**Palestine Technical University - Khadoorie**

**Faculty of Engineering and Technology**

**Department of Computer Systems Engineering**



**Graduation Project Thesis**



**DriveSafe: AI-Based Driver Drowsiness Detection System**

**Prepared by:**

Aseel Jayousi

Hanan Abuzainab

**Supervised by:**

Anas Melhm, Ph.D.

A graduation project submitted in partial fulfillment of the requirements for the

Bachelor's degree in Computer Systems Engineering.

Tulkarm – Palestine

January 2026

TABLE OF CONTENTS

[**LIST OF FIGURES** 3](#_Toc216011922)

[**LIST OF TABLES** 3](#_Toc216011923)

[ABSTRACT 4](#_Toc216011924)

[ملخص 5](#_Toc216011925)

[CHAPTER 1 – INTRODUCTION 6](#_Toc216011926)

[1.1 Motivation 7](#_Toc216011927)

[1.2 Problems 8](#_Toc216011928)

[1.3 Objectives 8](#_Toc216011929)

[CHAPTER 2 – LITERATURE REVIEW 9](#_Toc216011930)

[2.1 Traditional Drowsiness Detection Methods 10](#_Toc216011931)

[2.2 Vision-Based Drowsiness Detection Methods (Without AI) 11](#_Toc216011932)

[2.3 Deep Learning Methods for Detecting Drowsiness 12](#_Toc216011933)

[2.4 Comparison between Drowsiness Detection Methods 13](#_Toc216011934)

[2.5 Conclusion 13](#_Toc216011935)

[CHAPTER 3 - PROPOSED SOLUTION 14](#_Toc216011936)

[3.1 Proposed Model 15](#_Toc216011937)

[3.2 Proposed Hardware 17](#_Toc216011938)

[3.3 Diagrams 18](#_Toc216011939)

[CHAPTER 4 - IMPLEMENTATION 20](#_Toc216011940)

[4.1 Data Description 21](#_Toc216011941)

[4.2 Data Preparation 24](#_Toc216011942)

[4.3 Training process 28](#_Toc216011943)

[4.4 Software Deployment 36](#_Toc216011944)

[Reference list 53](#_Toc216011945)

**LIST OF FIGURES**

[Figure 1: Driving Drowsiness Monitoring using EGG signal 10](#_Toc216026370)

[Figure 2: Eye landmark points used for EAR calculation 11](#_Toc216026371)

[Figure 3: Performance comparison of YOLOv11 with previous YOLO versions 13](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026372)

[Figure 4:llustrates the drowsiness detection pipeline using a hybrid YOLO11 model. 16](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026373)

[Figure 5: Proposed hardware components 17](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026374)

[Figure 6: Activity diagram 18](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026375)

[Figure 7: Sequence diagram 19](#_Toc216026376)

[Figure 8: Sample images from the NTHU Dataset 21](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026377)

[Figure 9: Sample images from the Yawning Dataset 22](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026378)

[Figure 10: Sample closed-eye images from CEW dataset 22](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026379)

[Figure 11: Sample images from Driver Behavior Image dataset 22](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026380)

[Figure 12: Sample images from the Kaggle datasets 23](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026381)

[Figure 13: Absent class samples collected from web sources 23](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026382)

[Figure 14: Absent class samples captured using a mobile phone 23](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026383)

[Figure 15: Data Preparation Flowchart 24](#_Toc216026384)

[Figure 16: Dataset Split 26](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026385)

[Figure 17: Male vs. Female Ratio 27](#_Toc216026386)

[Figure 18: Training Process Flowchart 28](#_Toc216026387)

[Figure 19: Examples of Data Augmentation Applied During Training 30](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026388)

[Figure 20: : Confusion matrix 32](#_Toc216026389)

[Figure 21: : Normalized confusion matrix 33](#_Toc216026390)

[Figure 22: Training and Validation Loss, Top-1 Accuracy, and Top-5 Accuracy Curves 34](file:///C:\Users\lenovo\Downloads\editing%20december_last%20of%20ch4%20(1).docx#_Toc216026391)

[Figure 23 48](#_Toc216026392)

[Figure 24 48](#_Toc216026393)

[Figure 25 49](#_Toc216026394)

[Figure 26 49](#_Toc216026395)

[Figure 27 50](#_Toc216026396)

[Figure 28 50](#_Toc216026397)

[Figure 29 51](#_Toc216026398)

[Figure 30 51](#_Toc216026399)

[Figure 31 52](#_Toc216026400)

[Figure 32 52](#_Toc216026401)

**LIST OF TABLES**

[Table 1: The difference between Drowsiness Detection Methods 13](#_Toc215952976)

[Table 2: Dataset Distribution by Class, Gender, and Data Split 27](#_Toc215952977)

[Table 3: Model Performance Metrics 35](#_Toc215952978)

# ABSTRACT

Driver drowsiness is a major cause of road accidents, especially during long or late-night trips. This project introduces a real-time drowsiness detection system that monitors the driver’s full facial features using a front-facing camera. It detects signs of fatigue such as eye closure, head movement, and yawning, then alerts the driver before it becomes dangerous. While similar systems are usually only available in modern high-end vehicles, through intelligent monitoring, the solution contributes to safer roads and more attentive driving.

# ملخص

تُعدّ حالة النعاس أثناء القيادة من الأسباب الرئيسية لحوادث الطرق، خاصةً خلال الرحلات الطويلة أو أثناء القيادة ليلاً. يقدّم هذا المشروع نظامًا للكشف عن النعاس يعمل في الوقت الحقيقي، حيث يقوم بمراقبة ملامح وجه السائق بالكامل باستخدام كاميرا أمامية. يكتشف النظام علامات التعب مثل إغلاق العينين وحركة الرأس والتثاؤب، ثم يُصدر تنبيهًا للسائق قبل أن تتطور الحالة إلى وضع خطير. وعلى الرغم من أن أنظمة مشابهة تتوفر عادةً فقط في السيارات الحديثة والفاخرة، فإن هذا النظام يهدف من خلال المراقبة الذكية إلى المساهمة في تعزيز السلامة على الطرق ودعم التركيز والانتباه أثناء القيادة.

## 

# CHAPTER 1 – INTRODUCTION

This chapter introduces the project and highlights its main idea.

Traffic accidents are a major problem that causes material and moral losses and is one of the main causes of injury, disability, and death around the world [1][2] with the increasing number of vehicles on the roads. There are many causes of accidents, including excessive speed and the use of mobile phones while driving, among others especially in low-income countries, the causes of accidents are increasing [3]. One of the serious and slowly developing causes is drowsiness [4], a real problem in which the driver loses concentration and is less able to notice sudden changes in the road, making incorrect decisions, and having difficulty staying on the right lane. Factors that cause drowsiness include driving long distances or during night driving, or it may occur due to the driver's lack of sleep, which causes poor concentration. There are early signs of drowsiness, such as frequent yawning, frequent blinking, and heavy eyelids, but these signs are often ignored by drivers, leading to accidents that affect not only the driver but also passengers and other road users. Therefore, there is a need for systems that detect driver drowsiness in its early stages to reduce traffic accidents and their significant effects. Many cars still don't have drowsiness detection systems, like older or low-cost cars, leaving a segment of drivers unprotected from drowsiness and its consequences. Therefore, it has become necessary to find a smart, practical, and low-cost solution that suits all categories to monitor drivers and keep them alert. This project aims to develop a system to detect driver drowsiness, monitor their behavior in real time, and detect potential signs of drowsiness. If signs appear, the system alerts the driver. This system can reduce traffic accidents resulting from drowsiness, save lives, and make driving safer.

## 1.1 Motivation

The idea of a driver drowsiness detection system is motivated by the following key points:

1. **Providing safety for all drivers**: Every driver deserves to reach their destination safely and to use drowsiness detection systems that are easy to use, affordable, and effective. Drowsy driving is a major risk, especially in developing countries and low-cost vehicles that lack such systems.
2. **Save lives**: By preventing accidents that could be avoided by drowsiness detection, we aim to reduce the resulting harm and protect the lives of drivers, passengers, and other road users.
3. **Lack of driver awareness**: Many drivers ignore drowsiness and continue driving without considering it a risk, leading to accidents.

## 1.2 Problems

The following problems highlight the need for the proposed drowsiness detection system:

1. **Drowsiness increases the risk of accidents**: When drivers are tired, their focus and reaction time go down, especially during long trips or late at night. Some drivers even experience “microsleep,” where they briefly fall asleep without realizing it, which can lead to serious accidents.
2. **Such systems are not widely available**: Most cars don’t have tools that can detect signs of drowsiness. And the systems that do exist are usually expensive and found only in luxury vehicles, not in regular or public transport cars [5].
3. **Drivers often don’t realize how tired they are**: Many people think they’re fine to drive, even when they’re actually very tired. This makes it hard to judge their own alertness, which increases the risk without them noticing.

## 1.3 Objectives

The focus of this project is to design and implement an intelligent system that monitors driver alertness in real time. The main objectives are to:

1. **Enhance road safety** by providing a real-time alert system that helps prevent accidents caused by driver drowsiness.
2. **Design a cost-effective and adaptable solution** that can be installed in various types of vehicles, including personal cars, buses, and trucks, making it accessible to a wide range of users.
3. **Raise awareness about drowsy driving** by using the system to highlight the risks of driver fatigue and promote the use of intelligent safety technologies.

# CHAPTER 2 – LITERATURE REVIEW

This chapter presents a review of existing studies and related work relevant to the project.

## 2.1 Traditional Drowsiness Detection Methods

**2.1.1 Physiological Signal-Based Approaches (EEG, EOG, ECG)**

Some of the earliest methods for detecting drowsiness focused on physiological signals, in other words, measuring how the body responds internally when someone starts to feel tired. These approaches rely on specialized sensors that monitor brain activity, eye movement, and heart rate They detect changes that indicate fatigue.

* **EEG (Electroencephalogram):** This measures electrical activity in the brain. Studies have shown that as a person gets drowsy, high-frequency brain waves like beta waves start to decrease, while slower waves such as theta and delta increase. These shifts are considered early indicators of sleepiness.
* **EOG (Electrooculogram):** This tracks how the eyes move and blink. When someone is tired, their blinks slow down, and the eyes tend to remain closed for longer periods - both of which are strong signs of drowsiness.
* **ECG (Electrocardiogram):** This monitors heart rate and rhythm. A decrease in heart rate or irregular heartbeat patterns can also point to the onset of fatigue.

One study [6] used EEG signals to estimate PERCLOS - the percentage of eyelid closure over time - and achieved a very low error rate (RMSE = 0.117), which shows high accuracy in predicting drowsiness.

Also in [7], the authors proposed a hybrid drowsiness detection system based on EEG and ECG signals. They tested the approach on 22 subjects in a simulated driving environment and achieved an accuracy of 80% using only two electrodes - one for EEG and one for ECG.

Even though these methods are accurate, they’re not very practical for real-world use. Most of them require physical contact with the skin or scalp through electrodes, which can be uncomfortable and not ideal for daily driving situations. That’s why their use is mostly limited.



Figure 1: Driving Drowsiness Monitoring using EGG signal

**2.1.2 Vehicle Behavior-Based Approaches**

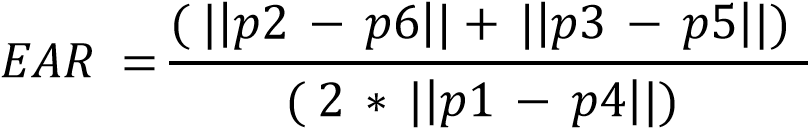
Another way to detect driver drowsiness is by observing how the driver controls the vehicle - like steering wheel movements, lane drifting, or braking patterns. These systems don’t require physical contact with the driver and often use sensors already built into modern vehicles.

In [8], the authors proposed a non-intrusive driver drowsiness detection model using steering wheel data. They applied an adaptive neuro-fuzzy feature selection method with a support vector machine classifier, achieving 98.12% accuracy in detecting whether the driver was drowsy or alert.

However, these methods usually detect drowsiness only after the driver’s behavior has already changed. They can also produce false alarms due to external factors like road curves, wind, or bumps. In addition, not all vehicles have the required sensors, and installing them can be expensive or impractical.

## 2.2 Vision-Based Drowsiness Detection Methods (Without AI)

Before the use of artificial intelligence, vision-based drowsiness detection systems relied on handcrafted geometric features extracted from facial landmarks. A key metric used in these systems is the **Eye Aspect Ratio (EAR)**, EAR is calculated as the ratio of vertical eye landmark distances to horizontal distances, reflecting how open the eye is and is defined as:



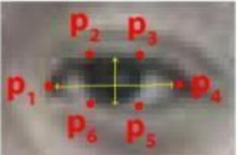


Figure 2: Eye landmark points used for EAR calculation

In [9], Soukupova and Cech proposed a real-time eye blink detection method based solely on EAR values and facial landmarks. Their system, which used Dlib for facial point detection, did not involve any machine learning techniques. Instead, it applied threshold-based classification to detect blinks in real time.

In [10], a real-time drowsiness detection method was proposed based on detecting the face and eyes, then measuring the percentage of time the eyes remain closed to determine drowsiness. The system applied simple threshold-based rules to estimate eye closure over time.

The methods worked well in normal conditions but were less accurate with low light or head movement.

