

ABSTRACT

Nowadays, driver safety in vehicles is one of the most wanted systems to avoid accidents. The drowsiness of drivers is one of the significant causes of road accidents. Every year, there is an increase in the number of deaths and fatal injuries globally due to drowsiness. By detecting the driver's drowsiness, we can reduce road accidents. The main objective of the project describes a machine learning approach for drowsiness detection. We use OpenCV, to detect faces and eyes and Support Vector Machine (SVM) for image classification to explore the level of the driver's drowsiness. Face detection is employed to locate the regions of the driver's eyes, which are used as templates for eye tracking in subsequent frames. Finally, the tracked eye images are used for drowsiness detection in order to generate warning alarms. Image processing is used to recognize the face of the driver and then it extracts the image of the eyes of the driver for detection of drowsiness. Thus, the proposed approach for real-time driver drowsiness detection is a low-cost and effective solution method.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	Title	i
	Certificate	ii
	Acknowledgment	iii
	Declaration	iv
	Abstract	a
	Table of Contents	b-c
	List of Figures	d
Chapter – 1	Introduction	1-2
Chapter – 2	Literature Survey	3-7
	2.1. Existing System	5-6
	2.2. Proposed System	7
Chapter – 3	System Requirements	8-13
	3.1. System Specifications	8
	3.2. Python	9-13
Chapter – 4	Methodology	14-18
Chapter – 5	Design	19-28
	5.1. Input Design	19
	5.2. Output Design	20
	5.3. Project Architecture	21-22
	5.4. UML Diagram	23
	5.5. Use Case Diagram	24
	5.6. Class Diagram	25
	5.7. Activity Diagram	26

	5.8. Collaboration Diagram	27
	5.9. Sequence Diagram	28
Chapter – 6	Code Implementation	29-47
Chapter – 7	Result	48-51
Chapter – 8	Conclusion and Scope of Future Work	52
Chapter – 9	Bibliography	53-54

LIST OF FIGURES

SL.NO	NAME OF THE FIGURE	PAGE NO
1.	Python Download	10
2.	Python Installation	10
3.	Face Landmarks	15
4.	Feature Extraction	15
5.	Eye Blinks	16
6.	Project Architecture	22
7.	Use Case Diagram	24
8.	Class Diagram	25
9.	Activity Diagram	26
10.	Collaboration Diagram	27
11.	Sequence Diagram	28

Chapter-1

INTRODUCTION

Humans have always invented machines and devised techniques to ease and protect their lives, for mundane activities like traveling to work, or for more interesting purposes like aircraft travel. In modern times, almost everyone in this world uses some sort of transportation every day. Some people are rich enough to have their own vehicles while others use public transportation. However, there are some rules and codes of conduct for those who drive irrespective of their social status.

One of them is staying alert and active while driving. Neglecting our duties toward safer travel has enabled hundreds of thousands of tragedies to get associated with this wonderful invention every year. One kind of carelessness is not admitting when we are too tired to drive. In order to monitor and prevent a destructive outcome from such negligence, at times, some of the points and observations made by the system are not accurate enough. Hence, to provide data and another perspective on the problem at hand, in order to improve their implementations and to further optimize the solution, this project has been done.

Driver safety is a concern for many nowadays. Millions of people every year lose their lives in traffic accidents, according to statistics from the World Health Organisation (WHO). Driver inattention and exhaustion account for the great majority of fatal collisions. The American Automobile Association claims that sleeping drivers are to blame for 21% of fatal traffic accidents and 7% of all accidents.

As relevant technologies have developed, numerous businesses as well as researchers have been investigating and creating driver fatigue systems in order to provide driver safety. Thus, the goal of the proposed system is to avoid traffic accidents brought on by tired drivers.

The technology created by Parmar [1] in 2002 uses a monochrome camera to detect the driver's eyes and determine whether they are open or closed. To determine the state detection of theirs, Yufeng et al. [2] analysed two photos of the drivers while they were on the road. In 2008, Horng and his colleagues [3] considered the drivers' eyes, counting the number of closed frames throughout 30 frames per second. In the system developed by Bajaj and colleagues [4] in 2010, face and mouth conditions are taken as basis, and using the obtained data and fuzzy logic algorithm are used to extract fatigue level. Coetzer et al. [5] compares YSA, SVM and

AdaBoost classification algorithms for eye detection, in the study they performed in 2011. Anumas and colleagues [6] focused on the causes of driver fatigue and driving impairment in their 2012 work. Zhang and his colleagues [7] used the convolutional neural network (CoNN)* algorithm in their work to try to detect the flex in the drivers. In an effort to identify insomnia, Zhang et al. [8] examined observations from 2017 and applied the CoNN approach to identify eye health. Bus drivers were utilised in a study by Mandal and colleagues [9] to identify driver weariness. Principal component analysis and the Adaboost algorithm were used by Savaş et al. [10] in an effort to detect driver weariness. Eye detection and PERCLOS were employed in this investigation. In their 2017 paper, Yuen and his colleagues [11] compare facial analysis techniques. As a result, the frame for facial detection is advised.

Therefore, there is a need to develop a system that will detect and notify a driver of her/him bad psychophysical condition, which could significantly reduce the number of fatigue-related car accidents. However, the biggest difficulties in the development of such a system are related to fast and proper recognition of a driver's fatigue symptoms. Due to the increasing number of vehicles on the road, which translates into road accidents directly, equipping a car with a fatigue detection system is a must. One of the technical possibilities to implement such a system is to use a vision-based approach. With the rapid development of image analysis techniques and methods, and a number of ready Component-on-the-Shelf solutions (e.g., high-resolution cameras, embedded systems, sensors), it can be envisaged, that introducing such systems into widespread use should be easy. Car drivers, truck drivers, taxi drivers, etc. should be allowed to use this solution to increase the safety of the passengers, other road users, and the goods they carry.

Chapter-2

LITERATURE SURVEY

In 2008, Hong Su et. al. described 'A Partial Least Squares Regression-Based Fusion Model for Predicting the Trend in Drowsiness'. They proposed a new technique of modeling driver drowsiness with multiple eyelid movement features based on an information fusion technique—partial least squares regression (PLSR), with which to cope with the problem of strong collinear relations among eyelid movement features and, thus, predicting the tendency of the drowsiness. The predictive precision and robustness of the model thus established are validated, which shows that it provides a novel way of fusing multi-features together for enhancing our capability of detecting and predicting the state of drowsiness.

In June 2010, Bin Yang et. al. described 'Camera-based Drowsiness Reference for Driver State Classification under Real Driving Conditions'. They proposed that measures of the driver's eyes are capable to detect drowsiness under simulator or experiment conditions. The performance of the latest eye-tracking-based in-vehicle fatigue prediction measures are evaluated. These measures are assessed statistically and by a classification method based on a large dataset of 90 hours of real road drives. The results show that eye-tracking drowsiness detection works well for some drivers as long as the blinks detection works properly. Even with some proposed improvements, however, there are still problems with bad light conditions and for persons wearing glasses. In summary, the camera-based sleepiness measures provide a valuable contribution to a drowsiness reference but are not reliable enough to be the only reference.

In 2011, M.J. Flores et. al. described a 'Driver drowsiness detection system under infrared illumination for an intelligent vehicle'. They proposed that to reduce the number of such fatalities, a module for an advanced driver assistance system, which caters to automatic driver drowsiness detection and also driver distraction, is presented. Artificial intelligence algorithms are used to process the visual information in order to locate, track and analyze both the driver's face and eyes to compute the drowsiness and distraction indexes. This real-time system works during nocturnal conditions as a result of a near-infrared lighting system.

Finally, examples of different driver images taken in a real vehicle at night time are shown to validate the proposed algorithms.

In June 2012, A. Cheng et. al. described 'Driver Drowsiness Recognition Based on Computer Vision Technology. They presented a nonintrusive drowsiness recognition method using eye-tracking and image processing. A robust eye detection algorithm is introduced to address the problems caused by changes in illumination and driver posture. Six measures are calculated with the percentage of eyelid closure, maximum closure duration, blink frequency, the average opening level of the eyes, opening velocity of the eyes, and closing velocity of the eyes. These measures are combined using Fisher's linear discriminated functions using a stepwise method to reduce the correlations and extract an independent index. Results with six participants in driving simulator experiments demonstrate the feasibility of this video-based drowsiness recognition method that provided 86% accuracy.

In 2013, G. Kong et. al. described 'Visual Analysis of Eye State and Head Pose for Driver Alertness Monitoring. They presented a visual analysis of eye state and head pose (HP) for continuous monitoring of the alertness of a vehicle driver. Most existing approaches to visual detection of non-alert driving patterns rely either on eye closure or head nodding angles to determine the driver's drowsiness or distraction level. The proposed scheme uses visual features such as eye index (EI), pupil activity (PA), and HP to extract critical information on the non-alertness of a vehicle driver. A support vector machine (SVM) classifies a sequence of video segments into alert or non-alert driving events. Experimental results show that the proposed scheme offers high classification accuracy with acceptably low errors and false alarms for people of various ethnicity and gender in real road driving conditions.

In June 2014, Eyosiyas et. al. described 'Driver Drowsiness Detection through \HMM based Dynamic Modelling'. They proposed a new method of analyzing the facial expression of the driver through Hidden Markov Model (HMM) based dynamic modeling to detect drowsiness. They have implemented the algorithm using a simulated driving setup. Experimental results verified the effectiveness of the proposed method.

In August 2014, García et. al. described 'Driver Monitoring Based on Low-Cost 3D Sensors'. They proposed a solution for driver monitoring and event detection based on 3-D

information from a range camera is presented. The system combines 2-D and 3-D techniques to provide head pose estimation and regions-of-interest identification. Based on the captured cloud of 3-D points from the sensor and analyzing the 2-D projection, the points corresponding to the head are determined and extracted for further analysis. Later, head poses estimation with three degrees of freedom (Euler angles) is estimated based on the iterative closest points algorithm. Finally, relevant regions of the face are identified and used for further analysis, e.g., event detection and behavior analysis. The resulting application is a 3-D driver monitoring system based on low-cost sensors. It represents an interesting tool for human factor research studies, allowing the automatic study of specific factors and the detection of special events related to the driver, e.g., driver drowsiness, inattention, or head pose.

2.1 Existing System

Majorly accidents are caused by drivers who are feeling drowsy. To control the accidents the systems are developed. To implement a system that can monitor a driver's drowsiness, to cameras are required. The first camera should be positioned to capture the driver's face while the second camera should be focused on the driver's eyes. The camera capturing the driver's face should be able to detect facial landmarks such as the eyes, nose, mouth, and eyebrows. This can be done using computer vision techniques such as facial recognition algorithms or machine learning models trained on facial landmark detection. The second camera monitoring the driver's eyes should be able to track eye movements and identify signs of drowsiness such as drooping eyelids or slow eye movements. This can be achieved using eye-tracking technology or machine learning models trained on eye movement patterns associated with drowsiness. Once both cameras are capturing the required information, the data can be analyzed using machine learning algorithms to determine the driver's level of drowsiness. The system can be programmed to sound an alarm or alert the driver if it detects signs of drowsiness, potentially preventing accidents caused by driver fatigue. The use of sensors attached to the human body, specifically for detecting drowsiness in drivers, is a common approach in many existing systems. However, the existing system may have some limitations in terms of detecting drowsiness in a timely and accurate manner. sensors can be attached to the driver's body to measure physiological signals such as heart rate, respiration rate, or brain activity. These signals can provide valuable information about the driver's level of alertness and help to identify signs of

drowsiness. These sensors can be combined with computer vision algorithms to analyze the driver's behavior in real-time and issue warnings when signs of drowsiness are detected. The old machine learning approach for drowsiness detection using convolutional neural networks (CN) involves training a model on a large dataset of images of drivers' faces and eyes to classify the images into drowsy and non-drowsy states. While this approach has been effective to some extent, it also has some disadvantages, including:

To detect drowsiness, it takes extra time. If the eyes are closed for five successive frames the system concludes that the driver is declining slumbering and issues a warning signal. Hence the existing system is not very helpful in detecting drowsiness even though it is detecting but it takes so much of time.

