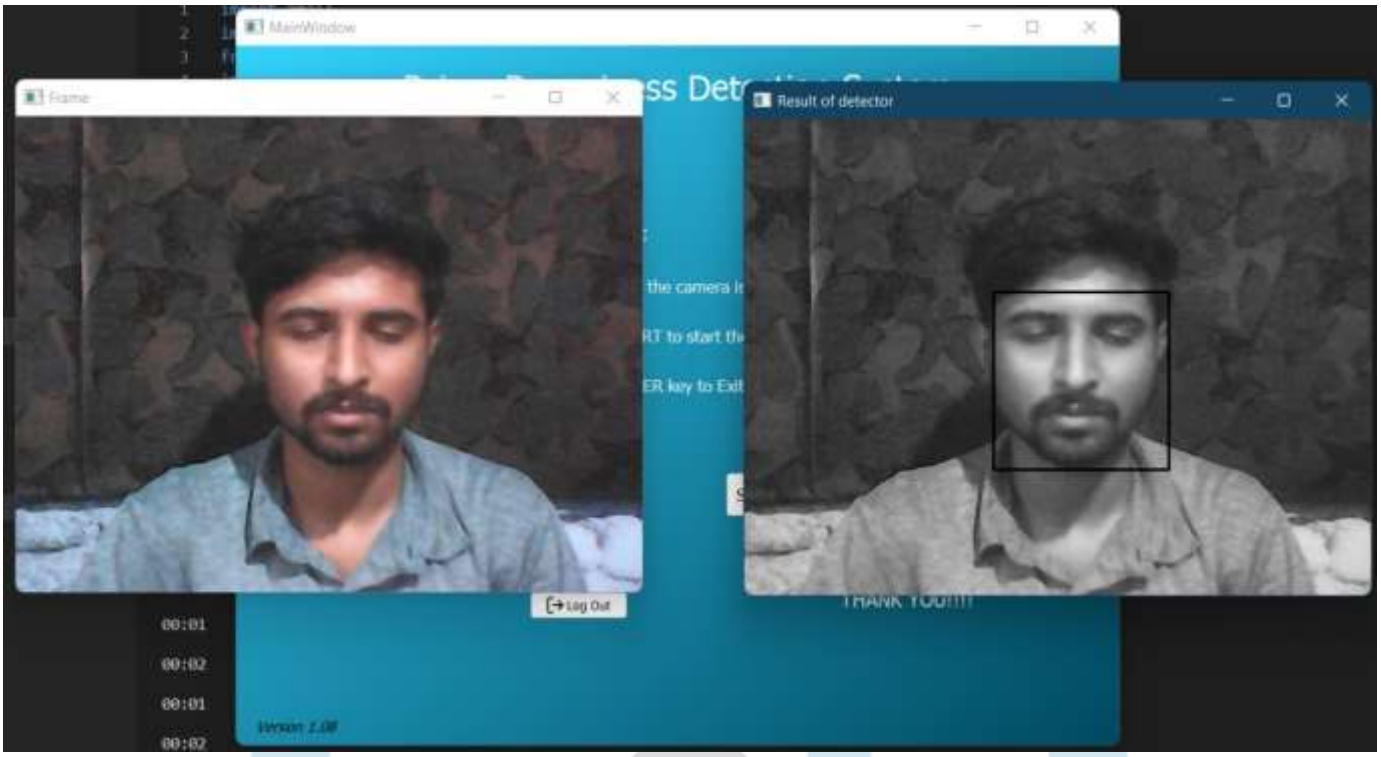


Figure 2: Detection stage

2.2.2 Tracking stage:

Once the driver's head and eyes are properly located and all the necessary features are extracted, the system enters the regular tracking (monitoring) stage. A key step in this stage is the continuous monitoring of the driver's eyes within a dynamically allocated tracking area. More specifically, in order to save some processing time, the system will determine the size of the tracking area based on the previous history of eye movements. For example, if the eyes were moving horizontally to the left for a number of frames it is to be expected that that trend with continue in the following frame also. So it is logical to expand the tracking area towards the expected direction of the eyes and shrink the area in other three directions. During this stage, the system must also determine the state of the eyes. All these tasks must be carried out in realtime; depending on the processor's abilities and current load, it might be necessary to occasionally skip a few frames, without sacrificing algorithmic accuracy.

2.2.3 Warning stage:

If the driver keeps his eyes closed for prolonged period of time or starts to nod, alertness has to be raised. The key step within this stage is close monitoring of driver's eyes. The system must determine whether the eyes are still closed, and what is the eyes' position relative to previously established thresholds. We cannot afford to skip frames in this stage. In practice, tracking of eyes is performed much in the same way as in the tracking stage with the addition of the following processes: calculation of velocity and trajectory of the eyes and threshold monitoring. These additional computations are required to improve the system's ability to determine whether the driver is drowsy or not.

