# PROJECT REPORT

# 1. INTRODUCTION

### 1.1 Project Overview

The system for detecting driver drowsiness in real-time and preventing accidents combines various physiological and behavioral indicators to accurately assess the driver's drowsiness level. These indicators include eye movement, head pose, and facial expressions, which are continuously monitored to identify patterns associated with drowsiness.

Eye Movement: The system tracks the driver's eye movements using specialized cameras or sensors. It analyzes factors such as blink rate, eye closure duration, and eye gaze patterns. Slow or irregular blinking, prolonged eye closures, and instances of microsleep (brief episodes of unintended sleep) can indicate drowsiness.

Head Pose: The system also observes the driver's head pose and movements. It uses cameras or sensors to track head position and orientation. A drooping or nodding head, as well as sudden jerks or head tilts, may suggest drowsiness.

Facial Expressions: By analyzing the driver's facial expressions, the system can identify specific signs of drowsiness. It looks for indicators such as drooping eyelids, yawning, or changes in facial muscle tone. These expressions can provide additional insights into the driver's level of alertness.

The system continuously analyzes and combines data from these physiological and behavioral indicators to determine the driver's drowsiness level. It uses machine learning algorithms to establish baseline patterns for each individual driver and compare real-time data against these baselines. This individualized approach helps to account for variations in drowsiness indicators across different people.

To ensure the system's accuracy, extensive training and validation are conducted using large datasets that include diverse driving scenarios, lighting conditions, and weather conditions. This allows the system to adapt and perform reliably across different environments.

When the system detects that the driver's drowsiness level is increasing and poses a potential risk, it provides timely alerts. These alerts can be in the form of visual, auditory, or haptic cues, designed to grab the driver's attention without causing distraction or panic. The nature and intensity of the alert can be customized based on the severity of drowsiness.

The ultimate purpose of this system is to prevent accidents caused by drowsy driving. By accurately detecting and addressing driver drowsiness in real-time, it aims to increase driver safety, reduce the number of accidents, and potentially save lives.

### 1.2 Purpose

The purpose of this system is to enhance driver safety and reduce the number of accidents caused by drowsy driving. Drowsiness is a major contributor to road accidents, as it impairs a driver's attention, reaction time, and decision-making abilities. By accurately detecting and addressing drowsiness in real-time, the system aims to prevent accidents and save lives.

By using a combination of physiological and behavioral indicators, the system can provide a more comprehensive assessment of the driver's drowsiness level. This approach helps to minimize false alarms and ensures that the system is sensitive enough to detect subtle signs of drowsiness, even under varying lighting and weather conditions.

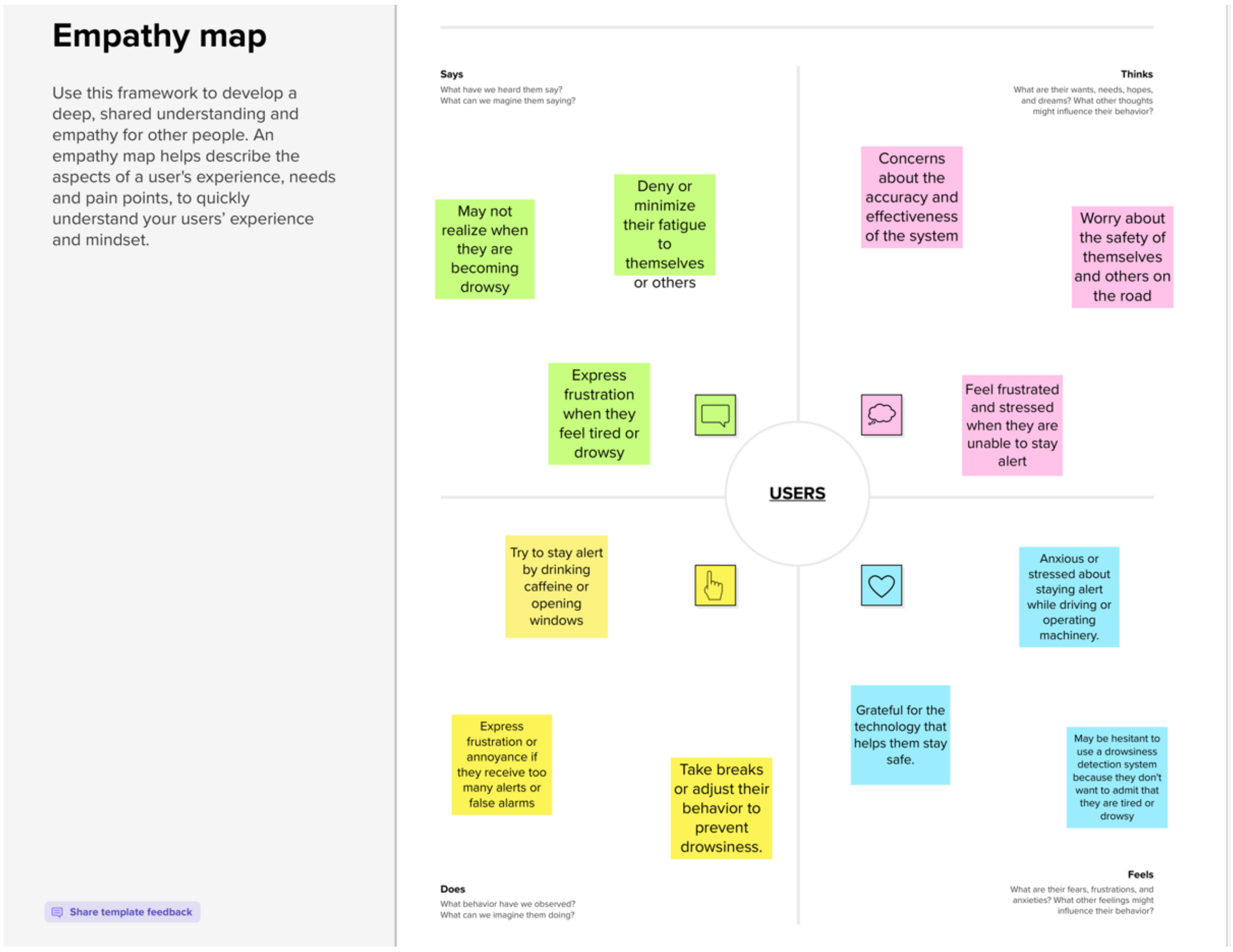
Overall, the system's primary goal is to improve driver safety by proactively detecting and alerting drowsy drivers, enabling them to take necessary actions to prevent accidents and maintain their alertness while on the road.