2.1.1 Disadvantages in the Existing System

High cost: The use of multiple cameras and sensors in the system increases the cost of the system, making it expensive for some users.

Physical discomfort: The use of sensors attached to the human body can cause physical discomfort to the driver, affecting their driving experience and potentially leading to incorrect readings.

High computational time: The processing of data from multiple cameras and sensors requires significant computational resources and can result in high computational time, leading to a delay in issuing warning signals to the driver.

Limited accuracy: It may not always accurately detect drowsiness, especially in cases where the driver's eyes are only partially closed, or when they are in a state of partial drowsiness.

Limited adaptability: CNN-based models are not easily adaptable to new and changing situations, such as different lighting conditions or different facial structures of drivers.

Limited scalability: CNN-based models can be computationally expensive, requiring significant computing resources for training and inference, which limits their scalability to larger datasets or real-time applications.

Limited generalization: CNN-based models trained on one specific dataset or population may not generalize well to other populations or datasets, leading to reduced accuracy.

2.2 Proposed System

To overcome the disadvantages in the existing system, we are exploring new approaches to drowsiness detection that rely on only a single camera by using OpenCV and SVM. OpenCV detects the face and eyes by using an SVM classifier. SVM for image classification. Instead of investing more amount of money, we can invest less cost to purchase requirements and our system gives the output quickly. If the driver feels drowsy the machine quickly detects and gives the alarm to alert the driver.

2.2.1 Advantages of the Proposed System

Low cost: Using only a single camera and readily available software like OpenCV and SVM, the proposed system is much more cost-effective compared to traditional drowsiness detection systems, which typically require multiple cameras and sensors.

Non-intrusive: The proposed system does not require any sensors or devices to be attached to the driver's body, which eliminates the possibility of physical discomfort or incorrect readings.

Fast processing time: The system uses machine learning algorithms that are optimized for fast processing, allowing for quick and accurate detection of drowsiness.

High accuracy: By using a combination of OpenCV and SVM, the proposed system can accurately detect facial features and classify images into drowsy and non-drowsy states, with a high degree of accuracy.

Easy implementation: OpenCV and SVM are widely used software tools, which makes the proposed system easy to implement and integrate with other software or hardware systems.

Better Alertness: It quickly alerts the driver very efficiently by giving the warning alarm. Decreasing major road accidents.

Chapter-3

SYSTEM REQUIREMENTS

3.1 System Specifications

3.1.1 Software Requirements:

- **Editor** : VS code
- **Programming Language** : Python
- **Operating System** : Windows 7 and above versions
- **Libraries** : OpenCV & D-lib

3.1.2 Hardware Requirements:

- **RAM** : minimum of 4GB and above is required.
- **Processor** : i3 and above.
- **Web cam**
- **Speaker**

3.2. PYTHON

3.2.1. Introduction:

Python is a widely used general-purpose, high-level programming language. It was created by Guido Van Rossum in 1991 and further developed by the python software foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

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

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding make it very interactive for Rapid Application development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

3.2.2 Setup of Python:

- Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install python.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of python <https://www.python.org/>.

3.2.2.1 Installation (using python IDLE):

- To start, go to python.org/downloads and click on the button to download the latest version of python.
- We can download python IDLE in windows, mac, and Linux operating systems also.



Fig 3.2.2.1.1. Python download

- Run the .exe file that you just downloaded and start the installation of python by clicking on install now.
- We can give environmental variables i.e., path after completion of downloading.



Figure 3.2.2.1.2. Python Installation

3.2.3. Features of Python:

1. **Readable:** Python is a readable language
2. **Easy to learn:** Learning python is easy as this is an expressive and high-level programming language, which means it is easy to understand the language and thus easy to learn.

3. **Cross-platform:** Python is available and can run on various operating systems such as Mac, Windows, Linux, UNIX, etc. This makes it a cross-platform and portable language.
4. **Open source:** Python is an open-source programming language
5. **Large Standard Library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in python.
6. **Free:** Python is free to download and use. This means you can download it for free and use it in your applications. Python is an example of a FLOSS (Free/Libre Open-Source Software), which means you can freely distribute copies of this software, read its source code and modify it.
7. **Supports Exception Handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program execution and can disrupt the normal flow of a program. Python supports exception handling which means we can write less error-prone code and can test various scenarios that can cause an exception later on.
8. **Advanced Features:** Supports generators and list comprehensions.
9. **Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. you do not have to bother about clearing the memory.

OpenCV:

OpenCV (Open Source Computer Vision Library) plays a significant role in drowsiness detection systems by providing essential computer vision functionalities. Here are some key roles of OpenCV in a drowsiness detection system:

Face detection: OpenCV offers pre-trained models and algorithms for face detection. It allows you to identify and locate faces within an image or video stream, which is the initial step in drowsiness detection. By detecting the face, you can then focus on the region of interest (ROI) where the eyes are located.

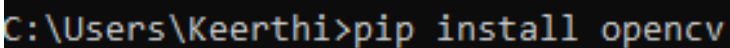
Eye detection: OpenCV provides algorithms for eye detection, enabling you to identify and locate the eyes within the detected face region. This step is crucial for analysing eye-related features and determining the drowsiness level.

Eye aspect ratio (EAR) calculation: OpenCV helps calculate the eye aspect ratio, a metric commonly used in drowsiness detection. By measuring the distances between specific landmarks on the eye (such as the inner and outer corners), OpenCV assists in determining if the eyes are open, closed, or partially closed.

Blink detection: With the calculated EAR values, OpenCV can perform blink detection by setting a threshold to determine if a blink has occurred. Analysing the sequence of eye states allows the system to detect drowsiness patterns, triggering appropriate alerts or interventions.

Visualization and alert mechanisms: OpenCV enables you to overlay visualizations on the video stream, such as drawing rectangles around the detected faces or eyes. Additionally, OpenCV can be integrated with other components of the system to trigger alerts, such as sounds, notifications, or vibrations, when drowsiness is detected.

Installation of OpenCV



```
C:\Users\Keerthi>pip install opencv
```

Dlib:

dlib is a powerful C++ library that is widely used for various computer vision tasks, including face detection, facial landmark detection, and shape prediction. In the context of a drowsiness detection system, dlib can play a role in detecting facial landmarks, such as eye corners, which are crucial for calculating the eye aspect ratio (EAR) and determining drowsiness levels.

Here are the key roles of dlib in a drowsiness detection system:

Facial landmark detection: dlib provides algorithms and pre-trained models for detecting facial landmarks, including the positions of eyes, nose, mouth, and other key points on a face. By detecting the eye corners, you can calculate the eye aspect ratio, which is a key feature for drowsiness detection.

Shape prediction: dlib's shape prediction functionality helps in accurately estimating the facial landmarks even when the face is partially occluded or not perfectly aligned. This robustness can be valuable in real-world scenarios where the person's face may not be fully visible or in the same pose as the training data.

Installation of dlib:

```
C:\Users\skusawa\Downloads>pip install "dlib-19.22.99-cp310-cp310-win_amd64.whl"
```

Scipy:

Scipy is a powerful scientific computing library in Python that provides various modules for numerical computations, optimization, signal processing, and more. While Scipy itself may not have a direct role in drowsiness detection, it can be useful for some additional functionalities and supporting computations in a drowsiness detection system. Here are a few potential roles of Scipy

Signal processing: Scipy's signal processing module offers a range of functions for filtering, Fourier analysis, spectral analysis, and other signal processing operations. In a drowsiness detection system, you can utilize these functions to pre-process the audio or video signals for noise reduction, frequency analysis, or other relevant signal operations.

Statistical analysis: Scipy's statistics module provides numerous statistical functions, probability distributions, and statistical tests. You can utilize these capabilities to analyse and interpret the data collected during the drowsiness detection process. For instance, you can compute statistical metrics, such as mean, standard deviation, or perform hypothesis testing to analyse the effectiveness of the drowsiness detection algorithm.

Optimization: Scipy's optimization module offers various optimization algorithms that can be useful for fine-tuning parameters or optimizing certain aspects of the drowsiness detection system. For instance, you may employ optimization techniques to adjust the threshold values or parameters used for blink detection based on the collected data.

Installation of Scipy:

```
C:\Users\Keerthi>pip install scipy
```

CHAPTER 4

METHODOLOGY

4.1 SVM Classifier:

In a Drowsiness Detection System, the role of a Support Vector Machines (SVM) classification algorithm is to classify whether a person is drowsy or not based on the input features. An SVM classifier takes input features that will present the state of the driver, such as eye blink duration and eye closure duration, and eye closure frequency, and outputs a binary classification indicating whether the driver is drowsy or not.

Support Vector Machine (SVM) is a relatively simple Supervised Machine Learning Algorithm that can be used for both classification and regression tasks. In Drowsiness Detection System, it is to classify and detects drowsiness based on input features.

By employing SVM in the drowsiness detection system, it becomes possible to train a model that can effectively distinguish between drowsy and alert states based on the extracted features. This enables the system to provide timely warnings or interventions when drowsiness is detected, promoting safety and preventing accidents, especially in scenarios such as driving or operating heavy machinery.

SVM has several advantages, such as its ability to handle high-dimensional data, resistance to overfitting, and effectiveness even with limited training data. However, SVM can be computationally expensive, especially for large datasets, and the choice of kernel and hyperparameters can greatly impact the results

4.1.1 Steps to Use SVM Classifier for Drowsiness Detection

Collect Data: SVM collects data from different drivers and records their eye state (open or closed) over time. Additionally, gather data on other features that may affect drowsiness such as time of day, driving speed, etc. The first step in developing a drowsiness detection system is to collect relevant data. This typically involves gathering features that can help differentiate between drowsy and awake states, such as eye closure duration, blinking rate, and head movements from facial landmarks.

