*A project report on*

**DRIVER DROWSINESS DETECTION**

*Submitted in partial fulfilment for the award of the degree of*

**INTEGRATED M-TECH**

*By*

**M SARANYA (19MIS7008)**

**R NAGA ROHITH (19MIS7109)**



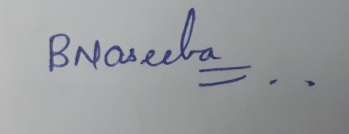
**AMARAVATI**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCOPE)**

May,2023

**CERTIFICATE**

This is to certify that the Capstone Project work titled “**DRIVER DROWSINESS DETECTION**” that is being submitted by **RAMANADHAM NAGA ROHITH (19MIS7109)** is in partial fulfillment of the requirements for the award of **Master of Technology(Integrated 5 Year) Software Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.



Dr. Beebi Naseeba

Guide Name & Signature

**The thesis is satisfactory**

**Approved by**

**PROGRAM CHAIR DEAN**

M. Tech. SE School Of Computer Science and Engineering

**ABSTRACT**

Effective sleepiness detection technologies are urgently needed to improve road safety due to the rising frequency of traffic incidents linked to driver fatigue. An overview of the methods and difficulties involved in driver sleepiness detection are given in this abstract.

Driver drowsiness has been detected using a variety of methods, including physiological measures and behavioral analyses. Monitoring variables like heart rate, electroencephalography (EEG) signals, eye movements, and skin conductance are all examples of physiological measurements. These readings can offer insightful information about the physiological state of the driver and aid in the early detection of tiredness. Behavioral analysis techniques, on the other hand, use computer vision algorithms to examine facial expressions, eye closure, head movements, and other visual clues to analyze indicators of tiredness.

Driver drowsiness detection systems frequently use machine learning algorithms to categorize and forecast sleepiness levels based on the gathered data. These algorithms are able to recognize patterns and features in physiological or behavioral data and can notify the driver of their tiredness in real time or activate the proper intervention systems to stop accidents.

The successful application of driver sleepiness detection systems, however, faces a number of obstacles. Significant issues include the heterogeneity of sleepiness patterns among people, the impact of outside variables like medication or road conditions, and the requirement for reliable and precise detection algorithms. In addition, it is crucial to focus on assuring user approval and reducing false alarms without sacrificing the functionality of the system.

The purpose of this project is to provide an overview of present approaches and technologies for detecting driver drowsiness, highlighting their strengths and weaknesses. By understanding the current state of research, practitioners and researchers can identify areas for improvement and pursue novel approaches to improve the accuracy, reliability, and user-friendliness of driver drowsiness detection systems, thereby contributing to a reduction in road accidents caused by driver fatigue.

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude to Dr. Beebi Naseeba, Associate Professor Grade - 1, VIT-AP, for her constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning (ML).

I would like to express my gratitude to Dr. G. Viswanathan, Dr. Sekar Viswanathan, Sankar Viswanathan, G. V. Selvam, Kadhambari S. Viswanathan, Dr. S. V. Kota Reddy, and Dr. Sudha S V, School of Computer Science and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Reeja S R, Program Chair Master of Technology in Integrated Software Engineering (MTSE), Associate Professor Grade-1, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 18-05-2023  **Naga Rohith Ramanadham**

**CONTENTS**

**Serial Title Page. No**

1 SYSTEM REQUIREMENTS SPECIFICATION 7

1.1 Introduction 7

1.1.1 Purpose 7

1.1.2 Problem Statement 7

1.2 Overview 8

1.2.1 Existing System 9

1.2.2 Disadvantages 9

1.2.4 Working Methodology 10

1.3 Objectives 12

1.6 Scope of the project 13

2 LITERATURE SURVEY 14

3 LIBRARIES 15

3.1 OpenCV 15

3.2 Imutils 16

3.3 Dlib 17

3.4 Scipy.spatial 18

3.5 NumPy 20

3.6 Pandas 21

3.7 Datetime 22

3.8 Winsound 23

3.9 OS 24

4 CODE 25

4.1 Code for Eye Shape Detection 25

4.2 Code for Mouth Shape Detection 26

5 FINAL CODE 31

5.1 Code 31

5.2 Output 38

6 CONCLUSION AND FUTURE WORK 40

6.1 Conclusion 40

6.2 Future Work 40

7 REFERENCES 41

**LIST OF FIGURES**

**Figure Title Page. No**

1.2.3 Block Diagram 10

1.2.5 Sequence Diagram 12

3.1 Sample Implementation of OpenCV 16

3.2 Sample Implementation of Imutils 17

3.3 Sample Implementation of Dlib 18

3.4 Sample Implementation of Scipy.spatial 19

3.5 Sample Implementation of NumPy 21

3.6 Sample Implementation of Pandas 22

3.7 Sample Implementation of Datetime 23

3.8 Sample Implementation of Winsound 23

3.9 Sample Implementation of OS 24

5.2 Code Output 38

**Chapter 1**

**SYSTEM REQUIREMENT SPECIFICATION**

* 1. **Introduction**

**1.1.1 PURPOSE**

The goal of the driver drowsiness detection project is to create and implement an effective system that can identify indicators of tiredness in drivers. The primary purpose is to improve road safety by issuing early warnings or intervening to avoid accidents caused by driver tiredness.

The project's goal is to address the major issue of drowsy driving, which endangers both the driver and other road users. The technology can notify the driver if it detects early signs of drowsiness, allowing them to take appropriate measures to battle exhaustion, such as taking a break, drinking coffee, or engaging in other stimulating activities.

Finally, the project aims to improve road safety by reducing the number of incidents caused by drowsy driving. The project's goal is to save lives, reduce injuries, and encourage safe driving practices by establishing an efficient and robust driver sleepiness detection system.

**1.1.2 PROBLEM STATEMENT**

**The problem of:**

1. This project contributes to the reduction of sleepy driving or driver weariness.
2. The project also addresses the issue of inadequate driver sleepiness awareness and prompt action. Many drivers may be unaware of their own fatigue levels or misjudge the risks of sleepy driving.
3. The initiative contributes to the reduction of sleepy driving by offering a dependable, real-time detection system that assists drivers in recognizing and addressing their sleepiness.

**Benefits:**

1. Improved Road Safety: By detecting driver tiredness accurately, the initiative improves road safety by lowering the dangers associated with drowsy driving. Timely alarms and interventions aid in the prevention of accidents caused by driver sleepiness, potentially saving lives and avoiding injuries.
2. Accident Prevention: Drowsy driving can cause significant accidents, typically resulting in fatalities or serious injuries. The project aids in the prevention of such accidents by providing drivers with early warnings, allowing them to take essential actions, such as sleeping or taking a break, before their tiredness becomes a hazard on the road.
3. Personal Safety: The project prioritizes individual driver safety and well-being. The device alerts drivers to their tiredness, allowing them to make informed judgements about their own safety, potentially avoiding accidents and ensuring their own physical well-being.
   1. **OVERVIEW**

The goal of the driver drowsiness detection project is to create and implement an efficient system for detecting indicators of tiredness in drivers and improving road safety. Algorithms and technology for reliably diagnosing and warning drivers about their fatigue levels are part of the project.

The study begins with a thorough examination of existing methodologies and technology for detecting driver drowsiness. This overview assists in understanding the advantages, disadvantages, and issues involved with various detection methods, including behavioral analysis.