# 2. IDEATION & PROPOSED SOLUTION

## 2.1 Problem Statement Definition

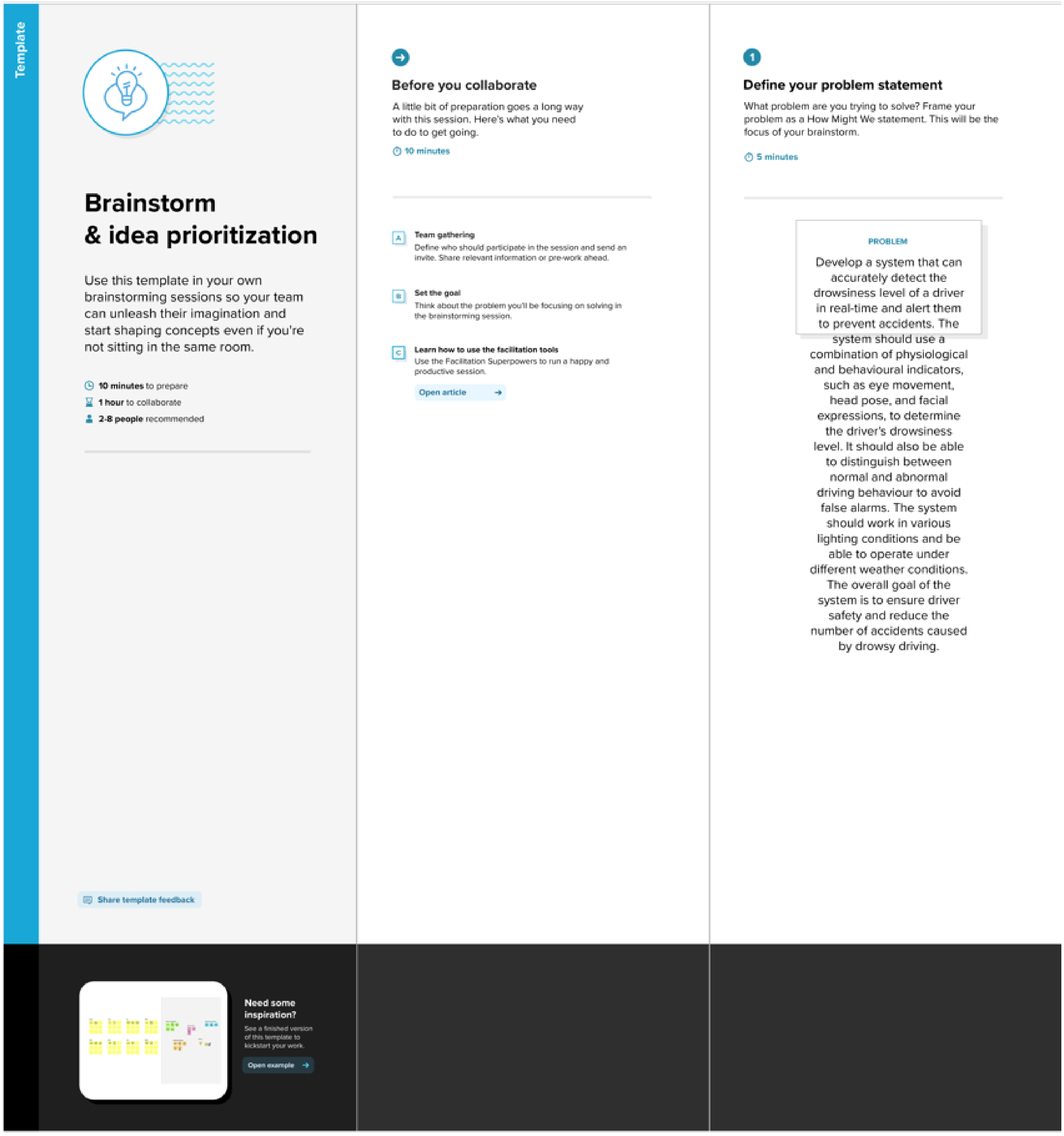
The aim of this project is to develop a real-time drowsiness detection system for drivers that accurately determines the level of drowsiness and alerts the driver to prevent potential accidents. The system will utilize a combination of physiological and behavioral indicators, including eye movement, head pose, and facial expressions, to assess the driver's drowsiness level. It should be capable of distinguishing between normal and abnormal driving behavior to avoid false alarms. The system must function effectively in various lighting conditions and be capable of operating under different weather conditions. The ultimate goal is to enhance driver safety and reduce the number of accidents caused by drowsy driving.

## 2.2 Empathy Map Canvas



Empathy map
Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.
Share template feedback
Says
What have we heard them say? What can we magine them saying?
Thinks
What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?
May not realize when they are becoming drowsy
Deny or minimize their fatigue to themselves or others
Concerns about the accuracy and effectiveness of the system
Worry about the safety of themselves and others on the road
Express frustration when they feel tired or drowsy
Feel frustrated and stressed when they are unable to stay alert
USERS
Try to stay alert by drinking caffeine or opening windows
Anxious or stressed about staying alert while driving or operating machinery.
Grateful for the technology that helps them stay safe.
Express frustration or annoyance if they receive too many alerts or false alarms
Take breaks or adjust their behavior to prevent drowsiness.
May be hesitant to use a drowsiness detection system because they don't want to admit that they are tired or drowsy
Does
What behavior have we observed? What can we imagine them doing?
Feels
What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

## 2.3 Ideation & Brainstorming



Template

**Brainstorm & idea prioritization**

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- **10 minutes** to prepare
- **1 hour** to collaborate
- **2-8 people** recommended

Share template feedback

**Need some inspiration?**

See a finished version of this template to kickstart your work.

Open example

**Before you collaborate**

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article

1 **Define your problem statement**

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

Develop a system that can accurately detect the drowsiness level of a driver in real-time and alert them to prevent accidents. The system should use a combination of physiological and behavioural indicators, such as eye movement, head pose, and facial expressions, to determine the driver's drowsiness level. It should also be able to distinguish between normal and abnormal driving behaviour to avoid false alarms. The system should work in various lighting conditions and be able to operate under different weather conditions. The overall goal of the system is to ensure driver safety and reduce the number of accidents caused by drowsy driving.

## 2.4 Proposed Solution

Real-Time Drowsiness Detection System for Drivers

Novelty: The proposed system uses a combination of physiological and behavioral indicators, including eye tracking, head pose detection, facial expression recognition, steering wheel and pedal analysis, and environmental factors, to accurately detect driver drowsiness in real-time. This multi-modal approach provides a more accurate and reliable prediction of drowsiness levels compared to existing systems that rely on a single indicator. Additionally, the proposed system can distinguish between normal and abnormal driving behavior to avoid false alarms.

Feasibility: The proposed system is feasible to develop as it uses existing technology such as cameras, sensors, and deep learning algorithms. The system can be installed in vehicles, including cars, trucks, buses, and other forms of transportation, providing a wide range of applications. The system's reliability and accuracy can be validated through testing in real-world driving conditions.

Business Model: The system can be marketed to automotive manufacturers, fleet management companies, and transportation services to improve road safety and reduce accidents caused by drowsy driving. The system can be sold as a standalone product or integrated into existing vehicle safety systems. A subscription-based business model could be implemented for ongoing maintenance, software updates, and support.

Social Impact: The proposed system can have a significant impact on road safety by preventing accidents caused by drowsy driving, reducing injuries, and saving lives. According to the National Highway Traffic Safety Administration, drowsy driving is responsible for an estimated 100,000 crashes, 71,000 injuries, and 1,550 fatalities in the United States annually. The proposed system can help reduce these numbers by alerting drivers to take necessary actions to avoid accidents caused by drowsiness.

Scalability: The proposed system is scalable as it can be implemented in a wide range of vehicles and transportation services. The system's accuracy and reliability can be improved by incorporating feedback and data from a large number of users, resulting in a continuously learning and improving system. As the demand for road safety increases, the proposed system's scalability will provide an opportunity for growth and expansion into new markets and industries.

Conclusion: The proposed real-time drowsiness detection system for drivers is a feasible and scalable solution with significant social impact potential. The proposed multi-modal approach provides a more accurate and reliable prediction of drowsiness levels, and the system's scalability provides an opportunity for growth and expansion into new markets and industries. The proposed system can improve road safety and reduce accidents caused by drowsy driving, ultimately saving lives and reducing injuries.

# 3. REQUIREMENT ANALYSIS

## 3.1 Functional requirement

Real-time Drowsiness Detection: The system should continuously monitor the driver's physiological and behavioral indicators, such as eye movement, head pose, and facial expressions, to detect drowsiness in real-time.

Drowsiness Level Determination: The system should accurately analyze the collected data to determine the level of drowsiness on a scale, indicating whether the driver is alert, mildly drowsy, or severely drowsy.

Alert Mechanism: When the system detects a high level of drowsiness, it should promptly alert the driver to take necessary actions. This can be achieved through visual alerts (e.g., blinking lights), auditory alerts (e.g., sound alarms), or haptic alerts (e.g., vibrations).

False Alarm Prevention: The system should be able to differentiate between normal driving behavior and actual drowsiness to avoid unnecessary or false alarms. It should consider factors such as time of day, road conditions, and driver-specific patterns to make accurate judgments.