## 2.3 Deep Learning Methods for Detecting Drowsiness

With the rapid advancement of deep learning, methods for driver drowsiness detection have increasingly shifted towards approaches that can automatically extract and analyze features from images and videos. Such models have proven highly effective for real-time monitoring of driver alertness.

**Convolutional Neural Networks (CNNs) for Drowsiness Detection**

Convolutional Neural Networks (CNNs) are widely used for image classification. In drowsiness detection, CNNs analyze facial features and eye states to determine whether a driver is alert or drowsy.

In [11] a system for driver drowsiness detection was proposed using computer vision and deep learning techniques. Two models were studied and compared: a custom-designed CNN model and a pre-trained VGG16 model. A dataset containing diverse images was used, and the accuracy of the CNN model was 97%, while the custom VGG16 model achieved 74%.

Also, in this study [12], a drowsiness detection system was developed using convolutional neural networks and computer vision. This system uses real-time video processing using OpenCV to extract and analyze facial features and eye ratios. The results achieved an accuracy of 97.2%. However, this system automatically detects drowsiness.

**YOLO Architectures for Real-Time Drowsiness Detection**

YOLO (You Only Look Once) is a CNN-based model, but it processes the entire image in a single step, making it faster and more efficient than traditional CNNs.

One relevant study that supports the use of YOLO architectures for driver drowsiness detection is *“Real Time Driver Drowsiness Detection Using Facial Analysis and Machine Learning Techniques”*[13]. In this work, the authors compared YOLO-based models with traditional machine learning and computer vision techniques. Their results showed that YOLOv5 and YOLOv8 achieved exceptional performance, reaching 100% precision and recall, with a mean average precision (mAP@0.5) of approximately 99.5%.

These findings highlight the strong capability of YOLO models for real-time, non-intrusive drowsiness detection, demonstrating their effectiveness in accurately identifying facial cues of fatigue

## 2.4 Comparison between Drowsiness Detection Methods

Table 1: The difference between Drowsiness Detection Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Traditional** | **Vision-Based (Without AI)** | **Deep Learning** |
| **Input type** | Sensors signals | Cropped eye frames | Raw face images |
| **Data requirements** | Low - works with signals or built-in vehicle sensors | |  | | --- | |  |  |  | | --- | | Low - relies on geometric measurements | | High - requires large annotated datasets |
| **Robustness to lighting & pose** | Very high (not camera dependent).  Moderate for the vehicle affected | Low - heavily affected by lighting, shadows, and head rotation | High - CNNs/YOLO handle variations well |
| **Processing Speed** | |  | | --- | | Medium | | High | High (YOLO extremely fast)-but it depends on hardware performance. |
| **Hardware Requirements** | Electrodes, Vehicle sensors | Camera only, almost no GPU/CPU load | Camera, a powerful computer or processing unit |
| **Limitations** | Intrusive, Uncomfortable, Costly | Prone to false alarms and not very reliable | High computational cost |

## 2.5 Conclusion

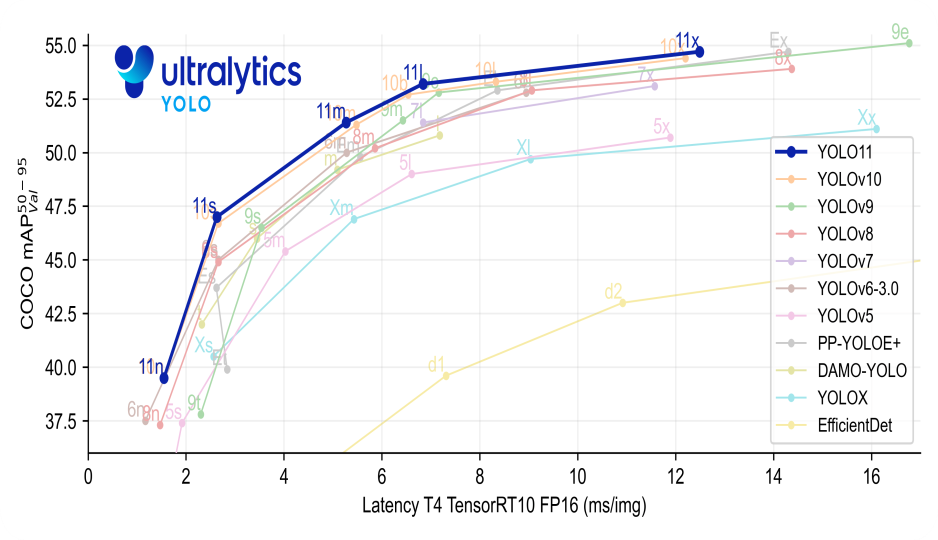
For this project, deep learning methods were chosen for driver drowsiness detection due to their ability to automatically extract features from video frames, high accuracy, robustness to variations in lighting and head pose, and excellent real-time performance.  
Specifically, YOLO models outperform traditional CNNs in real-time drowsiness detection, offering higher accuracy and faster processing. Versions like YOLOv5 and YOLOv8 have already demonstrated excellent results in detecting driver fatigue efficiently. Here, we adopt the latest YOLOv11, which provides further improvements in speed, accuracy, and robustness. A diagram illustrating YOLOv11’s performance will be presented below.

Figure 3: Performance comparison of YOLOv11 with previous YOLO versions

# CHAPTER 3 - PROPOSED SOLUTION

This chapter presents the proposed system and how it works

In this chapter, we present a proposed driver drowsiness detection system that initiates detection immediately using video inputs and determines the driver's state using a deep learning-based model (YOLO11). It uses indicators to determine the system's state. If drowsiness is detected, the system activates an alert to warn the driver.

## 3.1 Proposed Model

The proposed model consists of three main stages: Input, Processing, and Decision Making. These stages work together to enable real-time drowsiness detection, as illustrated in Figure 4.

**1. Input Stage**

The system uses real-time video captured from a front-facing **camera** mounted on the dashboard of the vehicle.

During the development phase, multiple driver-monitoring datasets were utilized. These datasets contain videos of drivers exhibiting various states, such as awake, blinking, yawning, drowsy, and asleep, recorded under different lighting conditions (including daytime, nighttime, and infrared). The data is divided into training and testing subsets.

**2. Processing Stage**

**Frame Acquisition:** Splits the video stream into individual frames. Then, each video frame captured by the camera is received by the system and prepared for analysis, ensuring a continuous real-time processing flow.

**Frame** **Preprocessing:** Each frame is converted to grayscale, resized, normalized, and formatted to meet the input requirements of the model.

**Frame Classification:** The preprocessed frame is passed to a YOLO-based classification model, which classifies the frame into one of three states.

**Drowsiness Tracking:** The system monitors consecutive frames and keeps track of how many of them are classified as drowsy. If the number of drowsy frames exceeds a predefined threshold (e.g., more than 20 consecutive frames), the system concludes that the driver is in a drowsy state.

**3. Decision-Making stage**

* **State Classification:**

After analyzing the input frame, the model classifies the driver’s condition into one of three states:

* **Awake**
* **Drowsy**
* **Absent**
* **System Response:**

Depending on the detected state, the system generates an immediate response to enhance driving safety:

* **Red LED:** The red LED is continuously turned on while the driver’s face is not visible, signaling a loss of monitoring.
* **Green LED:** The green LED is turned on whenever the driver’s face is visible, indicating that monitoring is active.
* **buzzer:** If the driver is classified as drowsy and the number of consecutive drowsy frames exceeds the predefined threshold, the buzzer warns the driver to regain attention.
* **Continuous monitoring:**

The system continuously updates its prediction as new frames are captured, ensuring that any change in the driver’s condition is quickly detected and handled.

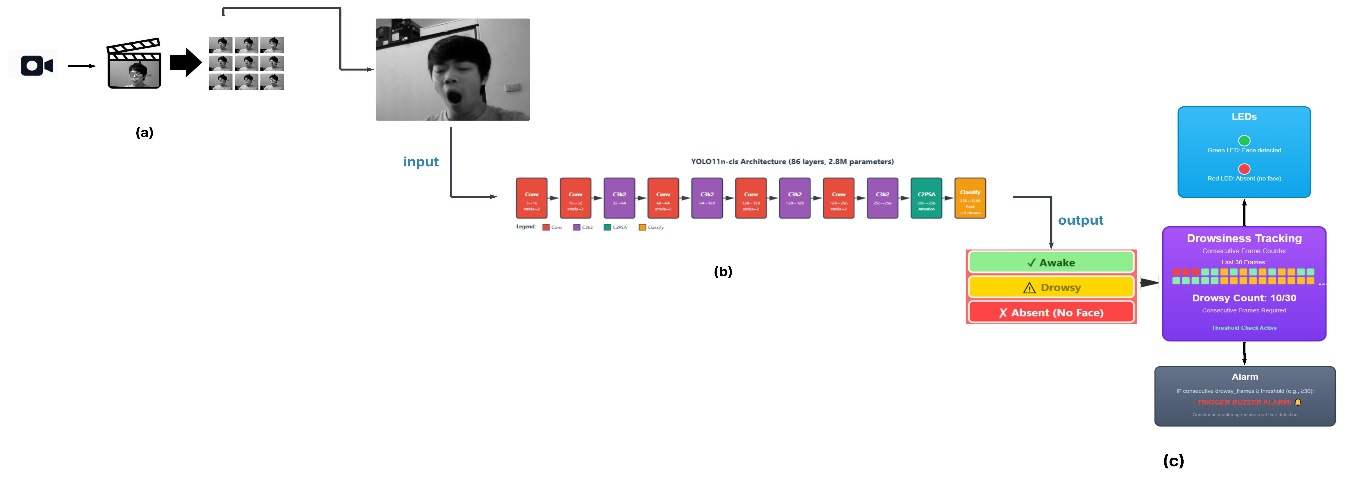


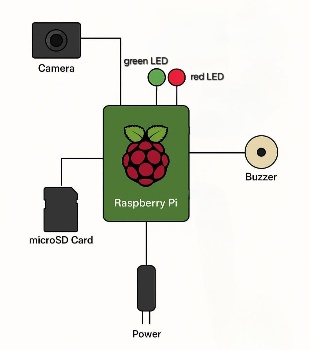
Figure 4:llustrates the drowsiness detection pipeline using a hybrid YOLO11 model.

**(a)** Real-time video capture from dashboard camera (30 fps).**(b)** Frame extraction and classification using YOLOv11 into three states: Awake, Drowsy, or Absent. **(c)** Temporal tracking of the last 30 frames with a dual alert system: **LEDs** for continuous monitoring (Green: awake, Red: absent) and a **Buzzer** for drowsy threshold exceedance.

## 3.2 Proposed Hardware

This driver drowsiness detection system is built using low-cost hardware components integrated with a processing unit to enable real-time monitoring. The main components include:

**Processing Unit:** Handles all computations for video processing and YOLOv11 inference.

**Camera:** Captures real-time video of the driver’s face.

**microSD Card:** Stores the operating system, code,

and trained model.

**LED Indicators:** Show system status (green and red).

**Buzzer:** Provides auditory alerts to notify the driver.

**Power Supply:** Powers the entire system.

Figure 5: Proposed hardware components

## 3.3 Diagrams

This section includes two diagrams that illustrate the internal behavior of the drowsiness detection system.

**3.3.1 Activity Diagram**

The following diagram illustrates the core workflow of the proposed drowsiness detection system.

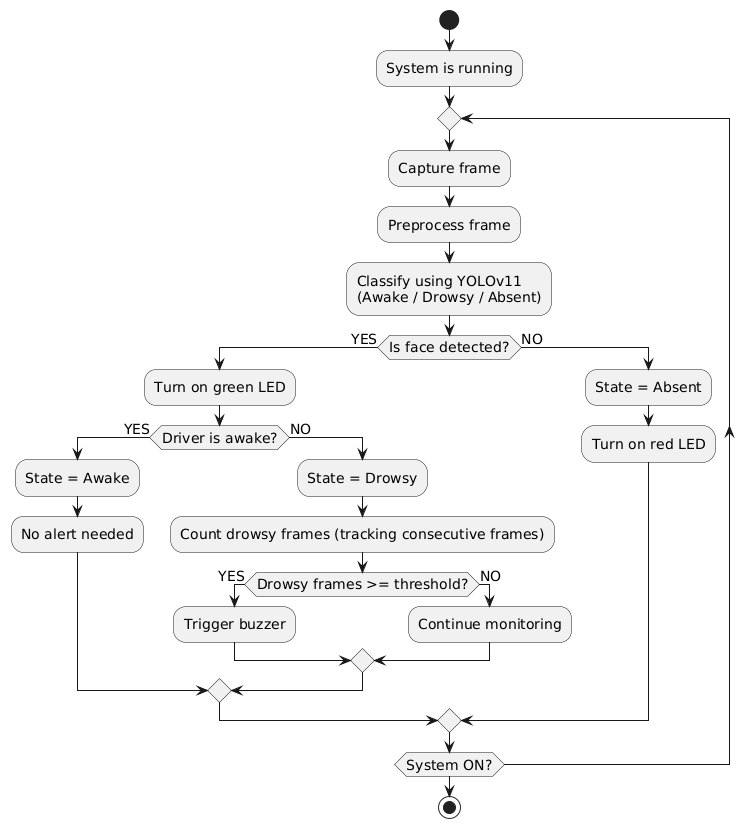
**

Figure 6: Activity diagram

**3.3.1 Sequence Diagram**

The following diagram illustrates a sequence diagram of the interactions between the proposed drowsiness detection system and the remaining components.