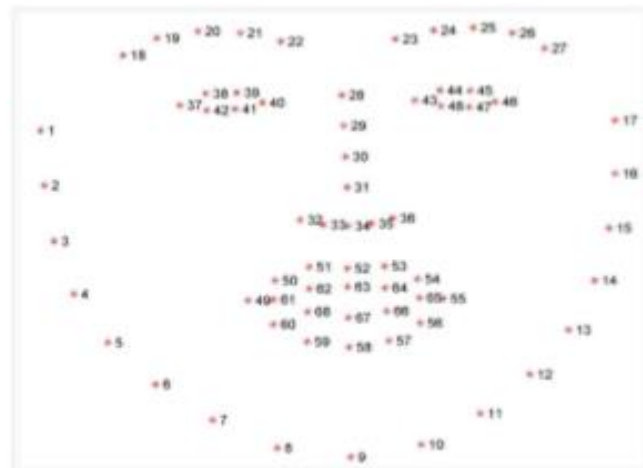


Figure 4.1.1.1. Face Landmarks

Feature Extraction: Extract features from the collected data. In the case of drowsiness detection, common features used are eye blink duration, eye closure duration, and eye closure frequency. Once the data is collected, relevant features need to be extracted from the raw data. Feature extraction techniques help to represent the collected data in a meaningful and compact way. This step aims to capture the most discriminative characteristics related to drowsiness.

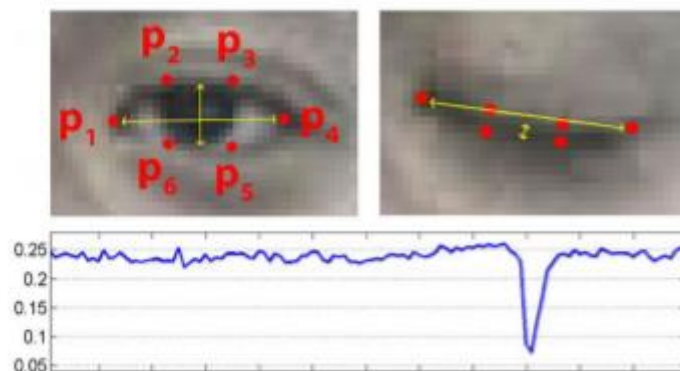


Figure 4.1.1.2. Feature Extraction

Data Pre-processing: Clean the collected data and perform any necessary pre-processing such as normalizing or scaling the features.: Before training an SVM classifier, pre-processing steps are often applied to the data. This can include filtering to remove noise or artifacts, normalization to bring features to a similar scale, and feature extraction techniques to derive informative features from raw data. Pre-processing helps improve the quality and effectiveness of the SVM classifier.

Train SVM: Train an SVM classifier using the extracted and pre-processed features. The SVM will learn to classify whether a person is drowsy or not based on the input features. The labeled data samples are used to train the SVM model. The SVM algorithm learns a decision boundary or hyperplane that best separates the drowsy and awake states in the feature space. The SVM algorithm aims to find the optimal hyperplane with the maximum margin of separation between the two classes.

Test and Evaluate: Test the SVM classifier on a new set of data and evaluate its performance. Common performance metrics for classifiers include accuracy, precision, recall, and F1 score.

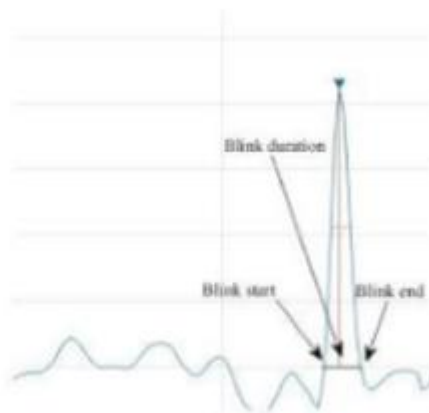


Figure 4.1.1.3. Eye Blink's

Deployment: Once the SVM classifier is trained and tested, it can be deployed in a drowsiness detection system where it can classify whether a driver is drowsy or not in real-time-based on input from a webcam and other data sources.

Overall, SVM is a popular choice for classification problems like drowsiness detection because it is effective at handling high-dimensional feature spaces and can generalize well to new data.

In practical applications, real-time drowsiness detection is crucial. Therefore, considerations should be made for the computational efficiency and speed of the SVM

classifier. Optimized implementations, feature dimensionality reduction techniques (e.g., Principal Component Analysis), or approximation methods can be employed to enable real-time processing.

It's important to note that the specific variant or modifications of SVM used in a drowsiness detection system can vary across different research studies or commercial implementations. The choice of features, pre-processing techniques, and other considerations depends on the specific requirements, available data, and the expertise of the researchers or developers involved in the project.

Pre-processing Steps or Feature Selection Techniques Applied to the Data in Drowsiness Detection System.

In a drowsiness detection system, various pre-processing steps and feature selection techniques can be applied to the data to enhance the quality and relevance of the features used for classification. Here are some common pre-processing steps and feature selection techniques employed in drowsiness detection systems:

- 1. Signal Filtering:** The input signals, such as EEG, EOG, or EMG, may be contaminated with noise or artifacts. Signal filtering techniques, such as high-pass, low-pass, band-pass, or notch filters, can be applied to remove unwanted noise and artifacts while retaining the relevant frequency components.
- 2. Artifact Removal:** Certain artifacts, such as eye blinks or muscle movements, can interfere with the drowsiness detection process. Various methods, including independent component analysis (ICA) or template matching, can be employed to identify and remove these artifacts from the signals.
- 3. Dimensionality Reduction:** High-dimensional feature spaces can be computationally expensive and may lead to overfitting. Dimensionality reduction techniques, such as principal component analysis (PCA) or linear discriminant analysis (LDA), can be applied to reduce the number of features while preserving the most relevant information.
- 4. Feature Normalization:** It is common to normalize the features to a common scale to ensure that they contribute equally during the classification process. Normalization techniques, such as z-score normalization or min-max scaling, can be used to scale the features to a standard range (e.g., between 0 and 1) or to have zero mean and unit variance.
- 5. Feature Extraction:** Drowsiness detection systems often require extracting relevant features from the input signals. Feature extraction techniques aim to capture informative

patterns and characteristics related to drowsiness. These techniques can include time-domain analysis (e.g., statistical measures, waveform parameters), frequency-domain analysis (e.g., power spectral density, spectral entropy), or time-frequency analysis (e.g., wavelet transform, spectrogram).

6. Feature Selection: In some cases, not all extracted features may be equally informative or contribute significantly to the drowsiness detection task. Feature selection methods can be applied to identify the most relevant features and discard redundant or less discriminative ones. Common feature selection techniques include correlation analysis, mutual information, recursive feature elimination (RFE), or genetic algorithms.

7. Windowing: To analyze continuous data streams, windowing techniques are often employed to segment the data into smaller frames or time windows. This allows for the examination of specific temporal patterns within the signals. Various window types, such as rectangular, Hamming, or Hanning windows, can be applied depending on the specific requirements of the analysis.

These pre-processing steps and feature selection techniques help optimize the input data for the drowsiness detection system, improving the performance and interpretability of the subsequent classification algorithms, such as SVM. The specific choices and combinations of these techniques can vary depending on the nature of the data, the sensor modality, and the requirements of the drowsiness detection system.

Chapter-5

DESIGN

5.1. INPUT DESIGN:

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data into a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps, and keeping the process simple. The input is designed in such a way that it provides security and ease of use with retaining privacy. Input design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog guides the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when errors occur.

OBJECTIVES:

1. Input design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
2. It is achieved by creating user-friendly screens for the data entry to handle large volumes of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also performs record viewing facilities.
3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus, the objective of input design is to create an input layout that is easy to follow.

5.2 OUTPUT DESIGN:

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other systems through outputs. In output design it is determined how the information is to be displaced for immediate and also the hard copy out. It is the most important and direct source of information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

- ❖ Designing computer output should proceed in an organized, well-thought-out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analyzing design computer output, they should identify the specific output that is needed to meet the requirements.
- ❖ Select methods for presenting the information.
- ❖ Create documents, reports, or other formats that contain information produced by the system. The output form of an information system should accomplish one or more of the following objectives.
- ❖ Convey information about past activities, current status, or projections of the future.
- ❖ Signal important events, opportunities, problems, or warnings.
- ❖ Trigger an action.
- ❖ Confirm an action.

5.3. Project Architecture:

This architecture presents an overview of the drowsiness detection system, which combines hardware and software components to monitor and analyze the level of driver drowsiness. The system employs a sequential process involving face detection, eye detection, determination of the level of fatigue, and generating an alarm. Additionally, it includes a feedback loop mechanism to ensure accurate detection in case any stage malfunctions. The purpose of this system is to enhance driver safety by alerting individuals about their fatigue levels and mitigating the risk of accidents.

The architecture of a drowsiness detection system is aimed at safeguarding driver and passenger well-being. It describes the sequential stages involved in the system, highlighting their significance in assessing the driver's fatigue levels. By employing state-of-the-art hardware and software components, this system offers an effective solution to reduce accidents caused by drowsy driving.

Face Detection

The initial stage of the drowsiness detection system involves capturing an image of the driver's face. Using advanced algorithms and image processing techniques, the system identifies and locates the driver's face within the captured image. This process provides the foundational step for subsequent stages by ensuring accurate identification of the driver.

Eye Detection

Once the driver's face has been successfully detected, the system proceeds to the eye detection stage. Utilizing sophisticated image analysis algorithms, the system isolates and extracts the driver's eyes from the previously detected face image. This stage is crucial as it focuses on the most indicative region for assessing drowsiness.

Level of Fatigue Determination

Having obtained the accurate eye region, the system analyzes various visual cues to determine the level of fatigue exhibited by the driver. By considering factors such as eyelid closure, eye movement patterns, and pupil dilation, the system employs advanced machine learning algorithms to assess the driver's fatigue level accurately. The outcome of this stage serves as a critical input for subsequent actions.

Generate Alarm

Upon determining the driver's level of fatigue, the system generates an alarm to alert the driver and mitigate potential accidents. The alarm may consist of visual, auditory, or tactile signals designed to capture the driver's attention. By providing timely warnings, this stage aims to prevent or minimize the consequences of drowsiness-related incidents.

Feedback Loop Mechanism

Recognizing the importance of reliable drowsiness detection, the system incorporates a feedback loop mechanism. This mechanism ensures that in the event of any malfunction or erroneous detection at any stage, the system reverts to the initial stage and repeats the sequential process. By continuously assessing and reevaluating the driver's state, the system maintains accuracy and minimizes false positives or false negatives.

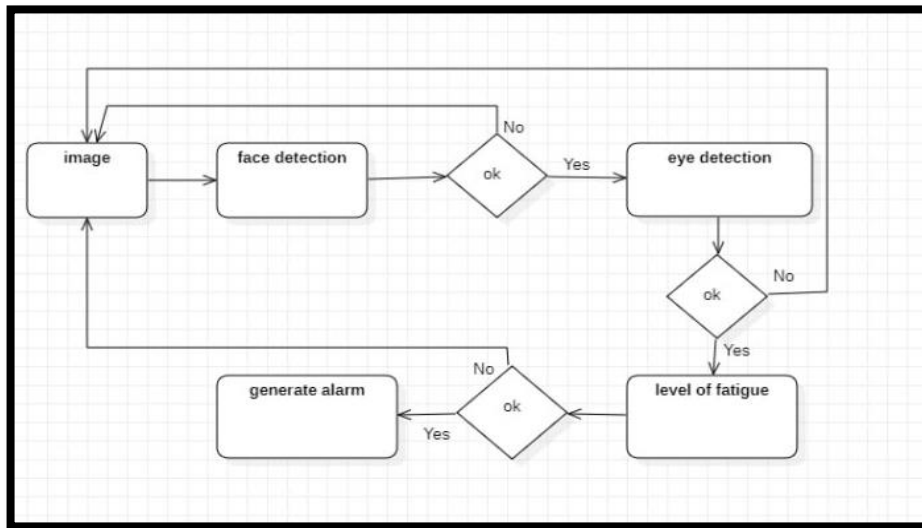


Figure 5.3.1 Project Architecture

5.4 UML DIAGRAMS:

UML stands for Unified Modelling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed and was created by, the Object Management Group.