Lighting and Weather Adaptability: The system should be designed to function effectively in various lighting conditions, including low light or bright sunlight, as well as adapt to different weather conditions, such as rain, fog, or snow.

User-Friendly Interface: The system should have a user-friendly interface that displays the driver's drowsiness level and provides clear alerts. The interface should be easy to understand and operate without causing distraction or confusion for the driver.

Data Logging and Analysis: The system should log the collected data, including physiological and behavioral indicators, for further analysis and evaluation. This can be useful for identifying patterns, improving the system's performance, and providing insights into the driver's drowsiness patterns over time.

Integration with Vehicle Systems: The drowsiness detection system should be designed for integration with existing vehicle systems, such as onboard computers or infotainment systems, to provide seamless functionality and enable easy installation in different vehicle models.

System Reliability and Stability: The system should be reliable and stable, capable of running continuously without experiencing frequent crashes or malfunctions. It should undergo thorough testing and quality assurance processes to ensure its robustness and reliability.

Privacy and Data Security: The system should adhere to privacy regulations and ensure the security of collected data. Driver data should be anonymized, encrypted, and stored securely to protect driver privacy and prevent unauthorized access.

Calibration and Personalization: The system should allow for calibration and personalization to adapt to individual driver characteristics, such as baseline behavior and physiological variations, for enhanced accuracy and reliability.

System Compatibility: The system should be designed to be compatible with a wide range of vehicles, including cars, trucks, and buses, to promote widespread adoption and maximize its impact in reducing drowsy driving accidents.

## 3.2 Non-Functional requirements

Performance: The drowsiness detection system should provide real-time monitoring and analysis with minimal latency. It should be capable of processing data quickly and accurately to ensure timely alerts and responses.

Accuracy: The system should strive for a high level of accuracy in detecting drowsiness levels, minimizing false positives and false negatives. It should undergo rigorous testing to validate its performance and calibration.

Robustness: The system should be able to handle variations in environmental conditions, such as different lighting and weather conditions, without compromising its performance. It should be resilient to external factors that may impact its functionality.

Usability: The system should have an intuitive and user-friendly interface that is easy to navigate and understand. It should be designed with driver safety in mind, ensuring that interactions with the system do not cause distraction or compromise attention on the road.

Reliability: The system should operate consistently and reliably without frequent failures or errors. It should be capable of recovering from failures gracefully and resume normal operation without manual intervention.

Scalability: The system should be scalable to accommodate different vehicle models and sizes. It should be able to handle increased data processing demands as more sensors and inputs are added to support future enhancements or upgrades.

Maintainability: The system should be designed with modular components and well-documented code to facilitate easy maintenance and updates. It should support future improvements, bug fixes, and compatibility with new technologies.

Data Privacy and Security: The system should adhere to privacy regulations and ensure the secure handling of driver data. Data transmission and storage should be encrypted, and access to sensitive information should be strictly controlled to prevent unauthorized access or breaches.

Integration: The system should be compatible and easily integratable with existing vehicle systems, such as CAN bus or vehicle control units, for seamless operation and minimal disruption to the vehicle's functionality.

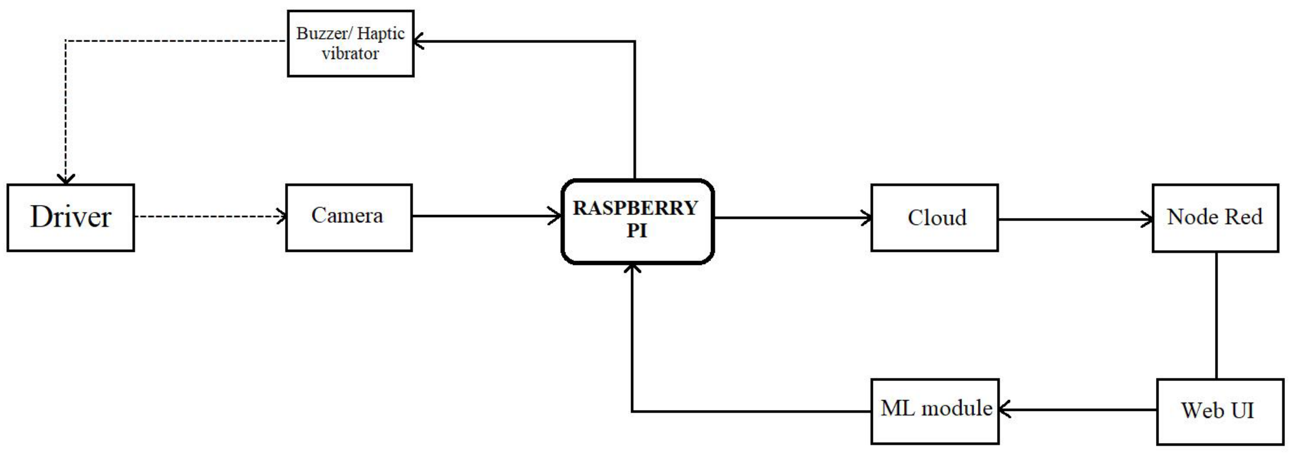
Power Efficiency: The system should be designed to minimize power consumption to avoid draining the vehicle's battery. It should employ efficient algorithms and utilize hardware components that are optimized for low power usage.

Compliance: The system should comply with relevant industry standards, regulations, and safety requirements for automotive systems. It should undergo appropriate testing and certification processes to ensure compliance and adherence to legal and regulatory frameworks.

Adaptability: The system should be adaptable to evolving technologies and advancements in drowsiness detection methodologies. It should be designed with flexibility to incorporate new features or upgrades without requiring significant system overhaul.

# 4. PROJECT DESIGN

## 4.1 Data Flow Diagram



## 4.2 Solution & Technical Architecture

**Solution Architecture:**

The drowsiness detection system can be designed using a combination of hardware and software components. Here is a high-level overview of the solution architecture:

- **Sensors:** Various sensors are used to capture the driver's physiological and behavioral indicators. This may include eye trackers, head pose sensors, and facial expression analysis tools. These sensors collect real-time data, which serves as input for the drowsiness detection system.

- **Data Acquisition:** The acquired data from the sensors is processed and transmitted to the central processing unit (CPU) for further analysis. Data acquisition techniques, such as analog-to-digital conversion, are employed to convert the sensor data into a digital format that can be processed by the system.

- **Data Preprocessing:** The raw data is preprocessed to remove noise, normalize signals, and enhance the quality of the data. Preprocessing techniques may involve filtering, signal conditioning, or feature extraction to prepare the data for subsequent analysis.

- **Drowsiness Detection Algorithm:** The preprocessed data is analyzed using a drowsiness detection algorithm. This algorithm utilizes machine learning techniques, such as deep learning models or pattern recognition algorithms, to extract meaningful features from the data and determine the driver's drowsiness level. The algorithm should be trained on a labeled dataset to enable accurate classification.

- **Alert Mechanism**: Based on the output of the drowsiness detection algorithm, an alert mechanism is triggered when a high level of drowsiness is detected. This can include visual alerts (e.g., blinking lights on the dashboard), auditory alerts (e.g., sound alarms), or haptic alerts (e.g., vibrating steering wheel). The alerts are designed to grab the driver's attention and prompt them to take necessary actions to prevent accidents.

**User Interface:** A user-friendly interface is provided to display the driver's drowsiness level and provide interaction options. The interface can be integrated into the vehicle's infotainment system or displayed on a dedicated screen. It should present information in a clear and concise manner to avoid distractions while driving.

**Technical Architecture:**

- **Hardware Components:** The hardware components include sensors (e.g., eye trackers, head pose sensors), microcontrollers or embedded systems to interface with the sensors, and communication modules for data transmission. These components are responsible for capturing and transmitting data to the central processing unit.

- **Central Processing Unit (CPU):** The CPU serves as the core processing unit of the system. It receives the sensor data, performs data preprocessing, executes the drowsiness detection algorithm, and triggers the alert mechanism. The CPU can be a high-performance processor capable of real-time data processing and analysis.

