## Project Report (CS758PC)

## on

# ADVANCED DRIVER MONITORING SYSTEM USING MACHINE LEARNING

*Submitted*

*in partial fulfillment of the requirements for the award of the degree of*

## Bachelor of Technology

in

## Computer Science and Engineering [Data Science]

By

**GAWLI VINAY (22265A6703)**

**G SONY (2226546702)**

Under the guidance of

**Dr. Barnali Gupta Banik**

**Mr. M. Srikanth Sagar Assistant Professor**



## Department of Emerging Technologies

Mahatma Gandhi Institute of Technology

Kokapet(V), Gandipet(M), Hyderabad.

Telangana - 500 075.

## 2024 - 2025

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

GANDIPET, HYDERABAD – 500075, Telangana

## CERTIFICATE



This is to certify that the project entitled ― **“ADVANCED DRIVER MONITORING SYSTEM USING MACHINE LEARNING”** is being submitted by **GAWLI VINAY** bearing Roll No. **22265A6703** and **G SONY** bearing Roll No. **22265A6702** in partial fulfillment of the requirements for the Major Project (CS758PC) in **COMPUTER SCIENCE AND ENGINEERING [DATA SCIENCE]** is a record of bonafide work carried out by him.

The results of the investigations enclosed in this report have been verified and found satisfactory.

|  |  |
| --- | --- |
| Project Guide: | Head of the Department |
| **Mr. M. Sunil Kumar**  Associate Professor | **Dr. M. Rama Bai**  Professor |
|  |  |

**External Examiner**

# DECLARATION

This is to certify that the work reported in this project titled ―**“ADVANCED DRIVER MONITORING SYSTEM”** is a record of work done by me in the Department of Emerging Technologies, Mahatma Gandhi Institute of Technology, Hyderabad.

No part of the work is copied from books/journals/internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the work done entirely by me and not copied from any other source.

**GAWLI VINAY (22265A6703)**

**G SONY (22265A6702)**

# ACKNOWLEDGEMENT

We would like to express my sincere thanks to **Prof. G. Chandra Mohan Reddy, Principal MGIT**, for providing the working facilities in college.

We wish to express my sincere thanks and gratitude to **Dr. M. Rama Bai, Professor and HoD**, Department of Emerging Technologies, MGIT, for all the timely support and valuable suggestions during the period of project.

We are extremely thankful to **Dr.Barnali Gupta Banik, Assistant Professor,** and **Dr.D Koteswara Rao, Assistant Professor,** Department of Emerging Technologies, MGIT, Major Project coordinators for their encouragement and support throughout the project.

We are extremely thankful and indebted to our mentors**Mr.M.Sunil Kumar**,**Assistant Professor**, Department of Emerging Technologies, for their constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank all the faculty and staff of Emerging Technologies Department who helped me directly or indirectly, for completing this project.

**GAWLI VINAY(22265A6703)**

**G.SONY(22265A6702)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TABLE OF CONTENTS** | | |  | | |
| **Certificate** | | | | | i | | |
| **Declaration** | | | | | ii | | |
| **Acknowledgement** | | | | | iii | | |
| **List of Figures** | | | | | iv | | |
| **Abstract** | | | | | v | | |
| **1. INTRODUCTION** | | | | | 1 | | |
| 1.1. Motivation | | | | 2 | | |
| 1.2. Problem Definition | | | | 3 | | |
| 1.3. Existing System | | | | 4 | | |
| 1.4. Proposed System | | | | 4 | | |
| 1.5. Requirement Specification | | | | 4 | | |
| 1.5.1. Software Requirements | | | 5 | | | |
| 1.5.2. Hardware Requirements | | | 5 | | | |
| **2. LITERATURE SURVEY** | | | | | 13 | | |
| **3. METHODOLOGY** | | | | | 16 | | |
| 3.1. Research Methodology | | | | 17 | | |
| 3.2. Data Collection and Analysis  3.3. Data Pre-processing | | | | 17 | | |
| 3.4. Model Selection | | | | 17 | | |
| 3.5. Integration of AI Recommendation | | | | 17 | | |
| 3.6. GeoLocation and Doctor Locator | | | | 17 | | |
| 3.7. Backend Development | | | | 18 | | |
| 3.8. Frontend Development  3.9. Training  3.10. Predicting the Result | | | | 18 | | |
| 3.11. Testing and Validation | | | | 18 | | |
| 3.12. Development | | | | 19 | | |
| 3.13. Maintenance and Updates | | | | 19 | | |
| 3.14. Diagrammatical Representation | | | | 20 | | |
| 3.14.1. Dataflow Diagram | | | 20 | | | |
| 3.14.2. Sequence Diagram | | | 21 | | | |
| 3.14.3. Workflow Diagram | | | 22 | | | |
| 3.14.4 Use Case Diagram | | 23 | | |
| **4. CODING AND IMPLEMENTATION** | | | | | | 28 | | |
| **5. EXECUTION AND RESULTS** | | | | | | 38 | | |

**6. CONCLUSION** 45

**7. FUTURE ENHANCEMENTS**  46

**BIBILIOGRAPHY** 47

**LIST OF FIGURES**

| **Figure** | **Description** | **Page** |
| --- | --- | --- |
| 1.1 | Visual Studio Code window on Windows operating system | 11 |
| 3.1 | 68\_face\_landmarks | 19 |
| 3.2 | Eye Detection | 20 |
| 3.2.1 | Formula for Eye Aspect Ratio (EAR) | 20 |
| 3.3 | Mouth Detection | 20 |
| 3.3.1 | Formula for Mouth Aspect Ratio (MAR) | 21 |
| 3.4 | System Design | 21 |
| 3.5 | Model Architecture | 22 |
| 3.6 | Use Case Diagram | 23 |
| 3.7 | Class Diagram | 24 |
| 3.8 | Sequence Diagram | 25 |
| 3.9 | Activity Diagram | 26 |
| 3.10 | Dataflow Diagram | 27 |
| 5.1 | No Face Detection | 39 |
| 5.2 | Face Not Clear | 39 |
| 5.3 | Eye Detection with (Closed Eyes and Open Eyes) | 40 |
| 5.4 | Yawning Detection with (Yawning and Non-Yawning) | 41 |
| 5.5 | Stages Of Web Application | 43 |
| 5.6 | Accuracy | 44 |
| 5.7 | Line Chart (EAR and MAR over frames) | 44 |

**ABSTRACT**

In response to the growing concern over driver fatigue and distraction leading to accidents, this research introduces an innovative Advanced Driver Monitoring System (ADMS) leveraging Machine Learning (ML) and OpenCV. This real-time system analyses video input from an in- car camera, focusing on crucial facial features – eyes, mouth, and head pose – to assess driver alertness.The core functionality relies on ML algorithms, potentially including Convolutional Neural Networks (CNNs) for accurate facial feature detection and classification. Eye blink rate and eyelid closure duration are meticulously monitored to identify potential drowsiness. Mouth opening and closing patterns are analysed to detect yawns, a strong indicator of fatigue. OpenCV plays a vital role in real-time image processing and feature extraction. The system continuously processes video frames, detecting facial landmarks and tracking their subtle changes over time. Based on the trained ML models and extracted features, the system predicts the driver's state (alert, drowsy, or distracted).

This ADMS aims to provide timely warnings in the form of visual alerts on the dashboard, audible cues, or even haptic feedback to alert the driver when drowsiness or distraction is detected. By promoting driver awareness and prompting corrective actions, the system has the potential to significantly reduce road accidents caused by fatigued or inattentive drivers.However, this Research looks beyond the immediate functionalities. We envision a future where the system can be extended to incorporate head pose estimation. This would allow for the detection of situations where the driver's gaze is diverted from the road for extended periods, indicating potential distraction from mobile devices or external factors. Additionally, the integration of facial expression recognition could unlock the ability to identify signs of stress or fatigue beyond simple drowsiness detection. By incorporating these advanced features and continuously refining the ML models with real-world data, the ADMS can evolve into a comprehensive driver monitoring solution, promoting safer and more responsible driving behaviour.

# INTRODUCTION

Driver drowsiness and sleep deprivation are among the leading causes of road accidents worldwide, posing significant threats to public safety. A drowsy driver experiences reduced concentration, slower reaction times, and impaired decision- making abilities, making them more prone to mistakes that can result in severe injuries, fatalities, and economic losses. Long-distance drivers, including truck drivers, bus operators, taxi drivers, and travellers on extended journeys, are particularly at risk due to long working hours and insufficient sleep. In India, where road conditions can often exacerbate risks, fatigue-related accidents remain a major concern, highlighting the urgent need for solutions to combat this issue effectively.

Accidents are inevitable with the growth of the population all around the globe. According to the World Health Organization (WHO), there are various reasons for road accidents, including sleeping while driving, texting while driving, over speeding, and alcohol consumption. Around 1.35 million deaths occur annually during road accidents globally, with driver fatigue being one of the leading contributors.

Globally, studies indicate that fatigue accounts for approximately 20% of all road accidents and as much as 50% on certain highways. Reports from the Australian Transport Bureau and the Ministry of Transportation of Ontario reveal that sleep deprivation is a key factor in many fatal crashes. In India, over 148,000 people lost their lives in road accidents in 2015 alone, with at least 21% of these fatalities linked to driver fatigue. Despite these alarming statistics, the role of fatigue is often underestimated, partly because it lacks measurable indicators, unlike alcohol or drug consumption, which can be easily tested. Combined with poor infrastructure, such as bad roads and bridges, fatigue becomes a recipe for disaster in developing countries like India.

Financial pressures further aggravate the problem. Many drivers push themselves to continue driving despite fatigue due to the lucrative nature of their jobs. This is especially true for transport truck drivers who often work long hours without proper rest, prioritizing earnings over safety. Although some countries have implemented regulations to limit driving hours, enforcement remains a challenge due to high implementation costs and resistance from the industry.Addressing driver fatigue requires a multifaceted approach. While raising awareness and encouraging drivers to recognize and admit their fatigue are essential, these measures alone are insufficient. Technological innovations offer a more effective solution. Advanced Driver Monitoring Systems (ADMS) that utilize cameras, sensors, and machine learning algorithms can detect signs of drowsiness, such as prolonged eye closure, facial fatigue, and decreased alertness.

The inclusion of infrared (IR) cameras has further improved the accuracy and reliability of these systems. IR cameras are particularly effective in monitoring driver eye movements and facial expressions under low-light or nighttime conditions, where traditional cameras may struggle. These cameras can detect subtle signs of drowsiness, such as blinking patterns and eye closure duration, ensuring comprehensive monitoring regardless of the time of day.

The integration of IoT (Internet of Things) and machine learning into these systems has further enhanced their efficiency. By continuously monitoring drivers and analysing data in real- time, these technologies provide a proactive approach to accident prevention. Additionally, these systems can be coupled with vehicle controls, such as automatic braking or lane assistance, to further minimize risks. Such advancements not only improve road safety but also help address the economic and social impacts of accidents caused by driver fatigue.