Based on the review findings, the project moves forward with the design and implementation of a driver drowsiness detection system. This system combines several libraries, including Scipy. spatial, Imutils, Dlib, Cv2, Tkinter, Datetime, OS, Winsound, Numpy, and Pandas, to collect real-time data on the driver's performance.

To analyze the acquired data and effectively diagnose the driver's sleepiness status, machine learning algorithms, and pattern recognition techniques are used. These algorithms are trained on big datasets to discover drowsiness-related patterns and features, allowing

them to deliver credible alarms and warnings.

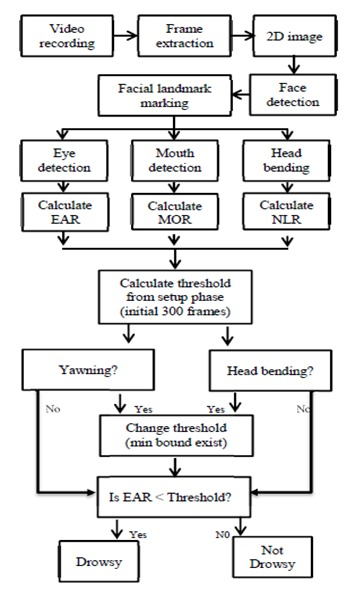
**1.2.1 EXISTING SYSTEM**

In the driver drowsiness detection project, the existing system combines sensor technologies, computer vision, machine learning, and human-machine interface components. It monitors facial expressions, eye movements, and head position with cameras and eye-tracking sensors, while physiological sensors assess heart rate and brain activity. Computer vision algorithms analyze face characteristics and eye states to obtain sleepiness indications. Based on the collected attributes, machine learning algorithms classify sleepiness levels, prompting warnings and interventions. The system incorporates a user interface that provides the driver with visual alerts, audio notifications, and haptic feedback. Data logging and analysis allow for system performance evaluation and continual development. Overall, the current system includes modern technology to monitor and identify drowsiness while also giving real-time notifications to improve road safety.

**1.2.2 DISADVANTAGES**

1. False Positives: The system may issue false positive alarms, detecting tiredness when the driver is awake. This can result in unneeded disruptions and distractions for the driver, potentially leading to frustration or a loss of trust in the system.
2. False Negatives: On the other hand, in some situations, the system may fail to detect drowsiness, resulting in missed opportunities for intervention. This is especially risky if the driver is feeling significant drowsiness or exhaustion, which increases the likelihood of an accident.
3. Individual Variations: Drowsiness patterns and responses differ between people, making it difficult to establish a one-size-fits-all approach. Age, health problems, and medication can all influence sleepiness levels, therefore the system must account for these individual variables in order to maintain accuracy and reliability.

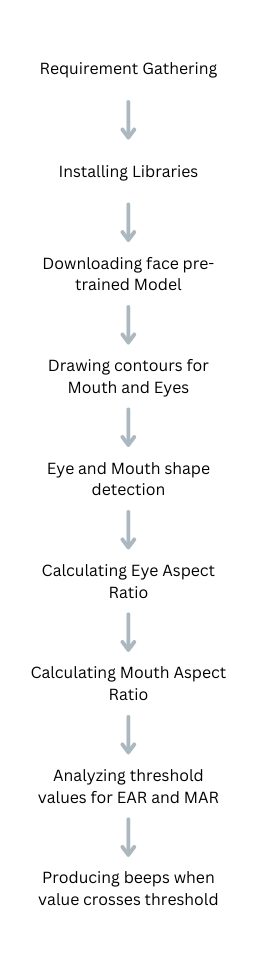
**1.2.3 BLOCK DIAGRAM**



**1.2.4 WORKING METHODOLOGY**

1. Requirement Analysis: Understanding the project objectives, target audience, and specific needs, such as the desired accuracy level, response time, and integration limitations, are all part of the requirement analysis.
2. Data Collection: Compile a diversified array of face data that can be utilized to recognize the driver's face.
3. Feature Extraction: Identify and extract relevant characteristics from preprocessed data. In computer vision, for example, features may include eye closure duration, blink patterns, or facial expressions. We must employ the Eyes and Mouth shape detection algorithms.
4. Algorithm Development: We used the face identification model's pre-trained SVM classifier dataset as the superset for the Eye and Mouth shape detection.
5. Algorithm Development: We used the face identification model's pre-trained SVM classifier dataset as the superset for the Eye and Mouth shape detection.
6. Cross-validation: Certain approaches can be used to ensure the model's robustness and generalization.
7. Feature Extraction: Identify and extract relevant characteristics from preprocessed data. In computer vision, for example, features may include eye closure duration, blink patterns, or facial expressions. Heart rate variability and brainwave patterns are examples of physiological characteristics.
8. Implementation: Integrate the learned model into a real-time system. Create software to acquire live data from cameras, eye-tracking sensors, and physiological sensors. In real-time, feed the data into the model for sleepiness categorization.
9. Warning and Intervention: Trigger relevant alerts and interventions to alert the driver based on the sleepiness categorization output. This includes auditory and visual indicators that advise the driver to take corrective action or relax.
10. System Evaluation and Optimisation: Extensive testing of the system in various driving scenarios and conditions is required. Gather user feedback and iterate on the system to improve accuracy, responsiveness, and user experience. Based on the evaluation results, optimize the algorithms and settings.

**1.2.5 SEQUENCE DIAGRAM**

****

**1.3 OBJECTIVES**

Increase Road Safety: The major goal is to increase road safety by building a system that can identify driver drowsiness effectively. The initiative attempts to lower the risk of accidents caused by drowsy driving by notifying drowsy drivers in real-time.

Preventing Accidents: The initiative intends to prevent accidents and the loss of life or property damage by giving drowsy drivers with early warnings and interventions. The device can advise drivers to take necessary rest breaks or other preventive actions by identifying tiredness early on.

Increase Driver Awareness: The project's goal is to raise driver awareness of their own sleepiness levels and the consequences that come with it. Drivers become more aware of their situation as a result of feedback and alerts, perhaps leading to better self-management of fatigue and sleepiness while driving.

Assist Professional Drivers: The initiative may specifically target professional drivers, such as truck drivers, bus drivers, or taxi drivers, who are more susceptible to long driving hours and fatigue-related issues. The goal is to assist these drivers in being safe and alert while on the road.

* 1. **SCOPE OF THE PROJECT**

Drowsiness Detection: The project's goal is to provide an accurate and dependable system for detecting driver drowsiness. This comprises the use of various sensor technologies, such as webcams, eye-tracking sensors, and physiological sensors, to collect and analyze pertinent sleepiness data.

Real-time Monitoring: The project intends to build a real-time monitoring system that continuously examines the driver's state of drowsiness. When drowsiness is detected, the system should deliver rapid notifications and interventions to prompt the driver to take proper steps.

System Integration: The project covers the integration of the sleepiness detection system into automobiles or other relevant platforms. This necessitates either a seamless connection with existing vehicle systems or the creation of standalone systems that drivers may readily install and use.

Scalability and adaptability: The project's scalability and adaptability to multiple vehicle

types, driving circumstances, and user preferences should be considered. The goal is to create a system that can be easily altered and customized for different cars and driver characteristics.