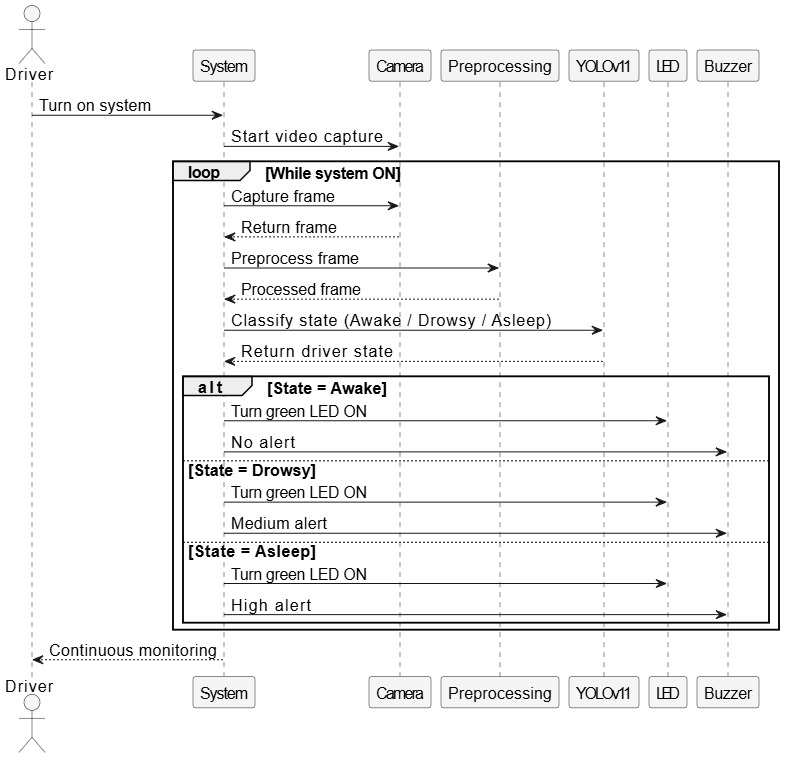


Figure 7: Sequence diagram

# CHAPTER 4 - IMPLEMENTATION

This chapter presents the implementation of the system and how it has been developed and deployed.

In this chapter, the procedural steps for developing the driver drowsiness detection system are outlined. This chapter delves into the intricacies of each step. The chapter provides the [source code](https://github.com/hananabuzainab/DriveSafe-AI-Based-Driver-Drowsiness-Detection-System.git)

## 4.1 Data Description

In this project, multiple datasets were used to train and evaluate the proposed driver drowsiness detection system. The **NTHU Drowsy Driver Detection Dataset (NTHU-DDD)** served as the primary source, complemented by additional images from **YawDD**, **Closed Eyes in the Wild (CEW)**, the **Driver Behavior Image Dataset**, and relevant collections from **Kaggle**. Together, these datasets provide diverse facial expressions, lighting conditions, and participant characteristics, thereby enhancing the model’s generalization capability.

The system also includes a third class, **Absent**, representing images of empty seats. These images were collected separately to ensure diversity and help the model avoid false positives.

**4.1.1 NTHU Drowsy Driver Detection Dataset (NTHU-DDD)**

The **NTHU Drowsy Driver Detection Dataset (NTHU-DDD)** [14] collected by the Computer Vision Laboratory at National Tsing Hua University, was used as the **primary dataset** in this project. This dataset is **not publicly available**, and access was granted after signing the Dataset License Agreement by the head of the department and obtaining approval.

The dataset includes **36 subjects (people)** of various ethnicities and genders, recorded under five driving scenarios: **BareFace (NoGlasses), Glasses, Sunglasses, Night-BareFace, and Night-Glasses**. Videos capture a range of driver behaviors, including yawning, slow blinking, nodding, laughing, talking, and looking aside, with two main statuses: **drowsy and non-drowsy**.

All videos were recorded using **active infrared (IR) illumination** in AVI format at 640x480 pixels. Night scenarios were recorded at **15 fps**, and other scenarios at **30 fps**. The dataset was split into **training and testing sets**, with 18 subjects (multiple videos per scenario) for training and the remaining 18 subjects (90 videos in total) for testing, covering a mixture of drowsy and non-drowsy behaviors.

This dataset was chosen as the primary source due to its **comprehensive coverage of drowsiness-related behaviors, lighting conditions, and facial characteristics**. Sample images are shown in figure 8 .



Figure 8: Sample images from the NTHU Dataset

**4.1.2 Yawning Detection Dataset (YawDD)**

The YawDD dataset [15] includes two sub-datasets: one with normal facial expressions and another with drivers yawning. Participants are diverse in age, gender, and glasses usage. This dataset supplemented NTHU-DDD by providing additional yawning examples.Sample images are shown in Figure 10 .

Figure 9: Sample images from the Yawning Dataset

**4.1.3 Closed Eyes in the Wild (CEW)**

A portion of the CEW dataset [16], which contains images of eyes labeled as “closed” or “open,” was used, where only the closed-eye images were selected to support the detection of eye closure in our model.



Figure 10: Sample closed-eye images from CEW dataset

**4.1.4 Driver Behavior Image Dataset**

The Driver Behavior Image Dataset [17] includes images capturing various driver behaviors, such as looking aside, talking, and other A portion of this dataset was used to enrich training and testing with diverse driving behaviors.

Figure 11: Sample images from Driver Behavior Image dataset

**4.1.5 Kaggle Datasets**

In addition to the previously mentioned datasets, supplementary data were collected from Kaggle to enhance variability in participants, conditions, and modalities. The following four Kaggle datasets were utilized:

**UTA Real-Life Drowsiness Dataset (videos) [18]:** Contains video sequences of drivers in real-life scenarios, providing natural variations in facial expressions and drowsiness behavior.

**Drowsiness Prediction Dataset[19]:** Includes labeled facial images for awake and drowsy states, supporting classification tasks in the system.

**NITYMED Dataset (videos)[20**]: Offers video data of drivers under different levels of alertness, enriching temporal features for drowsiness detection.

**Driver Inattention Detection Dataset (images)[21]:** Provides still images of drivers exhibiting attentive and inattentive states, helping improve robustness in distinguishing drowsiness cues.

****

Figure 12: Sample images from the Kaggle datasets

**4.1.6 Absent Class**

In addition to “Awake” and “Drowsy,” a third class, **Absent**, represents images where no driver is present (empty seats). This class helps the model distinguish between actual driver states and empty seats.These images are essential to avoid false positives and ensure accurate classification. They were captured with a mobile phone under different angles and lighting conditions (day and night), along with additional images collected from the web.



Figure 13: Absent class samples collected from web sources

Figure 14: Absent class samples captured using a mobile phone

## 4.2 Data Preparation

The data preparation process is a crucial step to ensure the quality and reliability of the driver drowsiness detection system. It includes selecting suitable subjects, creating metadata files, extracting and cleaning frames, labeling, preprocessing, and finally splitting the dataset into training and testing sets. These steps collectively ensure that the dataset is ready for effective model training and evaluation.

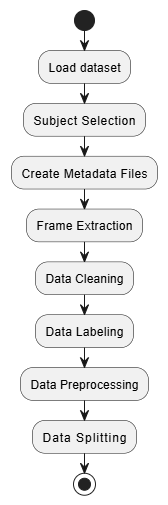


Figure 15: Data Preparation Flowchart

**4.2.1 Subject Selection**

The datasets were carefully examined to select suitable subjects. In the NTHU-DDD dataset, 21 subjects were selected out of the original 36, while 15 subjects were excluded due to unclear facial features, which could negatively impact the accuracy of the results.

In the NITYMED dataset, 10 subjects were selected out of 21 in order to maintain a balanced distribution of gender (male and female) across the dataset.

This process ensured that the final dataset was both high quality and balanced, making it more reliable for training and evaluation.

**4.2.2 Metadata Creation**

To ensure clear organization and management of the data, metadata files in CSV format were created for the videos in the NTHU-DDD dataset and for those recorded using a mobile phone. Each metadata file included the following information: video name, total number of frames, frame rate (frames per second), video duration, and video dimensions.

This step helped in understanding and organizing the dataset, making frame extraction easier.  
**4.2.3 Frame Extraction**

Videos from all datasets were split into individual frames so that they could be analyzed and used for model training and testing.

We manually reviewed several videos in the NTHU-DDD dataset and observed that the drivers’ movements were generally slow and did not change significantly from one frame to the next. Therefore, extracting every frame was unnecessary, and sampling fewer frames was sufficient to capture all meaningful variations while avoiding redundant, nearly identical frames.  
Based on this observation, frame sampling in the training set was adjusted according to each video’s frame rate: videos recorded at 30 fps were sampled by taking one frame every 30 frames, while 15 fps videos were sampled every 15 frames. This approach ensured that approximately one frame per second was extracted, reducing repetition while maintaining enough diversity for effective training.

For the testing set, all videos were 30 fps and generally longer than those in the training set. To control the number of extracted frames while still preserving essential content, one frame was taken every 60 frames (approximately one frame every two seconds)

In the dataset recorded using a mobile phone (absent), one frame was taken every 3 frames. This kept enough variety in the short videos while avoiding too many repeated frames.

For the NITYMED dataset, videos longer than 30 seconds had one frame taken every 25 frames to reduce repeated frames while keeping enough variation, while videos shorter than 30 seconds kept all frames to preserve all the information.

This gave a balanced and representative set of frames for all datasets while reducing repeated frames.  
**4.2.4 Data Cleaning**

After extracting frames from videos, all frames were carefully checked to ensure they were suitable for training and testing. Frames that were blurry, noisy, or otherwise low quality were removed.

For frames that were already provided as separate images from other datasets, each frame was manually reviewed, and any low-quality frames were removed.

This data cleaning process ensured that the final dataset was high-quality, consistent, and ready for model training and evaluation, reducing errors caused by poor-quality frames.

**4.2.5 Data Labeling**

The frames from the NTHU-DDD dataset were already labeled, but all frames were carefully reviewed manually after noticing some errors in the original labels. This ensured that the labels were accurate and reliable. Frames from the other datasets were also manually labeled and verified to make sure they were correct. This careful labeling process made sure that the dataset was fully ready for training, with all frames correctly classified into the target classes: Awake, Drowsy, and Absent.

**4.2.6 Data Preprocessing**

After cleaning and labeling, frames from all datasets were merged into a single dataset and converted to grayscale, simplifying the data while keeping all important facial movements, including eyes, mouth, and head, so the model could focus on them without being affected by color differences or other conditions.

This preprocessing ensured that all frames were consistent and ready for training and testing.

**4.2.7 Data Splitting**

After preprocessing, the dataset was divided into training and testing sets. Approximately 75% of the frames were used for training, while the remaining 25% were used for testing. The testing set included different subjects from those in the training set.

This ensured that the model was evaluated on unseen data, providing a reliable measure of its performance.

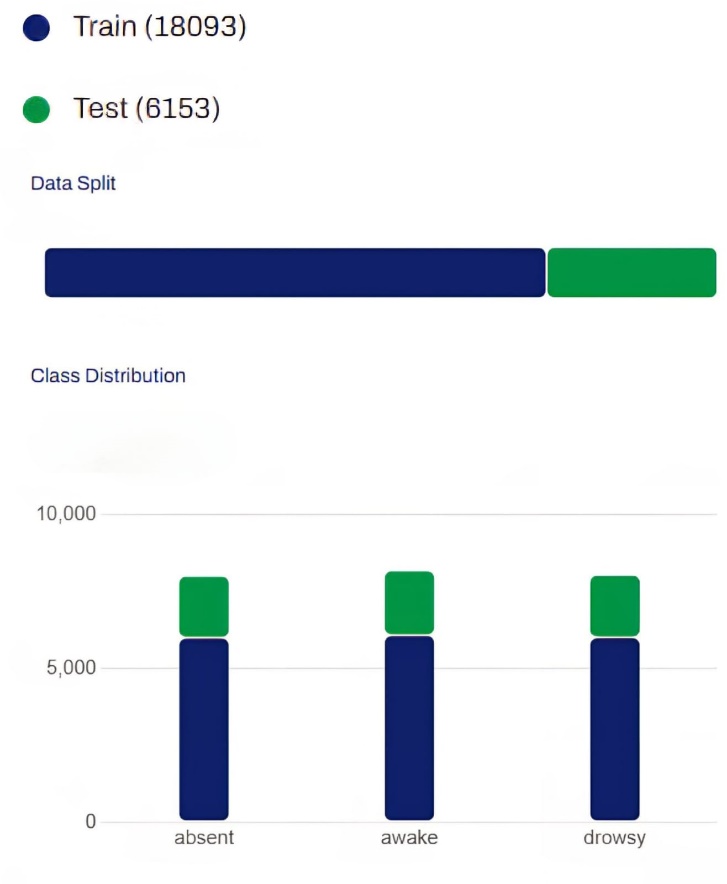


Figure 16: Dataset Split

**Final Dataset Characteristics**

After merging and cleaning all data sources, a unified and well-structured dataset was created to ensure consistency and representativeness. This section describes the main characteristics of the final dataset used for training the YOLOv11 model.