2.2.4 Alert stage:

Once it has been determined that the driver appears to be in an abnormal driving state, the system has to be proactive and alert the driver of potential dangers that can arise. Combination of audio/visual alerts are used to attract the driver's attention and raise their alertness level. Alerting has to be implemented in such a way as not to cause the opposite effect of intended and startle the driver into causing an accident.

2.3 FACE DETECTION

The proposed system will start by capturing the video frames one by one. OpenCV provides extensive support for processing live videos. The system will detect the face in the frame image for each frame. This system uses Viola-Jones object detector which is a machine learning approach for visual object detection (Paul Viola, 2004 and Paul Viola, 2001). This is achieved by making use of the Haar algorithm for face detection. The inbuilt OpenCV xml "haarcascade_frontalface_alt2.xml" file is used to search and detect the face in individual frames. This file contains a number of features of the face and constructed by using a number of positive and negative samples. First load the cascade file then pass the acquired frame to an edge detection function, which detects all the possible objects of different sizes in the frame. Since the face of the driver occupies a large part of the image, instead of detecting objects of all possible sizes, specify the edge detector to detect only objects of a particular size i.e. for face region. Next, the output the edge detector is stored and this output is compared with the cascade file to identify the face in the frame. The output of this module is a frame with face detected in it. Only disadvantage in Haar algorithm is that it cannot extrapolate and does not work

appropriately when the face is not in front of the camera axis. Once the face detection function has detected the face of the driver, the eyes detection function tries to detect the driver's eyes.

2.4 EYE DETECTION

Once the face detection function has detected the face of the driver, the eyes detection function tries to detect the automobile driver's eyes. After face detection find eye region by considering eyes are present only in upper part of the face and from top edge of the face, extract eyes Region Of Interest (ROI) by cropping mouth and hair, we mark it the region of interest. By considering the region of interest it is possible to reduce the amount of processing required and also speeds up the processing for getting exact eyes. After the region of interest is marked, the edge detection technique is applied only on the region of interest .Then search for eyes in ROI, Circular Hough Transformation is used here to find shape of eyes (Rhody Chester, 2005). The main advantage of the Hough transform technique is that it is liberal to gaps in feature boundary descriptions and is relatively unaffected by image noise, unlike edge detectors. The OpenCV function Hough Circles () is used to detect circles in an eye image. CHT ensure that at most two eyes found. With the eye detection technique we will only be able to detect the open state of eyes.

2.5 DROWSINESS DETECTION

Today, we are going to extend this method and use it to determine how long a given person's eyes have been closed for. If there eyes have been closed for a certain amount of time, we'll assume that they are starting to doze off and play an alarm to wake them up and grab their attention. To accomplish this task, I've broken down today's tutorial into three parts. In the first part, I'll show you how I setup my camera in my car so I could easily detect my face and apply facial landmark localization to monitor my eyes. I'll then demonstrate how we can implement our own drowsiness detector using OpenCV, dlib, and Python. Finally, I'll hop in my car and go for a drive (and pretend to be falling asleep as I do). As we'll see, the drowsiness detector works well and reliably alerts me each time I start to "snooze".

3. SYSTEM DESIGN

3.1 FRAMEWORK

OpenCV

OpenCV was started at Intel in 1999 by **Gary Bradsky**, and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day. OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language OpenCV is a popular and open-source computer vision library that is focussed on real-time applications. The library has a modular structure and includes several hundreds of computer vision algorithms. OpenCV includes a number of modules including image processing, video analysis, 2D feature framework, object detection, camera calibration, 3D reconstruction and more. We will be using OpenCV to capture the live video stream from webcam and to use it in the program.

3.2 DETAILS OF HARDWARE AND SOFTWARES

- **Hardware**

1.Laptop or RaspberryPi
2. ESP32
2.Arduino
3.Buzzer
4.Breadboard And Pushbutton

- **Software**

1.Python editor
2.Visual Studio Code
3.Visual Studio (for C++ Development)
4.PyQT Designer
5.Command Prompt for exe. File build
6.SQLite Database
7.Arduino IDE

- **Libraries**

1. Dlib: The dlib C++ library is a cross-platform package for threading, networking, numerical operations, machine learning, computer vision, and compression, placing a strong emphasis on extremely high-quality and portable code. For medium to large

image sizes Dlib is the fastest method on CPU. But it does not detect small sized factors. So, if you know that your applications will not be dealing with very small sized faces then the HoG based Face detector is the better option.