The Goal for UML is to become a common language for creating models of object-oriented computer software. In its current form, UML is comprised of two major components: a Meta model and a notation. In the future, some form of method or process may also be added to or associated with, UML.

The UML is a standard language for specifying, Visualization, Constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven the success in modeling large and complex systems.

GOALS:

The primary goal in the design of the UML is as follows:

1. Provide users with a ready-to-use, expressive visual modeling language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of a particular programming language and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns, and components.
7. Integrate best practices.

5.5 USE CASE DIAGRAM:

A Use Case Diagram in the UML is a type of behavioral diagram defined by and created from a use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use-case is to show what system functions are performed for which actor. The roles of the actors in the system can be depicted as follows:

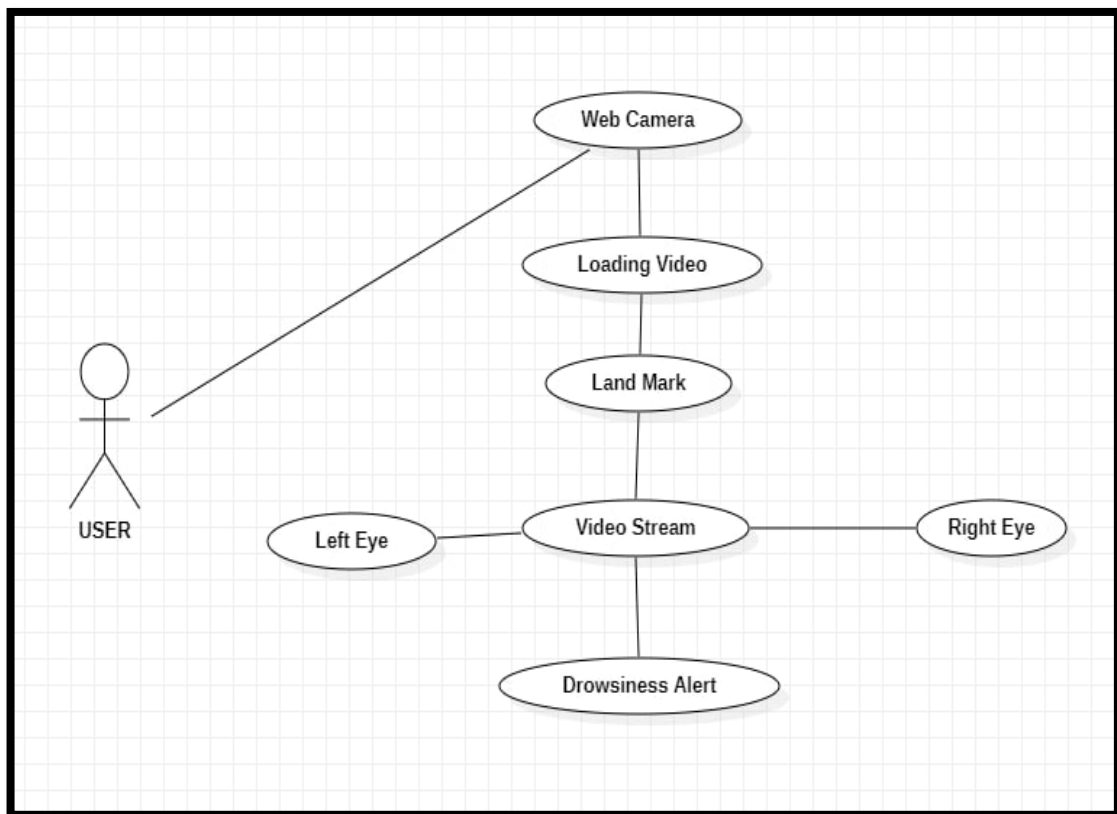


Figure 5.5 Use Case Diagram

5.6 CLASS DIAGRAM:

In software engineering, a class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, attributes, operations, and relationships among the classes. It explains which class contains information.

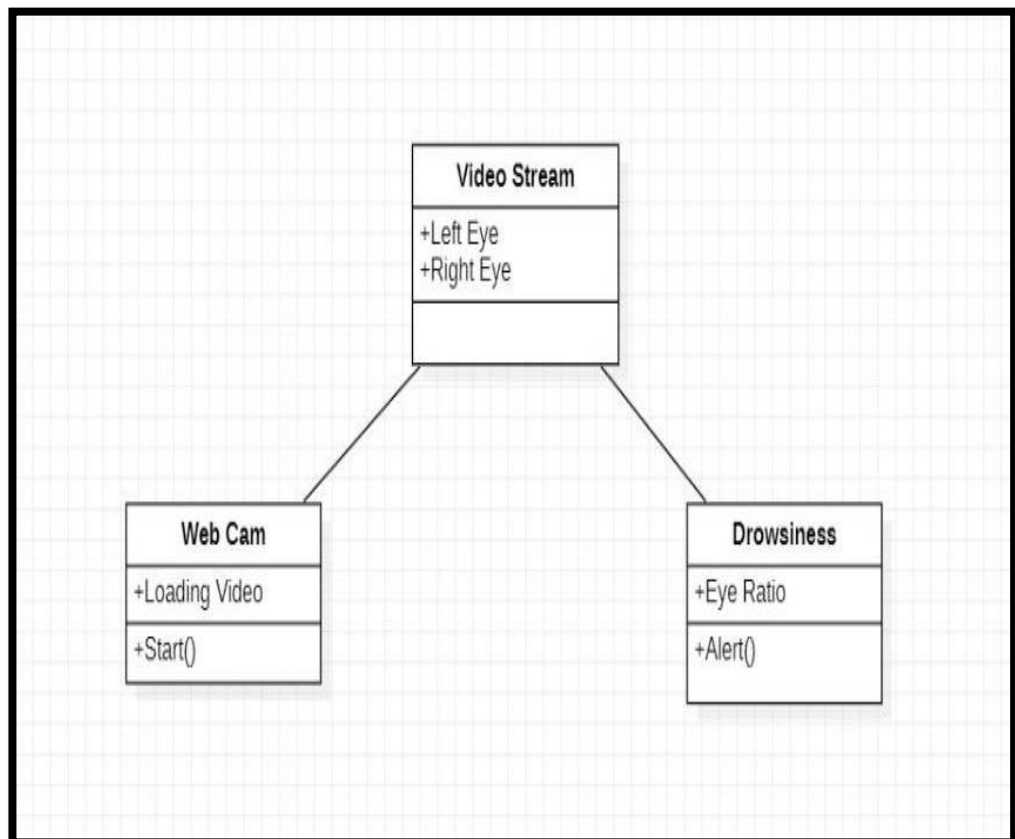


Figure 5.6 Class Diagram

5.7 ACTIVITY DIAGRAM

An activity diagram is basically a flowchart to represent the flow from one activity to another activity.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc. It captures the dynamic behavior of the system.

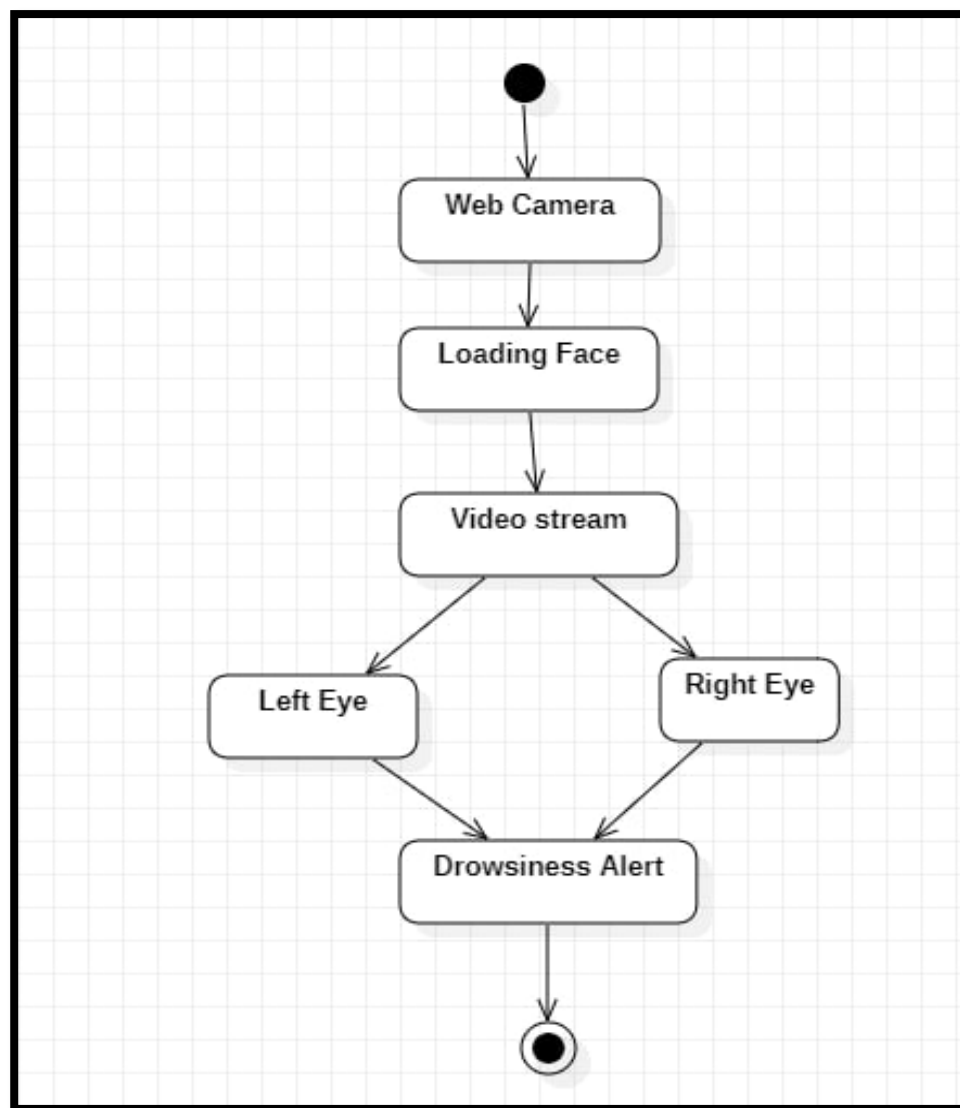


Figure 5.7 Activity Diagram.

5.8 COLLABORATION DIAGRAM

The collaboration Diagram depicts the relationships and interactions among software objects. They are used to understand the object architecture within a system rather than the flow of a message as in a sequence diagram. They are also known as “Communication Diagrams.” Collaboration Diagrams are used to explore the architecture of objects inside the system. The message flow between the objects can be represented using a collaboration diagram.