**Chapter 2**

**LITERATURE SURVEY**

Road accidents are a significant issue for everyone. Every day, a considerable number of valuable lives are lost as a result of car accidents due to driver drowsiness. It is the most critical field that requires significant exploration, given the high rates of fatalities related to road crashes. Driver error and delays in recognizing the driver’s drowsiness are the two most common causes. In order to prevent such accidents, a technology is required that detects the driver’s drowsiness and alerts the driver based on his unattentive nature. A device that can continuously monitor the driver and alerts him whenever he feels drowsy is needed in the present situations to reduce human errors. Numerous scholars have suggested a variety of automatic drowsiness warning systems in the research literature. Among them are IOT technology-based project which contains sensors such as camera, EEG (Electroencephalography) sensor, ECG (Electrocardiography) sensor and accelerometer and Software technology based which can be implemented using machine learning models and inbuilt libraries such as OpenCV. Every vehicle should have a driver drowsiness detection and necessary components installed in their vehicles. This research work provides a critical review of numerous emerging methodologies for forecasting and avoiding road accidents by emphasizing their benefits, shortcomings, and problems that must be resolved in order to ensure traffic safety and save lives.

Road accidents are one of the major causes of mortality around the world and over 1,300,000 people are killed annually in the road accidents. Out of these 10% of the accidents are due to driver un attentiveness.

The main objective of this paper is to design a system in order to make sure the driver will be notified whenever he feels drowsy while he is driving. This will reduce the number of accidents that are being happened due to drowsiness. The proposed system will make the driver be attentive by making strong beeps whenever he is feeling drowsy or when he is not attentive. So, it will reduce the unnecessary accidents which are not expected.

In general, a Driver Drowsiness Detection consists of a camera which is used to continuously monitor the driver, a machine learning model to detect the driver's face, and an algorithm to analyze the drivers Eye Aspect Ratio and Mouth Aspect Ratio.

Existing system analyses the driver drowsiness using completely machine learning algorithm which can give less accuracy whenever there is frequent change of the driver.

**Chapter 3**

**LIBRARIES**

**3.1 OpenCV**

OpenCV is an acronym for "Open-Source Computer Vision Library," and it offers a variety of functions and algorithms for image and video processing, object recognition and tracking, feature extraction, and other tasks.

OpenCV is written in C++ and supports a variety of computer languages, including Python. The Python interface, sometimes known as "cv2," enables developers to use Python to access OpenCV methods and execute computer vision tasks.

To install Open CV in windows we need to execute the below command in CMD

**pip install OpenCV-python**

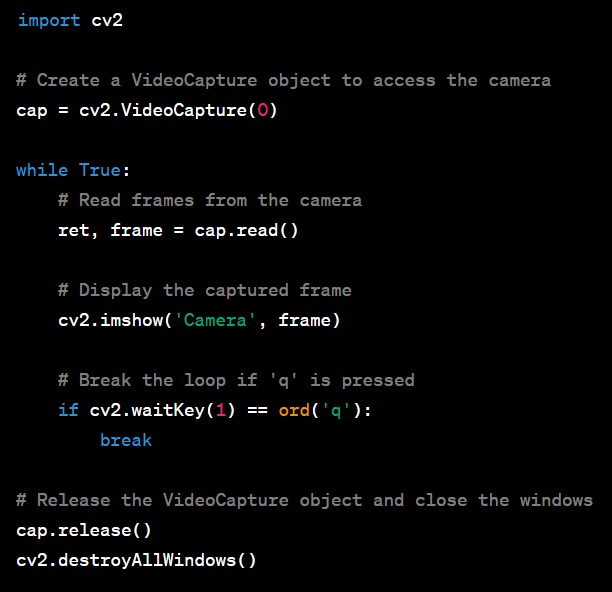
OpenCV (Open Source Computer Vision Library) is a sophisticated tool for computer vision applications that provides a wide range of image and video processing functionalities. We used OpenCV for:

Processing of Images and Videos: OpenCV includes functions for reading, manipulating, and processing images and videos. Resizing, cropping, rotating, filtering, blending, and color space conversions are all possible. OpenCV is to perform advanced image processing techniques including edge detection, image segmentation, and morphological operations.

Object Detection and Tracking: For object detection and tracking, OpenCV provides pre-trained models and algorithms. These models can be used to detect and track objects of interest in pictures or video streams. This is beneficial for applications such as facial recognition, object recognition, and motion tracking.

Feature Extraction: OpenCV includes methods for extracting and matching features from images, such as corners, edges, and key points. These capabilities can be applied to applications such as image registration, image stitching, and object detection. OpenCV also contains feature-matching algorithms for finding corresponding points or objects in diverse pictures.

**Sample implementation of OpenCV**



* 1. **Imutils**

The imutils library is a Python package that contains a set of functions for working with pictures in OpenCV. It facilitates the use of OpenCV functions and provides easier methods for performing typical image processing operations.

We used imutils for:

Image Resizing: imutils has a function named resize() that allows you to simply resize images. It maintains the image's aspect ratio and offers resizing options depending on width, height, or a specific measurement.

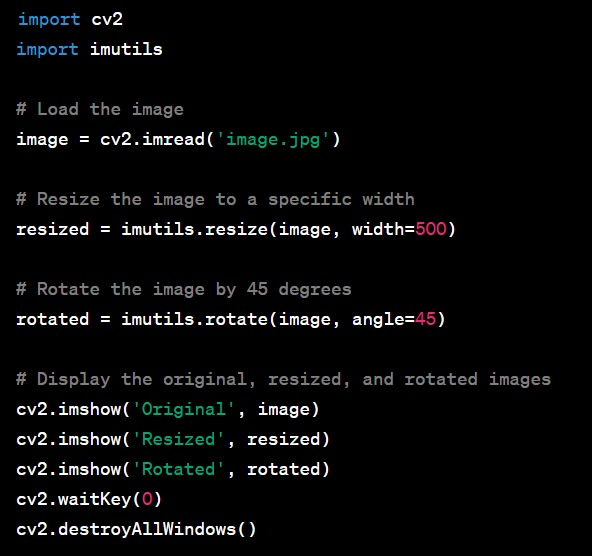
Cropping a ROI (Region of Interest): The crop() method in imutils makes it easier to extract a specific region of interest from an image. You can give the function the desired bounding box coordinates, and it will extract the relevant region.

Drawing Functions: Imutils offers drawing utilities such as drawBoundingBox() and drawText(), which make drawing bounding boxes and text on pictures easier.

To install imutils in windows we need to implement the following command in CMD

**pip install imutils**

**sample implementation of imutils**

****

* 1. **Dlib**

Dlib is a robust open-source library for machine learning, computer vision, and image processing. It's written in C++, but it also has a Python interface, so Python coders can use it. Face identification, facial landmark detection, object tracking, image classification, and other features are available in Dlib.

dlib library can be used for several applications such as:

Face Detection: Dlib includes a face detection algorithm that is based on a technique known as Histogram of Oriented Gradients (HOG). It can recognize faces in pictures or video frames under difficult settings such as shifting lighting and occlusions.

Facial landmark detection: Dlib features a facial landmark detection algorithm that can recognize key points on a face, such as the corners of the eyes, nose, and mouth. These landmarks can be used for a variety of purposes, including face alignment, emotion analysis, and facial expression identification.