2. Imutils: A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3.

3. Pygame: The pygame library is an open-source module for the Python programming language specifically intended to help you make games and other multimedia applications. Built on top of the highly portable SDL (Simple Direct Media Layer) development library, pygame can run across many platforms and operating systems

4. NumPy: NumPy stands for Numerical Python and it can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high level mathematical functions that operate on these arrays and matrices.

5. Pyfirmata: There are libraries that implement the Firmata protocol in order to communicate (from a computer, smartphone or tablet for example) with Firmata firmware running on a microcontroller platform. PyFirmata is a Python interface for the Firmata protocol and runs on python3.

6. PyQT5: PyQt5 is the latest version of a GUI widgets toolkit developed by Riverbank Computing. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt5 is a blend of Python programming language and the Qt library. This introductory tutorial will assist you in creating graphical applications with the help of PyQt.

7. SQLite3: Python SQLite3 module is used to integrate the SQLite database with Python. It is a standardized Python DBI API 2.0 and provides a straightforward and simple-to-use interface for interacting with SQLite databases. There is no need to install this module separately as it comes along with Python after the 2.5x version.

8. SMTPLib: Python provides smtplib module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. host − This is the host running your SMTP server. You can specify IP address of the host or a domain name like example.com.

9. PyMsgBox: PyMsgBox is simple, cross-platform, purely implemented in Python for message boxes as JavaScript has. It uses Python's built-in Tkinter module for its GUI.

3.3 IMPLEMENTATION AND FLOWCHART

In this Python project, we have used OpenCV for gathering the images from webcam and feed them into the deep learning model which will classify whether the person's eyes are 'Open' or 'Closed'. The approach we have used for this Python project is as follows:

Step 1 – Take image as input from a camera.

With a webcam, we will take images as input. So to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV to access the camera and set the capture object will read each frame and we store the image in a frame variable.

Step 2 – Detect the face in the image and create a Region of Interest (ROI).

To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes grey images in the input. We don't need color information to detect the objects.
Then we perform the detection. It returns an array of detections with x, y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

Step 3 – Detect the eyes from ROI and feed it to the classifier.

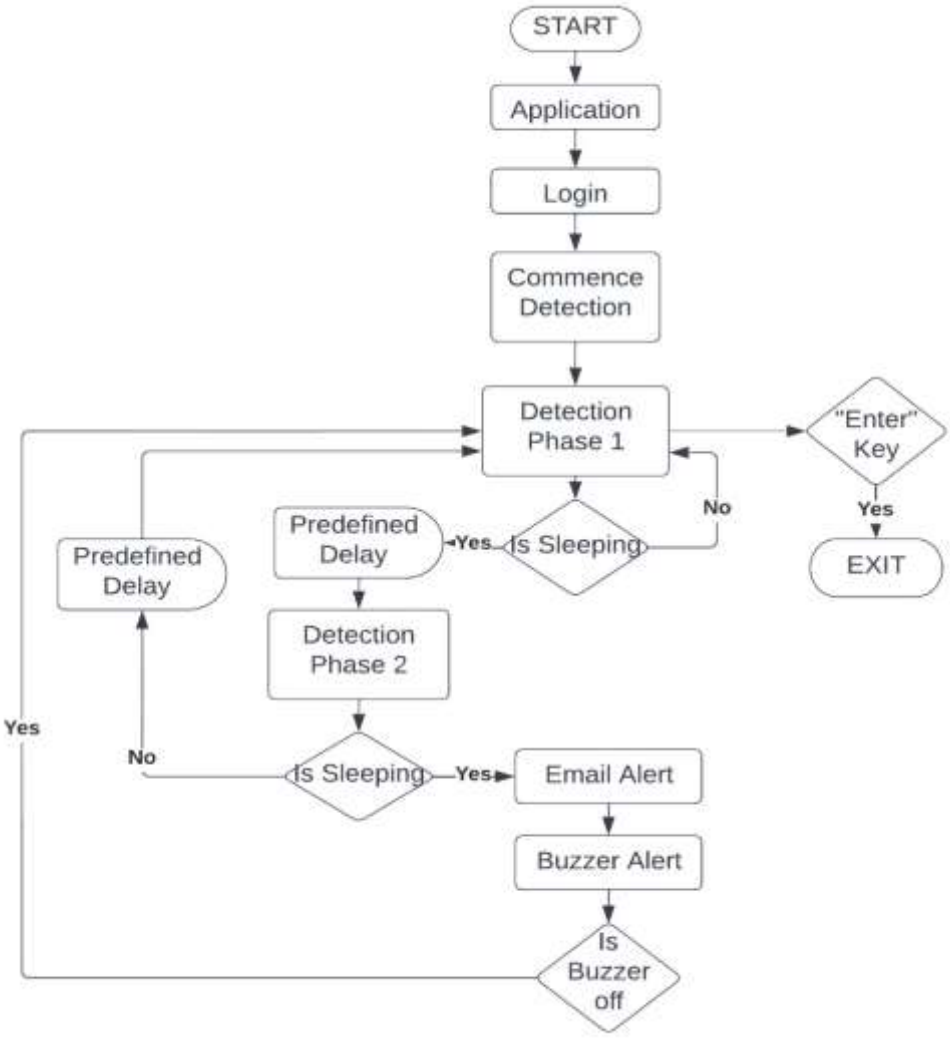
The same procedure to detect faces is used to detect eyes. First, we set the cascade classifier for eyes in **leye** and **reye** respectively then detect the eyes. Now we need to extract only the eyes data from the full image. This can be achieved by extracting the boundary box of the eye and then we can pull out the eye image from the frame