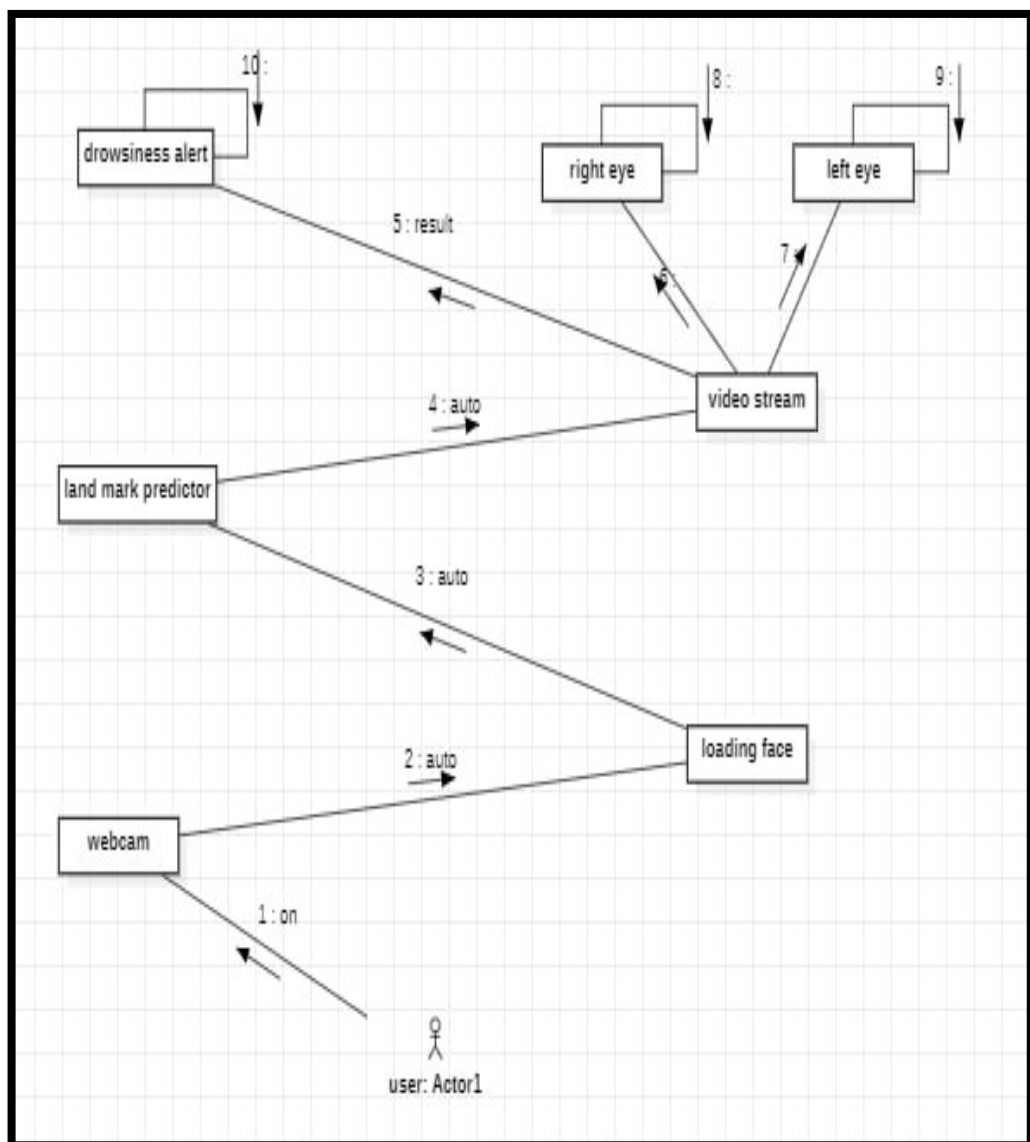


Figure 5.8 Collaboration Diagram

5.9 SEQUENCE DIAGRAM

A Sequence diagram in UML is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a message sequence chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

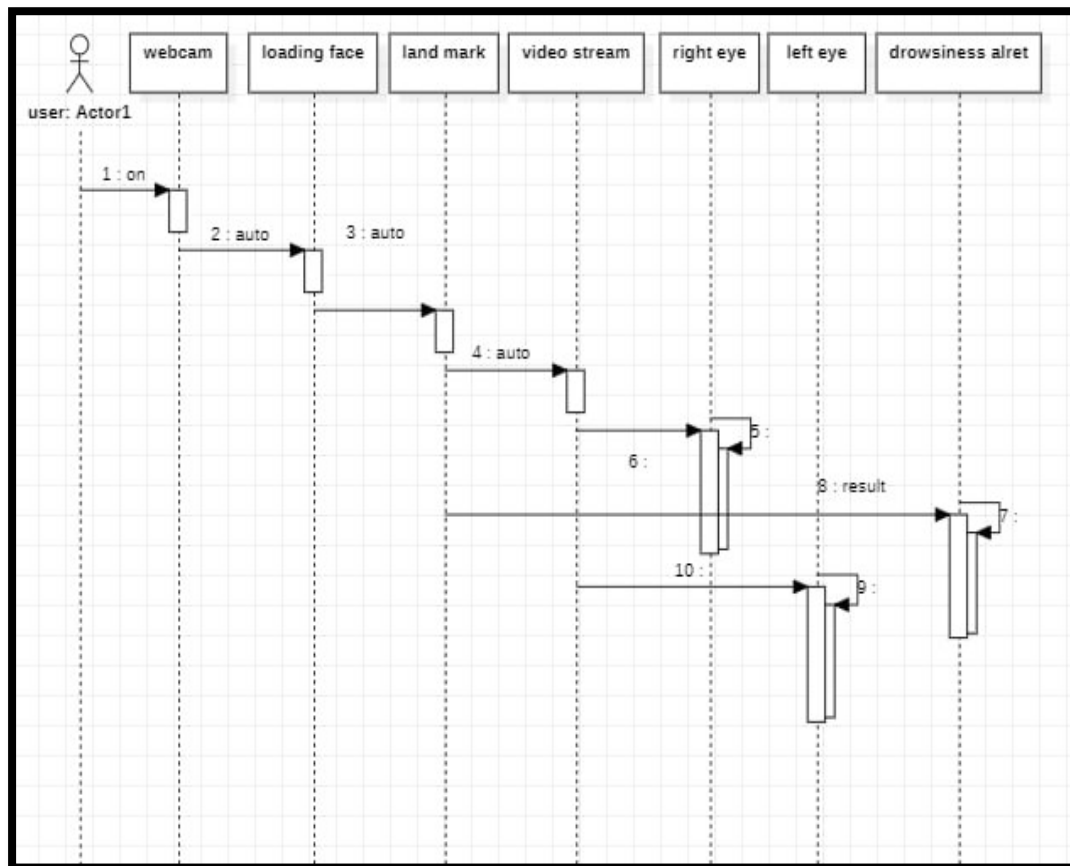


Figure 5.9 Sequence Diagram

Chapter-6

CODE IMPLEMENTATION

Once the required library is installed, import the required packages and method.

```
import dlib
import sys
import cv2
import time
import numpy as np
from scipy.spatial import distance as dist
from threading import Thread
import playsound
import queue
```

1. Importing Libraries:

dlib: A library that provides tools and algorithms for computer vision tasks, including face detection and facial landmark detection.

sys: A module providing access to some variables used or maintained by the interpreter and to functions that interact with the interpreter.

cv2: The OpenCV library, which is used for various computer vision tasks, including image and video processing.

time: A module providing various time-related functions.

numpy: A library for numerical computations in Python.

scipy.spatial.distance: A module from the Scipy library that provides distance computations between points or sets of points.

threading: A module for creating and managing threads in Python.

playsound: A library for playing sounds in Python.

queue: A module providing a queue data structure for multi-threaded programming.

```
FACE_DOWNSAMPLE_RATIO = 1.5
RESIZE_HEIGHT = 460

thresh = 0.27
modelPath = "models/shape_predictor_70_face_landmarks.dat"
sound_path = "alarm.wav"

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(modelPath)

leftEyeIndex = [36, 37, 38, 39, 40, 41]
rightEyeIndex = [42, 43, 44, 45, 46, 47]

blinkCount = 0
drowsy = 0
state = 0
blinkTime = 0.15 #150ms
drowsyTime = 1.5 #1200ms
ALARM_ON = False
GAMMA = 1.5
threadStatusQ = queue.Queue()

invGamma = 1.0/GAMMA
table = np.array([(i / 255.0) ** invGamma) * 255 for i in range(0, 256)]).astype("uint8")
```

This portion of the code sets up various parameters and variables used in the drowsiness detection system.

FACE_DOWNSAMPLE_RATIO = 1.5: This constant defines the ratio by which the captured face image is downscaled before further processing. A higher value will result in faster processing but lower accuracy.

RESIZE_HEIGHT = 460: This constant defines the desired height of the face image after downsampling. The width is adjusted automatically to maintain the aspect ratio.

thresh = 0.27: This variable sets a threshold value used for determining if the eyes are closed or open. It is used in calculating the eye-aspect ratio (EAR) later in the code.

modelPath = "models/shape_predictor_70_face_landmarks.dat": This variable specifies the file path to the pre-trained model for facial landmark detection. The model is used to locate specific points on the face, such as the eyes, which are essential for drowsiness detection.

sound_path = "alarm.wav": This variable holds the file path to the sound file that will be played when drowsiness is detected.

detector = dlib.get_frontal_face_detector(): This line initializes the face detector from the dlib library. The get_frontal_face_detector() function returns a face detection model that can identify faces in an image or video frame.

predictor = dlib.shape_predictor(modelPath): This line initializes the facial landmark predictor using the pre-trained model specified by modelPath. The shape_predictor is responsible for detecting the landmarks on the face, such as the eyes and other key points.

leftEyeIndex and rightEyeIndex: These variables contain lists of indices representing the landmarks corresponding to the left and right eyes, respectively. These indices are used later to calculate the eye aspect ratio (EAR).

blinkCount = 0: This variable keeps track of the number of blinks detected.

drowsy = 0: This variable tracks the drowsiness state. Its value is updated based on the blink count and time thresholds.

state = 0: This variable represents the current state of the drowsiness detection system. Its value can be used to indicate if the system is active or inactive.

blinkTime = 0.15: This variable defines the threshold time (in seconds) for considering an eye blink. If the duration between blinks is less than this value, it is considered a blink.

drowsyTime = 1.5: This variable sets the threshold time (in seconds) for considering a drowsy state. If the eyes are closed for a duration longer than this value, it indicates drowsiness.

ALARM_ON = False: This variable indicates the current state of the alarm sound. It is initially set to False, meaning the alarm is not active.

GAMMA = 1.5: This variable represents the gamma correction value for adjusting the brightness of the video frames.

threadStatusQ = queue.Queue(): This line creates a queue object used for communication and synchronization between different threads in the system.

invGamma = 1.0/GAMMA: This line calculates the inverse of the gamma value.