- **Software Modules:** The software modules include drivers or interfaces to interact with the hardware components, data preprocessing algorithms, the drowsiness detection algorithm, and the alert mechanism. These modules are implemented using programming languages such as Python, C++, or Java.

- **Integration with Vehicle Systems:** The drowsiness detection system needs to integrate with existing vehicle systems. This may involve integrating with the vehicle's onboard computer, infotainment system, or CAN bus. APIs or communication protocols like Bluetooth or Wi-Fi can be used for seamless integration and data exchange.

- **Data Storage and Management:** Collected data can be stored in a database or a cloud storage solution for further analysis, system improvement, and compliance with privacy regulations. Data security measures, such as encryption and access controls, should be implemented to protect sensitive driver information.

- **System Monitoring and Maintenance:** The architecture should include mechanisms for system monitoring, error handling, and logging. This allows for real-time monitoring of system performance, identification of potential issues, and maintenance activities, such as software updates and bug fixes.

- **Scalability and Performance:** The architecture should be designed to handle large-scale deployments and accommodate a growing number

## 4.3 User Stories

As a **driver**, I want the drowsiness detection system to provide timely and accurate alerts when it detects that I am becoming drowsy, so that I can take immediate action to prevent accidents. I expect the system to have a clear and attention-grabbing alert mechanism, such as visual alerts (e.g., blinking lights on the dashboard), auditory alerts (e.g., sound alarms), or haptic alerts (e.g., vibrating steering wheel). The alerts should be triggered when my drowsiness level reaches a dangerous threshold and should continue until I respond or regain alertness.

As a **fleet manager**, I want the drowsiness detection system to collect and store data on drowsiness levels for my drivers, so that I can analyze patterns and identify drivers who may be at higher risk of drowsy driving. I expect the system to have a data logging feature that records information about each driver's drowsiness levels over time. This data can help me monitor driver behavior, identify trends, and provide targeted interventions or training to reduce the risk of accidents caused by drowsiness.

As a **car manufacturer,** I want the drowsiness detection system to be easily integrated into different vehicle models, so that I can offer it as a standard safety feature to enhance driver safety. I expect the system to have a flexible and modular design that can be seamlessly integrated into the vehicle's existing systems, such as the onboard computer or infotainment system. The integration process should be straightforward and require minimal modifications to the vehicle's architecture.

As a **software developer,** I want the drowsiness detection system to have a well-documented and developer-friendly architecture, so that I can understand and extend its functionality. I expect the system to provide APIs or software development kits (SDKs) that allow me to access and utilize the system's capabilities, such as retrieving drowsiness data or customizing the alert mechanisms. The documentation should be comprehensive, including clear instructions, code examples, and relevant technical specifications.

As a **regulatory authority**, I want the drowsiness detection system to comply with relevant safety standards and regulations, so that I can ensure its effectiveness and reliability. I expect the system to undergo rigorous testing and validation processes to demonstrate its accuracy, robustness, and adherence to safety requirements. The system should also incorporate privacy measures to protect driver data and comply with data protection regulations.

As a **maintenance technician,** I want the drowsiness detection system to have built-in diagnostics and monitoring capabilities, so that I can efficiently troubleshoot and maintain the system. I expect the system to provide detailed logs and error messages that help me identify and resolve issues. Additionally, the system should support remote maintenance and software updates to minimize downtime and ensure optimal performance.

# 5. CODING & SOLUTIONING

## CODE EXPLANATION - NODE RED

The code provided appears to be a configuration file for a Node-RED flow. Here are the features identified in the code:

- **IBM Watson IoT Platform Integration:** The code includes an IBM Watson IoT node (ibmiot in) that is used to connect to the IBM Watson IoT Platform. It enables communication with IoT devices and the exchange of events and commands.

- **Debug Node:** There is a debug node that allows you to inspect the message payload during runtime for debugging purposes. It can be used to view and analyze the data flowing through the flow.

- **Function Node:** A function node is used to define custom JavaScript code logic. In this case, it receives a message payload and performs a conditional check. Depending on the value of the payload, it sets a corresponding message payload for the output.

- **UI Text Node:** The ui_text node is used to display text on the Node-RED Dashboard. It receives the message payload from the previous function node and displays it as dynamic text on the dashboard.

- **Node-RED Dashboard:** The code includes configuration for the Node-RED Dashboard. It defines the appearance and settings of the dashboard, such as the theme, site name, and layout.

These features collectively form a flow that integrates with the IBM Watson IoT Platform, processes data using custom JavaScript logic, and presents the results on a Node-RED Dashboard.

## CODE EXPLANATION - PYTHON

- **Importing libraries**The necessary libraries are imported, including cv2 for computer vision operations, dlib for face detection and landmark prediction, numpy for numerical operations, ibmiotf for IBM Watson IoT Platform integration, time for time-related operations, and pygame for playing audio.

- **IBM Watson IoT Platform Setup:**
  - The code sets up the credentials and connection options for the IBM Watson IoT Platform.
  - The organization, device type, device ID, authentication method, and authentication token are provided.
  - An options dictionary is created with the configuration details.

- **IBM Watson IoT Platform Client Connection:**
  - An IBM Watson IoT Platform client is created using the provided options.
  - The client is connected to the IBM Watson IoT Platform.

- **Eye Aspect Ratio Calculation:**
  - The eye_aspect_ratio function is defined, which calculates the eye aspect ratio based on the given eye landmarks.
  - The function uses the Euclidean distances between the eye landmarks to compute the eye aspect ratio.

- **Initialization:**
  - The code initializes the face detector and landmark predictor using the dlib library.
  - Constants such as EYE_AR_THRESH (eye aspect ratio threshold) and EYE_AR_CONSEC_FRAMES (number of consecutive frames for drowsiness detection) are defined.
  - Variables like COUNTER (counts consecutive frames with low eye aspect ratio), and ALARM_ON (tracks if the alarm is already on) are initialized.
  - Pygame is initialized to play the alarm sound.

- **Video Capture and Processing Loop:**
  - The code starts capturing frames from the video source (webcam).
  - The captured frame is converted to grayscale for face detection and landmark prediction.
  - The detector is used to detect faces in the grayscale frame.
  - For each detected face, the landmarks are predicted using the predictor.
  - The eye regions are extracted based on the predicted landmarks.
  - The eye aspect ratio is calculated using the eye_aspect_ratio function.
  - If the eye aspect ratio is below the threshold, the COUNTER is incremented.
  - If the COUNTER exceeds the consecutive frames threshold, it indicates drowsiness. The alarm is activated, and a message is published to the IBM Watson IoT Platform.
  - If the eye aspect ratio is above the threshold, the COUNTER is reset, and the alarm is turned off.
  - The eye aspect ratio is displayed on the frame.
  - The processed frame is displayed in a window.
  - The loop continues until the user presses 'q' to quit.
  - Once the loop ends, the video capture is released, and the windows are closed.

## FEATURES OF NODE RED

**Video Capture:** The code uses OpenCV (cv2) to capture video frames from the default webcam. It continuously processes each frame to detect drowsiness.

**Face Detection:** It utilizes the dlib library to perform face detection in the captured frames. The get_frontal_face_detector() function is used to create a face detector object, which is then applied to the grayscale frame to detect faces.

**Landmark Detection:** After detecting a face, the code uses a pre-trained shape predictor from dlib (shape_predictor_68_face_landmarks.dat) to identify the specific landmarks on the face, particularly the eyes. The shape.predictor() function is used to detect landmarks for each face detected.

**Eye Region Extraction:** Using the detected landmarks, the code extracts the regions of interest (ROI) corresponding to the left and right eyes. These regions will be used to calculate the eye aspect ratio.