**Gender Balance (Male vs. Female)**

The dataset maintained a nearly balanced distribution between male and female subjects, with 7,997 male and 8,236 female samples across all classes. This balance was intentionally preserved to prevent gender bias during the model’s training process and to ensure fair performance across different driver groups. Such a balance contributes to more reliable and unbiased detection of driver drowsiness and inattention.

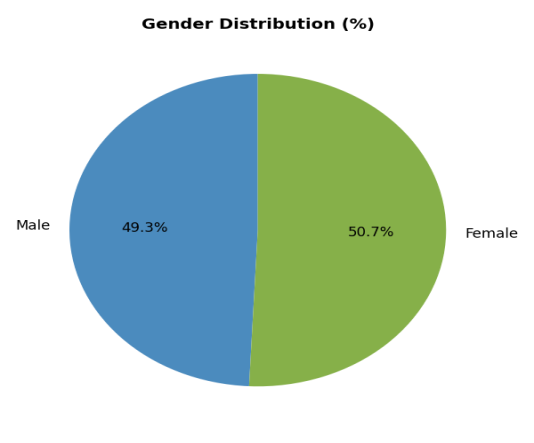


Figure 17: Male vs. Female Ratio

**Class Distribution**

The dataset showed a well-balanced distribution across the three driver states.

Awake: 8188 images

Drowsy: 8045 images

Absent: 8013 images

This near equal class distribution allows the YOLOv11 model to learn distinctive visual features for each driver state effectively.

A detailed breakdown by gender and data split shows that:

Table 2: Dataset Distribution by Class, Gender, and Data Split

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Data Split | Male | Female | Total |
| Awake | Training | 2996 | 3086 | 6082 |
|  | Testing | 951 | 1155 | 2106 |
| Drowsy | Training | 3004 | 3007 | 6011 |
|  | Testing | 1046 | 988 | 2034 |
| Absent | Training | ------ | ------ | 6000 |
|  | Testing | ------ | ------ | 2013 |
| Total | ------ | 7997 | 8236 | 24246 |

The Absent class has no gender labels. The table presents gender balance for Awake and Drowsy classes and total counts for all splits.

## 4.3 Training process

The YOLOv11 model was trained using Google Colab Pro. Google Colab, or Colaboratory, is an online platform where you can write and run Python code directly in the browser. It provides access to GPUs, which is extremely helpful for training deep learning models. Colab also allows easy sharing of notebooks and collaboration with others, enabling real-time comments and edits. After training, the YOLOv11 model is able to classify the driver's state into three categories: Awake, Drowsy, or Absent, providing accurate predictions on input images or video frames.

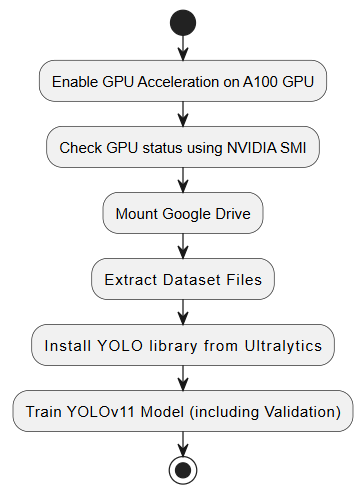
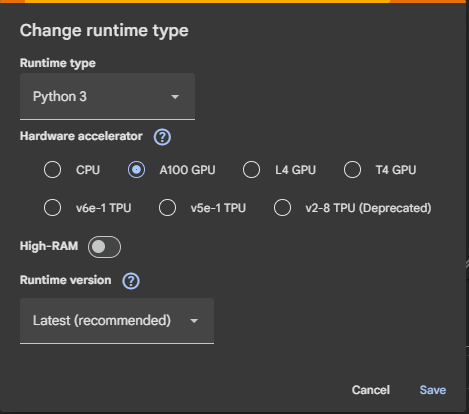


Figure 18: Training Process Flowchart

**1.Enabling GPU Acceleration:**To efficiently train YOLOv11 on a large dataset with multiple epochs, GPU acceleration was enabled using the NVIDIA A100, significantly reducing training time and ensuring smooth processing of high-resolution images.

**2. Check GPU Status using NVIDIA SMI:** After enabling GPU acceleration, the NVIDIA System Management Interface (NVIDIA SMI) was used to verify the GPU status. This command-line tool provides detailed information about the GPU, including memory usage, temperature, and active processes, ensuring that the A100 GPU is properly recognized and ready for training the YOLOv11 model.

!nvidia-smi

**3. Mount Google Drive:** Google Drive was mounted in the Colab environment to directly access the dataset and to store the training results, including model weights and logs. This approach allowed seamless data access and ensured that all outputs were safely saved for later use without relying on temporary Colab storage.

from google.colab import drive

drive.mount('/content/drive')

**4. Extract Dataset Files:** The dataset was extracted and organized into a clear hierarchical folder structure. The training set contains three subfolders corresponding to the classes: Awake, Drowsy, and Absent. Similarly, the testing set also contains the same three subfolders. This organization allows the YOLOv11 model to efficiently access and process images for both training and evaluation.

from zipfile import ZipFile

zip\_path = "/content/drive/MyDrive/drowsiness\_detection\_system\_dataset.zip"

extract\_path = "/content/dataset"

with ZipFile(zip\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_path)

**5. Install and Load YOLO Library from Ultralytics:** The Ultralytics YOLO library was installed in Google Colab (pip install ultralytics) to enable the use of YOLOv11. A pre-trained YOLOv11 model was loaded directly for classification, simplifying the training workflow and ensuring compatibility with the dataset.

#Ultralytics

!pip install ultralytics

# import YOLO

from ultralytics import YOLO

**6.Train YOLOv11 Model (including Validation):**A YOLOv11n classification model (yolo11n-cls.pt) was trained on a custom driver drowsiness dataset with three classes. The dataset included separate sets for training and testing. The model was trained for multiple epochs on a GPU with a batch size of 32 and an image size of 640×640. Throughout training, a consistent decrease in training loss and a noticeable improvement in testing accuracy were observed, indicating that the model was effectively learning the classification task. All training logs and results were stored in Google Drive for further analysis.

model = YOLO("yolo11n-cls.pt")

model.train(

    data="/content/dataset/drowsiness\_detection\_system\_dataset",

    epochs=100,

    batch=32,

    imgsz=640,

    device=0,

    project="/content/drive/MyDrive/yolo\_training",

    name="exp"

)

**Data Augmentation During Training**

During the training process, the YOLOv11 framework automatically applied data augmentation techniques to the input images to improve the model’s stability and ability to generalize. Data augmentation artificially increases the variety of the training dataset by creating modified versions of existing images through random transformations.

In this project, YOLOv11 applied augmentation automatically on each batch during training. These transformations included scaling, cropping, brightness adjustment, and color changes. Such operations help the model become less sensitive to changes in lighting, camera angle, and driver appearance.

By applying data augmentation during training, the YOLOv11 model became more robust and generalized better, allowing it to accurately recognize Awake, Drowsy, and Absent states under various lighting conditions and facial angles.

Figure 19: Examples of Data Augmentation Applied During Training

**Results of Training**

Upon completion of the training process, the YOLO model generates a comprehensive output directory containing several important files and folders that document the training results and model performance. These outputs include:

**1. Trained model weights** :

best.pt and last.pt

**2. Configuration file** :

The **(args.yaml)** file records all training parameters. Most parameters were kept at their default values, as they represent well-optimized settings commonly used in YOLO training, with only essential project-specific parameters being modified.

The main parameters in this configuration include:

* **task**: Defines the type of task for the model; here it is set to classify, meaning the model performs image classification.
* **mode**: Specifies the operation mode, which is train in this case.
* **model**: Indicates the pre-trained YOLOv11 model used as a starting point (yolo11n-cls.pt).
* **data**: The path to the dataset used for training and evaluation.
* **epochs**: The total number of training epochs (100 in this project), representing full passes over the training data.
* **batch**: Batch size (32), controlling how many images are processed before updating model weights.
* **imgsz**: Image size (640×640 pixels) for input images during training.
* **save**: Enables saving the trained model and associated outputs.
* **device**: Specifies the hardware used for training. '0' refers to GPU.
* **project**: The directory path where training outputs, including weights and logs, are saved.
* **name**: Name of the 'exp', used to organize results in subfolders.
* **optimizer**: Optimization algorithm for updating model weights, auto allows YOLO to select the best optimizer.
* **plots**: Enables generation of plots for training loss and accuracy.
* **lr0**: Initial learning rate for weight updates.
* **auto\_augment**: Activates automatic data augmentation (randaugment) to improve generalization and reduce overfitting.

**3. Confusion Matrix :**

The confusion matrix provides a comprehensive evaluation of the model's classification performance by comparing predicted labels against actual ground truth.



Figure 20: : Confusion matrix

**Analysis:**

**1- Absent Predictions:**

* Correctly predicted as Absent: 2013 instances
* Incorrectly predicted as Awake: 0 instances
* Incorrectly predicted as Drowsy: 0 instances

**2- Awake Predictions:**

* Correctly predicted as Awake: 2082 instances
* Incorrectly predicted as Absent: 0 instances
* Incorrectly predicted as Drowsy: 34 instances

**3- Drowsy Predictions:**

* Correctly predicted as Drowsy: 2000 instances
* Incorrectly predicted as Absent: 0 instances
* Incorrectly predicted as Awake: 24 instances

**Interpretation**

* The model achieves **high accuracy** across all classes.
* The **Absent** class is predicted **perfectly**.
* **Awake** and **Drowsy** are also classified correctly in most cases, indicating strong overall performance.

**4. Normalized Confusion Matrix :**

The normalized confusion matrix expresses each prediction as a percentage of the true class total, providing clearer insight into relative model performance.

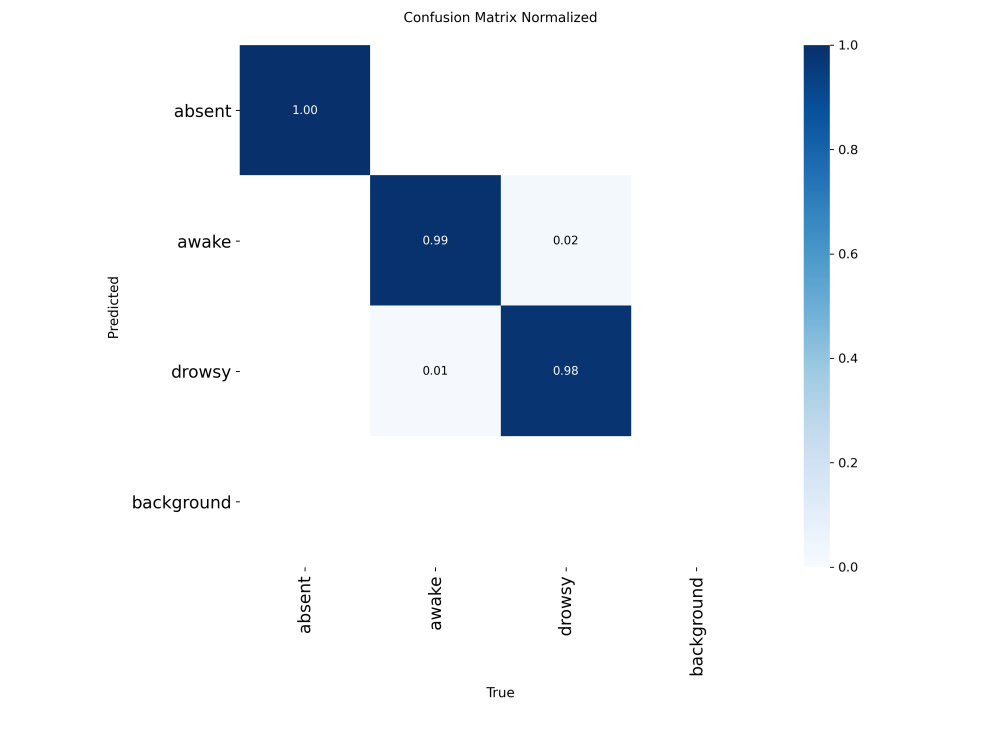


Figure 21: : Normalized confusion matrix

**Analysis:**

* Absent: Perfectly classified (100%), with no confusion with other classes.
* Awake: Mostly correct )99%( correctly predicted, with a slight misclassification as Drowsy.
* Drowsy: Mostly correct )98%( correctly predicted, with a small misclassification as Awake.

**5. Training Curves :**

The training process generated four key performance curves that illustrate the model's learning progression over 100 epochs:

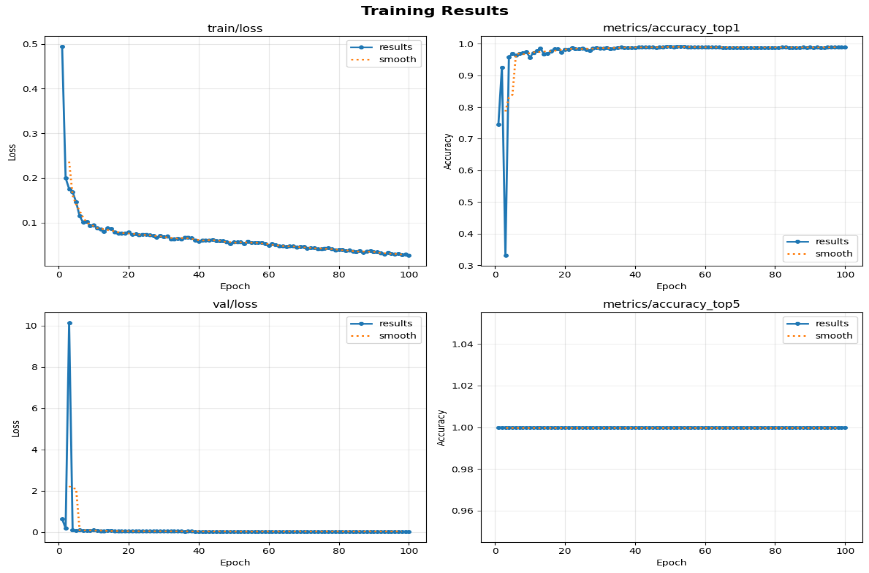


Figure 22: Training and Validation Loss, Top-1 Accuracy, and Top-5 Accuracy Curves

**Interpretation**:

**Train Loss:** Shows how the training loss decreases over epochs, indicating that the model is learning to fit the training data.