Step 4 – Classifier will categorize whether eyes are open or closed.

We are using CNN classifier for predicting the eye status. To feed our image into the model, we need to perform certain operations because the model needs the correct dimensions to start with

Step 5 – Calculate score to check whether the person is drowsy.

The score is basically a value we will use to determine how long the person has closed his eyes. So if both eyes are closed, we will keep on increasing score and when eyes are open, we decrease the score. We are drawing the result on the screen using cv2.putText() function which will display real time status of the person.



Flowchart

3.4 Sample Screenshots

- **Active state:**

This is the initialization stage of the system. Every time the system is started it needs to be set up and optimized for current user and conditions.In this state check the position of driver that is in Active state.and gives frame of Active s

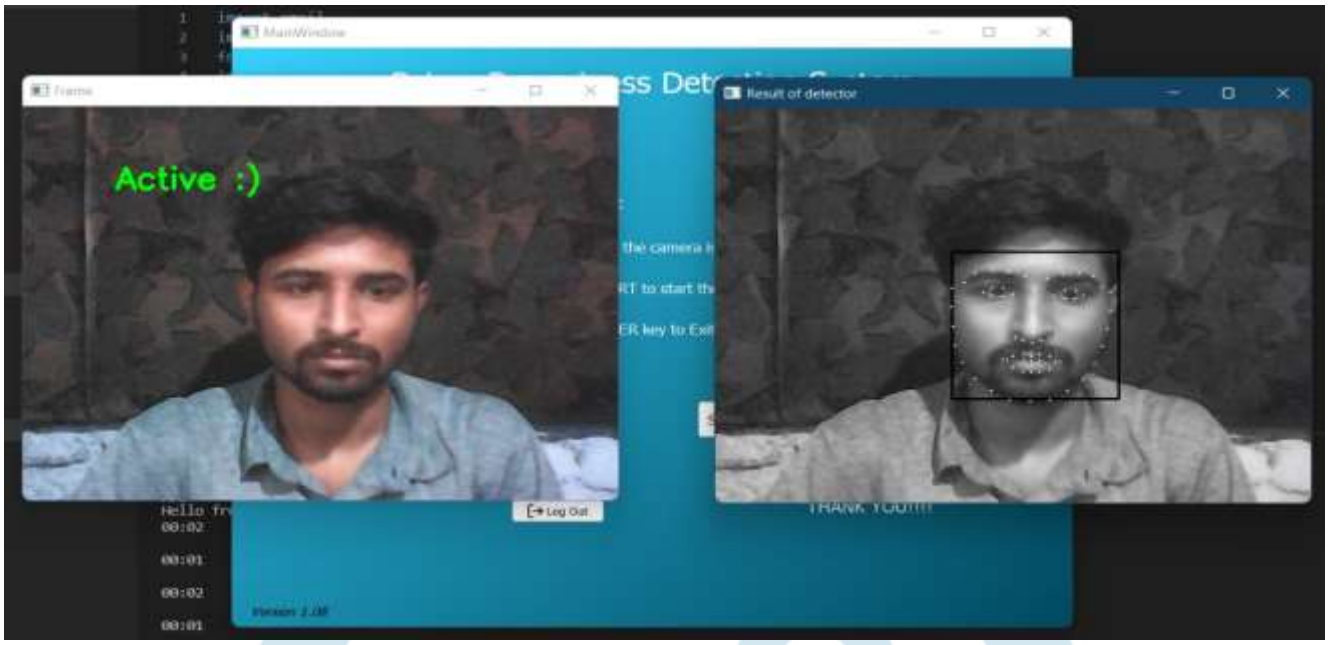


Figure 3 : Active State

- **Drowsy State:**

In this figure gives alert if driver eye become drowsy that gives drowsy state. If drivers could be warned before they became too drowsy to drive safely, some of these crashes could be prevented. In order to reliably detect the drowsiness, it depends on the presentation of timely warnings of drowsiness.