**Eye Aspect Ratio (EAR) Calculation:** The eye_aspect_ratio() function calculates the eye aspect ratio based on the coordinates of the eye landmarks. EAR is a measure of eye openness and is determined by the ratio of the vertical distance between the upper and lower eye landmarks to the horizontal distance between the inner and outer eye corners. It provides an indication of whether the eyes are closed or open.

**Drowsiness Detection:** The calculated eye aspect ratio is compared against a threshold value (EYE_AR_THRESH) to determine if the eyes are sufficiently closed, indicating drowsiness. If the average eye aspect ratio falls below the threshold for a certain number of consecutive frames (EYE_AR_CONSEC_FRAMES), an alarm is triggered.

**Alarm and Sound:** The code uses the pygame library to play an alarm sound when drowsiness is detected. It initializes the pygame mixer and loads an alarm sound file. The alarm sound is played when the drowsiness condition is met and is stopped when the eyes are open again.

**IBM Watson IoT Platform Integration:** The code connects to the IBM Watson IoT platform using the provided credentials and options. It publishes events related to drowsiness detection, sending messages indicating whether drowsiness is detected or not.

**Visualization:** The code overlays the calculated eye aspect ratio value on the video frame and displays it in a window. It also shows the live video feed with the overlay.

These features collectively enable the code to detect drowsiness by monitoring the eye aspect ratio and triggering an alarm when drowsiness is detected, while also integrating with the IBM Watson IoT platform for event reporting.

## FEATURES OF PYTHON CODE

**IBM Watson IoT Platform Integration:** The code integrates with the IBM Watson IoT Platform to publish messages/alerts when drowsiness is detected.

**Computer Vision:** The code utilizes computer vision techniques to detect faces, extract eye regions, and calculate the eye aspect ratio for drowsiness detection.

**Dlib Library:** The code uses the Dlib library, which provides pre-trained models for face detection and landmark prediction.

**Eye Aspect Ratio Calculation:** The code calculates the eye aspect ratio, which is a measure of the eye's openness, based on the relative distances between eye landmarks.

**Alarm Activation:** When drowsiness is detected, the code activates an alarm to alert the user.

**Video Capture:** The code captures frames from a video source, typically a webcam, to process for drowsiness detection.

**Pygame Audio:** The code uses the Pygame library to play an audio alarm when drowsiness is detected.

**Thresholds:** The code defines thresholds such as EYE_AR_THRESH (eye aspect ratio threshold) and EYE_AR_CONSEC_FRAMES (number of consecutive frames for drowsiness detection).

**Consecutive Frame Counting:** The code tracks the number of consecutive frames in which the eye aspect ratio is below the threshold to determine drowsiness.

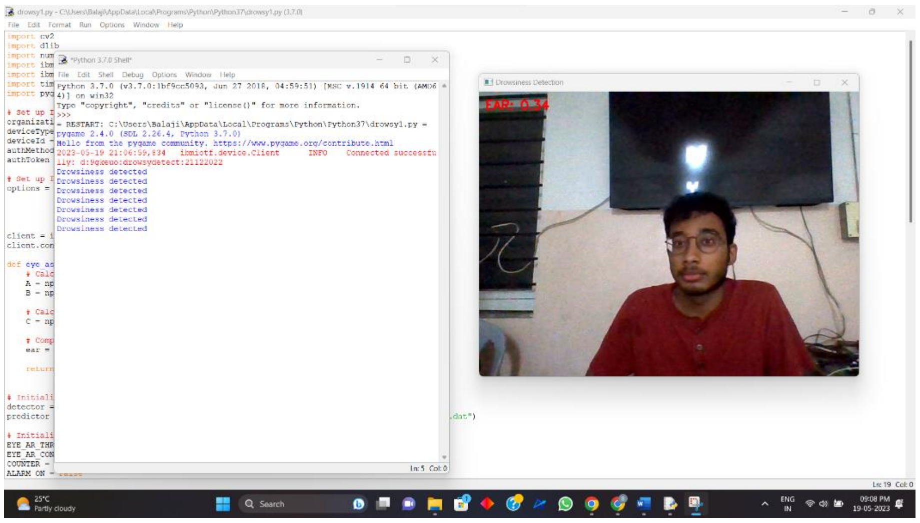
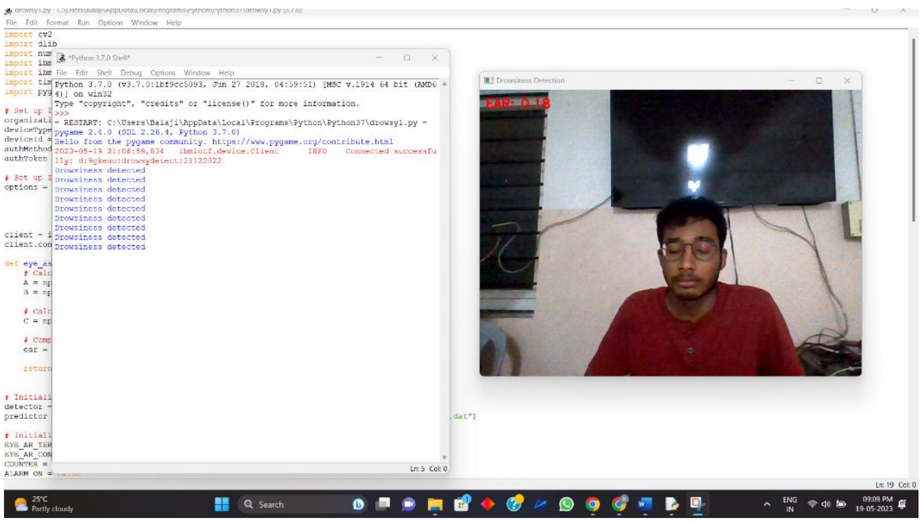
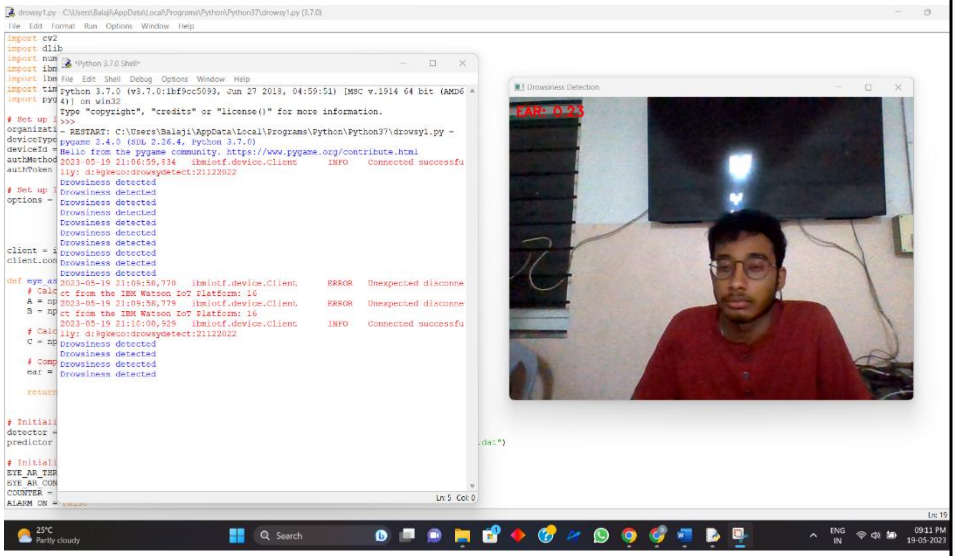
**Grayscale Conversion:** The code converts the captured frames to grayscale before performing face detection and landmark prediction.

**Real-time Display:** The processed frames, eye aspect ratio, and alarm status are displayed in real-time in a separate window.