**Validation Loss (Val Loss):** Reflects the model's performance on unseen validation data. A decreasing validation loss suggests good generalization.

**Training Accuracy (Metrics/Accuracy\_top1):** Measures the percentage of correctly predicted instances in the training set. Increasing accuracy indicates that the model is learning effectively, with the curve beginning to stabilize around epoch 40, after which further improvements become minimal and the accuracy remains consistently high.

**Top-5 Accuracy (Metrics/Accuracy\_top5):** Since the dataset contains only 3 classes, Top-5 accuracy is always 1 (100%), meaning the true class is always within the top 5 predictions.

**Calculation of Performance Metrics**

To evaluate the model's performance, four key metrics were calculated based on the confusion matrix results.

**1. Accuracy**

**Formula:** Accuracy = (Total Correct Predictions) / (Total Predictions)

**Calculation:** (2013 + 2082 + 2000) / (2013 + 2116 + 2024) = 6095 / 6153 = 0.9906 = 99.06%

**2. Precision (Macro Average)**

**Formula:** Precision = TP / (TP + FP), Macro Precision = Average of all classes

**Calculation:**

* Absent: 2013 / 2013 = 1.0000
* Awake: 2082 / 2106 = 0.9886
* Drowsy: 2000 / 2034 = 0.9833

Macro Precision = (1.0000 + 0.9886 + 0.9833) / 3 = 0.9906 = 99.06%

**3. Recall (Macro Average)**

**Formula:** Recall = TP / (TP + FN), Macro Recall = Average of all classes

**Calculation:**

* Absent: 2013 / 2013 = 1.0000
* Awake: 2082 / 2116 = 0.9839
* Drowsy: 2000 / 2024 = 0.9881

Macro Recall = (1.0000 + 0.9839 + 0.9881) / 3 = 0.9907 = 99.07%

**4. F1-Score (Macro Average)**

**Formula:** F1-Score = 2 × (Precision × Recall) / (Precision + Recall), Macro F1 = Average of all classes

**Calculation:**

* Absent: 2 × (1.0000 × 1.0000) / (1.0000 + 1.0000) = 1.0000
* Awake: 2 × (0.9886 × 0.9839) / (0.9886 + 0.9839) = 0.9861
* Drowsy: 2 × (0.9833 × 0.9881) / (0.9833 + 0.9881) = 0.9857

Macro F1-Score = (1.0000 + 0.9861 + 0.9857) / 3 = 0.9906 = 99.06%

**Summary**

Table 3: Model Performance Metrics

|  |  |
| --- | --- |
| Metric | Value |
| Accuracy | 99.06% |
| Precision | 99.06% |
| Recall | 99.07% |
| F1-Score | 99.06% |

The model achieved exceptional performance with all metrics exceeding 99%. These consistently high scores across accuracy, precision, recall, and F1-score demonstrate the model's reliability and balanced classification capability, confirming its readiness for real-time drowsiness detection applications.

**Conclusion from Training**

The training process of the YOLOv11 model for driver drowsiness detection showed high performance and stability. Over 100 epochs, the model successfully learned to recognize key visual cues, such as eye closure, yawning, and head position, which are critical indicators of driver states. Both training and validation losses decreased steadily and converged, indicating effective learning without signs of overfitting.

Evaluation using the confusion matrix showed that the **Absent** class was perfectly classified, while the **Awake** and **Drowsy** classes achieved accuracy above 98%, with only slight confusion between them. These results confirm that the model generalized well to unseen data.

Automatic data augmentation further enhanced the model's robustness, enabling it to handle variations in lighting, facial orientation, and expressions. The best-performing weights were saved as best.pt, representing the epoch with the highest validation accuracy, and are ready for deployment in real-time detection.

Overall, the training process proved that the YOLOv11 model is accurate, stable, and capable of real-time driver drowsiness detection, making it suitable for practical applications.

## 4.4 Software Deployment

**4.4.1 Introduction**

Before deploying the system on the embedded hardware, the trained model was first tested on a laptop to ensure it performs correctly in real-time. A complete desktop application was developed using PyQt5. The interface was designed using an AI-powered design tool to produce a clean, modern, and user-friendly layout, enabling evaluation of the system’s behavior with a live camera feed, measurement of processing speed, and verification of the alarm mechanism. This step allowed us to confirm the reliability of the model and the overall workflow in a controlled environment.

**4.4.2 Source code**

This section presents the source code of the desktop application in organized segments, followed by detailed explanations describing the purpose and functionality of each part.

**Imports and Dependencies:**

import sys

import os

import cv2

import time

from pygame import mixer

from PyQt5.QtWidgets import (

    QApplication, QWidget, QLabel, QVBoxLayout, QPushButton,

    QHBoxLayout, QFrame, QSizePolicy, QSpacerItem, QMessageBox, QToolButton

)

from PyQt5.QtCore import QThread, pyqtSignal, Qt, QSize

from PyQt5.QtGui import QImage, QPixmap, QFont, QIcon

from ultralytics import YOLO

**sys:** Provides access to system-level variables and runtime environment functions.

**os:** Offers utilities for file and directory management across operating systems.

**cv2:** OpenCV library used for general computer vision operations and image handling.

**time:** Standard Python module for working with time-related functions.

**pygame.mixer:** Module from the Pygame library used for audio playback and sound management.

**PyQt5:** A comprehensive framework for building graphical user interfaces in Python. It provides widgets, layout managers, event handling, image rendering tools, and multithreading support.

**Ultralytics:** Used to load and run YOLO-based deep learning models for image classification.

**Resource Path Handling:**

def resource\_path(*relative\_path*):

    """ Get the correct path for files whether running as exe or python script """

    try:

*# If running as exe*

        base\_path = sys.\_MEIPASS

    except Exception:

        base\_path = os.path.abspath(".")

    return os.path.join(base\_path, *relative\_path*)

A helper function that helps the program find files (like images and sounds) whether it's running as a regular Python script or as a compiled .exe application.

**Audio System Initialization:**

Initializes the pygame mixer to enable the application to play alarm sounds when drowsiness is detected.

mixer.init()

**Video Processing Thread (QThread):**

class VideoThread(QThread):

*# Convert image*

    change\_pixmap\_signal = pyqtSignal(QImage)

    status\_signal = pyqtSignal(str, int, bool)  *# label\_name, drowsy\_counter, alarm\_active*

    def \_\_init\_\_(*self*, *model*, *labels\_dict*, *window\_size*=30, *alert\_cooldown*=1):

        super().\_\_init\_\_()

*self*.model = *model*

*self*.labels\_dict = *labels\_dict*

*self*.window\_size = *window\_size*

*self*.alert\_cooldown = *alert\_cooldown*

*self*.running = False

*self*.cap = None

*self*.drowsy\_counter = 0

*self*.alert\_triggered = False

*self*.last\_alert\_time = 0

*# Load alarm sound file*

        try:

            mixer.music.load(resource\_path("alarm.mp3"))

        except Exception as e:

            print(f"Error loading alarm.mp3: {e}")

    def run(*self*):

*self*.cap = cv2.VideoCapture(0)

*self*.running = True

        while *self*.running:

            ret, frame = *self*.cap.read()

            if not ret:

                continue

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

            input\_frame = cv2.cvtColor(gray, cv2.COLOR\_GRAY2BGR)

            results = *self*.model.predict(input\_frame, *verbose*=False)

            label\_index = int(results[0].probs.top1)

            label\_name = *self*.labels\_dict.get(label\_index, "unknown")

*# Variable to track alarm state*

            alarm\_is\_active = False

            if label\_name == "drowsy":

*self*.drowsy\_counter += 1

            else:

*self*.drowsy\_counter = 0

*self*.alert\_triggered = False

                try:

                    mixer.music.stop()

                except:

                    pass

*# Activate alarm only when threshold is exceeded*

            if *self*.drowsy\_counter >= *self*.window\_size and not *self*.alert\_triggered:

                current\_time = time.time()

                if current\_time - *self*.last\_alert\_time > *self*.alert\_cooldown:

                    try:

                        mixer.music.play(-1)

                    except Exception as e:

                        print(f"Error playing alarm: {e}")

*self*.last\_alert\_time = current\_time

*self*.alert\_triggered = True

*# Determine if alarm is currently active*

            if *self*.drowsy\_counter >= *self*.window\_size:

                alarm\_is\_active = True

*# Show the frame on the screen to user*

            frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

            h, w, ch = frame\_rgb.shape

            bytes\_per\_line = ch \* w

            qimg = QImage(frame\_rgb.data, w, h, bytes\_per\_line, QImage.Format\_RGB888)

*self*.change\_pixmap\_signal.emit(qimg)

*# Send alarm state with data*

*self*.status\_signal.emit(label\_name, *self*.drowsy\_counter, alarm\_is\_active)

        if *self*.cap is not None:

*self*.cap.release()

**1- Class Definition:**

VideoThread is a subclass of QThread and is responsible for handling video capture and model inference in a separate execution thread. This prevents the GUI from freezing during continuous frame processing.

**2- Signals:**

The class defines two PyQt signals:

change\_pixmap\_signal: sends the processed video frame (QImage) to the GUI for display.

status\_signal: sends the predicted label, the drowsiness counter, and alarm status to update the on-screen indicators.

**3- Initialization:**

During initialization, the thread receives the trained YOLO model, a dictionary mapping numeric class values to text labels, the window size that determines how many consecutive drowsy frames are required to trigger the alarm, and the alert cooldown duration. The constructor also loads the alarm audio file using mixer.music.load(...) and initializes internal state variables such as drowsy\_counter, alert\_triggered, and last\_alert\_time, which are used to control the alert behavior throughout execution.

**4- run Method (Main Processing Loop(:**

The run() method contains the continuous processing loop executed while the thread is active. Within this loop, the webcam is initialized, frames are captured in real time, and each frame is converted to grayscale and then expanded back to a 3-channel image to match the YOLO model requirements. The model predicts the driver’s state for every frame, after which the system updates the drowsiness counter and checks whether the alert threshold is reached. If the alarm condition is met, an audio warning is triggered. The processed frame is then converted to a QImage and emitted to the GUI, while the detection results and alarm status are sent through the status\_signal for interface updates.

**5- Alarm Handling:**

An internal logic activates the alarm only when a predefined number of consecutive frames are classified as drowsy, ensuring reliable alert generation and preventing false alarms caused by isolated misclassified frames.

**6- Thread Shutdown:**

When the thread stops, the webcam resource is safely released to prevent device locking or runtime errors.

**GUI Construction (Main Window Initialization):**

class DrowsinessApp(QWidget):

    def \_\_init\_\_(*self*):

        super().\_\_init\_\_()

*# Title*

*self*.setWindowTitle("Drowsiness Detection System")

*# Window size*

*self*.setMinimumSize(950, 900)

*# Load model*

*self*.model = YOLO(resource\_path("best.pt"))

*self*.labels\_dict = {0: 'absent', 1: 'awake', 2: 'drowsy'}

*self*.create\_ui()

*# Create thread at the beginning*

*self*.thread = VideoThread(*self*.model, *self*.labels\_dict, *window\_size*=30, *alert\_cooldown*=1)

*self*.thread.change\_pixmap\_signal.connect(*self*.update\_image)

*self*.thread.status\_signal.connect(*self*.update\_status)

Sets up the main window, loads the YOLO model, defines the labels (absent, awake, drowsy), and connects the video thread to the GUI for real-time updates.