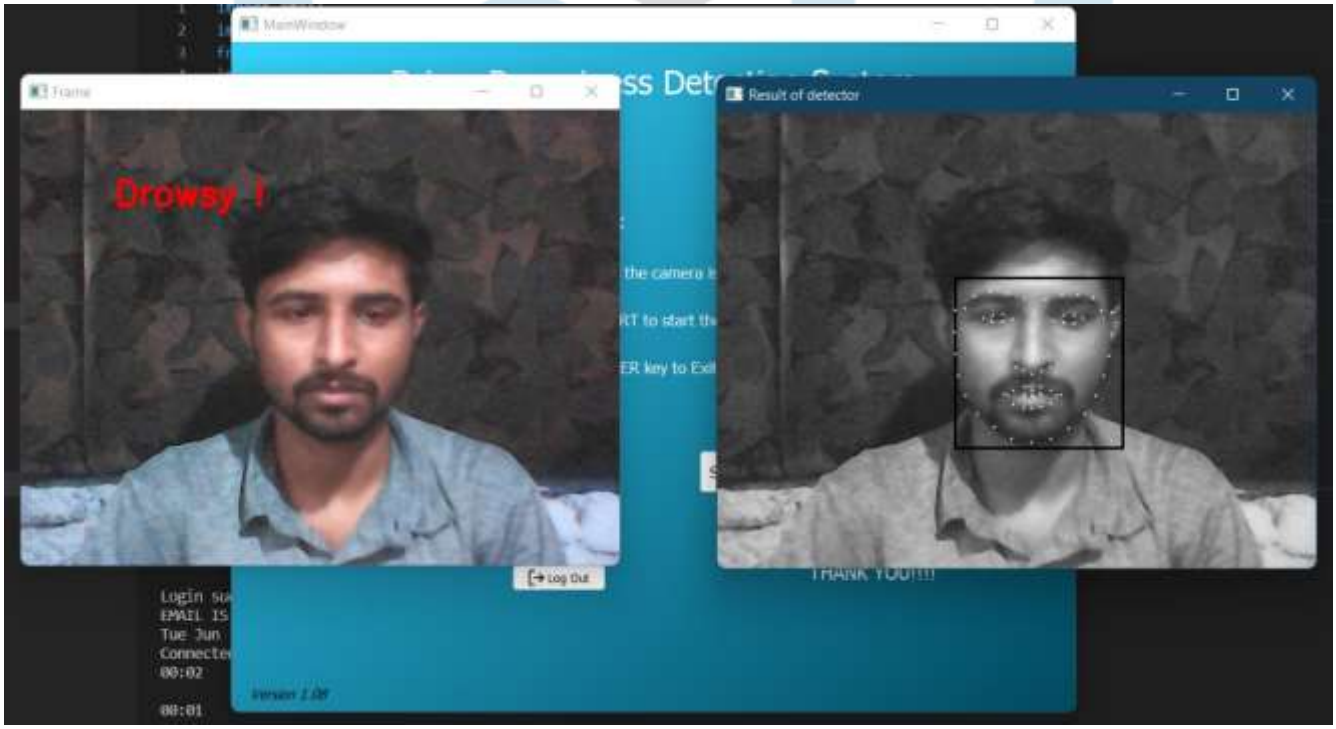


Figure 4: Drowsy State

- **Detection stage:**

The driver might be asked to sit comfortably in its normal driving position so that system can determine upper and lower thresholds needed for detecting potential nodding. The driver might also be asked to hold their eyes closed and then open for a matter of few seconds each time. This is enough to get the system started. Over time, the system will expand the dataset of obtained images and will become more error resistant and overall, more robust.