**Keyboard Input Handling:** The code listens for keyboard input and exits the loop when the user presses 'q' to quit.

## 6. RESULTS

### 6.1 Performance Metrics

| **Parameter** | **Values** | **Screenshot** |
|---|---|---|
| Accuracy: using eye aspect ratio which is displayed in EAR<br><br>1)With glasses and eyes fully opened | EAR: 0.34<br><br>STATUS: AWAKE |  |
| 2)With glasses and eyes fully closed | EAR: 0.18<br><br>STATUS: DROWSINESS DETECTED |  |
| 3)With glasses and eyes partially closed | EAR: 0.23 |  |

Screenshot 1 (EAR: 0.34):

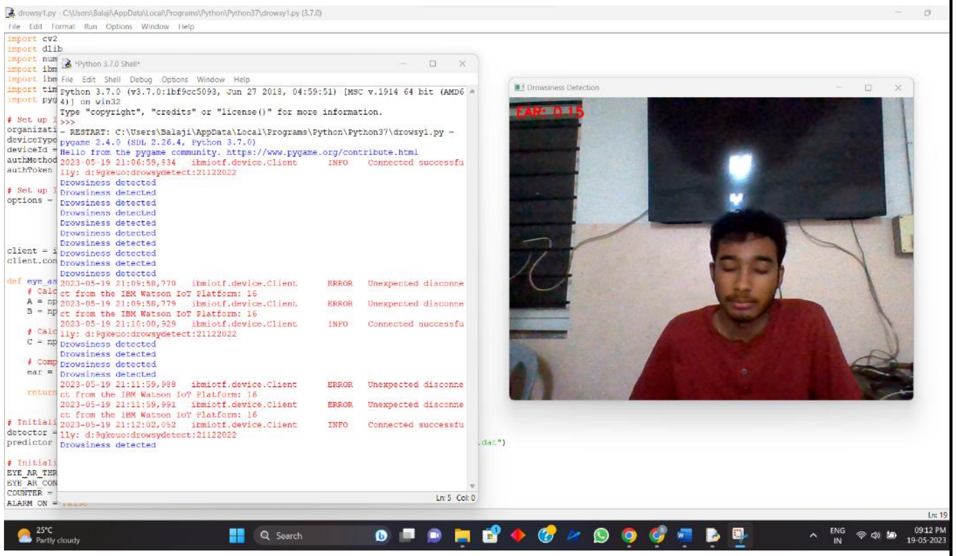
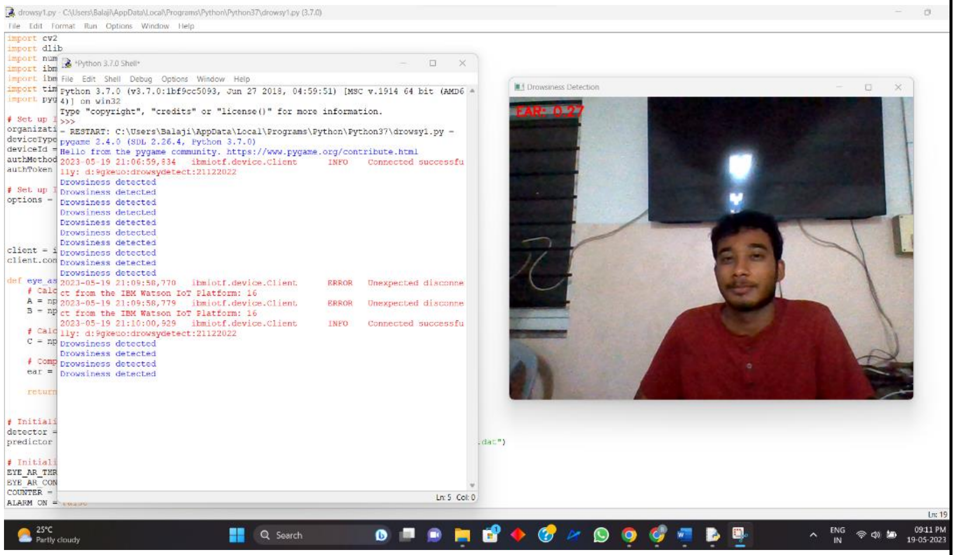
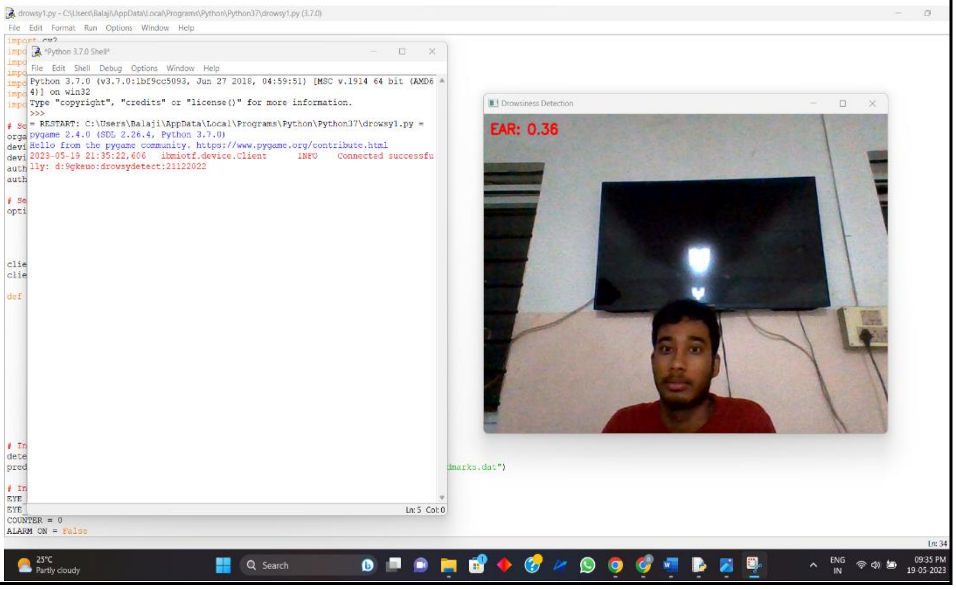
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:06:59,834   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
```

Screenshot 2 (EAR: 0.18):

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:06:59,834   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
```

Screenshot 3 (EAR: 0.23):

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:06:59,834   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
2023-05-19 21:09:58,770   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:09:58,779   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:10:00,929   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
```

|  |  |  |
|---|---|---|
|  | STATUS: AWAKE |  |
| 4)Without glasses and eyes fully closed | EAR: 0.15<br><br>STATUS: DROWSINESS DETECTED |  |
| 5)Without glasses and eyes partially opened | EAR: 0.27<br><br>STATUS: AWAKE |  |
| 6)Without glasses and eyes | EAR: 0.36 |  |

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:06:59,834   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
2023-05-19 21:09:58,770   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:09:58,779   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:10:00,929   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
2023-05-19 21:11:59,988   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:11:59,991   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:12:02,052   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:06:59,834   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
2023-05-19 21:09:58,770   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:09:58,779   ibmiotf.device.Client      ERROR   Unexpected disconne
ct from the IBM Watson IoT Platform: 16
2023-05-19 21:10:00,929   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
Drowsiness detected
Drowsiness detected
Drowsiness detected
Drowsiness detected
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Balaji\AppData\Local\Programs\Python\Python37\drowsy1.py =
pygame 2.4.0 (SDL 2.26.4, Python 3.7.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
2023-05-19 21:35:22,606   ibmiotf.device.Client      INFO    Connected successfu
lly: d:9gkeuo:drowsydetect:21122022
```

|  |  |  |
|---|---|---|
| Fully opened | STATUS: AWAKE |  |

## 7. ADVANTAGES & DISADVANTAGES

**Advantages:**

**Increased Driver Safety:** The drowsiness detection system helps enhance driver safety by providing timely alerts when drowsiness is detected. This allows drivers to take necessary actions, such as taking a break or adjusting their driving behavior, to prevent accidents caused by drowsy driving.
Example: The system detects that a driver's eye movements have become slow and heavy, indicating drowsiness. It immediately triggers an alert, prompting the driver to pull over and rest, avoiding a potential collision.