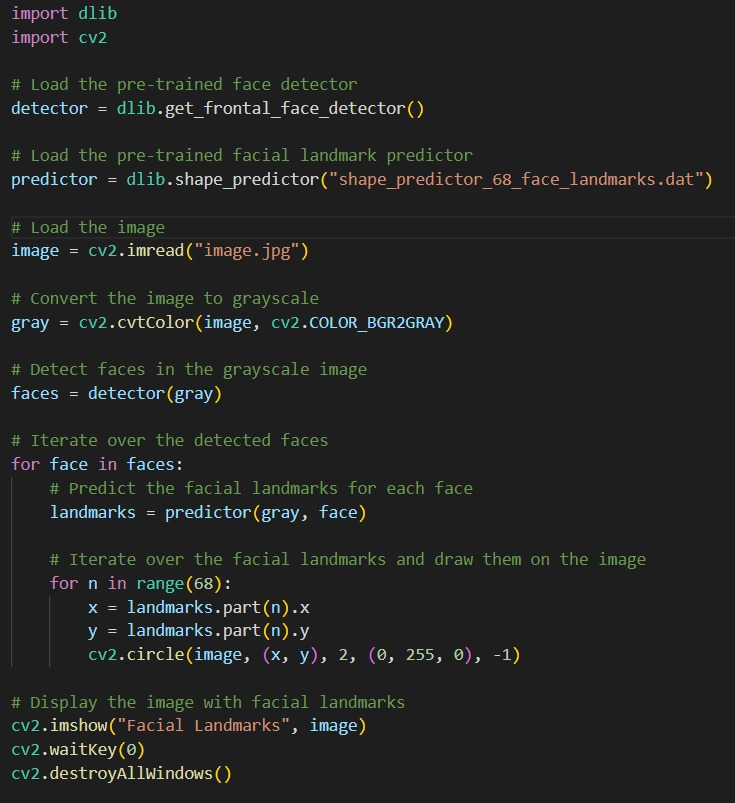
Object detection and tracking: Dlib provides techniques for training and deploying object identification models based on HOG features and Support Vector Machines (SVM). Custom object detectors can be trained to detect specific things in pictures or video streams. Dlib includes object-tracking methods such as the correlation tracker.

Integration with OpenCV: Dlib can be smoothly connected with OpenCV, allowing you to combine the features of both libraries. Dlib can be used for tasks such as face detection and facial landmark detection, while OpenCV can be used for image processing and display.

To install dlib in Windows we need to enter the below command in CMD

**pip install dlib**

**Sample implementation of dlib**

****

* 1. **Scipy.spatial**

scipy.spatial is a SciPy sub library that includes functions for working with spatial data and calculating distances between points, such as k-dimensional trees, Voronoi diagrams, and Delaunay triangulations.

The scipy.spatial.distance module contains functions for computing distances between two points or between a collection of points and another set of points. Distance measurements such as Euclidean distance, Manhattan distance, and cosine distance are included.

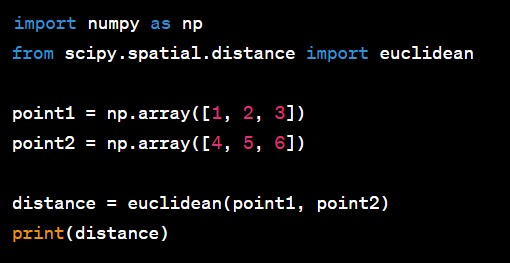
scipy.spatial library is used for:

Distance Metrics: The library provides several distance metrics, including Euclidean distance, Manhattan distance, and cosine distance. These distance functions can be used to determine the similarity or dissimilarity of points or vectors in a multidimensional space.

closest Neighbour Search: The scipy.spatial package includes data structures and algorithms for doing efficient closest neighbour searches. The scipy.spatial module. The KDTree class enables the creation of k-dimensional trees, which allow for rapid nearest neighbour searches and can be used for tasks such as locating the closest points to a given point or conducting k-nearest neighbour classification.

Complex Hull computation: Functions for computing the convex hull of a set of points in 2D or 3D space are included in the library. The convex hull is the smallest convex polygon that encloses all of the provided points and is helpful in computational geometry, collision detection, and image processing.

**Sample implementation of scipy.spatial**



* 1. **NumPy**

The NumPy library (short for Numerical Python) is a foundational Python module for scientific computing. It is a key library for numerical operations and data manipulation because it has sophisticated capabilities for working with arrays, matrices, and mathematical functions.

NumPy library is mostly used for applications such as:

Multidimensional Arrays: NumPy introduces the ndarray object, which is a strong N-dimensional array that enables fast storing and processing of homogeneous data. Arrays are the fundamental data format for numerical operations in NumPy and can have any number of dimensions.

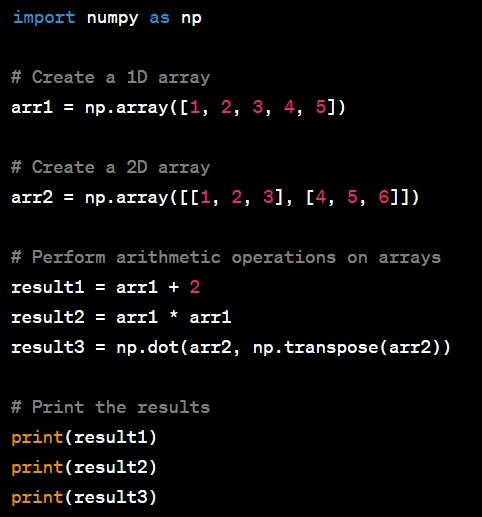
Mathematical Operations: Numpy includes a complete range of mathematical functions that operate effectively on ndarray objects. Basic arithmetic operations (addition, subtraction, multiplication, and division), trigonometric functions, exponential and logarithmic functions, linear algebra operations, and other operations are included. These functions are meant to work element-by-element on arrays, making sophisticated computations on enormous datasets simple.

Array Manipulation: Numpy provides a number of functions for manipulating and reshaping arrays. Functions such as reshape(), resize(), transpose(), and flatten() can be used to change the shape, size, and dimensions of an array. Furthermore, NumPy supports strong indexing and slicing, allowing you to extract certain elements or sub-arrays from an array.

To install NumPy library, we need to execute the below command in cmd

**pip install NumPy**

**sample implementation of numpy library**

****

* 1. **Pandas**

The pandas package for Python is a robust open-source data manipulation and analysis tool. It is a popular choice for working with structured data since it provides simple data structures and data analysis features.

Pandas library can be used for certain applications such as:

Pandas introduces two major data structures: Series and DataFrame.

A series is a one-dimensional labelled array that can hold any form of data. It is analogous to a spreadsheet column or a single column in a database table.

DataFrame is a two-dimensional labelled data structure with potentially diverse sorts of columns. It represents a tabular structure, much like a spreadsheet or SQL table.

Data Manipulation: Pandas has a plethora of functions and methods for manipulating and transforming data. Filtering, sorting, combining, joining, grouping, and altering data are all possible. You can use these processes to extract relevant insights, clean and preprocess data, and prepare it for further analysis.