As road networks and vehicle usage continue to expand globally, investing in technologies to driver fatigue is crucial. By leveraging innovative solutions, including advanced infrared camera systems, governments and industries can work together to create safer roads and reduce the consequences of drowsy driving.

* 1. **Motivation**

The motivation behind developing a driver drowsiness detection system stems from the critical need to address the increasing number of accidents caused by driver fatigue, which is a leading contributor to road incidents worldwide. The system aims to save lives by preventing fatigue- related accidents, which are responsible for a significant portion of road fatalities and injuries. By enabling real- time detection and intervention, the system provides an opportunity to reduce these incidents, ultimately saving lives and minimizing injuries. Additionally, the system enhances safety in low-light conditions, where traditional detection methods often fail. By utilizing infrared cameras, it ensures accurate monitoring of the driver’s face and eyes, even in poorly lit environments, such as nighttime driving.

A key aspect of the motivation is the shift from reactive to proactive accident prevention. Through continuous monitoring and real-time analysis of metrics such as PERCLOS, the system detects signs of drowsiness early, allowing drivers to take breaks before reaching a critical point. This proactive approach helps reduce the risk of accidents caused by delayed responses. Furthermore, the system aims to lower the economic burden of road accidents, which often result in substantial medical costs, property damage, and loss of productivity. By preventing accidents, the system contributes to reducing these costs and mitigating the broader societal impact.  
In addition, the development of this system aligns with advancements in transportation technology, such as vehicle automation. By integrating AI-driven drowsiness detection with autonomous vehicle functionalities like automatic braking and lane assistance, the system paves the way for smarter, safer transportation solutions. Finally, fostering driver awareness and responsibility is another motivation. By issuing alerts and encouraging timely breaks, the system raises awareness about the importance of managing fatigue and ensures drivers take responsibility for their safety on the road.

## Problem Definition

Despite numerous advancements in road safety technology, driver fatigue remains a significant contributor to road accidents worldwide. Traditional methods of detecting fatigue, such as visual observation and self-reporting, are often unreliable and insufficient for timely intervention. As a result, there is a pressing need for an efficient and accurate system capable of real-time detection and alerting of driver fatigue. This paper aims to address this challenge by proposing the implementation of a machine learning-driven system designed specifically for driver fatigue detection and alerting. The system will leverage a combination of physiological and behavioural indicators to accurately identify fatigue patterns and promptly alert drivers, thereby enhancing road safety and reducing the risk of accidents caused by drowsy driving. Through this problem statement, we highlight the urgency and importance of developing such a system to mitigate the detrimental effects of driver fatigue on road safety.

**1.3 Existing System**

Currently, most driver fatigue detection systems rely on basic methods, like the driver or passengers observing signs of tiredness, which can be inaccurate. Some vehicles have systems that monitor steering wheel movements and lane changes, but these systems are not advanced enough to reliably detect fatigue. There are also wearable devices that track fatigue, but these aren't widely used because they can be uncomfortable.Recently, there's been more interest in using machine learning to detect fatigue. These systems look at things like eye movements, facial expressions, and heart rate to figure out if the driver is tired. While these systems show promise in controlled experiments, they still face challenges in real-world conditions. They need to be more reliable in different environments, work with individual drivers, and integrate well into vehicles.

Most of these machine learning systems are still in the testing phase and are not yet used in regular cars. There are still many hurdles to overcome, such as improving their performance and making them ready for widespread use. In short, while driver fatigue detection systems, especially those using machine learning, hold promise, they still need a lot of development and testing before they can be used in everyday vehicles.

## Proposed System

The proposed model for a real-time drowsiness detection system focuses on monitoring driver fatigue by analyzing key indicators such as eye movements, facial expressions, and head position. It tracks parameters like blink rate, yawning frequency, and head tilt to detect signs of fatigue. The system uses Convolutional Neural Networks (CNNs), a powerful machine learning technique, to process visual data captured by cameras monitoring the driver’s face. This enables the system to detect signs of drowsiness with high accuracy, even under challenging lighting conditions, by utilizing infrared cameras and optimizing the system settings for different environments.

In addition to detecting fatigue, the system is designed to reduce false alarms, ensuring that the alerts remain reliable across various lighting conditions and driver behaviours. If the system detects drowsiness and the driver fails to respond to the initial alarm, it takes further action by automatically slowing down the vehicle and activating the parking lights. This response ensures that the vehicle is safely brought to a stop, reducing the risk of accidents caused by driver fatigue. The combination of real-time monitoring, machine learning, and responsive actions makes this system an effective solution for improving driver safety on the road.

## Requirements Specification

### Software Requirements

### Operating System:

* + - Windows 10 or later, macOS, or a Linux distribution (e.g., Ubuntu).
    - 64-bit OS is recommended for better performance

### Arduino IDE:

* + - Arduino Integrated Development Environment (IDE) for programming and uploading code to Arduino Uno.

### Python:

* + - Python 3.7 or later.
    - Installed as part of the Anaconda distribution.

### Anaconda:

* + - Latest version of Anaconda, which includes Python, Jupyter Notebook, and many useful data science libraries.

### Visual Studio Code (VS Code):

* + - Latest version of VS Code.
    - Install Python extension for VS Code for better Python support.
    1. **Hardware Requirements**
* **Processor**: Minimum of quad-core CPU (e.g., Intel i5/i7 or AMD Ryzen 5/7)
* **Memory (RAM):** At least 8 GB, recommended 16 GB
* **Storage:** 20 GB of free disk space
* **Camera:** HD webcam or in-car camera (minimum 720p resolution) for real-time facial monitoring

**1.5.1.1 Python**

Python is a high-level, interpreted, interactive and object-oriented scripting language created by Guido Rossum in 1989. Python is designed to be highly readable. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large- scale projects. It is ideally designed for rapid prototyping of complex applications.

It has interfaces to many OS system calls and libraries and is extensible to C or C++. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science.

### History of Python

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language, capable of exception handling and interfacing with the Amoeba operating system. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle- detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python .Features of Python

Python's features include:

* **Easy-to-learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* **Easy-to-read**: Python code is more clearly defined and visible to the eyes.
* Easy-to-maintain: Python's source code is fairly easy-to-maintain.
* **A broad standard library**: Python's bulk of the library is very portable and cross- platform compatible on UNIX, Windows, and Macintosh.
* **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* **Databases**: Python provides interfaces to all major commercial databases.
* **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* **Scalable**: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

* IT supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte-code for building large applications.
* It provides very high-level dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.

**1.5.1.2 Python Libraries and Packages:**

### Machine Learning Libraries:

**Scipy :**

This module from the SciPy library is used for spatial calculations, particularly in

measuring the distance between points or vectors. The dist submodule provides various distance metrics such as Euclidean, Manhattan, and cosine distances. These can be used in machine learning, pattern recognition, and other applications where you need to measure the similarity or dissimilarity between data points.

### imutils.face\_utils :

The imutils library provides a set of convenience functions to make working with OpenCV easier. The face\_utils module within imutils offers several utilities to process facial landmarks, such as detecting the positions of facial features (e.g., eyes, mouth, nose). This is useful in face recognition, emotion detection, and drowsiness detection systems, where identifying specific facial landmarks is important.

### imutils:

imutils is a Python library designed to simplify working with OpenCV, making image processing tasks more intuitive and concise. It includes functions for resizing, rotating, and performing basic image transformations. It's often used in computer vision projects where you need to manipulate images easily, like resizing for faster processing or rotating images to align objects.

### matplotlib.pyplot:

matplotlib.pyplot is a plotting library used to create static, animated, or interactive visualizations in Python. It's commonly used for creating charts and graphs to visualize data. In drowsiness detection, matplotlib might be used to visualize the results of analysis, such as plotting the accuracy of the model over time or visualizing the facial landmarks on an image to verify the detection process.

### NumPy:

NumPy is a core library in Python for numerical computing. It provides support for large multi-dimensional arrays and matrices, along with a variety of mathematical functions to manipulate them. In drowsiness detection, NumPy is used to handle large datasets of image or video frames and perform mathematical operations on the data, such as array manipulation and statistical analysis.

### cv2 (OpenCV):

OpenCV (Open Source Computer Vision Library) is an open-source library for computer vision tasks. The cv2 module is widely used for image and video processing, including reading images, performing operations like blurring, thresholding, and edge detection, and handling video streams. In facial drowsiness detection, OpenCV helps capture and process video or image frames to detect driver behavior

### winsound:

The winsound module is a simple way to generate sound on Windows systems. It's used to play beeps or other sound effects, which can be useful in alert systems. For example, in driver drowsiness detection, winsound can be used to trigger an alarm when the system detects signs of drowsiness, such as eye closure or yawning

### sklearn:

These functions come from sklearn (Scikit-learn), a machine learning library. classification\_report provides a detailed summary of the performance of a classification model, including precision, recall, and F1-score for each class. accuracy score calculates the accuracy of the model by comparing the predicted values with the actual labels. These functions are essential for evaluating how well a machine learning model is performing, such as in a drowsiness detection system that classifies the driver as either alert or drowsy

### dlib:

dlib is a powerful toolkit used for machine learning and computer vision tasks. It provides

robust facial landmark detection, object detection, and facial recognition capabilities. In drowsiness detection or face recognition systems, dlib is used to locate and track key facial features (like eyes and mouth) in real-time. It's highly efficient and accurate in these tasks.

**1.5.1.3 Anaconda Distribution**

**Figure 1.2** Anaconda installation window on Windows operating system

Anaconda is a free, open-source distribution of the Python and R programming languages, developed for data science, machine learning, and scientific computing. Created by **Anaconda, Inc.**—founded by **Peter Wang** and **Travis Oliphant** in 2012—this platform simplifies package management and deployment for Windows, Linux, and macOS systems.

Anaconda comes pre-installed with many useful packages like NumPy, pandas, scikit-learn, and Jupyter Notebook, making it a go-to tool for data scientists and ML developers. It is available in different editions, including the **Individual Edition** (free), **Team Edition**, and **Enterprise Edition** (paid). With over **20 million users** worldwide, it is one of the most popular platforms for managing Python/R-based data science workflows.

**Usage of Anaconda**  
To use Anaconda, users can download the installer from the official website, selecting the version based on their operating system. Once installed, it provides access to tools like **Anaconda Navigator** (a GUI for managing environments and packages) and **conda** (a command-line package manager). Users can launch environments, install libraries, and run applications like Jupyter Notebook directly. Anaconda also allows easy switching between different Python versions and virtual environments.

**Main Features of Anaconda**

* **Pre-installed Libraries:** Comes with 250+ scientific libraries for data analysis, ML, and AI.
* **Cross-Platform Support:** Available for Windows, macOS, and Linux.
* **Environment Management:** Create and manage isolated environments using conda.
* **Integrated Tools:** Includes Jupyter Notebook, Spyder IDE, and Anaconda Navigator.
* **Scalability:** Suitable for single-machine use and scalable to enterprise-level solutions.