**Accident Prevention:** By alerting drivers to their drowsiness levels, the system reduces the risk of accidents caused by drowsy driving. It helps drivers maintain focus and attentiveness on the road, mitigating the potential dangers associated with fatigue.
Example: The system detects that a driver's head pose is consistently tilted downwards, indicating drowsiness. It promptly alerts the driver, preventing them from veering off the road or rear-ending another vehicle.

**Customizable and Adaptive:** A drowsiness detection system can be personalized and calibrated to suit individual drivers' characteristics, such as their normal eye movement patterns or head poses. This customization enhances the accuracy and effectiveness of the system.
Example: The system is calibrated to account for a specific driver's baseline eye movement patterns. It can accurately detect deviations from the norm and issue alerts tailored to that driver's behavior.

**Disadvantages:**

**False Alarms:** The drowsiness detection system may occasionally generate false alarms, triggering alerts when the driver is not actually drowsy. This can be due to inaccuracies in the detection algorithm or misinterpretation of the driver's physiological or behavioral indicators.
Example: The system mistakenly identifies a driver's prolonged blink as a sign of drowsiness and triggers an alert, causing unnecessary distraction and potentially frustrating the driver.

**Technical Limitations:** Drowsiness detection systems may have limitations in certain conditions or scenarios. For example, they may struggle to accurately detect drowsiness levels in extreme weather conditions or when the driver is wearing sunglasses that obstruct the view of their eyes.
Example: The system's accuracy may be compromised during heavy rain or fog, as the visibility of the driver's eyes is significantly reduced, making it difficult to accurately assess drowsiness levels.

**Reliance on Sensor Accuracy:** The effectiveness of a drowsiness detection system relies heavily on the accuracy and reliability of the sensors used to capture the driver's physiological and behavioral indicators. Inaccurate sensor readings can lead to false detections or missed instances of drowsiness.
Example: A malfunctioning eye tracker sensor fails to accurately track the driver's eye movements, resulting in inaccurate drowsiness assessments and potentially missing critical instances of drowsiness.

It is important to consider these advantages and disadvantages when designing, implementing, and using a drowsiness detection system to ensure its optimal performance and address any potential limitations.

## 8. CONCLUSION

In conclusion, a drowsiness detection system that accurately assesses the drowsiness level of a driver in real-time and provides timely alerts is a valuable tool for enhancing driver safety and reducing accidents caused by drowsy driving. By analyzing a combination of physiological and behavioral indicators such as eye movement, head pose, and facial expressions, the system can detect signs of drowsiness and prompt drivers to take necessary actions to prevent accidents.

The advantages of a drowsiness detection system include increased driver safety, accident prevention, and the ability to customize and adapt to individual driver characteristics. By alerting drivers to their drowsiness levels, the system enables them to respond appropriately, such as taking breaks or adjusting their driving behavior, reducing the risk of accidents.

However, it is important to consider the potential disadvantages, such as false alarms and technical limitations. False alarms may lead to unnecessary distractions for drivers, while technical limitations, such as challenges in extreme weather conditions or reliance on sensor accuracy, may impact the system's effectiveness.

To overcome these limitations, ongoing research and development efforts are necessary to improve the accuracy, reliability, and adaptability of drowsiness detection systems. Furthermore, continuous testing, validation, and calibration of the system are essential to ensure its optimal performance in various driving conditions and scenarios.

Overall, the implementation of a robust and reliable drowsiness detection system, integrated seamlessly into vehicle systems, can play a significant role in promoting driver safety, reducing accidents, and ultimately saving lives. By addressing drowsy driving, we can make a positive impact on road safety and strive towards a future with fewer accidents caused by drowsiness.

## 9. FUTURE SCOPE

The future scope of drowsiness detection systems is promising, with potential advancements and opportunities for improvement. Here are some areas of future development and expansion:

**Advanced Sensor Technologies:** Continuous advancements in sensor technologies can enhance the accuracy and reliability of drowsiness detection systems. Integration of advanced sensors, such as infrared cameras or depth sensors, can provide more precise measurements of physiological and behavioral indicators, enabling more accurate drowsiness assessments.

**Multi-Modal Data Fusion:** Future systems can leverage the power of multi-modal data fusion, combining data from various sensors such as eye trackers, facial expression analysis, and heart rate monitors. By integrating and analyzing data from multiple sources, the system can achieve higher accuracy in detecting drowsiness levels and differentiating them from other factors like distraction or cognitive load.

**Artificial Intelligence and Machine Learning:** The application of advanced machine learning and artificial intelligence techniques can improve the accuracy and adaptability of drowsiness detection systems. Deep learning algorithms can learn from large datasets, enabling the system to detect subtle patterns and variations in driver behavior, resulting in more precise and personalized drowsiness assessments.

**Real-Time Monitoring and Intervention:** Future systems can incorporate real-time monitoring of driver behavior and provide interventions beyond simple alerts. For example, the system could automatically adjust the vehicle's cabin environment, such as temperature or ventilation, to help keep the driver alert. Additionally, the system could integrate with other safety features like lane departure warning systems or adaptive cruise control to actively assist the driver and prevent accidents.

**Integration with Driver Assistance Systems:** The integration of drowsiness detection systems with existing driver assistance systems can further enhance safety on the roads. By combining drowsiness detection with technologies like lane-keeping assist, adaptive cruise control, or collision avoidance systems, a comprehensive driver safety ecosystem can be created to proactively mitigate the risks associated with drowsy driving.

**Data Analytics and Insights:** Collecting and analyzing drowsiness data from multiple drivers can provide valuable insights and trends. By applying data analytics techniques, patterns related to drowsy driving can be identified, leading to the development of targeted interventions, driver training programs, or fatigue management strategies.

**Wearable Devices and Mobile Applications:** The integration of drowsiness detection technologies with wearable devices or mobile applications can extend the scope of monitoring beyond the confines of the vehicle. Drivers can benefit from continuous drowsiness monitoring even when they are not driving, helping them manage their sleep patterns and fatigue levels more effectively.

As technology continues to advance, the future scope of drowsiness detection systems holds great potential to further improve driver safety, reduce accidents caused by drowsy driving, and promote overall road safety. Continued research, innovation, and collaboration among industry stakeholders will drive the evolution of these systems, making roads safer for everyone.

# 10. APPENDIX

## 10.1 Source Code - Python

```python
import cv2
import dlib
import numpy as np
import ibmiotf.application
import ibmiotf.device
import time
import pygame

# Set up IBM Watson IoT platform credentials
organization = "9gkeuo"
deviceType = "drowsydetect"
deviceId = "21122022"
authMethod = "token"
authToken = "14102108"

# Set up IBM Watson IoT platform client
options = {"org": organization,
           "type": deviceType,
           "id": deviceId,
           "auth-method": authMethod,
           "auth-token": authToken}
client = ibmiotf.device.Client(options)
client.connect()

def eye_aspect_ratio(eye):
    # Calculate the Euclidean distances between the vertical eye landmarks
    A = np.linalg.norm(eye[1] - eye[5])
    B = np.linalg.norm(eye[2] - eye[4])

    # Calculate the Euclidean distance between the horizontal eye landmarks
    C = np.linalg.norm(eye[0] - eye[3])

    # Compute the eye aspect ratio
    ear = (A + B) / (2.0 * C)

    return ear


# Initialize face detector and landmark predictor
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("C:/Users/Balaji/Downloads/shape_predictor_68_face_landmarks.dat")

# Initialize constants
EYE_AR_THRESH = 0.25
EYE_AR_CONSEC_FRAMES = 48
COUNTER = 0
ALARM_ON = False

# Initialize pygame for alarm sound
pygame.mixer.init()
sound = pygame.mixer.Sound("C:/Users/Balaji/Downloads/oversimplified-alarm-clock-113180.mp3")

# Start video capture
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
        break

    # Convert to grayscale and detect faces
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 0)

    # Loop over face detections
    for rect in rects:
        # Detect landmarks and extract eye regions
        shape = predictor(gray, rect)
        leftEye = np.array([(shape.part(36).x, shape.part(36).y),
                            (shape.part(37).x, shape.part(37).y),
                            (shape.part(38).x, shape.part(38).y),
                            (shape.part(39).x, shape.part(39).y),
                            (shape.part(40).x, shape.part(40).y),
                            (shape.part(41).x, shape.part(41).y)], np.int32)
        rightEye = np.array([(shape.part(42).x, shape.part(42).y),
                             (shape.part(43).x, shape.part(43).y),
                             (shape.part(44).x, shape.part(44).y),
                             (shape.part(45).x, shape.part(45).y),
                             (shape.part(46).x, shape.part(46).y),
                             (shape.part(47).x, shape.part(47).y)], np.int32)

        # Calculate eye aspect ratio and check for drowsiness

        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)
        ear = (leftEAR + rightEAR) / 2.0
        if ear < EYE_AR_THRESH:
            COUNTER += 1
            if COUNTER >= EYE_AR_CONSEC_FRAMES:
                if not ALARM_ON:
                    ALARM_ON = True
                    print("Drowsiness detected")
                    client.publishEvent("alert", "json", {"drowsiness": "true"})
                    # Play alarm sound
                    sound.play()
            else :
                    ALARM_ON = False
                    client.publishEvent("awake", "json", {"drowsiness": "false"})
        else:
            COUNTER = 0
            ALARM_ON = False
            # Stop alarm sound
            sound.stop()

    # Show live feed with overlay

        cv2.putText(frame, f"EAR: {ear:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX,0.7, (0, 0, 255),
2)
        cv2.imshow("Drowsiness Detection", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

## Source code - Node RED