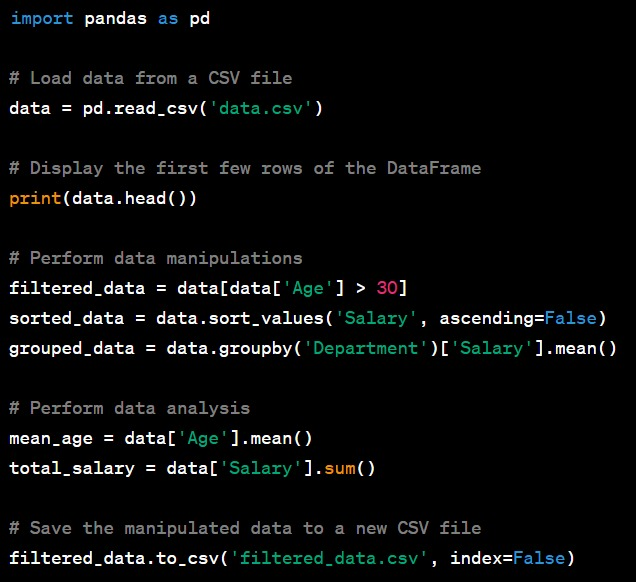
Data Analysis: Pandas provides a broad collection of data analysis tools. On your data, you can do statistical calculations, data aggregation, summarization, and descriptive statistics. It includes tools for dealing with missing data, working with time series data, and performing mathematical operations on your data.

Data Input/Output: Pandas can read and write data in a variety of formats, including CSV, Excel, SQL databases, JSON, and others. It makes it easier to load data from external sources and save the results of your analysis.

To install pandas in windows, we need to execute the below command in CMD.

**pip install pandas**

**Sample Implementation of pandas**

****

* 1. **Datetime**

The datetime library is a Python standard library that contains classes and functions for working with dates, times, and intervals. It lets us to modify, format, and calculate dates and times in a variety of forms.

Date and Time Representation: The datetime library includes numerous classes that can be used to represent dates, times, and intervals. The primary classes are as follows:

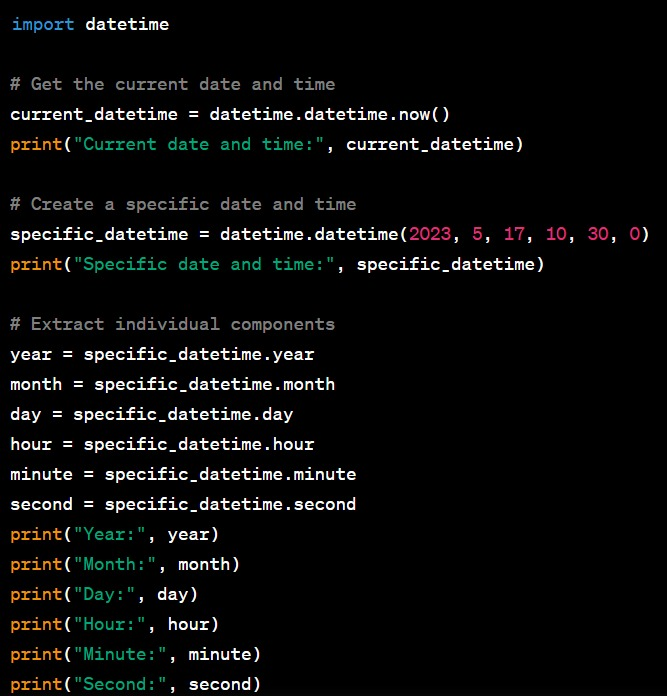
datetime: A date and time that includes the year, month, day, hour, minute, second, and microsecond.

date: A date without a time component that includes the year, month, and day.

time: A time without a date component that includes the hour, minute, second, and microsecond.

timedelta: Represents the difference or duration of two datetime objects, allowing you to execute arithmetic operations on dates and times.

**sample implementation of datetime library**

****

* 1. **Winsound**

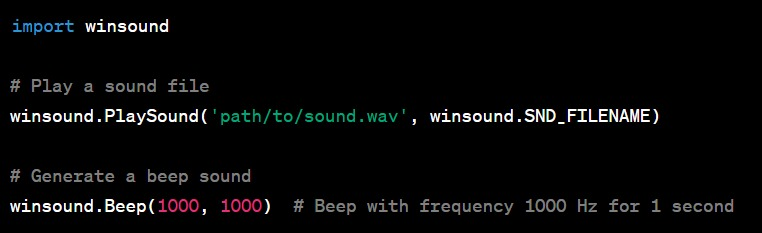
The winsound library is a Python built-in library that gives access to the Windows operating system's sound-playing features. It lets you play sound files and create simple sounds from predefined system sounds.

winsound library is used for several applications such as:

Playing Sound Files: You can use the winsound library to play sound files in various formats, such as WAV files. To play a sound file, use the PlaySound() function. As input, the function takes the path to the sound file and plays it asynchronously.

System noises: winsound has tools for creating simple system noises. To make a beep sound, for example, use the Beep() method. As parameters, you can choose the frequency and length of the beep sound.

**Sample implementation of winsound**

****

* 1. **OS**

The os library is a Python built-in library that allows you to communicate with the operating system. It provides a number of functions for carrying out operating-system-related tasks, such as file and directory operations, environment variables, process management, and so on.

os library is used for several applications such as:

File and Directory Operations: The operating system library includes functions for performing file and directory operations such as creating, renaming, and deleting files or directories, determining whether a file or directory exists, and retrieving information about files and directories.

Environment Variables: The os library can be used to retrieve and alter environment variables. It includes functions for retrieving the value of an environment variable, setting the value of an environment variable, and retrieving all environment variables.

**sample os implementation**

****

**Chapter 4**

**CODE**

**4.1 Code for Eye Shape Detection**

from imutils import face\_utils

import imutils

import dlib

import cv2

webcamera = cv2.VideoCapture(0)

svm\_predictor\_path = 'SVMclassifier.dat'

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

svm\_predictor = dlib.shape\_predictor(svm\_predictor\_path)

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

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

while True:

ret, frame = webcamera.read()

frame = imutils.resize(frame, width=640)

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

rects = svm\_detector(gray, 0)

for rect in rects:

shape = svm\_predictor(gray, rect)

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

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

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

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

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord("q"):

break

cv2.destroyAllWindows()

webcamera.release()

**4.2 Code for Mouth Shape Detection**

from scipy.spatial import distance as dist

from imutils import face\_utils

import imutils

import dlib

import cv2

import tkinter

page = tkinter.Tk()

page.title("Driver Drowsiness Monitoring")

# page.geometry("550x400")

page.geometry("{}x{}".format(page.winfo\_screenwidth(), page.winfo\_screenheight()))

font = ('times', 17, 'bold')

# title = tkinter.Label(page, text='Driver Drowsiness Detection',anchor=tkinter.W, justify=tkinter.CENTER)

title = tkinter.Label(page, text='Driver Drowsiness Detection', anchor='center', justify='center')

title.pack(fill='both', expand=True)

title.config(bg='black', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

def EAR(drivereye):

point1 = dist.euclidean(drivereye[1], drivereye[5])

point2 = dist.euclidean(drivereye[2], drivereye[4])

# computing the euclidean distance between the horizontal

distance = dist.euclidean(drivereye[0], drivereye[3])

# computing the eye aspect ratio

eye\_aspect\_ratio = (point1 + point2) / (2.0 \* distance)

return eye\_aspect\_ratio

def MOR(drivermouth):

# computing the euclidean distances between the horizontal

point = dist.euclidean(drivermouth[0], drivermouth[6])