### 1.5.1.3 Visual Studio Code

**Figure 1.1** Visual Studio Code window on Windows operating system

* + - Visual Studio Code (VS Code) is a free open-source text editor by Microsoft. VS Code is available for Windows, Linux, and macOS. Although the editor is relatively lightweight, it includes some powerful features that have made VS Code one of the most popular development environment tools in recent times.

### MACHINE LEARNING

* + Machine Learning is a field of artificial intelligence (AI) that focuses on developing algorithms and techniques that enable computers to learn from data and make predictions or decisions without being explicitly programmed for each task. It involves creating models that can learn patterns and relationships from large datasets, allowing computers to perform tasks or make decisions based on that learned knowledge.
* The primary goal of machine learning is to develop algorithms that can improve their performance over time as they are exposed to more data. This process of learning from data is often referred to as training, where the algorithm adjusts its parameters or structure to optimize its performance on a specific task**.**
* There are several types of machine learning algorithms, including:

1. **Supervised Learning:** In supervised learning, the algorithm is trained on labelled data, where each example in the dataset is paired with a corresponding label or outcome. The algorithm learns to map inputs to outputs based on the provided examples, allowing it to make predictions on new, unseen data.
2. **Unsupervised Learning:** Unsupervised learning involves training algorithms on unlabelled data, where the algorithm must discover patterns or structures in the data without explicit guidance. This type of learning is often used for tasks such as clustering similar data points together or dimensionality reduction.
3. **Reinforcement Learning:** Reinforcement learning is a type of learning where an agent learns to interact with an environment by performing actions and receiving feedback or rewards. The goal of the agent is to learn a policy that maximizes its cumulative reward over time, leading to behaviours that are optimal for the given task.

### APPLICATION OF MACHINE LEARNING

* **Predictive Analytics:**
* Predicting future outcomes or trends based on historical data, such as forecasting sales or stock prices.

### 2.Image and Speech Recognition:

* Automatically identifying objects, people, or speech patterns in images or audio recordings.
* **Natural Language Processing:** Understanding and generating human language, enabling tasks such as translation, sentiment analysis, and chatbots.

### Healthcare:

* Assisting in medical diagnosis, personalized treatment recommendations, and drug .

# LITERATURE SURVEY

[1]. Saini et al. she have introduced titled "Real-Time Driver Drowsiness Detection for Autonomous Vehicles" by Vandna Saini (2016) explores the use of Convolutional Neural Networks (CNN) for detecting driver drowsiness in real time. The study focuses on monitoring eye closure, head movements, and yawning as indicators of fatigue. CNNs are applied to process visual data and track facial features to detect these signs of drowsiness, which can be crucial for preventing accidents in autonomous vehicles. However, the study presents two key limitations. First, the system demands high computational power, as real-time image processing using CNNs requires substantial resources, making it less feasible for use in low-power or embedded systems, such as those in vehicles. This computational intensity can increase costs and reduce the system's practicality for wide-scale deployment. Second, the paper mentions a limited dataset, which hinders the model’s ability to generalize well across diverse conditions. A small or unrepresentative dataset limits the system's effectiveness in detecting drowsiness under varied real-world conditions, such as different drivers, lighting environments, or driving scenarios, reducing the overall robustness and accuracy of the detection system. Despite these challenges, the study lays a foundation for further research in the field of drowsiness detection in autonomous vehicles.

[2]. Jagbeer Singh et al. he was introduced project with the titled Driver Drowsiness Detection System – An Approach by Machine Learning Application by Jagbeer Singh et al., a key limitation discussed is the system's performance drop under extreme lighting conditions. While the system utilizes a combination of face detection and eye-tracking techniques to assess drowsiness, it faces challenges when the driver is exposed to either very bright or very dim lighting. Extreme lighting can interfere with the camera's ability to accurately detect facial landmarks, such as the eyes and their movement, which are critical for monitoring drowsiness. In situations of intense backlighting or glare, the system may fail to accurately distinguish between open and closed eyes, leading to false positives or missed alerts. Similarly, under low light conditions, the camera may struggle to capture clear images, affecting the precision of eye detection. These lighting issues can degrade the accuracy of the system and result in less reliable performance, potentially compromising its ability to alert the driver in real-time. Therefore, one of the challenges identified in the paper is improving the system's robustness to various lighting conditions, which is crucial for its widespread deployment and effectiveness in real-world scenarios. Further research and optimization, such as adjusting camera settings or integrating infrared sensors, could help address these limitations.

[3]. Md. Ebrahim Shaik et al. In the context of driver drowsiness detection research, one of the key limitations is the requirement for large datasets to ensure the model's accuracy and reliability. Drowsiness detection systems rely on a variety of data inputs, such as facial recognition, eye-tracking, and physiological measurements, which must be collected over a substantial number of driving hours across diverse environments, times of day, and conditions. This extensive data collection can be time-consuming, costly, and logistically challenging, as it requires continuous monitoring of drivers in real-world scenarios, often over extended periods.

Furthermore, the setup for these systems can be complex. Integrating various sensors, such as cameras for facial and eye detection, physiological monitors for heart rate or brain activity, and vehicle-based data such as steering wheel behaviour or lane position, requires a sophisticated infrastructure. Ensuring these sensors work seamlessly together, along with the software needed to process and analyse the data, adds another layer of complexity. Additionally, fine-tuning the models to accurately detect drowsiness without generating false positives or negatives requires extensive testing, which can only be achieved with large, diverse datasets. These complexities and data demands can limit the scalability and implementation of drowsiness detection systems, making them challenging to deploy in real-world applications on a wide scale.

[4]. Sadegh Aref Nezhad et al. *"*Driver Drowsiness Detection Based on Steering Wheel Data Applying Adaptive Neuro-Fuzzy Feature Selection*"* (published in *Sensors*, Vol. 19, No. 4, p. 943, 2019) explores an innovative method for improving driver drowsiness detection systems using steering wheel data. The key focus of this study is the development of a component selection approach for creating subtle, yet effective driver indifference detection systems, which are crucial for enhancing road safety.

The proposed method involves adaptive neuro-fuzzy inference systems (ANFIS) for feature selection. This system aims to identify the most relevant features related to drowsiness levels by analyzing various steering wheel data channels. The feature selection is carried out using an advanced technique that combines the analysis of multiple input channels and utilizes neuro-fuzzy structures to fine-tune the system's performance.

Once the relevant features are selected, they are fed into a support vector machine (SVM) classifier for further analysis and decision-making. The classifier's role is to classify the driver's state into two distinct categories: *inertia* (normal driving behaviour) and *alarm* (indicating potential drowsiness). The research demonstrates the application of particle swarm optimization (PSO) to enhance the accuracy of the SVM classifier by adjusting the operating conditions and tuning the limits of the adaptive damping structure.

The results from the study, based on a dataset of 20.5 hours of driver operation, show significant improvement in drowsiness detection accuracy compared to traditional methods. The technique effectively detects drowsiness based on the steering wheel data, with the feature selection process ensuring that the most relevant indicators are used for classification. This smart feature selection and classification approach results in a more accurate and reliable driver monitoring system.

the study's approach using adaptive neuro-fuzzy feature selection and SVM classification presents a promising solution for detecting driver drowsiness, offering better accuracy than existing systems. The use of PSO for classifier tuning further enhances the performance of the system, making it a significant contribution to the field of driver monitoring technologies.

Table 1: Comparison of Literature survey

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Title** | **Authors** | **Year** | **Objective** | **Methodology** | **Results** | **Future Scope** |
| 1 | Driver Fatigue Detection Based on Visual Signals | Li X., Zhao L. | 2018 | Detect driver fatigue based on facial signals such as eye closure and yawning. | Support Vector Machines (SVM) used to classify eye states (open/closed) and detect yawning. | Achieved high accuracy in detecting driver fatigue; some false positives in non-fatigue cases. | Improvement of real-time processing speed and integration with stress detection systems. |
| 2 | Real-Time Driver Drowsiness Detection Using Deep Learning | Patel S., Kumar M. | 2020 | Implement a deep learning-based driver drowsiness detection system to enhance road safety. | CNNs used for real-time facial feature extraction, focusing on eye closure and head tilting for drowsiness detection. | Achieved over 85% accuracy in real-time drowsiness detection. | Expand to include more complex behavior like prolonged gaze diversion and more environmental variations. |
| 3 | Driver Distraction Detection Using Convolutional Neural Networks | Khan A., Ahmed Z. | 2021 | Detect driver distractions through facial feature analysis using deep learning techniques. | CNNs trained on real-time video data to recognize distracted states based on gaze direction and head pose estimation. | Successful detection of distracted drivers in controlled environments. | Integration with more environmental variables such as weather conditions, road types, and diverse lighting scenarios. |
| 4 | Machine Learning-based Drowsiness Detection Systems | Kim J., Choi Y. | 2019 | To develop a reliable system to detect drowsiness using machine learning algorithms and video input. | Combined Haar cascades and CNNs for facial landmark detection and classification of driver alertness using a trained dataset. | Detected drowsiness with significant precision, alerts triggered at early stages of drowsiness. | Expand the system for other health conditions like stress, sudden health deterioration, and non-visual cues. |
| 5 | Fatigue and Drowsiness Detection Using Blink Features | Smith J., Brown P. | 2022 | Investigate drowsiness detection by analyzing blink duration and frequency in real-time driving conditions. | Machine learning models analyzing blink rates and facial expressions from continuous video feeds. | Successfully detected early drowsiness stages in long-duration drives with 90% accuracy. | Extend analysis to head tilting and yawning detection for more robust systems. |
| 6 | Driver Stress Detection Using Multimodal Signals | Zhang Y., Wang T. | 2023 | Identify stress in drivers through facial expressions and multimodal physiological signals. | Used machine learning models combining facial features, heart rate, and galvanic skin response to detect stress. | Effective detection of stress levels correlated with aggressive or risky driving behavior. | Further exploration of long-term stress monitoring and inclusion of driving behavior patterns to enhance detection. |

**3.METHODOLOGY**

## Research Methodology

The research methodology for developing a real-time drowsiness detection system starts with collecting data through cameras that monitor the driver’s face. This data includes signs of fatigue like eye movements, blinking rate, yawning, and head tilt. The goal is to capture data from various drivers under different driving conditions, including varying lighting and driver behaviour, so the system can be effective in real-world situations.

Next, the collected data is cleaned and pre-processed to remove any noise, ensuring that the important features (like eye closure and head position) can be accurately extracted. These features are then used to train a machine learning model. The primary method used is Convolutional Neural Networks (CNNs), a type of machine learning that works well with visual data. The CNN is trained to identify drowsiness signals, and the model will be fine-tuned to minimize false alarms, improving accuracy.

To ensure the system works well in different lighting conditions, infrared cameras will be used alongside regular cameras. This helps maintain reliable detection even in low light or glare. Once drowsiness is detected, the system will alert the driver. If the driver doesn't respond, the system will automatically slow down the vehicle and turn on the parking lights as an additional safety feature.