```
[
    {
        "id": "456e3d3163987cbe",
        "type": "tab",
        "label": "Flow 1",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "2ee16081c55f862b",
        "type": "tab",
        "label": "Flow 2",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "7e64a772f049e44e",
        "type": "tab",
        "label": "Flow 5",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "ad48ff63bbf485d5",
        "type": "tab",
        "label": "Flow 3",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "19a09963733aeb64",
        "type": "tab",
        "label": "Flow 4",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "1b19699648c71f10",
        "type": "ibmiot",
        "name": "",
        "keepalive": "60",
        "serverName": "",
        "cleansession": true,
        "appId": "",
        "shared": false
    },
    {
        "id": "93d32c025b723470",
        "type": "ui_tab",
        "name": "Detection System",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    },
    {
        "id": "f50165e736a09b66",
        "type": "ui_base",
        "theme": {
            "name": "theme-light",
            "lightTheme": {
                "default": "#0094CE",
                "baseColor": "#0094CE",
                "baseFont": "-apple-system,BlinkMacSystemFont,Segoe UI,Roboto,Oxygen-
Sans,Ubuntu,Cantarell,Helvetica Neue,sans-serif",
                "edited": true,
                "reset": false
            },
            "darkTheme": {
                "default": "#097479",
                "baseColor": "#097479",
                "baseFont": "-apple-system,BlinkMacSystemFont,Segoe UI,Roboto,Oxygen-
Sans,Ubuntu,Cantarell,Helvetica Neue,sans-serif",
                "edited": false
            },
            "customTheme": {
                "name": "Untitled Theme 1",
                "default": "#4B7930",
                "baseColor": "#4B7930",
                "baseFont": "-apple-system,BlinkMacSystemFont,Segoe UI,Roboto,Oxygen-
Sans,Ubuntu,Cantarell,Helvetica Neue,sans-serif"
            },
            "themeState": {
                "base-color": {
                    "default": "#0094CE",
                    "value": "#0094CE",
                    "edited": false
                },
                "page-titlebar-backgroundColor": {
                    "value": "#0094CE",
                    "edited": false
                },
                "page-backgroundColor": {
                    "value": "#fafafa",
                    "edited": false
                },
                "page-sidebar-backgroundColor": {
                    "value": "#ffffff",
                    "edited": false
                },
                "group-textColor": {
                    "value": "#1bbfff",
                    "edited": false
                },
                "group-borderColor": {
                    "value": "#ffffff",
                    "edited": false
                },
                "group-backgroundColor": {
                    "value": "#ffffff",
                    "edited": false
                },
                "widget-textColor": {
                    "value": "#111111",
                    "edited": false
                },
                "widget-backgroundColor": {
                    "value": "#0094ce",
                    "edited": false
                },
                "widget-borderColor": {
                    "value": "#ffffff",
                    "edited": false
                },
                "base-font": {
                    "value":                  "-apple-system,BlinkMacSystemFont,Segoe
UI,Roboto,Oxygen-Sans,Ubuntu,Cantarell,Helvetica Neue,sans-serif"
                }
            },
            "angularTheme": {
                "primary": "indigo",
                "accents": "blue",
                "warn": "red",
                "background": "grey",
                "palette": "light"
            }
        },
        "site": {
            "name": "Node-RED Dashboard",
            "hideToolbar": "false",
            "allowSwipe": "false",
            "lockMenu": "false",
            "allowTempTheme": "true",
            "dateFormat": "DD/MM/YYYY",
            "sizes": {
                "sx": 48,
                "sy": 48,
                "gx": 6,
                "gy": 6,
                "cx": 6,
                "cy": 6,
                "px": 0,
                "py": 0
            }
        }
    },
    {
        "id": "2eb972124fc842af",
        "type": "ui_group",
        "name": "Status",
        "tab": "93d32c025b723470",
        "order": 1,
        "disp": true,
        "width": "6",
        "collapse": false,
        "className": ""
    },
    {
        "id": "ec517124c92cd7fa",
        "type": "ibmiot in",
        "z": "456e3d3163987cbe",
        "authentication": "apiKey",
        "apiKey": "1b19699648c71f10",
        "inputType": "evt",
        "logicalInterface": "",
        "ruleId": "",
        "deviceId": "21122022",
        "applicationId": "",
        "deviceType": "+",
        "eventType": "+",
        "commandType": "",
        "format": "json",
        "name": "IBM IoT",
        "service": "registered",
        "allDevices": "",
        "allApplications": "",
        "allDeviceTypes": true,
        "allLogicalInterfaces": "",
        "allEvents": true,
        "allCommands": "",
        "allFormats": "",
        "qos": 0,
        "x": 130,
        "y": 160,
        "wires": [
            [
                "55ee11edcaae95cf",
                "89fc6a82f6fedbeb"
            ]
        ]
    },
    {
        "id": "55ee11edcaae95cf",
        "type": "debug",
        "z": "456e3d3163987cbe",
        "name": "debug 2",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 380,
        "y": 100,
        "wires": []
    },
    {
        "id": "89fc6a82f6fedbeb",
        "type": "function",
        "z": "456e3d3163987cbe",
        "name": "Message Status",
        "func":            "msg.payload=msg.payload.Status\n\nif(msg.payload==0)\n{\n
msg.payload=\"GOOD TO GO!\";\n}\nelse\n{\n      msg.payload=\"Wake Up!\";\n}\nreturn
msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "\nmsg.payload=\"Good To GO!\";\nreturn msg;",
        "finalize": "",
        "libs": [],
        "x": 320,
        "y": 240,
        "wires": [
            [
                "e5843b5085b6c2de"
            ]
        ]
    },
    {
        "id": "e5843b5085b6c2de",
        "type": "ui_text",
        "z": "456e3d3163987cbe",
        "group": "2eb972124fc842af",
        "order": 0,
        "width": 0,
        "height": 0,
        "name": "",
        "label": "",
        "format": "{{msg.payload}}",
        "layout": "row-center",
        "className": "",
        "x": 490,
        "y": 240,
        "wires": []
    }
]
```

**GitHub & Project Video Demo Link**

GitHub: https://github.com/naanmudhalvan-SI/IBM--12028-1682491618.git

Demo Video: https://drive.google.com/file/d/1dcE4IZIcwsMfT55KVGw92Auwlu7VhQa6/view?usp=sharing