# computing the euclidean distances between the vertical

point1 = dist.euclidean(drivermouth[2], drivermouth[10])

point2 = dist.euclidean(drivermouth[4], drivermouth[8])

# taking average

Ypoint = (point1+point2)/2.0

# computing mouth aspect ratio

mouth\_aspect\_ratio = Ypoint/point

return mouth\_aspect\_ratio

def startDetection():

webcamera = cv2.VideoCapture(0)

svm\_predictor\_path = 'SVMclassifier.dat'

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

svm\_predictor = dlib.shape\_predictor(svm\_predictor\_path)

(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"]

while True:

ret, frame = webcamera.read()

frame = imutils.resize(frame, width=640)

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

rects = svm\_detector(gray, 0)

for rect in rects:

shape = svm\_predictor(gray, rect)

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

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

mouth = shape[mStart:mEnd]

leftEAR = EAR(leftEye)

rightEAR = EAR(rightEye)

ear = (leftEAR + rightEAR) / 2.0

mar = MOR(mouth)

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

mouthHull = cv2.convexHull(mouth)

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

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

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

cv2.putText(frame, "EAR: {:.2f}".format(ear), (480, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.putText(frame, "MAR: {:.2f}".format(mar), (480, 60),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord("q"):

break

cv2.destroyAllWindows()

webcamera.release()

font1 = ('times', 15, 'bold')

button = tkinter.Button(page, text="Start Detecting", command=startDetection)

# button.place(x=50,y=200)

button.place(relx=0.5, rely=0.5, anchor='center')

button.config(font=font1)

pathlabel = tkinter.Label(page)

pathlabel.config(bg='DarkOrange1', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=50,y=250)

page.config(bg='chocolate1')

page.mainloop()

**Chapter 5**

**FINAL CODE**

**5.1 Code**

**import datetime**

**import os**

**import winsound**

**import numpy as np**

**import pandas as pd**

**from scipy.spatial import distance as dist**

**from imutils import face\_utils**

**import imutils**

**import dlib**

**import cv2**

**import tkinter**

**page = tkinter.Tk()**

**page.title("Driver Drowsiness Monitoring")**

**# page.geometry("550x400")**

**page.geometry("{}x{}".format(page.winfo\_screenwidth(), page.winfo\_screenheight()))**

**font = ('times', 17, 'bold')**

**# title = tkinter.Label(page, text='Driver Drowsiness Detection',anchor=tkinter.W, justify=tkinter.CENTER)**

**title = tkinter.Label(page, text='Driver Drowsiness Detection', anchor='center', justify='center')**

**title.pack(fill='both', expand=True)**

**title.config(bg='black', fg='white')**

**title.config(font=font)**

**title.config(height=3, width=120)**

**title.place(x=0,y=5)**

**def EAR(drivereye):**

**point1 = dist.euclidean(drivereye[1], drivereye[5])**

**point2 = dist.euclidean(drivereye[2], drivereye[4])**

**# computing the euclidean distance between the horizontal**

**distance = dist.euclidean(drivereye[0], drivereye[3])**

**# computing the eye aspect ratio**

**eye\_aspect\_ratio = (point1 + point2) / (2.0 \* distance)**

**return eye\_aspect\_ratio**

**def MOR(drivermouth):**

**# computing the euclidean distances between the horizontal**

**point = dist.euclidean(drivermouth[0], drivermouth[6])**

**# computing the euclidean distances between the vertical**

**point1 = dist.euclidean(drivermouth[2], drivermouth[10])**

**point2 = dist.euclidean(drivermouth[4], drivermouth[8])**

**# taking average**

**Ypoint = (point1+point2)/2.0**

**# computing mouth aspect ratio**

**mouth\_aspect\_ratio = Ypoint/point**

**return mouth\_aspect\_ratio**

**def createLog(message):**

**# current\_time = datetime.datetime.now()**

**current\_time = datetime.datetime.now()**

**date\_str = current\_time.strftime("%d-%m-%y")**

**time\_str = current\_time.strftime("%H:%M:%S")**

**if os.path.exists("log.xlsx") == True:**

**data = pd.read\_excel("log.xlsx")**

**data = data.values**

**data = np.vstack([data, [message, str(date\_str), str(time\_str)]])**

**else:**

**data = []**

**data.append([message, str(date\_str), str(time\_str)])**

**data = np.asarray(data)**

**df = pd.DataFrame(data, columns=['Alert','Date','Time'])**

**df.to\_excel("log.xlsx", index=False)**

**def beep():**

**frequency = 2500 # Set Frequency To 2500 Hertz**

**duration = 1000 # Set Duration To 1000 ms == 1 second**

**winsound.Beep(frequency, duration)**

**def startDetection():**

**webcamera = cv2.VideoCapture(0)**

**svm\_predictor\_path = 'SVMclassifier.dat'**

**svm\_detector = dlib.get\_frontal\_face\_detector()**

**svm\_predictor = dlib.shape\_predictor(svm\_predictor\_path)**

**(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"]**

**eyeThreshValue = 0.25**

**eyeRelativeFrames = 10**

**mouthThreshValue = 0.75**

**frameCount = 0**

**yawnStatus = False**

**yawnCount = 0**

**while True:**

**ret, frame = webcamera.read()**

**frame = imutils.resize(frame, width=640)**

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

**rects = svm\_detector(gray, 0)**

**recentYawnStatus = yawnStatus**

**for rect in rects:**

**shape = svm\_predictor(gray, rect)**

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

**leftEye = shape[lStart:lEnd]**

**rightEye = shape[rStart:rEnd]**

**mouth = shape[mStart:mEnd]**

**leftEAR = EAR(leftEye)**

**rightEAR = EAR(rightEye)**

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

**mar = MOR(mouth)**

**leftEyeHull = cv2.convexHull(leftEye)**

**rightEyeHull = cv2.convexHull(rightEye)**

**mouthHull = cv2.convexHull(mouth)**

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

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

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

**# cv2.putText(frame, "MAR: {:.2f}".format(mar), (480, 60),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**if ear < eyeThreshValue:**

**frameCount += 1**

**cv2.putText(frame, "Eyes Closed ", (10, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**if frameCount >= eyeRelativeFrames:**

**cv2.putText(frame, "DROWSINESS ALERT!", (10, 50),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**createLog("Eyes Closed Alert")**

**beep()**

**else:**

**frameCount = 0**

**cv2.putText(frame, "Eyes Open ", (10, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2)**

**cv2.putText(frame, "EAR: {:.2f}".format(ear), (480, 30),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**if mar > mouthThreshValue:**

**cv2.putText(frame, "Yawning, DROWSINESS ALERT! ", (10, 70),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**yawnStatus = True**