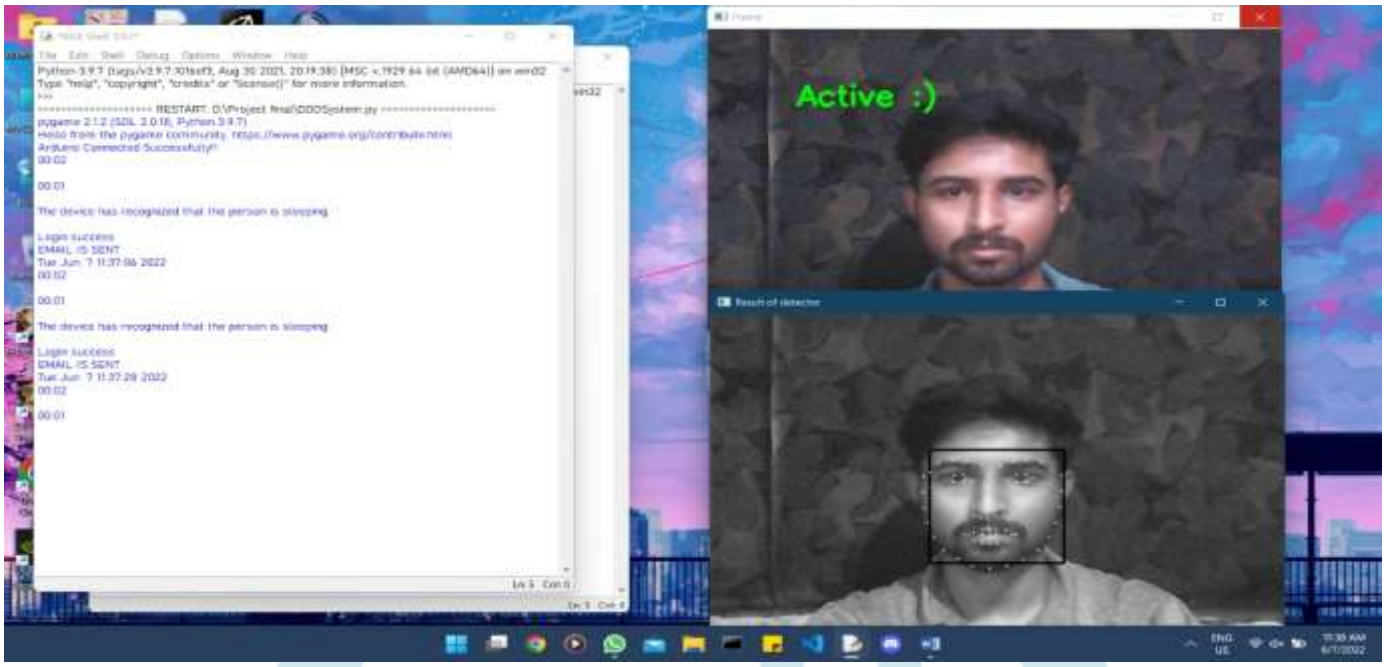


Figure 5: Detection stage

- **After all stage and Buzzer stage:**

After all stages are performed if driver is in drowsy stage it gives result of drowsy state and buzzer can work as a alert for driver.