Finally, the system will be tested in real driving scenarios to make sure it works under various conditions. This will help refine the system’s performance and ensure it remains reliable for detecting driver fatigue and improving road safety.

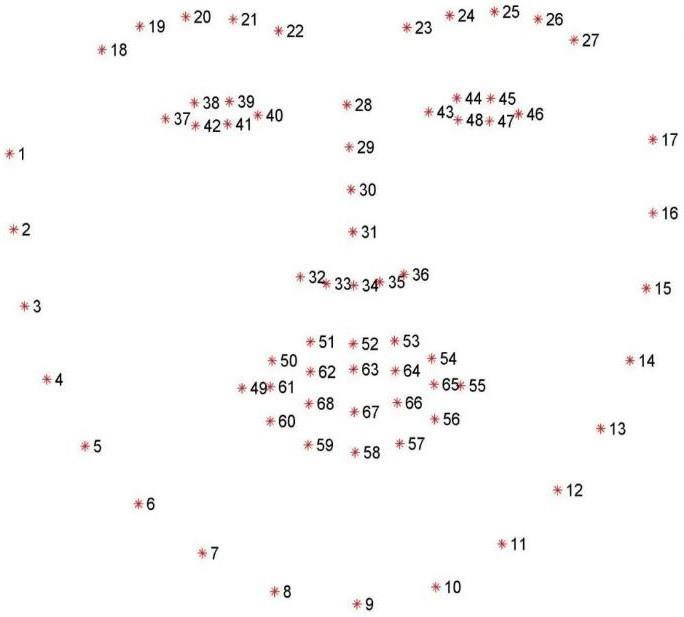
## Data Collection And Analysis

Collecting and analysing data for drowsiness detection using machine learning involves several steps to ensure model effectiveness and accuracy. The data collection primarily relies on pre-trained datasets and advanced facial recognition techniques.

The dataset, a free and open-source dataset from Kaggle, is used to train a CNN model. This dataset is divided into two parts: 75% for training and 25% for testing. Key feature values, such as eye and mouth characteristics, are used for evaluation.

The Dlib library is instrumental in detecting drowsiness through its pre-trained facial landmark detector (shape\_predictor\_68\_face\_landmarks.dat). This detector identifies 68 specific landmarks on the face, corresponding to critical facial regions such as the eyes, mouth, eyebrows, nose, and jawline. Each of these landmarks is vital for analysing drowsiness.

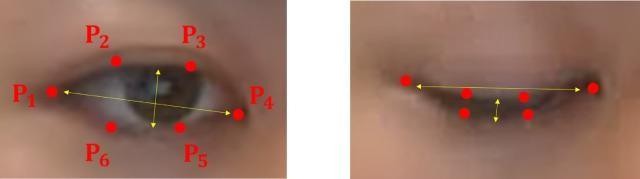
For example, the Eye Aspect Ratio (EAR) is calculated using the landmarks [36–41] for the left eye and [42–47] for the right eye. The EAR measures the degree of eye openness, and consistently low EAR values indicate prolonged eye closure, a key sign of drowsiness. Similarly, the Mouth Aspect Ratio (MAR), derived from landmarks [48–67], helps identify yawning, another significant indicator of fatigue.



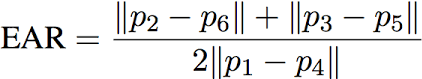
**Fig 3.1 68\_face\_landmarks**

### Eye Aspect Ratio (EAR) Detection:

The Eye Aspect Ratio (EAR) is a critical parameter for detecting drowsiness or fatigue by monitoring the openness of the eyes. EAR is calculated using specific facial landmarks around the eyes, typically involving the points that define the contours of the eyes and their surrounding regions. By measuring the vertical and horizontal distances between these landmarks, the EAR provides a numerical value that indicates the degree of eye openness. A consistently low EAR value, which occurs when the eyes remain closed for extended periods, is a strong indication of drowsiness. The threshold for EAR is usually set based on experimentation, with values below a certain point (commonly 0.2) signalling the driver is likely fatigued. If the EAR remains below the threshold for a specific time duration, the system can issue an alert to warn the driver of drowsiness.



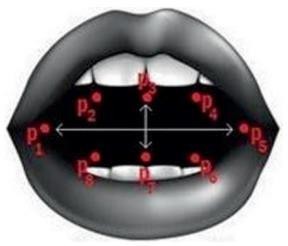
**Fig 3.2 Eye Detection**

****

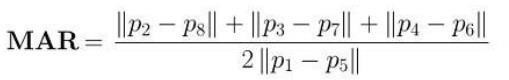
**Fig 3.2.1 formula for Eye Aspect Ratio (EAR)**

### Mouth Aspect Ratio (MAR) Detection:

The Mouth Aspect Ratio (MAR) is another important indicator in detecting drowsiness, particularly by analysing yawning behaviour. Yawning is often associated with fatigue and is an easily identifiable sign. The MAR is calculated by measuring the vertical distance between the upper and lower lip and dividing it by the horizontal distance between the corners of the mouth. A higher MAR value generally indicates that the mouth is wide open, a key sign of yawning. Yawning frequency can be a critical indicator of drowsiness, as drivers who are tired tend to yawn frequently. By monitoring the MAR, the system can detect these yawning events, which can serve as a signal for the onset of fatigue. MAR detection can be particularly effective when combined with EAR, as both parameters work together to provide a comprehensive assessment of the driver's alertness. Together, EAR and MAR form the foundation for a robust real-time drowsiness detection system, offering early warnings of fatigue and promoting safer driving practices.



**Fig 3.3 Mouth Detection**

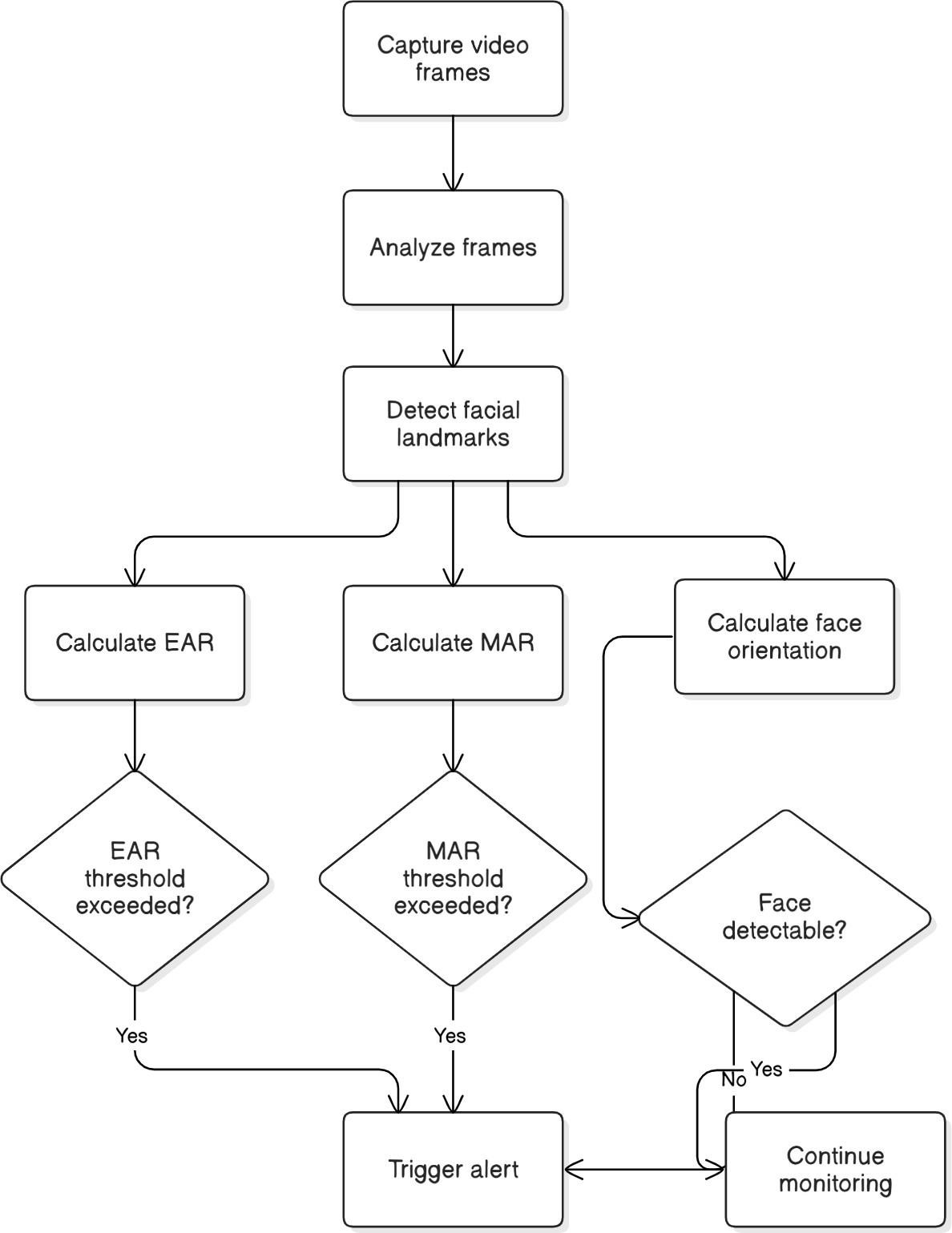


**Fig 3.3.1 formula for Mouth Aspect Ratio (MAR)**

**3.14. DIAGRAMMATICAL REPRESENTATION:**

## System Design

The System Design for drowsiness detection using machine learning focuses on integrating machine learning models with facial landmark detection to create an efficient and reliable solution for real-time drowsiness monitoring. The system is structured around several key components that work together to process images, detect facial landmarks, and identify drowsiness-related features, such as eye closure and yawning.



**Fig 3.4 System Design**

## The Model Architecture

the given project definition, we have used two different approaches towards the solution. First being the Deep learning model in which we built a Convolutional Neural Networks (CNN) with keras. A convolutional neural network is a special

type of deep neural network which performs extremely well for image classification purposes. A CNN basically consists of an input layer, an output layer and a hidden layer which can have multiple numbers of layers. A convolution operation is performed on these layers using a filter that performs 2D matrix multiplication on the layer and filter. It consists of multiple layers namely Convolutional layer, Fully connected layer, Input layer and output (Final) layer. Each layer has an activation function and an optimizer except the final(output) layer.