**output\_text = "Yawn Count: " + str(yawnCount + 1)**

**cv2.putText(frame, output\_text, (10,100),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7,(255,0,0),2)**

**createLog("Yawning Alert")**

**beep()**

**else:**

**yawnStatus = False**

**if recentYawnStatus == True and yawnStatus == False:**

**yawnCount+=1**

**cv2.putText(frame, "MAR: {:.2f}".format(mar), (480, 60),cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**

**cv2.imshow("Frame", frame)**

**key = cv2.waitKey(1) & 0xFF**

**if key == ord("q"):**

**break**

**cv2.destroyAllWindows()**

**webcamera.release()**

**font1 = ('times', 15, 'bold')**

**button = tkinter.Button(page, text="Start Detecting", command=startDetection)**

**# button.place(x=50,y=200)**

**button.place(relx=0.5, rely=0.5, anchor='center')**

**button.config(font=font1)**

**pathlabel = tkinter.Label(page)**

**pathlabel.config(bg='DarkOrange1', fg='white')**

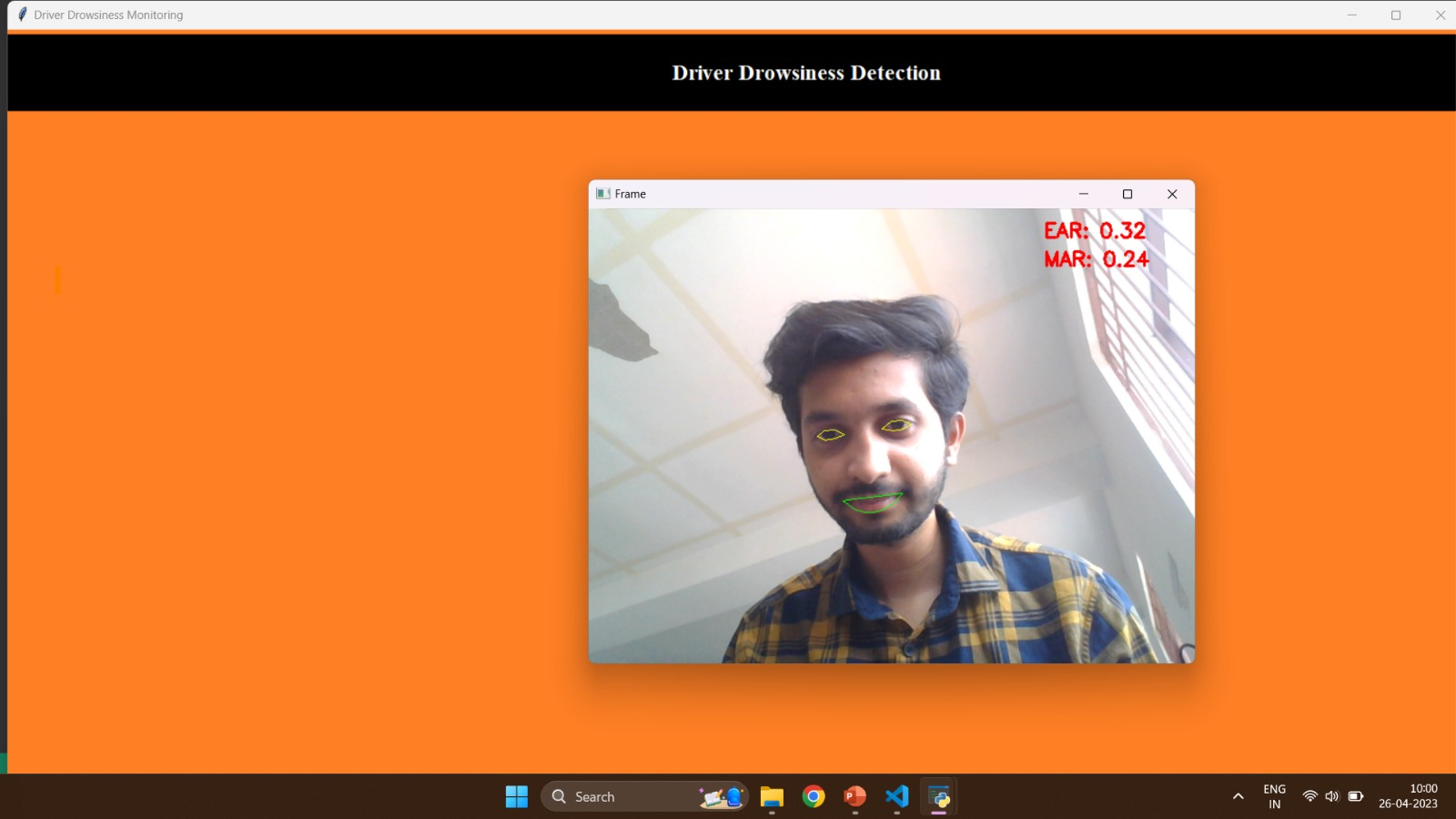
**pathlabel.config(font=font1)**

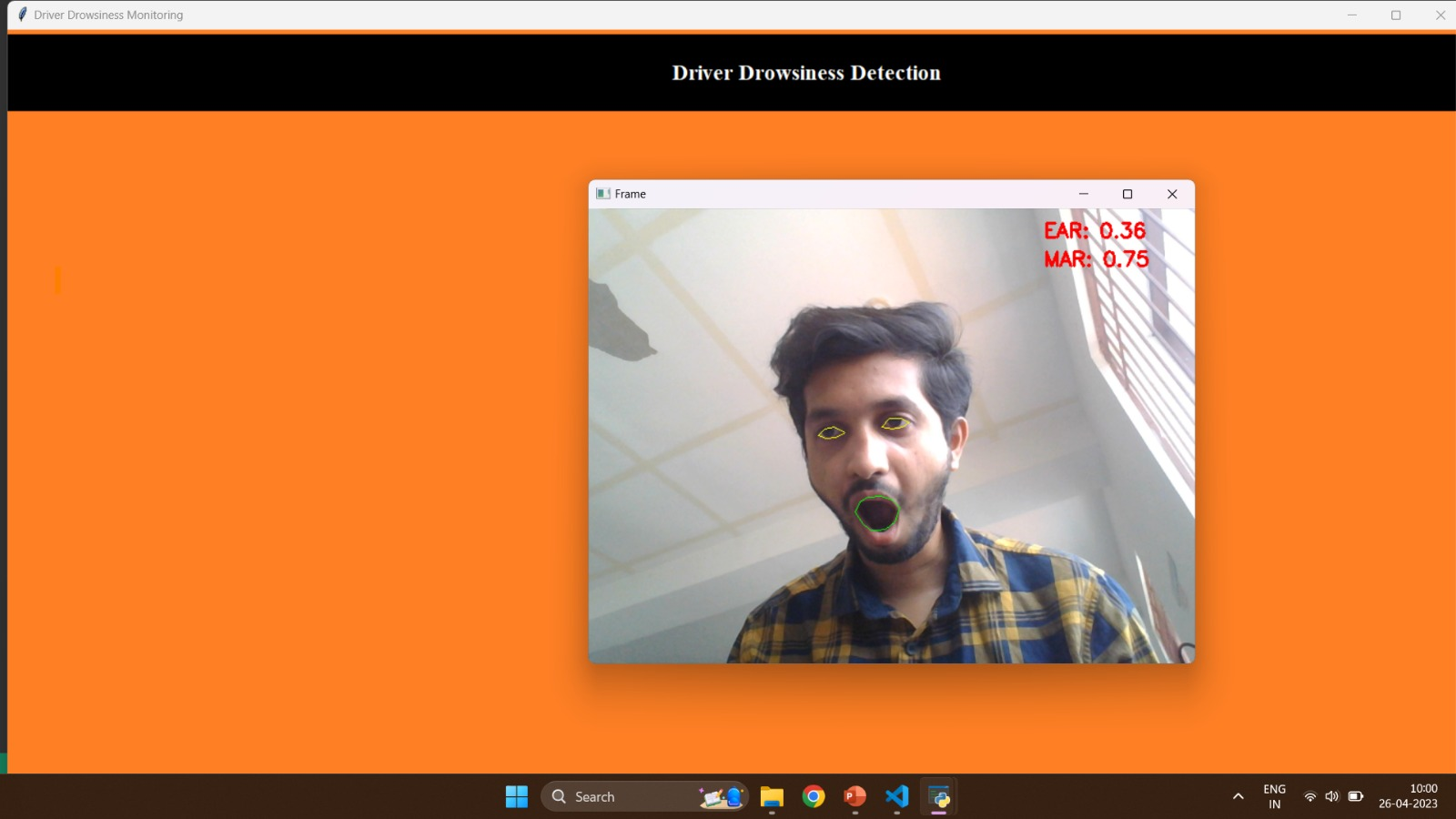
**pathlabel.place(x=50,y=250)**

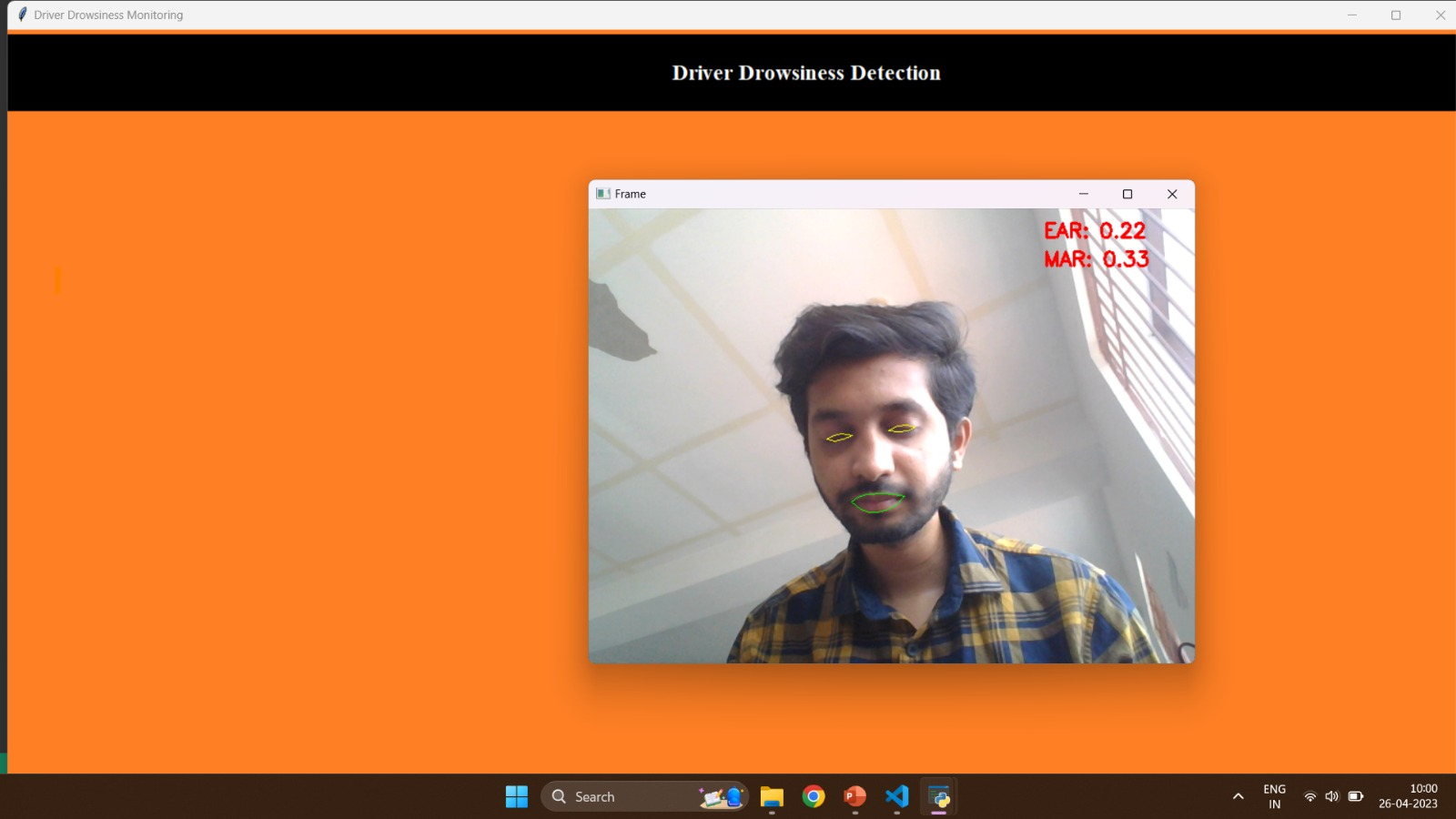
**page.config(bg='chocolate1')**

**page.mainloop()**

**5.2 Output**

****





**Chapter 6**

**CONCLUSION & FUTURE WORK**

**6.1 Conclusion**

Finally, the driver drowsiness detection project is a worthwhile endeavour that addresses the essential issue of driver alertness on the road. The project intends to detect indicators of tiredness in drivers and deliver early notifications to prevent potential accidents by using innovative technologies such as computer vision and machine learning. The project has various advantages, such as higher road safety, fewer accidents due by driver inattention, and a better overall driving experience.

The project can efficiently monitor driver behaviour and detect indicators of tiredness or distraction by implementing a comprehensive system that includes components such as cameras, sensors, image processing algorithms, and machine learning models. The technology can reliably identify probable sleepiness indicators by analysing facial expressions, eye movements, head position, and other pertinent factors.