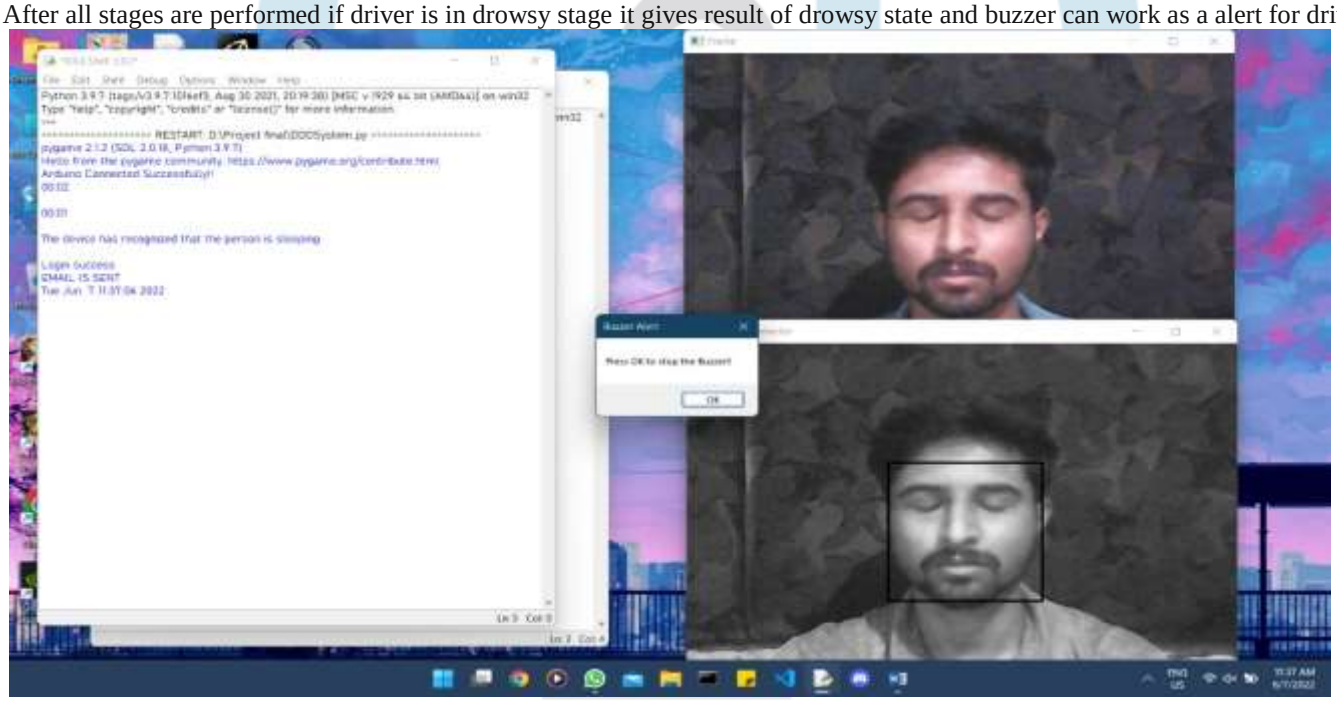


Figure 6: After all stage and Buzzer stage

- **Back to detection stage 1 after several iteration:**

After all stages are performed than start from initial stage that is detection stage and performed all step one by one this step continue performed.