`table = np.array([((i / 255.0) ** invGamma) * 255 for i in range(0, 256)]).astype('uint8')`: This line creates a lookup table that maps pixel intensities based on the gamma correction value.

```
def gamma_correction(image):
    return cv2.LUT(image, table)

def histogram_equalization(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.equalizeHist(gray)

def soundAlert(path, threadStatusQ):
    while True:
        if not threadStatusQ.empty():
            FINISHED = threadStatusQ.get()
            if FINISHED:
                break
        playsound.playsound(path)

def eye_aspect_ratio(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
```

This part of the code contains several functions used in the drowsiness detection system. Let's go through each function:

gamma_correction(image): This function applies gamma correction to an image using the lookup table `table` defined earlier. It takes an image as input and returns the gamma-corrected image.

histogram_equalization(image): This function performs histogram equalization on an image to enhance its contrast. It converts the image to grayscale and then applies the `cv2.equalizeHist()` function to equalize the histogram. It takes an image as input and returns the histogram-equalized image.

soundAlert(path, threadStatusQ): This function plays a sound alert repeatedly until a flag in the threadStatusQ queue is set to True. It continuously checks the queue's status, and if the flag is set to True, indicating that the thread should finish, it breaks out of the loop. It uses the playsound. playsound() function to play the sound file specified by the path argument.

eye_aspect_ratio(eye): This function calculates the eye aspect ratio (EAR) based on the coordinates of the eye landmarks. It takes a list of eye landmarks as input and returns the calculated EAR. The EAR is computed by measuring the Euclidean distances between the landmarks and using a formula $(A + B) / (2.0 * C)$, where A and B are distances between specific landmarks, and C is the distance between another pair of landmarks.

These functions likely serve specific purposes in the drowsiness detection system:

gamma_correction() and histogram_equalization() may be used for image preprocessing and enhancement to improve the accuracy of the subsequent processing steps.

soundAlert() is responsible for playing the sound alert when drowsiness is detected.

eye_aspect_ratio() calculates the eye aspect ratio, which is a crucial metric used to determine the state of the eyes (open or closed) and detect drowsiness.

```
def checkEyeStatus(landmarks):
    mask = np.zeros(frame.shape[:2], dtype = np.float32)

    hullLeftEye = []
    for i in range(0, len(leftEyeIndex)):
        hullLeftEye.append((landmarks[leftEyeIndex[i]][0], landmarks[leftEyeIndex[i]][1]))

    cv2.fillConvexPoly(mask, np.int32(hullLeftEye), 255)

    hullRightEye = []
    for i in range(0, len(rightEyeIndex)):
        hullRightEye.append((landmarks[rightEyeIndex[i]][0], landmarks[rightEyeIndex[i]][1]))

    cv2.fillConvexPoly(mask, np.int32(hullRightEye), 255)
```

This is the **checkEyeStatus** function, which is responsible for determining the eye status (open or closed) based on the facial landmarks.

mask = np.zeros(frame.shape[:2], dtype = np.float32): Creating a binary mask with the same dimensions as the frame. This mask will be used to isolate the regions corresponding to the left and right eyes.

Looping over the indices of the landmarks corresponding to the left eye (leftEyeIndex) and appending the coordinates of each landmark to the hullLeftEye list.

cv2.fillConvexPoly(mask, np.int32(hullLeftEye), 255): Filling a convex polygon on the mask using the coordinates of the landmarks of the left eye. The polygon is filled with white (pixel intensity value of 255) in the mask.

cv2.fillConvexPoly(mask, np.int32(hullRightEye), 255): Filling another convex polygon on the mask using the coordinates of the landmarks of the right eye. This polygon is also filled with white (pixel intensity value of 255) in the mask.

At this point, we have filled both convex polygons on the mask, representing the left and right eye regions.

The purpose of creating these polygons and filling them with white on the mask is to isolate the eye regions from the rest of the face. By doing so, we can focus specifically on the eye regions for further analysis, such as calculating the eye aspect ratio or determining the eye status (open or closed).

```
#####
leftEAR = eye_aspect_ratio(hullLeftEye)
rightEAR = eye_aspect_ratio(hullRightEye)

ear = (leftEAR + rightEAR) / 2.0
#####

eyeStatus = 1      # 1 -> Open, 0 -> closed
if (ear < thresh):
    eyeStatus = 0

return eyeStatus
```

leftEAR = eye_aspect_ratio(hullLeftEye): This line calculates the aspect ratio of the left eye. The **eye_aspect_ratio()** function takes the coordinates of the landmarks defining the left eye (stored in **hullLeftEye**) and computes the aspect ratio based on those points.

rightEAR = eye_aspect_ratio(hullRightEye): Similar to the previous line, this calculates the aspect ratio of the right eye using the coordinates of the landmarks defining the right eye (stored in **hullRightEye**).

ear = (leftEAR + rightEAR) / 2.0: The average of the left and right eye aspect ratios is computed and stored in the variable **ear**. This provides an overall measure of eye openness.

eyeStatus = 1: A variable named **eyeStatus** is initialized with a value of 1, which represents open eyes.

if (ear < thresh): This line checks if the calculated **ear** value is less than a predefined threshold value called **thresh**. The threshold value is typically determined through experimentation and can vary based on the specific requirements of the drowsiness detection system.

eyeStatus = 0: If the **ear** value is below the threshold, it implies that the eyes are closed or partially closed. Therefore, the **eyeStatus** variable is updated to 0, indicating closed eyes.

return eyeStatus: The final step in the function is to return the **eyeStatus** value, which represents the detected state of the eyes (1 for open and 0 for closed).

```
def checkBlinkStatus(eyeStatus):
    global state, blinkCount, drowsy
    if(state >= 0 and state <= falseBlinkLimit):
        if(eyeStatus):
            state = 0

        else:
            state += 1

    elif(state >= falseBlinkLimit and state < drowsyLimit):
        if(eyeStatus):
            blinkCount += 1
            state = 0

        else:
            state += 1

    else:
        if(eyeStatus):
            state = 0
            drowsy = 1
            blinkCount += 1

        else:
            drowsy = 1
```

The checkBlinkStatus function takes the eyeStatus as input, which represents the current status of the eyes (whether they are open or closed).

global state, blinkCount, drowsy: Declares these variables as global so that their values can be modified within the function.

if(state >= 0 and state <= falseBlinkLimit): Checks if the state variable is within the range of false blink limit. The false blink limit is a threshold value that determines the number of consecutive frames with closed eyes before considering it as a blink. If the state is within this range, it means the previous frames have not exceeded the false blink limit.

if(eyeStatus): Checks if the current eyeStatus is True, indicating that the eyes are open.

state = 0: If the eyes are open, reset the state to 0, indicating the absence of consecutive closed-eye frames.

Else, if the eyes are closed, increment the state by 1.

elif(state >= falseBlinkLimit and state < drowsyLimit): Checks if the state is within the range of false blink limit to drowsy limit. The drowsy limit is a threshold value that determines the number of consecutive frames with closed eyes before considering it as

drowsiness. If the state is within this range, it means the previous frames have exceeded the false blink limit but have not reached the drowsy limit.

if(eyeStatus): Checks if the current eyeStatus is True, indicating that the eyes are open.

blinkCount += 1: If the eyes are open, it means a blink has occurred. Increment the blinkCount variable.

state = 0: Reset the state to 0, indicating the absence of consecutive closed-eye frames.

Else, if the eyes are closed, increment the state by 1.

else: If the state exceeds the drowsy limit, it means the previous frames have reached or exceeded the drowsy limit.

if(eyeStatus): Checks if the current eyeStatus is True, indicating that the eyes are open.

state = 0: Reset the state to 0, indicating the absence of consecutive closed-eye frames.

drowsy = 1: Set the drowsy variable to 1, indicating drowsiness.

blinkCount += 1: Increment the blinkCount variable.

Else, if the eyes are closed, set the drowsy variable to 1, indicating drowsiness.

The purpose of the checkBlinkStatus function is to keep track of the state of the eyes (open or closed) over consecutive frames and determine if a blink or drowsiness has occurred based on predefined thresholds (falseBlinkLimit and drowsyLimit). It updates the state, blinkCount, and drowsy variables accordingly.

```
def getLandmarks(im):
    imSmall = cv2.resize(im, None,
                          fx = 1.0/FACE_DOWNSAMPLE_RATIO,
                          fy = 1.0/FACE_DOWNSAMPLE_RATIO,
                          interpolation = cv2.INTER_LINEAR)

    rects = detector(imSmall, 0)
    if len(rects) == 0:
        return 0

    newRect = dlib.rectangle(int(rects[0].left() * FACE_DOWNSAMPLE_RATIO),
                              int(rects[0].top() * FACE_DOWNSAMPLE_RATIO),
                              int(rects[0].right() * FACE_DOWNSAMPLE_RATIO),
                              int(rects[0].bottom() * FACE_DOWNSAMPLE_RATIO))

    points = []
    [points.append((p.x, p.y)) for p in predictor(im, newRect).parts()]
    return points
```

getLandmarks that is used in a drowsiness detection system. This function extracts facial landmarks from an input image to identify specific points on the face.

imSmall = cv2.resize(im, None, fx=1.0/FACE_DOWNSAMPLE_RATIO, fy=1.0/FACE_DOWNSAMPLE_RATIO, interpolation=cv2.INTER_LINEAR): The input image im is resized to a smaller size using OpenCV's resize function. The resizing is performed to improve the speed and accuracy of facial landmark detection. The downsampling ratio used is defined by the FACE_DOWNSAMPLE_RATIO variable.

rects = detector(imSmall, 0): A face detector (not shown in the code snippet) is used to detect faces in the downscaled image imSmall. The detector function is applied to imSmall, and the resulting bounding box rectangles of the detected faces are stored in the rects variable.

if len(rects) == 0: return 0: If no faces are detected (i.e., len(rects) is 0), the function returns 0, indicating that no facial landmarks could be extracted.

newRect = dlib.rectangle(int(rects[0].left() * FACE_DOWNSAMPLE_RATIO), int(rects[0].top() * FACE_DOWNSAMPLE_RATIO), int(rects[0].right() * FACE_DOWNSAMPLE_RATIO), int(rects[0].bottom() * FACE_DOWNSAMPLE_RATIO)): The bounding box coordinates of the first detected face are extracted from rects. These coordinates are then scaled up by the FACE_DOWNSAMPLE_RATIO to match the original image size. A new dlib.rectangle object named newRect is created using the scaled coordinates.

points = []: An empty list named points is initialized to store the coordinates of the facial landmarks.