While the current approach provides a framework for detecting driver drowsiness, there are certain limitations to consider. These may include the detection process's occasional false positives or false negatives, reliance on good illumination conditions, and the requirement for constant calibration and development of the algorithms. Furthermore, the system's performance may vary depending on the individual driver's attributes and specific road scenarios.

**6.2 Future Work**

Despite these constraints, the project offers a lot of room for growth and improvement. Future improvements could include adding more sensors for more thorough monitoring, enhancing machine learning algorithms to improve accuracy, and implementing real-time data analysis and decision-making capabilities.

Overall, the driver drowsiness detection project has enormous potential to improve road safety by tackling the serious issue of driver inattention. With greater study, development, and implementation, this technology has the potential to significantly reduce accidents caused by drowsy driving while also providing a better and more secure transportation environment for all road users.

**Chapter 7**

**REFERENCES**

Guo, S., Zhu, Y., & Sheng, W. (2020). A Survey on Driver Drowsiness Detection Systems. IEEE Transactions on Intelligent Transportation Systems, 21(10), 4338-4359.

Singh, A., Chaudhary, V., & Chaudhary, R. (2020). Driver Drowsiness Detection Techniques: A Comprehensive Review. International Journal of Intelligent Transportation Systems Research, 18(3), 1144-1177.

Chen, F., & Li, B. (2021). Driver Drowsiness Detection Based on Multimodal Information: A Review. IEEE Transactions on Human-Machine Systems, 51(2), 177-188.

Sharma, A., & Garg, S. (2020). A Survey of Vision-Based Systems for Driver Drowsiness Detection. Computer Vision and Image Understanding, 190, 102881.

Islam, M. M., & Kim, J. M. (2020). A Review on Non-Contact Driver Drowsiness Detection Using Physiological Signals. Sensors, 20(15), 4205.