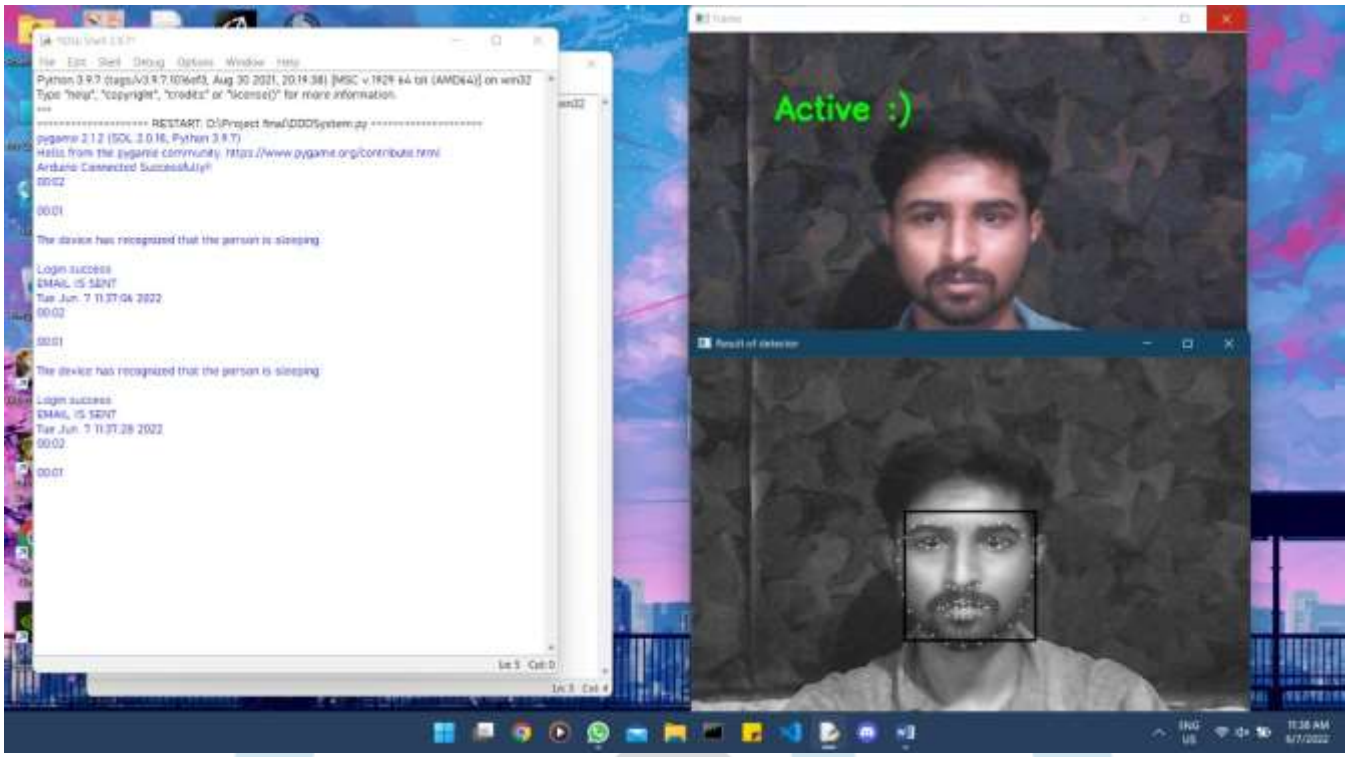


Figure 7: Back to detection stage 1 after several iteration

4. CONCLUSION AND FUTURE WORK

In this paper, we have reviewed the various methods available to determine the drowsiness state of a driver. Although there is no universally accepted definition for drowsiness, the various definitions and the reasons behind them were discussed. This paper also discusses the various ways in which drowsiness can be manipulated in a simulated environment. The various measures used to detect drowsiness include subjective, vehicle-based, physiological and behavioral measures; these were also discussed in detail and the advantages and disadvantages of each measure were described. Although the accuracy rate of using physiological measures to detect drowsiness is high, these are highly intrusive. However, this intrusive nature can be resolved by using contactless electrode placement. Hence, it would be worth fusing physiological measures, such as ECG, with behavioral and vehicle-based measures in the development of an efficient drowsiness detection system. In addition, it is important to consider the driving environment to obtain optimal results

The future works may focus on the utilization of outer factors such as vehicle states, sleeping hours, weather conditions, mechanical data, etc, for fatigue measurement. Driver drowsiness pose a major threat to highway safety, and the problem is particularly severe for commercial motor vehicle operators. Twenty-four-hour operations, high annual mileage, exposure to challenging environmental conditions, and demanding work schedules all contribute to this serious safety issue. Monitoring the driver's state of drowsiness and vigilance and providing feedback on their condition so that they can take appropriate action is one crucial step in a series of preventive measures necessary to address this problem. Currently there is not adjustment in zoom or direction of the camera during operation. Future work may be to automatically zoom in on the eyes once they are localized.

REFERENCES

[1] Python website. Python versions [online]. Available: https://www.python.org/download.

[2] OpenCV website. Installation in Linux [online] Available:http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_instal.html.

[3] Arduino.cc. ArduinoUno Overview [online]. Availablehttp://arduino.cc/en/Main/arduinoBoardUno.

[4] https://ijesc.org/upload/6e82078065c27e9b8da77c87f771a be2.RealTime%20Driver%20Drowsiness%20Detection%20System%20Based %20on%20Visual%20Information.pdf.

[5] https://sci-hub.mksa.top/10.1109/iembs.2010.5626013.

[6] http://www.fraunhofer.de/en/press/research-news/2010/10/eye-tracker-driver-drowsiness.html (accessed on 27 July 2012).

[7] Statistics related to fatique due to drivers {Available – 2017.06.01: http://www.orlenbezpiecznedrogi.pl/downloads/material-backgroundowy.doc} (in Polish).

[8] Bosch Drowsiness System {Available – 2017.06.01: http://www.bosch-prasa.pl/informacja.php?idinformacji=1356}.

[9] Information about the driver fatique vision system developed by the PSA in cooperation with the Technical University of Lausanne {Available – 2017.06.01: https://www.autocar.co.uk/car-news/industry/psa-peugeot-citroen-shows-new-cabin-technology}.

[10] Dasgupta, A.; Rahman, D.; Routray, A. A smartphone-based drowsiness detection and warning system for automotive drivers. IEEE Trans. Intell. Transp. Syst. 2018, 20, 4045–4054.

[11] T. Danisman, I. M. Bilasco, C. Djeraba, and N. Ihaddadene, "Drowsy driver detection system using eye blink patterns," in Machine and Web Intelligence (ICMWI), 2010 International Conference on, 2010, pp. 230-233.
[12] S. T. Lin, Y. Y. Tan, P. Y. Chua, L. K. Tey, and C. H. Ang, "PERCLOS Threshold for Drowsiness Detection during Real Driving," Journal of Vision, vol. 12, pp. 546-546, 2012.
[13] B Sangle, R Rathore, A Rathod, A Yadav, ; V Yadav, A Varghese, S Shenoy, K P Ks, ; R Remya, PatraDriver drowsiness monitoring system using visual behaviour and machine learning ISCAIE 2018 -2018 IEEE Symposium on Computer Applications and Industrial Electronics, volume 2018, p. 339 – 344 Posted: 2018.
[14] S Nagargoje, D S Shilvant Drowsiness Detection System for Car Assisted Driver Using Image Processing International Journal of Electrical and Electronics Research ISSN, volume 3, issue 4, p. 175 – 179 Posted: 2015.
[15] https://iopscience.iop.org/article/10.1088/1757-899X/767/1/012066.