**UI Layout and Components:**

Creates all the visual elements: header with logo, title, and info button, video display area, status indicators (LED and text), control buttons (Start/Stop), and footer. Everything is organized in a clean, modern layout.

    def create\_ui(*self*):

*# ------------ Header title ------------*

        title = QLabel("Drowsiness Detection System")

        title.setAlignment(Qt.AlignCenter)

        title.setFont(QFont("Arial", 25, QFont.Bold))

        title.setObjectName("title\_label")

*# ------------ Logo ----------------*

        logo = QLabel()

        logo.setObjectName("logo\_label")

        pixmap = QPixmap(resource\_path("images/logo.jpeg"))

        pixmap = pixmap.scaled(100, 100, Qt.KeepAspectRatio, Qt.SmoothTransformation)

        logo.setPixmap(pixmap)

        logo.setAlignment(Qt.AlignCenter)

*# ------------ Info Button ------------*

        info\_btn = QToolButton()

        info\_btn.setIcon(QIcon(resource\_path("images/info\_icon.png")))

        info\_btn.setIconSize(QSize(40, 40))

        info\_btn.setToolTip("About Drowsiness Detection System")

        info\_btn.setStyleSheet("""

            QToolButton {

                background: transparent;

                border: none;

                color: white;

                font-size: 18px;

            }

            QToolButton:hover {

                color: #a8c8ff;

            }

        """)

        info\_btn.clicked.connect(*self*.show\_info\_dialog)

*# ------------ Header Layout ------------*

        header\_frame = QFrame()

        header\_frame.setStyleSheet("""

            QFrame {

                background:#0e5f8a;

                border-radius: 8px;

                padding: 12px;

            }

            QLabel#title\_label {

                color: white;

            }

        """)

        header\_layout = QHBoxLayout()

        header\_layout.addWidget(logo)

        header\_layout.addSpacing(10)

        header\_layout.addWidget(title)

        header\_layout.addStretch()

        header\_layout.addWidget(info\_btn)

        header\_frame.setLayout(header\_layout)

*# ------------ Video area ------------*

*self*.video\_label = QLabel()

*self*.video\_label.setStyleSheet("background: #2b3240; border-radius: 8px;")

*self*.video\_label.setAlignment(Qt.AlignCenter)

*self*.video\_label.setText("Click Start to begin detection")

*self*.video\_label.setStyleSheet("color: #9fb3c8; background:#1f2a36; border-radius:10px;")

*self*.video\_label.setMinimumSize(720, 420)

*self*.video\_label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

*# ------------ Status bar under video ------------*

*# Driver state*

        status\_label = QLabel("Current Status:")

        status\_label.setFont(QFont("Arial", 10))

*self*.status\_dot = QLabel()

*self*.status\_dot.setFixedSize(16, 16)

*self*.status\_dot.setStyleSheet("background: #808080; border-radius:8px;")

*self*.status\_text = QLabel("Inactive")

*self*.status\_text.setFont(QFont("Arial", 11))

*self*.status\_text.setStyleSheet("color: #0078d7; font-weight: bold;")

*# Camera state*

        detection\_label = QLabel("Detection Mode:")

        detection\_label.setFont(QFont("Arial", 10))

*self*.mode\_text = QLabel("Inactive")

*self*.mode\_text.setFont(QFont("Arial", 11))

*self*.mode\_text.setStyleSheet("color: #0078d7; font-weight: bold;")

*# Put together in layout*

        status\_row = QHBoxLayout()

        status\_row.addWidget(status\_label)

        status\_row.addWidget(*self*.status\_dot)

        status\_row.addSpacing(8)

        status\_row.addWidget(*self*.status\_text)

        status\_row.addStretch()

        status\_row.addWidget(detection\_label)

        status\_row.addWidget(*self*.mode\_text)

        status\_frame = QFrame()

        status\_frame.setStyleSheet("background: #f4f7f9; border-radius:8px; padding:12px;")

        status\_frame.setLayout(status\_row)

*# ------------ Buttons ------------*

*self*.start\_btn = QPushButton("▶ Start Detection")

*self*.stop\_btn = QPushButton("◼ Stop Detection")

*self*.start\_btn.setStyleSheet("QPushButton{background:#16a3ab; color:white; padding:10px 22px; border-radius:8px;} QPushButton:hover{background:#13b3bb;}")

*self*.stop\_btn.setStyleSheet("QPushButton{background:#bdbdbd; color:#444; padding:10px 22px; border-radius:8px;} QPushButton:hover{background:#d32f2f; color:white;}")

*self*.stop\_btn.setEnabled(False)

*# Actions*

*self*.start\_btn.clicked.connect(*self*.on\_start)

*self*.stop\_btn.clicked.connect(*self*.on\_stop)

*# Layout*

        btn\_layout = QHBoxLayout()

        btn\_layout.addSpacerItem(QSpacerItem(40, 20))

        btn\_layout.addWidget(*self*.start\_btn)

        btn\_layout.addSpacing(12)

        btn\_layout.addWidget(*self*.stop\_btn)

        btn\_layout.addSpacerItem(QSpacerItem(40, 20))

*# ------------ Footer ------------*

        footer = QLabel("Real-time drowsiness monitoring system • Built with computer vision ")

        footer.setStyleSheet("color: #6b7a85;")

        footer.setAlignment(Qt.AlignCenter)

*# ------------ Main layout (contains everything) ------------*

        main\_layout = QVBoxLayout()

        main\_layout.addWidget(header\_frame)

        main\_layout.addSpacing(12)

        main\_layout.addWidget(*self*.video\_label, *stretch*=1)

        main\_layout.addSpacing(12)

        main\_layout.addWidget(status\_frame)

        main\_layout.addSpacing(12)

        main\_layout.addLayout(btn\_layout)

        main\_layout.addSpacing(8)

        main\_layout.addWidget(footer)

        main\_layout.setContentsMargins(12, 12, 12, 12)

*self*.setLayout(main\_layout)

**Show Information Dialog:**

   def show\_info\_dialog(*self*):

        info = QMessageBox(*self*)

        info.setWindowTitle("About Drowsiness Detection System")

        info.setIcon(QMessageBox.Information)

        info.setTextFormat(Qt.RichText)

        info.setText("""

        <h3>Drowsiness Detection System</h3>

        <p>An app designed to keep drivers safe by monitoring alertness and delivering instant warnings when signs of drowsiness are detected.</p><br>

        <b>Purpose:</b>

        <p>To support safer driving and reduce the risk of accidents caused by fatigue.</p><br>

        <b>Model:</b>

        <p>YOLO Nano v11-class</p><br>

        <i>Drive Safe • Stay Alert • Save Lives</i>

        """)

        info.setStandardButtons(QMessageBox.Ok)

        info.exec\_()

Shows a pop-up window with information about the system.

**Start and Stop Control Logic:**

    def on\_start(*self*):

*self*.start\_btn.setEnabled(False)

*self*.stop\_btn.setEnabled(True)

*self*.mode\_text.setText("Active")

        if not *self*.thread.isRunning():

*self*.thread = VideoThread(*self*.model, *self*.labels\_dict, *window\_size*=30, *alert\_cooldown*=1)

*self*.thread.change\_pixmap\_signal.connect(*self*.update\_image)

*self*.thread.status\_signal.connect(*self*.update\_status)

*self*.thread.start()

    def on\_stop(*self*):

*self*.stop\_btn.setEnabled(False)

*self*.start\_btn.setEnabled(True)

*self*.mode\_text.setText("Inactive")

        if *self*.thread.isRunning():

*self*.thread.running = False

*self*.thread.wait()

        try:

*self*.thread.change\_pixmap\_signal.disconnect(*self*.update\_image)

*self*.thread.status\_signal.disconnect(*self*.update\_status)

        except TypeError:

            pass

        try:

            mixer.music.stop()

        except:

            pass

*self*.video\_label.clear()

*self*.video\_label.setText("Click Start to begin detection")

*self*.video\_label.setAlignment(Qt.AlignCenter)

*self*.video\_label.setStyleSheet("color: #9fb3c8; background:#1f2a36; border-radius:10px;")

*self*.status\_text.setText("Inactive")

*self*.status\_dot.setStyleSheet("background: #808080; border-radius:8px;")

*self*.status\_text.setStyleSheet("color: #0078d7; font-weight: bold;")

*self*.thread.drowsy\_counter = 0

*self*.thread.alert\_triggered = False

*self*.thread.running = False

**1- on\_start():**

* Update UI: Disables the "Start Detection" button and enables the "Stop Detection" button.
* Set System State: Updates the detection mode text (mode\_text) to "Active".
* Start Video Thread: Creates a new instance of VideoThread if it is not already running, passing the model, class dictionary, window size, and alert cooldown.
* Connects the thread signals: Links the thread’s signals to the UI update functions (update\_image and update\_status).
* Run Thread: Starts the video thread to begin real-time video processing and display.

**2- on\_stop():**

* Update UI: Disables the "Stop Detection" button and enables the "Start Detection" button.
* Set System State: Updates the detection mode text (mode\_text) to "Inactive".
* Stop Video Thread: Stops the thread if running, waits for it to finish, and disconnects its signals from the UI functions.
* Stop Alarm Sound: Stops any active alarm using mixer.music.stop().
* Reset Video and Status: Clears the video label, restores the initial message, resets text style, background, sets the status text to "Inactive" with the colored dot gray, and resets counters and flags.

**GUI Update Functions (Image + Status):**

    def update\_image(*self*, *qimg*):

        if not *self*.thread.running:

            return

        pix = QPixmap.fromImage(*qimg*)

        scaled = pix.scaled(*self*.video\_label.size(), Qt.KeepAspectRatio, Qt.SmoothTransformation)

*self*.video\_label.setPixmap(scaled)

    def update\_status(*self*, *label\_name*, *counter*, *alarm\_active*):

        """

        Status update logic:

        1. Green LED when face is present (awake or drowsy before threshold)

        2. Red LED only when absent

        3. "Drowsy" text turns red when alarm is triggered (after threshold)

        4. Status changes to Drowsy only when threshold is exceeded

        """

        if *label\_name* == "absent":

*# Red LED - Face not detected*

*self*.status\_text.setText("Face not detected")

*self*.status\_text.setStyleSheet("color: #e67e22; font-weight: bold;")

*self*.status\_dot.setStyleSheet("background: #e74c3c; border-radius:8px;")

        elif *label\_name* == "awake":

*# Green LED - Person is awake*

*self*.status\_text.setText("Awake")

*self*.status\_text.setStyleSheet("color: #2ecc71; font-weight: bold;")

*self*.status\_dot.setStyleSheet("background: #2ecc71; border-radius:8px;")

        elif *label\_name* == "drowsy":

*# Green LED - Face is present (before reaching threshold)*

*self*.status\_dot.setStyleSheet("background: #2ecc71; border-radius:8px;")

            if *alarm\_active*:

*# After threshold - Text turns red and status changes to Drowsy*

*self*.status\_text.setText(" DROWSY - ALERT")

*self*.status\_text.setStyleSheet("color: #e74c3c; font-weight: bold;")

            else:

*# Before threshold - Status remains Awake (but model detected drowsy)*

*self*.status\_text.setText("Awake")

*self*.status\_text.setStyleSheet("color: #2ecc71; font-weight: bold;")

Two functions that continuously update the GUI: one displays the camera feed on screen, and the other updates the status indicators (LED color and status text) based on detection results (awake/drowsy/absent) and alarm state.

**Application Entry Point:**

def main():

    app = QApplication(sys.argv)

    win = DrowsinessApp()

    win.show()

    sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

    main()

The main() function initializes the PyQt5 application, creates an instance of DrowsinessApp, displays the main window, and starts the application’s event loop. This section ensures that the GUI is launched correctly when the program is executed.

**Executable Conversion:**

To simplify deployment and allow the system to run on computers without requiring a Python environment, the desktop application was converted into a standalone executable (.exe) using PyInstaller. This process bundles the Python interpreter, all dependencies, resources (images, audio, and model files), and the application code into a single file. As a result, users can launch the system directly by double-clicking the executable, ensuring ease of use and consistent behavior across different machines.

**4.4.3 Output:**

This section presents sample outputs from the desktop application, including:

**1. Application Interface Before Starting Detection**

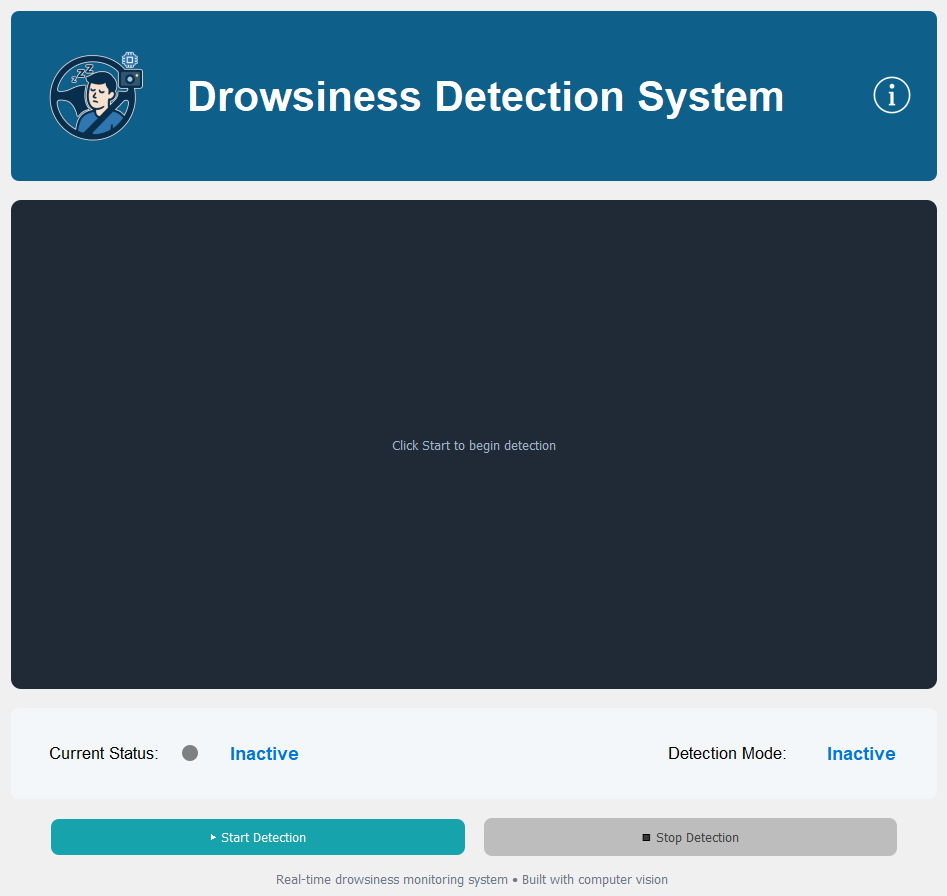
****

Figure 23

**2. Awake Detection**

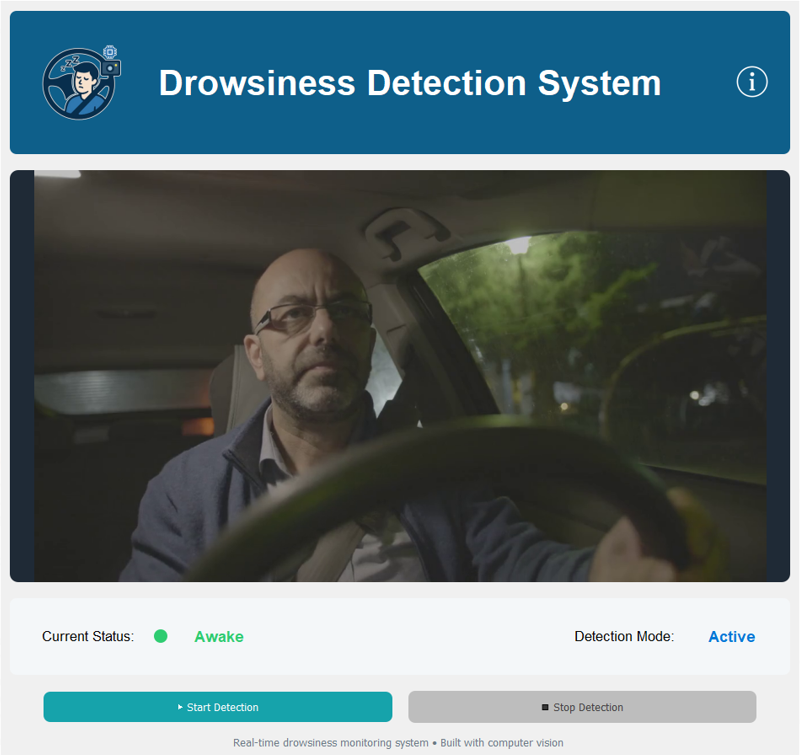
****

Figure 24

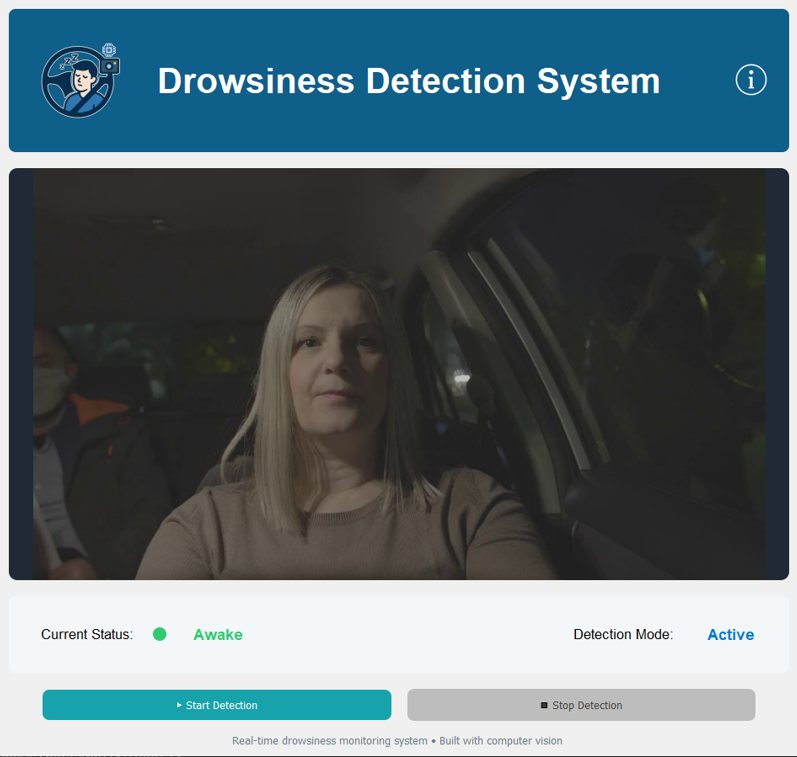
****

Figure 25

**3. Drowsy Detection**

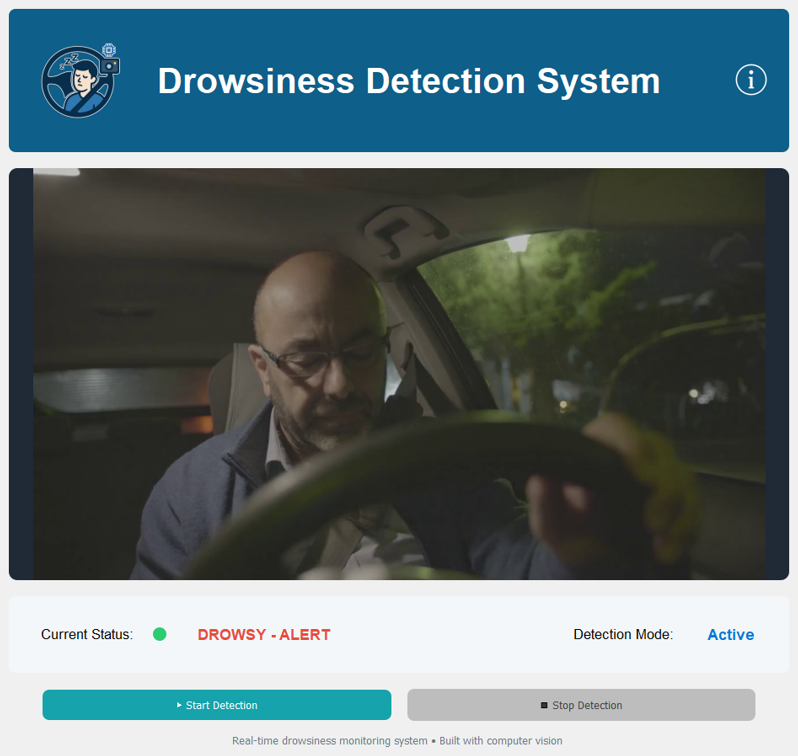
****

Figure 26

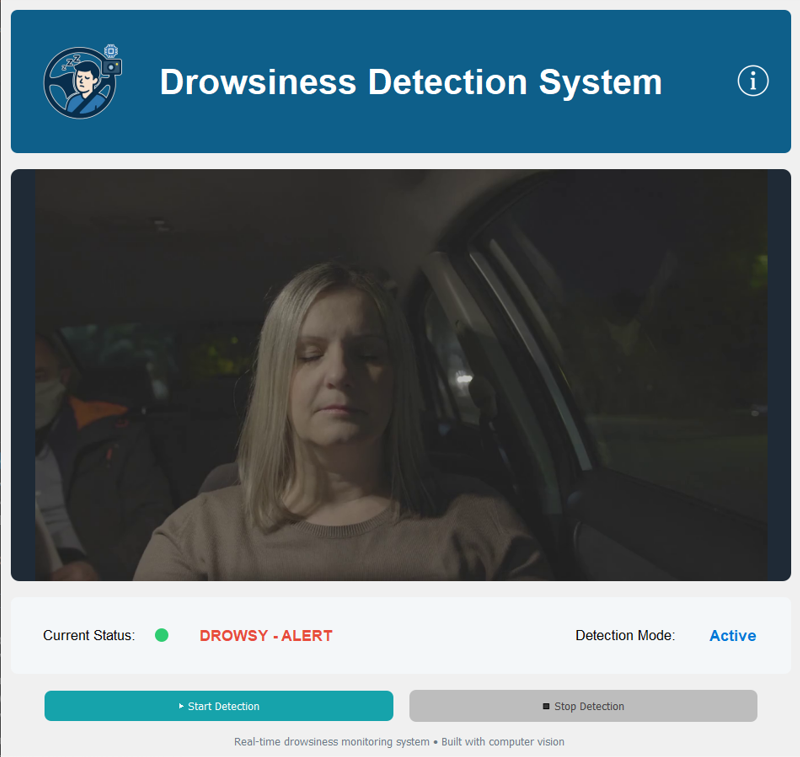
****

Figure 27

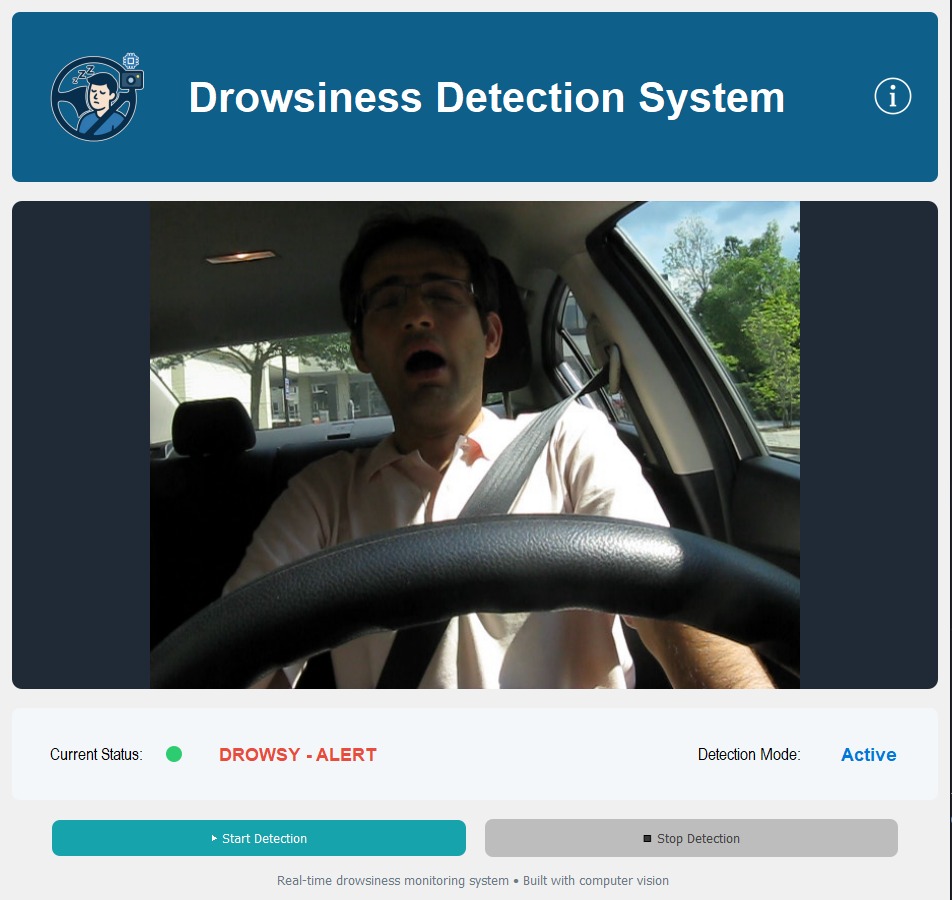


Figure 28

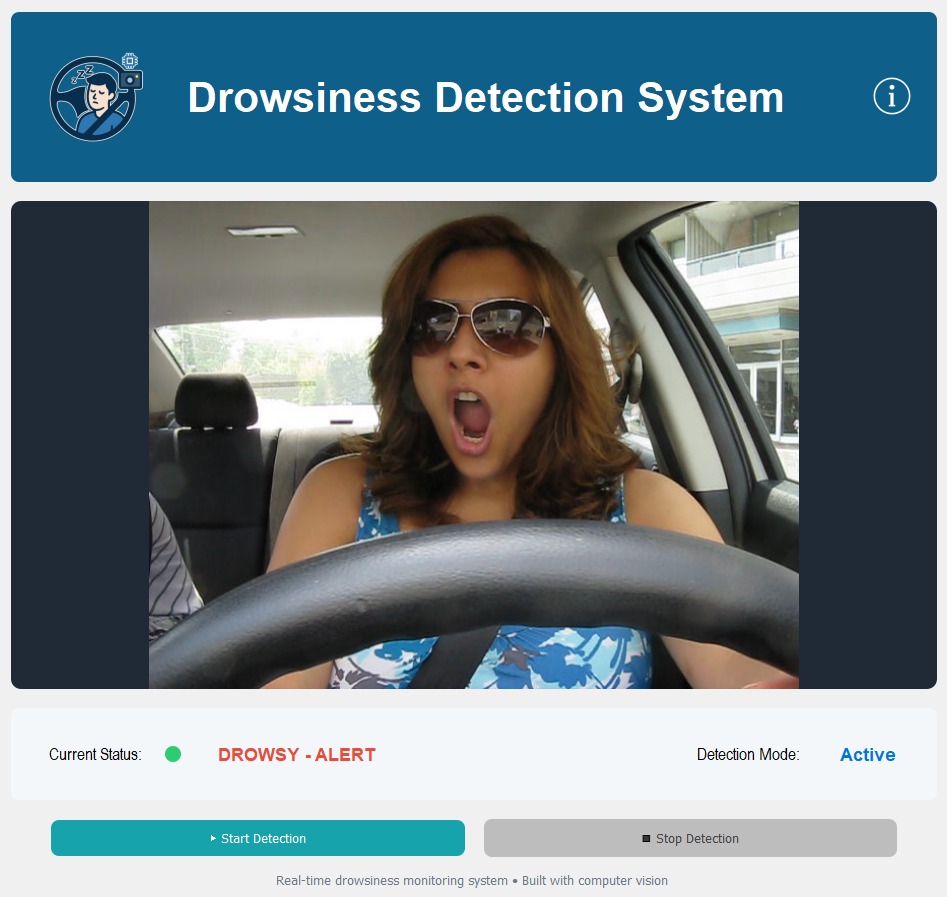


Figure 29

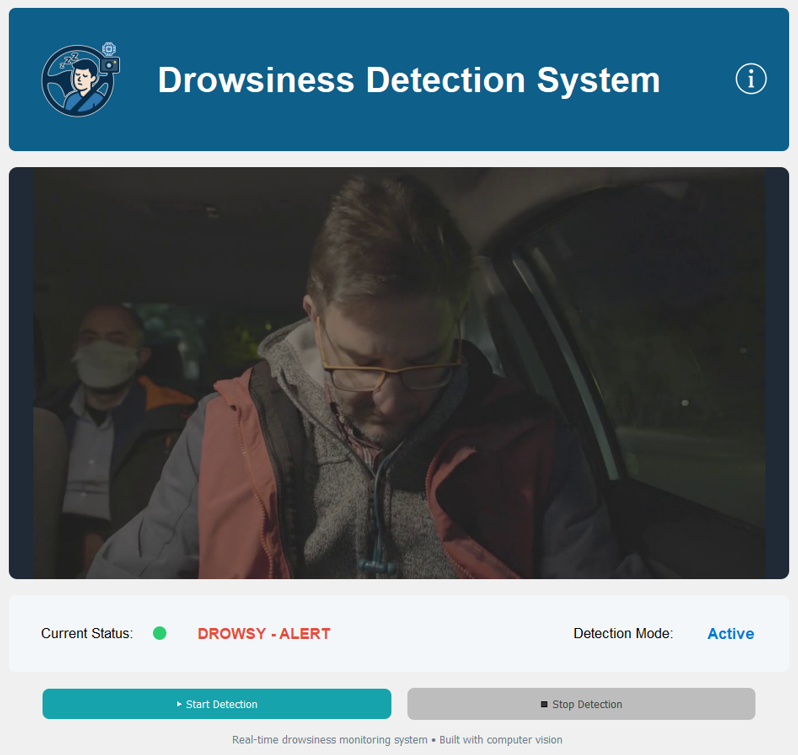


Figure 30

**3. Face Not Detected – Absent State**

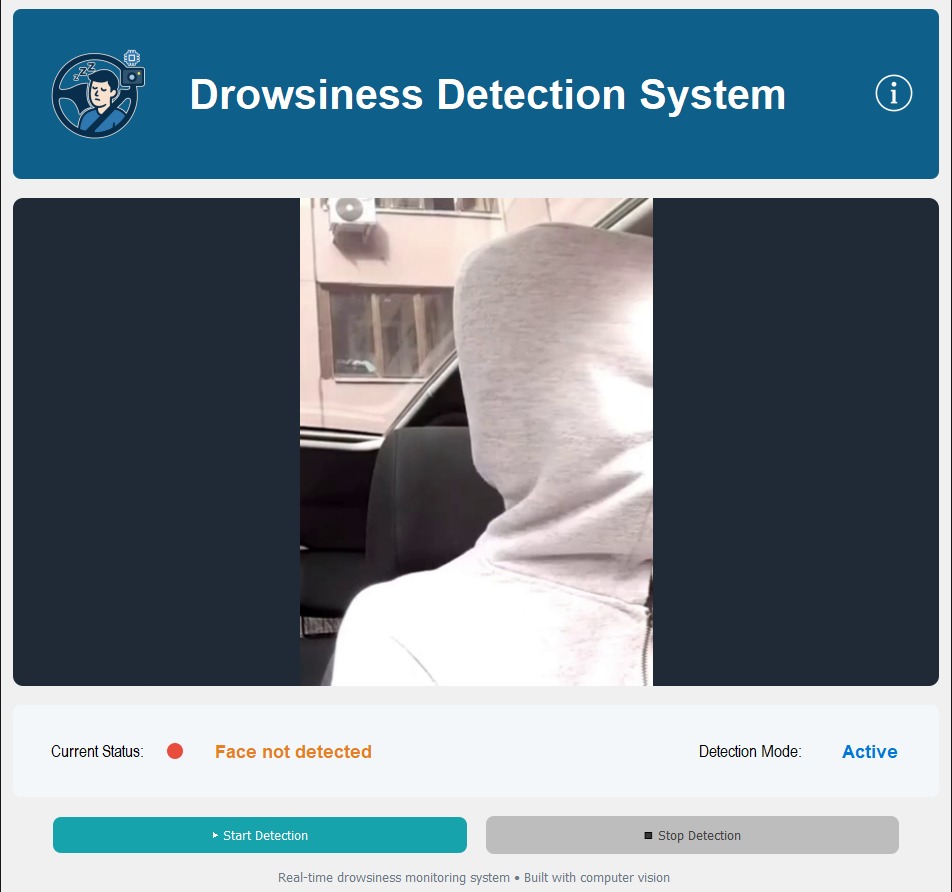
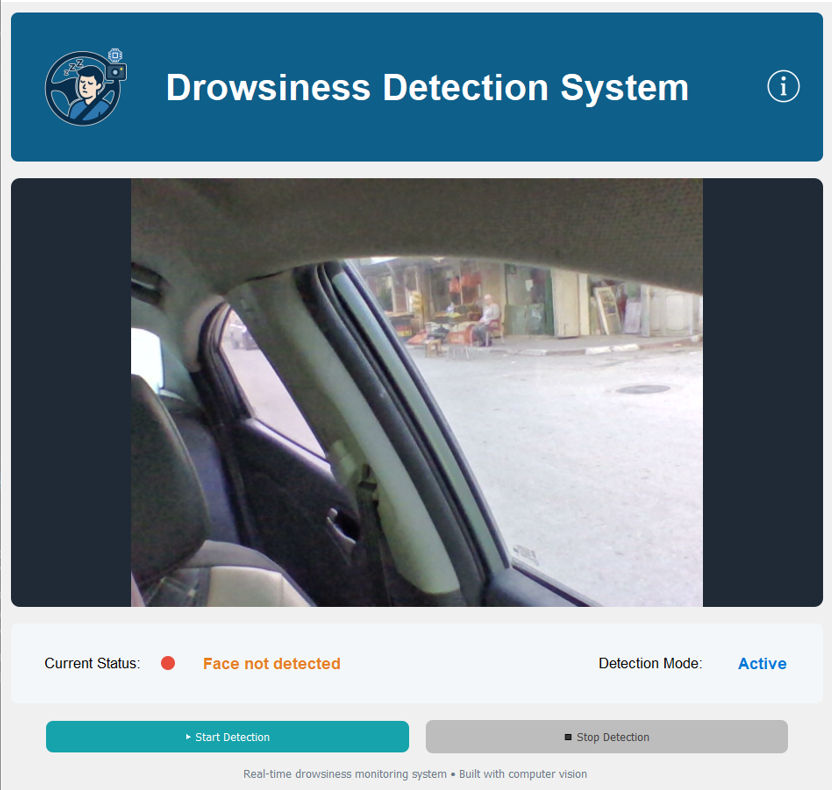
****

Figure 31



Figure

Figure 23 – Figure 32: Desktop Application Output from Our Drowsiness Detection Model

# Reference list

[1] World Health Organization. (2023, December). Road traffic injuries. World Health Organization. <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>

[2] Palestinian Central Bureau of Statistics. (2023, September 4). Casualties in road traffic accidents in Palestine by governorate and type of injury, 2022. <https://www.pcbs.gov.ps/statisticsIndicatorsTables.aspx?lang=en&table_id=2037>

[3] Amoadu, M., Ansah, E. W., & Sarfo, J. O. (2023). Psychosocial work factors, road traffic accidents and risky driving behaviours in low-and middle-income countries: a scoping review. IATSS research, 47(2), 240-250. <https://doi.org/10.1016/j.iatssr.2023.03.005>

[4] National Highway Traffic Safety Administration, “Drowsy Driving.” Available: <https://www.nhtsa.gov/risky-driving/drowsy-driving>.

[5] Fundación MAPFRE, "Fatigue detection system," Fundación MAPFRE, 2023. [Online]. Available: <https://www.fundacionmapfre.org/en/education-outreach/road-safety/adas-systems/types/fatigue-detection-system>

[6] M. Arefnezhad, M. K. Daneshvar, and R. Boostani, “PERCLOS estimation from EEG signals using dynamic regression models,” \*Scientific Reports\*, vol. 12, no. 1, 2022. [Online]. Available: <https://www.nature.com/articles/s41598-022-05810-x>

[7] M. Awais, N. Badruddin, and M. Drieberg, “Driver drowsiness detection using physiological signals: A review,” \*Sensors\*, vol. 17, no. 3, p. 495, Mar. 2017. [Online]. Available: https://[www.mdpi.com/1424-](http://www.mdpi.com/1424-) 8220/17/9/1991

[8] M. Arefnezhad, B. Sadeghi, and A. A. Pouyan, “Driver drowsiness detection based on steering behavior,” \*Journal of Medical Signals and Sensors\*, vol. 9, no. 1, pp. 45–52, 2019. [Online]. Available: <https://doi.org/10.3390/s17030495>

[9] T. Soukupova and J. Cech, “Real-time eye blink detection using facial landmarks,” in \*Proc. 21st Computer Vision Winter Workshop\*, Rimske Toplice, Slovenia, 2016. [Online]. Available:

<https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf>