**Fig 3.5 Model Architecture**

The second approach to the solution of the Driver Drowsiness detection problem is the transfer learning method. Transfer learning is an efficient method for training models on large datasets. Transfer Learning basically takes a pre-existing model and retrains the model on the dataset provided by the user. The main advantage this technique provides is that the total training time for the model is less. But as we are taking an already trained model on a general dataset the model lacks the accuracy for the particular data. So, we can say that transfer learning is a trade off between model training time and model accuracy

## UML Diagrams

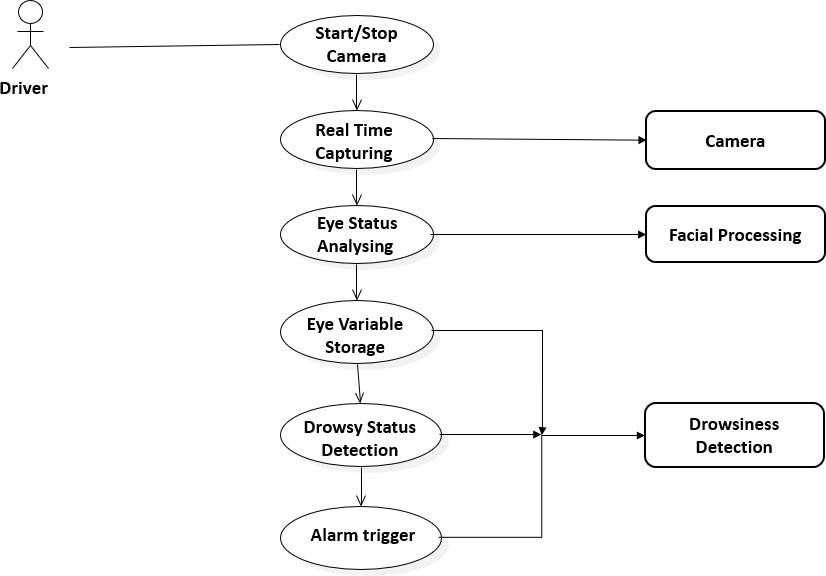
The Unified Modelling Language (UML) is a standardized visual language used to specify, visualize, modify, construct, and document the various artifacts of a system. In the case of the Drowsiness Detection System using machine learning and facial landmark detection, UML can be used to represent the system's architecture, components, processes, and interactions. This helps in effectively planning, communicating, and understanding the overall system design. The use of UML diagrams ensures that both technical and non-technical stakeholders can have a clear understanding of the system's components and workflow.

Key UML Diagrams for Drowsiness Detection System:

1. Use Case Diagram
2. Class Diagram
3. Sequence Diagram
4. Activity Diagram

### Use Case Diagram:

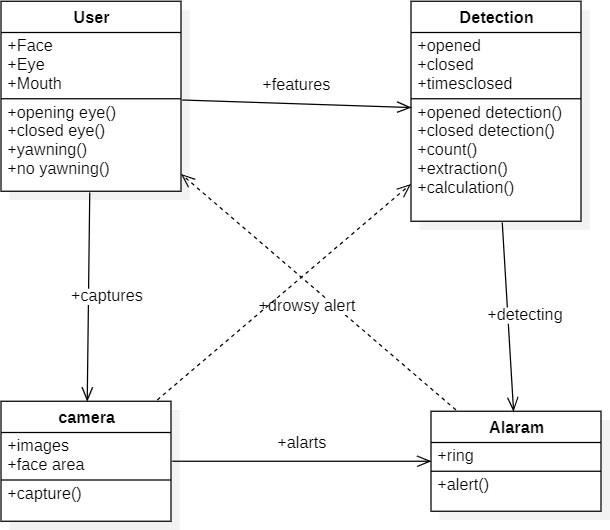
The Use Case Diagram for this system shows how different actors interact with the system. The main actor is the Driver, whose facial features are monitored for signs of drowsiness. The System processes these inputs and detects fatigue using facial analysis algorithms. If drowsiness is detected, the Alert System sends notifications to warn the driver. In some cases, the system can trigger automatic safety measures, such as slowing down the vehicle. Secondary actors, like an External Monitoring System, may also interact with the system for data logging or analysis. This diagram highlights how these actors work together to prevent accidents caused by driver fatigue.



**Fig 3.6 Use Case Diagram**

### Class Diagram

A Class Diagram is used to show the structure of a system by illustrating its classes, their attributes, and the relationships between them. For the drowsiness detection system, the main classes include Camera, which captures real-time images, and Face Detector, which detects the driver's face in these images. The Landmark Detector extracts facial landmarks used to calculate the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) with the help of the EAR Calculator and MAR Calculator classes. The Drowsiness Model uses the EAR and MAR values to detect drowsiness, while the Alert System sends warnings or notifications when drowsiness is detected. Finally, the Database stores images, detection results, and logs for analysis. The relationships between these classes show how they depend on and interact with each other, providing a clear blueprint for building the system.

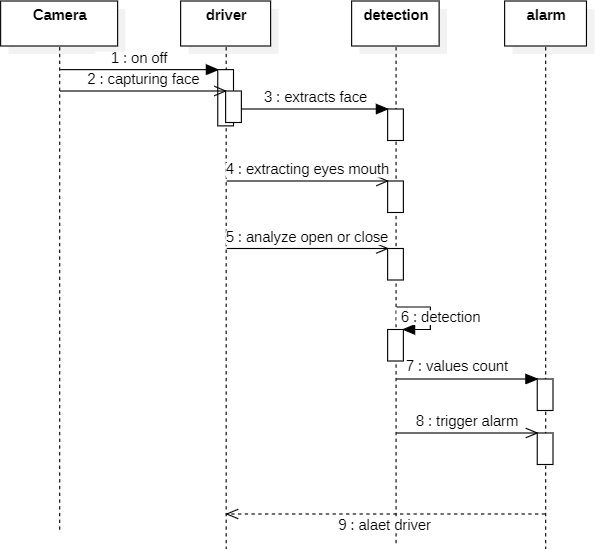


**Fig 3.7 Class Diagram**

### Sequence Diagram:

A Sequence Diagram illustrates the step-by-step interaction between components of a system, providing a clear visualization of their behavior in a specific order. For the drowsiness detection system, the process begins with the Camera capturing an image of the driver. This image is sent to the Face Detector, which identifies the location of the face in the frame. The Landmark Detector then pinpoints key facial features such as the eyes, nose, and mouth. These landmarks are critical inputs for subsequent calculations. The EAR (Eye Aspect Ratio) Calculator and MAR (Mouth Aspect Ratio) Calculator analyze these facial points to derive numerical values that indicate eye closure and yawning frequency.

These values are then fed into the Drowsiness Model, an AI-powered algorithm trained to detect signs of fatigue based on thresholds and patterns. If the model identifies signs of drowsiness, such as prolonged eye closure or frequent yawning, the Alert System is triggered to warn the driver. This warning can take various forms, such as an audible alarm, a vibrating seat, or flashing lights, ensuring the driver’s attention is quickly regained. By outlining these interactions, the sequence diagram not only highlights the functionality of each component but also emphasizes their integration, ensuring a seamless and effective drowsiness detection process.

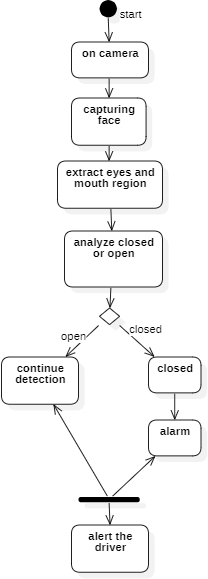


**Fig 3.8 Sequence Diagram**

### Activity Diagram:

The Activity Diagram outlines the workflow of the drowsiness detection system, beginning with the camera capturing real-time images of the driver. These images are processed to detect the driver’s face and extract facial landmarks using the Dlib library. These landmarks are then analyzed to calculate key metrics, including the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), which serve as critical indicators of the driver's alertness. The system continuously monitors these values, comparing them to predefined thresholds that differentiate between alert and fatigued states. This ensures that the system remains vigilant in detecting early signs of drowsiness.

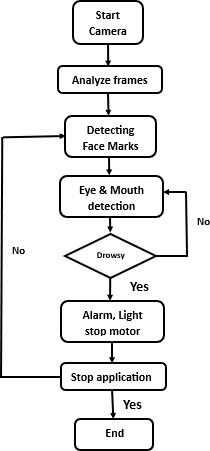
When the EAR or MAR values fall below their respective thresholds, the system flags potential signs of fatigue. It immediately triggers an alert to notify the driver through mechanisms such as audible alarms or visual warnings. These interventions prompt the driver to take corrective action, such as resting or regaining focus. Operating in real-time, the system ensures timely detection and response, significantly enhancing safety by reducing the likelihood of accidents caused by driver fatigue. This proactive approach underscores the system's effectiveness in addressing a critical issue in road safety.



**Fig 3.9 Activity Diagram**

### Dataflow Diagram :

the flow of a Drowsiness Detection System. It begins by starting the camera to capture video frames of the driver. These frames are analyzed to detect key facial features, such as the eyes and mouth. The system monitors these features for signs of drowsiness, like prolonged eye closure, frequent blinking, or yawning. If no drowsiness is detected, the system continues monitoring. If drowsiness is detected, it triggers safety measures such as sounding an alarm, flashing lights, or stopping the vehicle's motor after a delay. Finally, the system either continues monitoring or stops if the process is complete. This ensures real-time detection and response to prevent accidents caused by driver fatigue.



**Fig 3.10 Dataflow Diagram**

**4. CODING AND IMPLEMENTATION**

from flask import Flask, jsonify, Response, render\_template, request, send\_from\_directory, redirect

import threading

import cv2

import time

import numpy as np

import winsound

import os

from datetime import datetime

app = Flask(\_\_name\_\_)

detection\_running = False

status\_message = "Idle"

# Global variables for detection metrics

ear\_value = 0.0

mar\_value = 0.0

is\_alert = False

alert\_message = ""

# Adjust detection parameters for better accuracy

# Constants for thresholds

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml')

# Counter for consecutive frames with closed eyes

eye\_closed\_counter = 0

max\_closed\_frames = 15  # Increased threshold to reduce false positives

# Sound alert settings

frequency = 4000  # Increased from 2500 for a sharper sound

duration = 2000   # Increased from 1000 for a longer, more noticeable alert

# Minimum consecutive alert frames to trigger drowsiness

min\_alert\_frames = 3

alert\_counter = 0

# Original start\_detection route removed to fix duplicate route issue

@app.route('/stop\_detection', methods=['POST'])

def stop\_detection():

    global detection\_running

    detection\_running = False

    return jsonify({'status': 'stopped'})

@app.route('/status')

def get\_status():

    global status\_message, ear\_value, mar\_value, is\_alert, alert\_message

    return jsonify({

        'status': status\_message,

        'ear': ear\_value,

        'mar': mar\_value,

        'isAlert': is\_alert,

        'alertMessage': alert\_message

    })

@app.route('/login', methods=['POST'])

def login():

    data = request.get\_json()

    username = data.get('username')

    password = data.get('password')

    # Simple hardcoded authentication for demonstration

    if username == 'admin' and password == 'admin123':

        return jsonify({'success': True})

    else:

        return jsonify({'success': False, 'message': 'Invalid credentials'})

@app.route('/login.html')

def serve\_login():

    return send\_from\_directory('.', 'login.html')

@app.route('/dashboard.html')

def serve\_dashboard():

    return send\_from\_directory('.', 'dashboard.html')

@app.route('/app.js')

def serve\_app\_js():

    return send\_from\_directory('.', 'app.js')