[points.append((p.x, p.y)) for p in predictor(im, newRect).parts()]: The facial landmarks are obtained using a facial landmark predictor (not shown in the code snippet). The predictor function takes the original image im and the scaled bounding box newRect to predict the facial landmarks. The parts() function returns a list of landmark points, and each point's (x, y) coordinates are appended to the points list.

return points: The function returns the points list, which contains the coordinates of the extracted facial landmarks.

```
capture = cv2.VideoCapture(0)

for i in range(10):
    ret, frame = capture.read()

totalTime = 0.0
validFrames = 0
dummyFrames = 100

print("Caliberation in Progress!")
while(validFrames < dummyFrames):
    validFrames += 1
    t = time.time()
    ret, frame = capture.read()
    height, width = frame.shape[:2]
    IMAGE_RESIZE = np.float32(height)/RESIZE_HEIGHT
    frame = cv2.resize(frame, None,
                       fx = 1/IMAGE_RESIZE,
                       fy = 1/IMAGE_RESIZE,
                       interpolation = cv2.INTER_LINEAR)

    # adjusted = gamma_correction(frame)
    adjusted = histogram_equalization(frame)

    landmarks = getLandmarks(adjusted)
    timeLandmarks = time.time() - t
```

The provided code snippet captures video frames from the webcam using OpenCV. Let's break down the code:

capture = cv2.VideoCapture(0): Initializes the video capture object, capture, to capture frames from the default webcam (index 0).

for i in range(10):: Captures and discards the first 10 frames from the webcam. This is often done to allow the camera to adjust to the lighting conditions before starting the actual processing.

totalTime = 0.0: Initializes the totalTime variable to track the total time taken for capturing frames.

validFrames = 0: Initializes the validFrames variable to count the number of valid frames captured.

dummyFrames = 100: Specifies the number of frames to be captured for calibration. In this case, it is set to 100.

print("Calibration in Progress!"): Displays a message indicating that the calibration process is in progress.

while(validFrames < dummyFrames):: Enters a loop that continues until the number of valid frames captured (validFrames) reaches the number of dummy frames (dummyFrames).

validFrames += 1: Increments the validFrames counter by 1.

t = time.time(): Records the current time before capturing a frame.

ret, frame = capture.read(): Reads a frame from the video capture object and stores it in the frame variable. The ret variable indicates whether the frame was successfully read or not.

height, width = frame.shape[:2]: Retrieves the height and width of the captured frame.

IMAGE_RESIZE = np.float32(height)/RESIZE_HEIGHT: Calculates the resize factor based on the ratio of the frame's height to the desired height (RESIZE_HEIGHT). This factor is used to resize the frame to a standard size for further processing.

frame = cv2.resize(frame, None, fx=1/IMAGE_RESIZE, fy=1/IMAGE_RESIZE, interpolation=cv2.INTER_LINEAR): Resizes the frame using the calculated IMAGE_RESIZE factor. The fx and fy parameters represent the scale factors for resizing in the horizontal and vertical directions, respectively. The interpolation parameter specifies the interpolation method to be used during resizing.

adjusted = histogram_equalization(frame): Applies histogram equalization to the resized frame. This is likely a function call to perform histogram equalization on the frame.

landmarks = getLandmarks(adjusted): Calls a function getLandmarks to extract facial landmarks from the adjusted frame. The adjusted frame is passed as an argument to the function.

timeLandmarks = time.time() - t: Calculates the time taken to extract the facial landmarks by subtracting the previously recorded time (t) from the current time.

The code snippet appears to perform calibration by capturing frames, resizing them, applying histogram equalization, and extracting facial landmarks. The exact purpose of the calibration

process and the subsequent usage of the captured frames and landmarks are not shown in the provided code snippet.

```
if landmarks == 0:
    validFrames -= 1
    cv2.putText(frame, "Unable to detect face, Please check proper lighting", (10, 30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
    cv2.putText(frame, "or decrease FACE_DOWNSAMPLE_RATIO", (10, 50), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
    cv2.imshow("Blink Detection Demo", frame)
    if cv2.waitKey(1) & 0xFF == 27:
        sys.exit()
else:
    totalTime += timeLandmarks
```

if landmarks == 0: Checks if the landmarks variable is equal to 0, indicating that no facial landmarks were detected in the current frame.

validFrames -= 1: Decrements the validFrames counter by 1 since the current frame is not considered valid.

cv2.putText(...): Displays a text message on the frame indicating that a face was not detected properly or suggesting to adjust the lighting conditions or decrease the FACE_DOWNSAMPLE_RATIO. The cv2.putText() function is used to draw the text on the frame.

cv2.imshow("Blink Detection Demo", frame): Displays the frame with the text message using the OpenCV imshow() function.

if cv2.waitKey(1) & 0xFF == 27: sys.exit(): Checks if the user presses the ESC key (27 in ASCII) while the frame is displayed. If so, the program terminates by calling sys.exit().

else:: If the landmarks variable is not equal to 0, it means that facial landmarks were successfully detected in the current frame.

totalTime += timeLandmarks: Adds the time taken to extract the facial landmarks (timeLandmarks) to the totalTime variable, which accumulates the total time taken for landmark extraction.

This code snippet handles the case when facial landmarks are not detected properly and provides feedback to the user. It subtracts the invalid frames from the validFrames count, displays a message on the frame, and allows the user to exit the program if desired.

```
print("Caliberation Complete!")

spf = totalTime/dummyFrames
print("Current SPF (seconds per frame) is {:.2f} ms".format(spf * 1000))

drowsyLimit = drowsyTime/spf
falseBlinkLimit = blinkTime/spf
print("drowsy limit: {}, false blink limit: {}".format(drowsyLimit, falseBlinkLimit))
```

print("Calibration Complete!"): This line simply prints the message "Calibration Complete!" to the console, indicating that the calibration process of the drowsiness detection system has finished.

spf = totalTime/dummyFrames: Here, totalTime represents the total time elapsed during the calibration process, and dummyFrames represents the number of frames captured during calibration. The line calculates the average time taken per frame, known as "seconds per frame" (SPF), by dividing the total time by the number of frames. The result is stored in the variable spf.

print("Current SPF (seconds per frame) is {:.2f} ms".format(spf * 1000)): This line prints the value of SPF in milliseconds (ms) by multiplying spf by 1000 (to convert it from seconds to milliseconds) and formatting it to two decimal places. The message is displayed in the console, providing information about the current performance of the system.

drowsyLimit = drowsyTime/spf: Here, drowsyTime represents the duration (in seconds) considered as a threshold for detecting drowsiness. The line calculates the number of frames that would correspond to the drowsyTime duration based on the SPF value. The result is stored in the variable drowsyLimit, representing the limit of frames before considering the user as drowsy.

falseBlinkLimit = blinkTime/spf: Similar to the previous line, blinkTime represents the duration (in seconds) considered as a threshold for detecting false blinks. The line calculates

the number of frames that would correspond to the blinkTime duration based on the SPF value. The result is stored in the variable falseBlinkLimit, representing the limit of frames before considering a blink as a false positive.

print('drowsy limit: {}, false blink limit: {}'.format(drowsyLimit, falseBlinkLimit)):

This line prints the values of drowsyLimit and falseBlinkLimit to the console. The message displayed shows the calculated limits for detecting drowsiness and false blinks, respectively.

```
if __name__ == "__main__":
    vid_writer = cv2.VideoWriter('output-low-light-2.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 15, (frame.shape[1],frame.shape[0]))
    while(1):
        try:
            t = time.time()
            ret, frame = capture.read()
            height, width = frame.shape[:2]
            IMAGE_RESIZE = np.float32(height)/RESIZE_HEIGHT
            frame = cv2.resize(frame, None,
                               fx = 1/IMAGE_RESIZE,
                               fy = 1/IMAGE_RESIZE,
                               interpolation = cv2.INTER_LINEAR)

            # adjusted = gamma_correction(frame)
            adjusted = histogram_equalization(frame)

            landmarks = getLandmarks(adjusted)
            if landmarks == 0:
                validFrames -= 1
                cv2.putText(frame, "Unable to detect face, Please check proper lighting", (10, 30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                cv2.putText(frame, "or decrease FACE_DOWNSAMPLE_RATIO", (10, 50), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                cv2.imshow("Blink Detection Demo", frame)
                if cv2.waitKey(1) & 0xFF == 27:
                    break
                continue
```

if name == "main": This line checks if the script is being run as the main module. It ensures that the code inside the following block is only executed when the script is run directly, rather than being imported as a module.

vid_writer = cv2.VideoWriter('output-low-light-2.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 15, (frame.shape[1],frame.shape[0])):

This line creates a VideoWriter object named vid_writer to write the output frames to a video file. It specifies the output file name as 'output-low-light-2.avi', the video codec as 'MJPG', the frame rate as 15 frames per second, and the frame dimensions based on the shape of the frame variable.

while(1): This initiates an infinite loop, which will continue until it is manually interrupted or a break statement is encountered.

try: The code inside the try block is executed, and any exceptions that occur are caught and handled.

t = time.time(): This line captures the current time using the time.time() function and stores it in the variable t.

ret, frame = capture.read(): This line reads the next frame from the video capture source (capture) and assigns it to the variables ret and frame. ret indicates whether the frame was successfully read, and frame contains the actual frame data.

height, width = frame.shape[:2]: The height and width of the frame are extracted using the shape attribute of the frame object.

IMAGE_RESIZE = np.float32(height)/RESIZE_HEIGHT: The variable IMAGE_RESIZE is calculated by dividing the height of the frame by RESIZE_HEIGHT. This is used to resize the frame to a specific height.

frame = cv2.resize(frame, None, fx=1/IMAGE_RESIZE, fy=1/IMAGE_RESIZE, interpolation=cv2.INTER_LINEAR): The frame is resized using OpenCV's resize function. The resizing is performed to maintain a consistent size for further processing. The frame is scaled down by 1/IMAGE_RESIZE in both the horizontal and vertical directions.

adjusted = histogram_equalization(frame): The histogram_equalization function (not shown in the code snippet) is applied to the resized frame. This function performs histogram equalization to enhance the contrast of the frame.

landmarks = getLandmarks(adjusted): The getLandmarks function (not shown in the code snippet) is called with the adjusted frame as input to extract facial landmarks from the frame. The resulting landmarks are stored in the landmarks variable.

if landmarks == 0: This line checks if the landmarks variable is equal to 0, indicating that no facial landmarks could be detected.

validFrames -= 1: If no facial landmarks are detected, the validFrames variable (not shown in the code snippet) is decremented. This variable likely keeps track of the number of valid frames processed.

The following three cv2.putText lines display warning messages on the frame, informing the user about the failure to detect a face. The messages are shown using OpenCV's putText function, specifying the text, position, font, color, thickness, and line type.

cv2.imshow("Blink Detection Demo", frame): The current frame, with the warning messages and any modifications, is displayed in a window titled "Blink Detection Demo" using `

```
eyeStatus = checkEyeStatus(landmarks)
checkBlinkStatus(eyeStatus)

for i in range(0, len(leftEyeIndex)):
    cv2.circle(frame, (landmarks[leftEyeIndex[i]][0], landmarks[leftEyeIndex[i]][1]), 1, (0, 0, 255), -1, lineType=cv2.LINE_AA)

for i in range(0, len(rightEyeIndex)):
    cv2.circle(frame, (landmarks[rightEyeIndex[i]][0], landmarks[rightEyeIndex[i]][1]), 1, (0, 0, 255), -1, lineType=cv2.LINE_AA)

if drowsy:
    cv2.putText(frame, "!!! DROWSINESS ALERT !!!", (70, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
    if not ALARM_ON:
        ALARM_ON = True
        threadStatusQ.put(not ALARM_ON)
        thread = Thread(target=soundAlert, args=(sound_path, threadStatusQ,))
        thread.setDaemon(True)
        thread.start()
else:
    cv2.putText(frame, "Blinks : {}".format(blinkCount), (460, 80), cv2.FONT_HERSHEY_COMPLEX, 0.8, (0,0,255), 2, cv2.LINE_AA)
    # (0, 400)
    ALARM_ON = False
```

checkEyeStatus() and **checkBlinkStatus()** functions are called to determine the eye status and update the blink status based on the detected landmarks. Let's break down the code further:

eyeStatus = checkEyeStatus(landmarks): The checkEyeStatus() function is called with the landmarks variable as an argument. It returns the eye status, which is then assigned to the eyeStatus variable.

checkBlinkStatus(eyeStatus): The checkBlinkStatus() function is called with the eyeStatus variable as an argument. This function updates the blinking status (state, blinkCount, drowsy) based on the current eye status.