[10] Junaedi, S., & Akbar, H. (2018). *Driver drowsiness detection based on face feature and PERCLOS.* Journal of Physics: Conference Series, 1090, 012037. <https://doi.org/10.1088/1742-6596/1090/1/012037>

[11] Ahmed, M. I. B., Alabdulkarem, H., Alomair, F., Aldossary, D., Alahmari, M., Alhumaidan, M., ... & Zaman, G. (2023). A deep-learning approach to driver drowsiness detection. *Safety*, *9*(3), 65. https://[www.mdpi.com/2313-576X/9/3/65](http://www.mdpi.com/2313-576X/9/3/65)

[12] Madane, A., Singh, A., Fargade, S., & Dongare, A. (2025). *Real-time driver drowsiness detection using deep learning and computer vision techniques*. International Journal of Scientific Research in Science,Engineering and Technology, 12(3), 222–229. <https://doi.org/10.32628/IJSRSET2512335>

[13] Essahraui, S., Lamaakal, I., El Hamly, I., Maleh, Y., Ouahbi, I., El Makkaoui, K., Filali Bouami, M., Pławiak, P., Alfarraj, O., & Abd El‑Latif, A. A. (2025). Real‑Time Driver Drowsiness Detection Using Facial Analysis and Machine Learning Techniques. Sensors, 25(3), 812. <https://doi.org/10.3390/s25030812> ([mdpi.com](https://www.mdpi.com/1424-8220/25/3/812?utm_source=chatgpt.com))

[14] C.-H. Chou, "NTHU Drowsy Driver Detection Dataset (NTHU-DDD)," National Tsing Hua University, [Online]. Available: <http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/>

[15] utarlddv1, “yawdd,” Roboflow Universe, 2024. [Online]. Available: [https://universe.roboflow.com/utarlddv1/yawdd-nx0vr](https://universe.roboflow.com/utarlddv1/yawdd-nx0vr?utm_source=chatgpt.com). [Accessed: Dec. 5, 2025].

[16] F. Song, X. Tan, X. Liu and S. Chen, “Eyes closeness detection from still images with multi‑scale histograms of Principal Oriented Gradients,” Pattern Recognition, 2014. Available: <https://parnec.nuaa.edu.cn/_upload/tpl/02/db/731/template731/pages/xtan/ClosedEyeDatabases.html>

[17]Sahin Afridi, Arafat; Kafy, Arafath ; Nessa Moon, Ms. Nazmun; Shakil, Md. Shahriar (2024), “Driver Behavior Image Dataset ”, Mendeley Data, V2, doi: 10.17632/6y3g6vs2k4.2

[Online]. Available: <https://data.mendeley.com/datasets/6y3g6vs2k4/2>

[18] R. Ghoddoosian, M. Galib, and V. Athitsos, “UTA Real‑Life Drowsiness Dataset (UTA‑RLDD),” Kaggle, 2019. [Online]. Available:

<https://www.kaggle.com/datasets/rishab260/uta-reallife-drowsiness-dataset>

[19] C. Rakibuleceruet, “Drowsiness Prediction Dataset,” Kaggle, 2025. [Online].

Available: <https://www.kaggle.com/datasets/rakibuleceruet/drowsiness-prediction-dataset>.

[20]N. Petrellis, “NITYMED Dataset,” Kaggle, 2025. [Online]. Available: <https://www.kaggle.com/datasets/nikospetrellis/nitymed>

[21] Z. Mashhour, “Driver Inattention Detection Dataset,” Kaggle, 2025. [Online]. Available: <https://www.kaggle.com/datasets/zeyad1mashhour/driver-inattention-detection-dataset>