@app.route('/index.html')

def serve\_index():

    return redirect('/dashboard.html')

@app.route('/')

def root():

    return redirect('/login.html')

# Global variables for video recording

recording = False

output\_video = None

recording\_start\_time = None

recorded\_videos = []

@app.route('/start\_recording', methods=['POST'])

def start\_recording():

    global recording, output\_video, recording\_start\_time

    if not recording:

        # Create recordings directory if it doesn't exist

        if not os.path.exists('recordings'):

            os.makedirs('recordings')

        # Generate filename with timestamp

        timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

        filename = f"recordings/drowsiness\_recording\_{timestamp}.avi"

        # Get webcam properties using the shared camera function

        cam = get\_camera()

        if cam is not None and cam.isOpened():

            ret, frame = cam.read()

            if ret:

                height, width, \_ = frame.shape

                fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

                output\_video = cv2.VideoWriter(filename, fourcc, 10.0, (width, height))

                recording = True

                recording\_start\_time = datetime.now()

                return jsonify({'status': 'recording\_started', 'filename': filename})

        # If we get here, camera access failed

        return jsonify({'status': 'error', 'message': 'Could not access webcam'})

    return jsonify({'status': 'already\_recording'})

@app.route('/stop\_recording', methods=['POST'])

def stop\_recording():

    global recording, output\_video, recording\_start\_time, recorded\_videos

    if recording and output\_video is not None:

        recording = False

        output\_video.release()

        # Calculate recording duration

        duration = (datetime.now() - recording\_start\_time).total\_seconds()

        # Add to list of recorded videos

        recorded\_videos.append({

            'filename': output\_video.filename,

            'timestamp': recording\_start\_time.strftime("%Y-%m-%d %H:%M:%S"),

            'duration': f"{int(duration // 60)}:{int(duration % 60):02d}"

        })

        output\_video = None

        return jsonify({'status': 'recording\_stopped', 'duration': duration})

    return jsonify({'status': 'not\_recording'})

@app.route('/recordings')

def get\_recordings():

    global recorded\_videos

    return jsonify({'recordings': recorded\_videos})

# Completely rewrite the video\_feed function and detection\_loop

# Global camera instance to be shared

camera = None

def get\_camera():

    global camera

    if camera is None:

        # Try multiple camera indices

        for camera\_index in range(3):  # Try indices 0, 1, 2

            print(f"Attempting to open camera with index {camera\_index}")

            camera = cv2.VideoCapture(camera\_index)

            if camera.isOpened():

                print(f"Successfully opened camera with index {camera\_index}")

                return camera

        # If we get here, no camera could be opened

        print("Error: Could not open any camera.")

        camera = None

        return None

    return camera

# Remove OpenCV cascade classifiers since we're using dlib now

# face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml')

# mouth\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_smile.xml')

# Define the start\_detection route with dlib-based detection

@app.route('/start\_detection', methods=['POST'])

def start\_detection():

    global detection\_running, eyeCounter, mouthCounter

    if not detection\_running:

        detection\_running = True

        # Reset all counters

        eyeCounter = 0

        mouthCounter = 0

        threading.Thread(target=detection\_loop).start()

    return jsonify({'status': 'started'})

@app.route('/video\_feed')

def video\_feed():

    def generate():

        global detection\_running, recording, output\_video, status\_message, is\_alert, alert\_message, ear\_value, mar\_value

        try:

            while True:

                # Get camera (may be None if no camera available)

                cam = get\_camera()

                if cam is None:

                    # Create a black frame as fallback

                    frame = np.zeros((480, 640, 3), dtype=np.uint8)

                    cv2.putText(frame, "NO CAMERA AVAILABLE", (120, 240), cv2.FONT\_HERSHEY\_SIMPLEX,

                                1, (0, 0, 255), 2, cv2.LINE\_AA)

                    # Update status

                    status\_message = "Error: Cannot access webcam"

                    is\_alert = True

                    alert\_message = "Cannot access webcam. Please check your camera."

                else:

                    # Capture frame with error handling

                    ret, frame = cam.read()

                    if not ret or frame is None:

                        # Create a black frame as fallback

                        frame = np.zeros((480, 640, 3), dtype=np.uint8)

                        cv2.putText(frame, "CAMERA ERROR", (180, 240), cv2.FONT\_HERSHEY\_SIMPLEX,

                                    1, (0, 0, 255), 2, cv2.LINE\_AA)

                        # Update status

                        status\_message = "Error: Cannot access webcam"

                        is\_alert = True

                        alert\_message = "Cannot access webcam. Please check your camera."

                        # Try to reinitialize camera

                        release\_camera()

                        time.sleep(0.5)

                        continue

                # Save frame to video if recording

                if recording and output\_video is not None and frame is not None:

                    try:

                        output\_video.write(frame)

                    except Exception as e:

                        print(f"Error writing to video: {e}")

                # Process the frame for detection if detection is running

                if detection\_running and frame is not None:

                    try:

                        process\_frame\_for\_detection(frame)

                    except Exception as e:

                        print(f"Error in detection: {e}")

                # Add recording indicator if recording

                if recording:

                    cv2.putText(frame, "RECORDING", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX,

                                1, (0, 0, 255), 2, cv2.LINE\_AA)

                # Encode the frame as JPEG

                try:

                    ret, jpeg = cv2.imencode('.jpg', frame)

                    frame\_bytes = jpeg.tobytes()

                    # Yield the frame in the format expected by multipart/x-mixed-replace

                    yield (b'--frame\r\n'

                           b'Content-Type: image/jpeg\r\n\r\n' + frame\_bytes + b'\r\n')

                except Exception as e:

                    print(f"Error encoding frame: {e}")

                    time.sleep(0.5)

                    continue

                # Use a much shorter delay to prevent freezing

                time.sleep(0.03)

        except Exception as e:

            print(f"Error in video feed: {e}")

        finally:

            pass  # Don't release the camera here, it's shared

    # Return a streaming response

    return Response(generate(),

                    mimetype='multipart/x-mixed-replace; boundary=frame')

# Add mouth cascade classifier

mouth\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_smile.xml')

def calculate\_mar(mouth\_points):

    """Calculate a simplified Mouth Aspect Ratio based on mouth rectangle"""

    # Extract width and height from the mouth rectangle

    x, y, w, h = mouth\_points

    # Calculate a simplified MAR (height/width ratio)

    # Normalize to be in similar range as typical MAR values (0.0-1.0)

    mar = min(1.0, (h / w) \* 2.0) if w > 0 else 0.5

    return mar

def process\_frame\_for\_detection(frame):

    global status\_message, ear\_value, mar\_value, is\_alert, alert\_message, eye\_closed\_counter, alert\_counter

    global eyeCounter, mouthCounter

    # Resize frame for faster processing

    frame = imutils.resize(frame, width=450)

    # Convert to grayscale for detection

    gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    # Detect faces using dlib's face detector

    rects = detector(gray, 0)

    # Check if no face is detected

    if len(rects) == 0:

        status\_message = "No face detected"

        is\_alert = True

        alert\_message = "No face detected! Please position yourself in front of the camera."

        cv2.putText(frame, "ALERT: NO FACE DETECTED!", (10, 30),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

        winsound.Beep(frequency, duration)

        return

    # Process each face

    for rect in rects:

        # Determine facial landmarks

        shape = predictor(gray, rect)

        shape = face\_utils.shape\_to\_np(shape)

        # Calculate face orientation

        orientation = faceOrientation(shape)

        # Check if the orientation is below the threshold (face not clear or not facing camera)

        if orientation < orientationThresh:

            status\_message = "Face turned"

            is\_alert = True

            alert\_message = "Face turned! Please face the camera directly."

            cv2.putText(frame, "ALERT: FACE TURNED!", (10, 90),

                        cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

            winsound.Beep(frequency, duration)

            continue

        # Check for face clarity

        if not faceClarity(shape):

            status\_message = "Face not clear"

            is\_alert = True

            alert\_message = "Face not clear! Please ensure proper lighting and distance."

            cv2.putText(frame, "ALERT: FACE NOT CLEAR!", (10, 120),

                        cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

            winsound.Beep(frequency, duration)

            continue

        # Extract eye regions and compute EAR

        leftEye = shape[lStart:lEnd]

        rightEye = shape[rStart:rEnd]

        leftEAR = eyeAspectRatio(leftEye)

        rightEAR = eyeAspectRatio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0

        ear\_value = ear  # Update global EAR value

        # Extract mouth region and compute MAR

        mouth = shape[mStart:mEnd]

        mar = mouthAspectRatio(mouth)

        mar\_value = mar  # Update global MAR value

        # Draw contours around the eyes and mouth for visualization

        leftEyeHull = cv2.convexHull(leftEye)

        rightEyeHull = cv2.convexHull(rightEye)

        mouthHull = cv2.convexHull(mouth)

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

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

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

        # Print EAR and MAR values on the frame

        cv2.putText(frame, f"EAR: {ear:.2f}", (300, 30),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2)

        cv2.putText(frame, f"MAR: {mar:.2f}", (300, 60),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2)

        # Check for eye closure (drowsiness detection)

        if ear < earThresh:

            eyeCounter += 1

            if eyeCounter >= earFrames:

                status\_message = "DROWSINESS DETECTED (EYES)"

                is\_alert = True

                alert\_message = "WAKE UP! Eyes closed for too long!"

                cv2.putText(frame, "DROWSINESS DETECTED (EYES)", (10, 30),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

                # Play alert sound

                winsound.Beep(frequency, duration)

        else:

            eyeCounter = max(0, eyeCounter - 2)  # Decrease counter faster when eyes are open

            # If eyes are open and no other alerts, update status

            if eyeCounter == 0 and mouthCounter == 0:

                status\_message = "Alert"

                is\_alert = False

                alert\_message = ""

        # Check for yawning (mouth open)

        if mar > marThresh:

            mouthCounter += 1

            if mouthCounter >= marFrames:

                status\_message = "DROWSINESS DETECTED (YAWNING)"

                is\_alert = True

                alert\_message = "WAKE UP! Yawning detected!"

                cv2.putText(frame, "DROWSINESS DETECTED (YAWNING)", (10, 60),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

                # Play alert sound

                winsound.Beep(frequency, duration)

        else:

            mouthCounter = max(0, mouthCounter - 1)  # Gradually decrease counter

        # If no drowsiness detected, update status

        if eyeCounter == 0 and mouthCounter == 0 and not is\_alert:

            status\_message = "Alert"

            is\_alert = False

            alert\_message = ""

def detection\_loop():

    global detection\_running, status\_message, is\_alert, alert\_message

    while detection\_running:

        # Just sleep a bit to prevent high CPU usage

        # All detection logic is now in process\_frame\_for\_detection

        time.sleep(0.1)

    status\_message = "Stopped"

    is\_alert = False

    alert\_message = ""

def release\_camera():

    global camera

    if camera is not None:

        camera.release()

        camera = None

# Add required imports for dlib-based detection

from scipy.spatial import distance as dist

from imutils import face\_utils

import imutils

import dlib

# Update global variables and constants for dlib-based detection

# Constants for thresholds and frame checks

earThresh = 0.3

earFrames = 15  # Reduced from 48 in original for faster response in web app

marThresh = 0.5

marFrames = 10  # Reduced from 15 in original for faster response

orientationThresh = 20  # Threshold for face orientation (in degrees)

# Initialize Dlib's face detector and shape predictor

shapePredictor = r"shape\_predictor\_68\_face\_landmarks (1).dat"

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor(shapePredictor)

# Get indexes of facial landmarks for eyes and mouth

(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]

(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]

(mStart, mEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["mouth"]

# Initialize counters for drowsiness detection

eyeCounter = 0

mouthCounter = 0

# Function to calculate eye aspect ratio (EAR)

def eyeAspectRatio(eye):

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

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

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

    ear = (A + B) / (2.0 \* C)

    return ear

# Function to calculate mouth aspect ratio (MAR)

def mouthAspectRatio(mouth):

    A = dist.euclidean(mouth[2], mouth[10])  # 51, 59

    B = dist.euclidean(mouth[4], mouth[8])   # 53, 57

    C = dist.euclidean(mouth[0], mouth[6])   # 49, 55

    mar = (A + B) / (2.0 \* C)

    return mar

# Function to calculate face orientation using landmarks

def faceOrientation(shape):

    # Calculate angle between nose tip and chin (vector) vs. horizontal (x-axis)

    nose\_tip = shape[30]

    chin = shape[8]

    dx = nose\_tip[0] - chin[0]

    dy = nose\_tip[1] - chin[1]

    angle = np.degrees(np.arctan2(dy, dx))

    return abs(angle)

# Function to check face clarity

def faceClarity(shape, minLandmarkDistance=15):

    # Calculate distances between some key landmarks (nose tip to eyes, etc.)

    nose\_tip = shape[30]

    left\_eye = shape[42]

    right\_eye = shape[39]

    nose\_to\_left\_eye\_dist = dist.euclidean(nose\_tip, left\_eye)

    nose\_to\_right\_eye\_dist = dist.euclidean(nose\_tip, right\_eye)

    # A rough threshold to decide if the face is clear

    if nose\_to\_left\_eye\_dist < minLandmarkDistance or nose\_to\_right\_eye\_dist < minLandmarkDistance:

        return False  # Face is not clear

    else:

        return True  # Face is clear

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

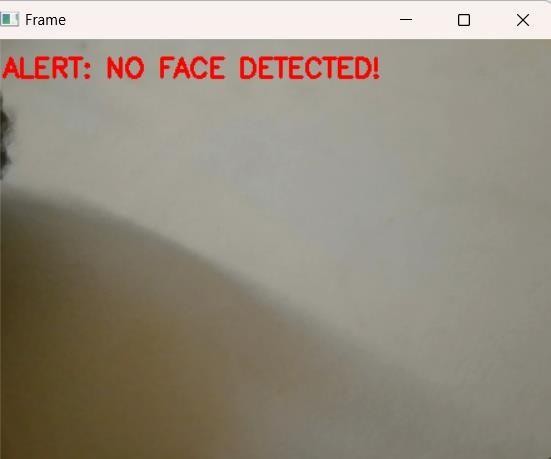
**5. EXECUTION AND RESULTS**

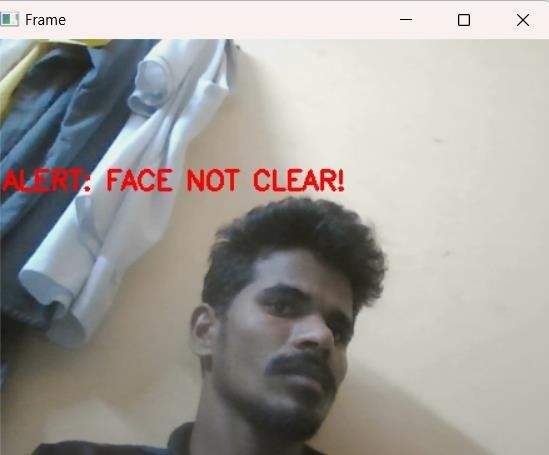
This Driver Drowsiness Detection System utilizes an infrared (IR) camera along with Machine Learning to detect signs of fatigue based on body posture, eye closure, and mouth yawning. The IR camera enables reliable monitoring even in low-light or nighttime conditions, ensuring continuous assessment of the driver’s posture, eye movements, and facial expressions. When drowsiness is detected, the system raises immediate alerts and displays a "Drowsy" message (Eye Detection), prompting the driver to take corrective action and enhancing road safety.

The IR camera also monitors the driver’s face in real time. If the face is not visible to the camera, the system triggers an immediate alert and displays a "Drowsy" message (No Face Detection). This ensures that the driver remains focused, helping to prevent accidents and promoting safe driving behavior at all times.

The camera frame will capturing the driver face to detect the drowsiness form the driver face when the driver is in normal condition the camera continuously framing the driver face when he closed the eyes the frame start detecting system raises immediate alerts and shows message as drowsy.

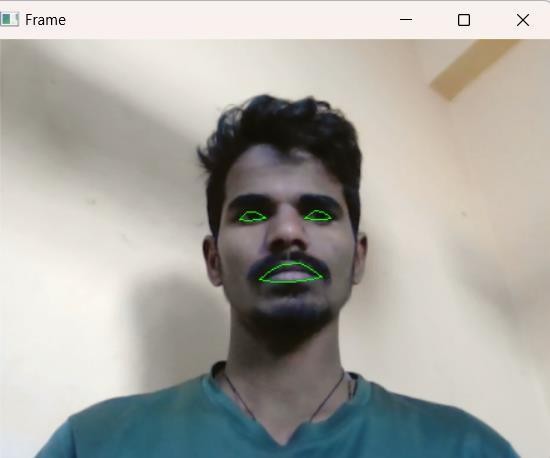
**Fig 5.1 No Face Detection**

**** The Driver Drowsiness Detection System uses an IR camera and Machine Learning to detect fatigue through eye movement, posture, and yawning. If drowsiness is detected, it alerts the driver with a "Drowsy" message. If the driver’s face isn’t visible (e.g., turning their head), the system raises an alert to keep the driver focused and ensure safety.



**Fig 5.2 Face Not Clear**

**Fig 5.3 Eye Detection with (Closed Eyes and Open Eyes )**

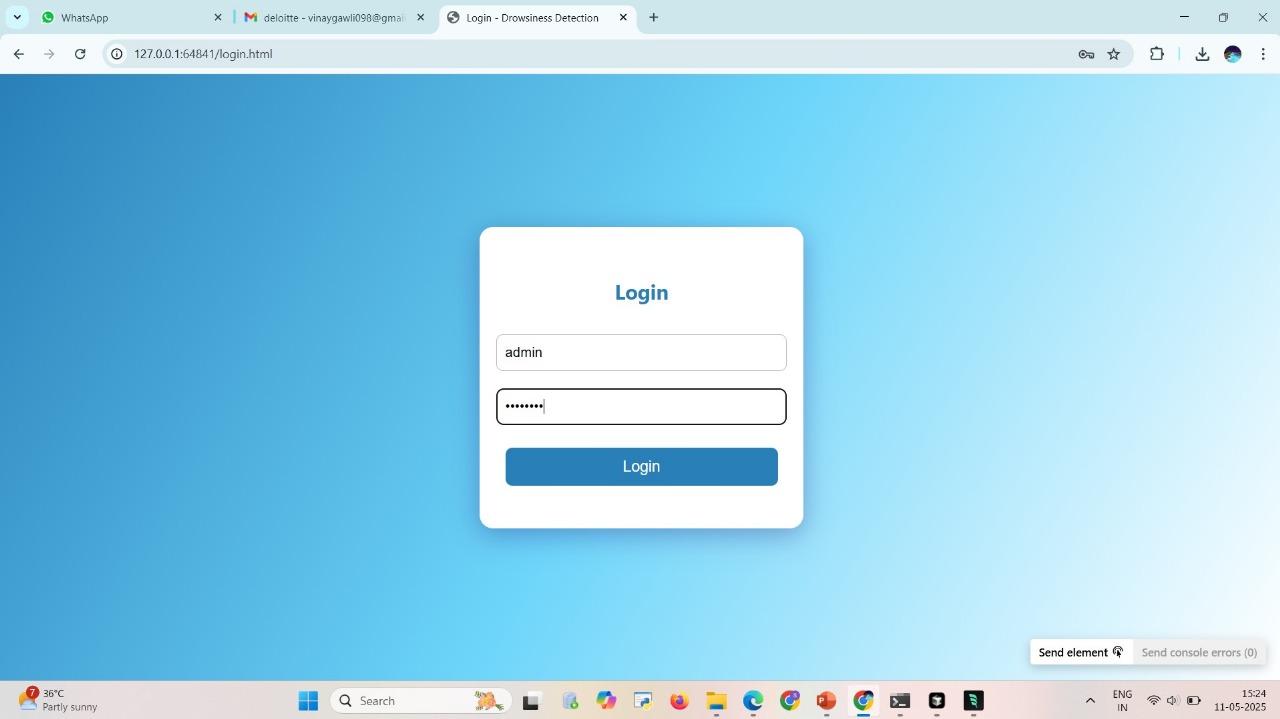


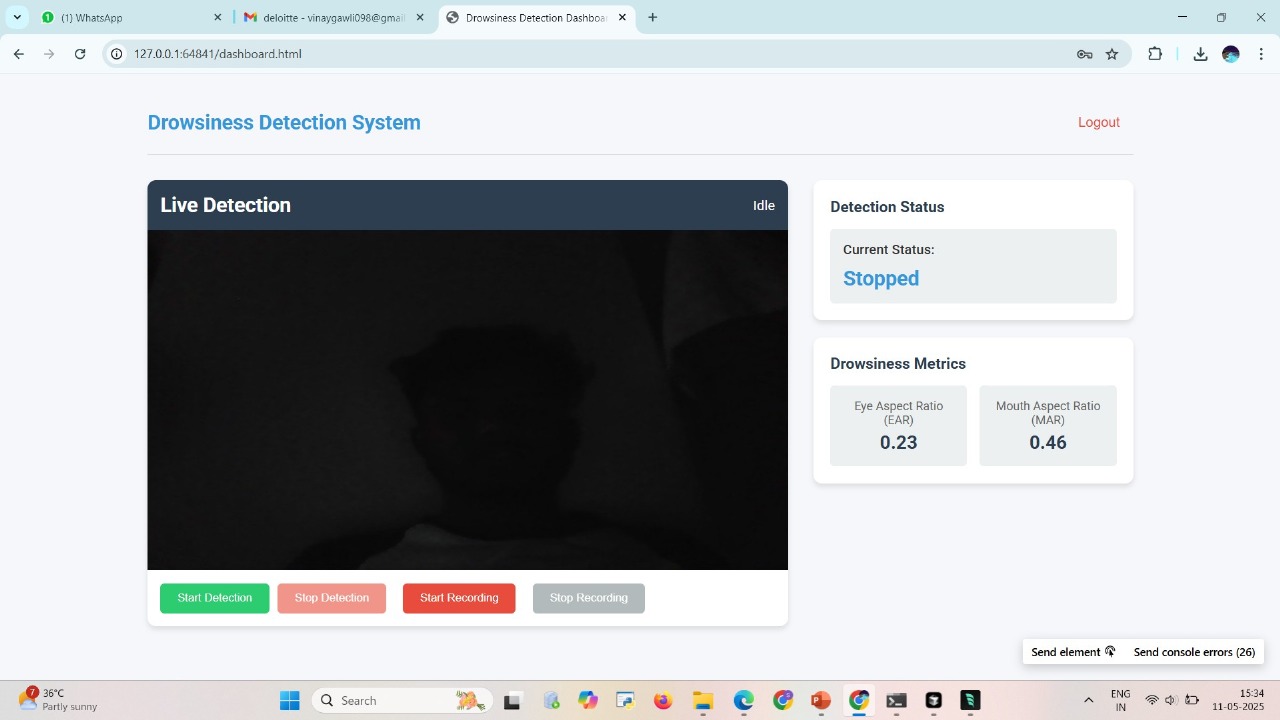
When the driver is in a normal condition, the camera continuously frames their face. If the driver starts yawning and opens their mouth, the system detects this and immediately raises an alert, showing a "Drowsy" message.