Looping over the left and right eye indices, the code draws circles on the frame to visualize the landmarks of the left and right eyes.

If the drowsy flag is set to true, indicating drowsiness, the code displays a warning message on the frame and initiates a sound alert if the ALARM_ON flag is not already set. The sound alert is handled in a separate thread.

If the drowsy flag is false, indicating no drowsiness, the code displays the number of blinks (blinkCount) on the frame.

The ALARM_ON flag is set to false to indicate that no alarm is currently active.

This code snippet integrates the eye and blink status into the frame visualization. It draws circles for the eye landmarks, displays drowsiness alerts, and shows the blink count on the frame.

```
cv2.imshow("Blink Detection Demo", frame)
vid_writer.write(frame)

k = cv2.waitKey(1)
if k == ord('r'):
    state = 0
    drowsy = 0
    ALARM_ON = False
    threadStatusQ.put(not ALARM_ON)

elif k == 27:
    break

# print("Time taken", time.time() - t)

except Exception as e:
    print(e)

capture.release()
vid_writer.release()
cv2.destroyAllWindows()
```

cv2.imshow("Blink Detection Demo", frame): The frame with the visualizations is displayed in a window with the title "Blink Detection Demo."

vid_writer.write(frame): The frame is written to a video file using the vid_writer object. This is useful if you want to save the processed frames with visualizations as a video.

k = cv2.waitKey(1): The code waits for a keyboard event for 1 millisecond and captures the key pressed.

If the key "r" is pressed (`k == ord('r')`), the code resets the state, drowsy, and ALARM_ON flags to their initial values. It also updates the thread status queue to reflect the new ALARM_ON value.

If the Esc key (key code 27) is pressed (`k == 27`), the code breaks out of the loop and ends the program.

Any exceptions that occur during the execution of the code are caught and printed.

Finally, after the loop ends, the following actions are performed:

capture.release(): The video capture object is released, closing the connection to the camera or video file.

vid_writer.release(): The video writer object is released, finalizing the video file writing process.

cv2.destroyAllWindows(): All open windows created by OpenCV are closed.

These actions ensure the proper release of resources and clean program termination.

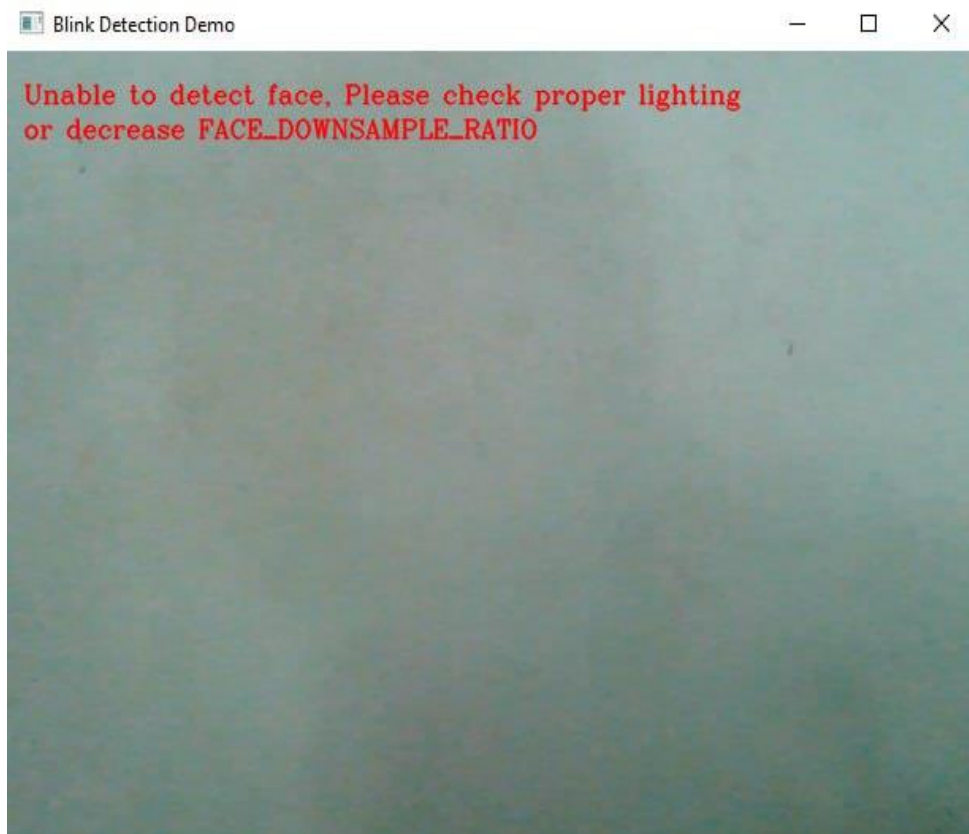
Chapter-7

RESULT

- ❖ After running the code, the application will connect with the webcam and initiates a streaming interface, enabling real-time video input. Once the webcam is connected and the streaming begins, the system continuously captures frames or images from the video feed. Each frame represents a single instance of data that will be processed by the SVM classifier .



- ❖ After the webcam is connected and the streaming begins, there may be a chance to get, the message **"Unable to detect a face, please check proper lighting or decreasing FACE_DOWN SAMPLE_RATIO"** which suggests that the drowsiness detection system is encountering difficulties in detecting and locating the person's face in the webcam feed. This issue can arise due to two potential reasons: improper lighting conditions or a need to adjust the FACE_DOWN SAMPLE_RATIO parameter.



- ❖ In a drowsiness detection system, the application continuously monitors all frames from the webcam feed to determine if the person's eyes are open or closed. It counts the number of eye blinks and calculates the blink ratio to assess whether the blink frequency is within a normal range. An increased blink time or a lower blink ratio may indicate drowsiness or fatigue, prompting the system to issue alerts or warnings. This monitoring and analysis of eye behavior help identify signs of drowsiness and enable timely intervention to prevent accidents.



- ❖ In the drowsiness detection system, if the application detects that the person's eyes are closed for an extended period, it triggers a Drowsiness Alert message. This alert serves as a warning that the person may be drowsy or falling asleep. The timely delivery of the alert helps prevent potential accidents by prompting the person to take corrective actions or seek rest.



In the drowsiness detection system, once drowsiness is detected, the application triggers a warning alarm to alert the driver. The warning alarm serves as an auditory signal that notifies the driver about their drowsy state, prompting them to take immediate action to avoid potential accidents. The alarm acts as an attention-grabbing stimulus to jolt the driver out of their drowsy state and regain focus on the road. By providing this audible warning, the system aims to prevent accidents by alerting the driver and promoting safe driving practices.

Chapter – 8

Conclusion and Future Scope

The implementation of a Drivers Drowsiness Detection System using the SVM classifier offers significant benefits in enhancing road safety and preventing accidents caused by drowsy driving. The SVM classifier, a robust and widely-used machine learning algorithm, through the analysis of features such as eye closure, and blink duration, the SVM classifier can effectively differentiate between the alert and drowsy states of the driver. This information can be utilized to generate timely alarm warnings to the driver, thereby mitigating the risk of accidents caused by drowsiness.

Additionally, this technology can be particularly beneficial for professional drivers, such as truckers or long-haul drivers, who are more susceptible to drowsy driving due to extended hours on the road. The Drivers Drowsiness Detection System using the SVM classifier is a promising solution to fight against drowsy driving and make the roads safer. With continued advancements and deployment in vehicles, this technology has the potential to play a crucial role in preventing accidents and saving lives on the road.

CHAPTER-9

BIBLIOGRAPHY

- Tefft, Brian C. "Acute sleep deprivation and risk of motor vehicle crash involvement." (2016).
- [1] Parmar, Neeta. "Drowsy driver detection system." Engineering Design Project Thesis, Ryerson University (2002).
- [2] Lu, Yufeng, and Zengcai Wang. "Detecting driver yawning in successive images." Bioinformatics and Biomedical Engineering, 2007. ICBBE 2007. The 1st International Conference on. IEEE, 2007.
- [3] Horng, Wen-Bing, and Chih-Yuan Chen. "A real-time driver fatigue detection system based on eye tracking and dynamic template matching." 11.1 (2008): 65-72.
- [4] Devi, Mandalapu Sarada, and Preeti R. Bajaj. "Fuzzy based driver fatigue detection." Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on. IEEE, 2010. [5] Coetzer, Reinier C., and Gerhard P. Hancke. "Eye detection for a real-time vehicle driver fatigue monitoring system." Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE, 2011.
- Burcu Kır Savaş and Yaşar Becerikli.
- [6] Anumas, Sasiporn, and Soo-chan Kim. "Driver fatigue monitoring system using video face images & physiological information." Biomedical Engineering International Conference (BMEiCON), 2011. IEEE, 2012.
- [7] Zhang, Weiwei, et al. "Driver yawning detection based on deep convolutional neural learning and robust nose tracking." Neural Networks (IJCNN), 2015 International Joint Conference on. IEEE, 2015.
- [8] Zhang, Fang, et al. "Driver fatigue detection based on eye state recognition." Machine Vision and Information Technology (CMVIT), International Conference on. IEEE, 2017.
- [9] Mandal, Bappaditya, et al. "Towards detection of bus driver fatigue based on robust visual analysis of eye state." IEEE Transactions on Intelligent Transportation Systems 18.3 (2017): 545-557.
- [10] Kır Savaş, B., and Becerikli, Y. "Development of Driver Fatigue Detection System By Using Video Images", Akıllı Sistemlerde Yenilikler Ve Uygulamaları (ASYU 2017), 2017.

- [11] Yuen, Kevan, and Mohan M. Trivedi. "An Occluded Stacked Hourglass Approach to Facial Landmark Localization and Occlusion Estimation." IEEE Transactions on Intelligent Vehicles 2.4 (2017): 321-331.
- <https://www.researchgate.net/publication/334168196> **Real Time Driver Fatigue Detection Based on SVM Algorithm**
- <https://www.researchgate.net/publication/235655049> **Driver fatigue monitoring system using Support Vector Machines**
- Tefft, Brain C." Acute sleep deprivation and risk of motor vehicle crash involvement." (2016).
- Parmar, Neeta. "Drowsy driver detection system." Engineering Design Project Thesis, Ryerson University (2002).
- Lu, Yufeng, and Zengcai Wang. "Detecting driver yawning in successive images." Bioinformatics and Biomedical Engineering, 2007.ICBBE 2007.The 1stInternational Conference on. IEEE,2007.
- Horng, Wen-Being, and Chih-Yuan Chen. "A real-time driver fatigue detection system based on eye tracking and dynamic template matching."11.1(2008):65-72.
- Coetzer,Renier C., and Gerhard P. Hancke. "Eye detection for a real-time vehicle driver fatigue monitoring system." Intelligent Vehicles Symposium (IV),2011 IEEE.IEEE,2011.
- Zhang, Fang, et al. "Driver fatigue detection based on eye state recognition ." Machine Vision and Information Technology (CM VIT), International Conference on. IEEE,2017.