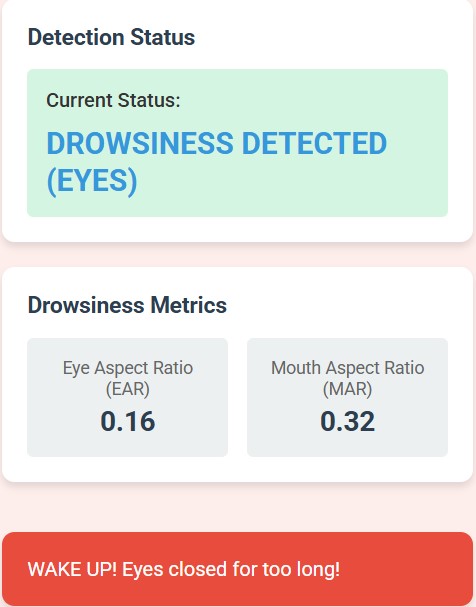
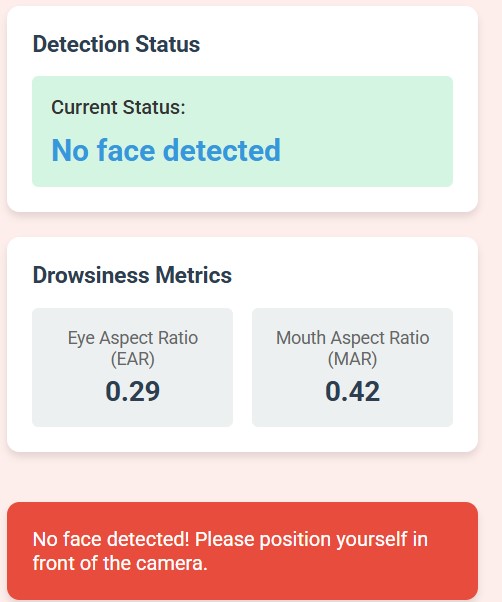


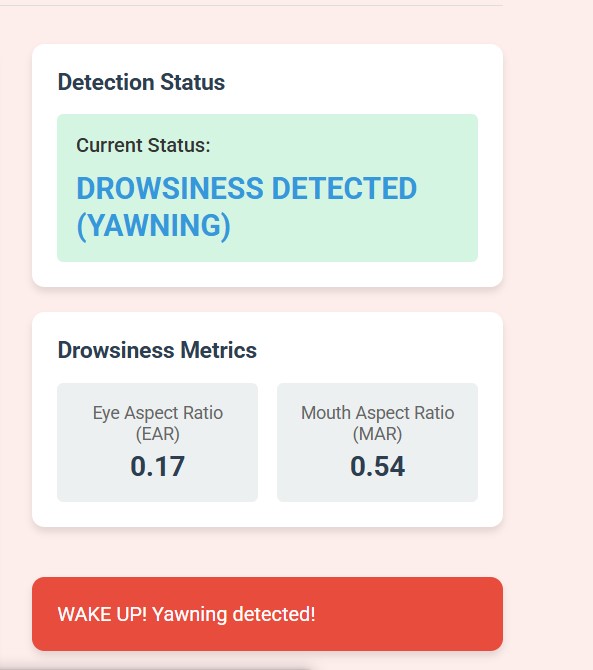
**Fig 5.4 Yawning Detection with (Yawning and Non-Yawning )**

When receiving a signal from the Python program, the Arduino activates the buzzer, hazard lights, and stops the motor on "A" (alert) or deactivates them and restarts the motor on "S" (safe).

****

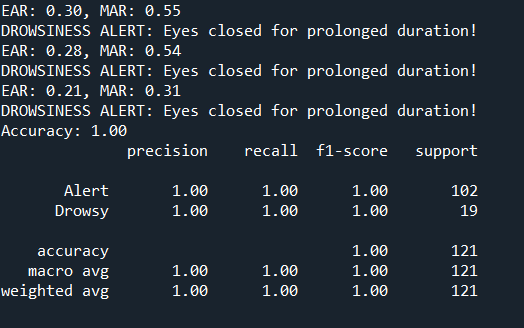
****

****

****

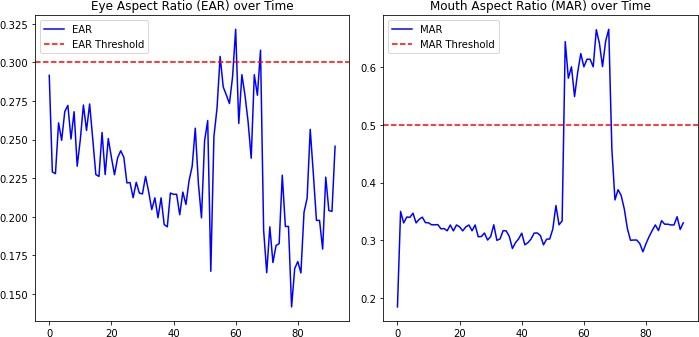
**Fig 5.5 Stages Of Web Application:**

Model accuracy for the drowsiness detection



**Fig 5.6 Accuracy**

line chart that displays the EAR and MAR values over time (frames).



**Fig 5.7 Line Chart**

1. **CONCLUSION**

Our current project develops a system for detecting drowsiness of the driver. This project is built using Python, OpenCV, ML model, Dlib and other open-source libraries. The system uses eye aspect ratio and mouth aspect ratio to detect blinks and yawning respectively and also a ML model is trained to draw the result based on them to achieve the main objective of project i.e., Driver’s Drowsiness. The framework has reached a stable state in which all bugs have been eliminated. The results are discussed in testing section and are found satisfactory. Our project, provides a way through which a number of road accidents might be avoided if an alert is sent to a driver that is deemed drowsy. Our model is not only useful to the person who will installs it in their vehicle but also for the other cars, trucks, buses and humans moving around it.

1. **FUTURE ENHANCEMENT**

Here are some Features to enhance and expand your Driver Drowsiness Detection System. By integrating additional sensors and data sources like EEG to track brain activity and GPS for location-based risk analysis, the system can become more accurate and proactive. It can also detect medical conditions like diabetes and improve overall driver comfort and safety.

### Expand Detection Capabilities:

**Drowsiness and Fatigue:**

* Analyse micro-sleeps (brief periods of unconsciousness) through eyelid flutter patterns and head movement. o Integrate bio-signals like heart rate variability for a more comprehensive drowsiness assessment.

### Distraction Detection:

* Track head orientation and pupil dilation to identify if the driver is looking at phones, navigation systems, or outside the vehicle.
* Analyse facial expressions to potentially detect cognitive distraction (lost in thought).

### Health Monitoring:

* Monitor signs of stress or anger through facial expressions and physiological changes. o Partner with medical professionals to detect potential health issues based on specific parameters (e.g., excessive yawning as a sign of sleep).

### Enhance System Functionality:

**Adaptive Alerts:** Personalize alerts based on driver data (baseline blink rate, typical fatigue patterns) to avoid false positives.

* **Interventions:** Combine visual (on-screen warnings) with auditory (alerts, voice messages) or haptic (seat vibrations) feedback for better driver engagement.
* **Driver Coaching:** Provide personalized feedback on driving behaviour to promote long- term safe driving habits.

### Additional Considerations:

* **Privacy Concerns:** Address privacy concerns by ensuring data anonymization, secure storage, and clear communication to users about data collection practices.
* **System Calibration:** Explore ways for the system to adapt to individual drivers (e.g., baseline eye closure duration)

**BIBILOGRAPHY**

1. Manu, B. N. 2016. Facial features monitoring for real time drowsiness detection. 2016 12th International Conference on Innovations in Information Technology (IIT).
2. Alshaqaqi, B., Baquhaizel, A. S., Amine Ouis, M. E., Boumehed, M., Ouamri, A., & Keche, M. 2013. Driver drowsiness detection
3. system. 2013 28th International Workshop on Systems, Signal Processing and Their Applications (WoSSPA).
4. Baek, J. W., Han, B.-G., Kim, K.-J., Chung, Y.-S., & Lee, S.-I. 2018. Real-Time Drowsiness Detection Algorithm for Driver State
5. Monitoring Systems. 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN).
6. You, F., Li, X., Gong, Y., Wang, H., & Li, H. 2019. A Real-time Driving Drowsiness Detection Algorithm with Individual
7. Differences Consideration*.* IEEE Access*,* vol. 7, pp. 179396-179408.
8. Dasgupta, A., Rahman, D., & Routray, A. 2018. A Smartphone-Based Drowsiness Detection and Warning System for Automotive
9. Drivers. IEEE Transactions on Intelligent Transportation Systems, 1–10.
10. Eraldo, B., Quispe, G., Chavez-Arias, H., Raymundo-Ibanez, C., & Dominguez, F. 2019. Design of a control and monitoring system
11. to reduce traffic accidents due to drowsiness through image processing. 2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX).
12. Lashkov, I., Kashevnik, A., Shilov, N., Parfenov, V., & Shabaev, A. 2019. Driver Dangerous State Detection Based on OpenCV.
13. Dlib Libraries Using Mobile Video Processing. 2019 IEEE International Conference on Computational Science and Engineering
14. (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC).
15. Zhang, S., & Wang, X. 2013. Human detection and object tracking based on Histograms of Oriented Gradients. 2013 Ninth
16. International Conference on Natural Computation (ICNC).
17. Kamran, M., Mannan, M., & Jeong, Y. 2019. Drowsiness, Fatigue and Poor Sleep ‘s Causes and Detection: A Comprehensive Study*.*