KLE Technological University | Creating Value Leveraging Knowledge
Dr. M. S. Sheshgiri Campus, Belagavi

Department of Electronics and Communication Engineering

Senior Design Project Report on

ADVANCED DRIVER AWARENESS AND SAFETY SYSTEM

By:

1. **Snehanshu Gunjal** USN: 02FE22BEC103
2. **Sourabh Pawar** USN: 02FE22BEC105
3. **Suchitkumar Khadakabhavi** USN: 02FE22BEC109
4. **Hiba Ashekhan** USN: 02FE22BEC121

Semester: VII, 2025

Under the guidance of

Dr. Arun Tigadi

KLE Technological University,
Dr. M. S. Sheshgiri College of Engineering and Technology
BELAGAVI-590 008
2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that project entitled **"Advanced Driver Awareness and Safety System"** is a bonafide work carried out by the student team of **"Snehanshu Gunjal (02FE22BEC103) , Sourabh Pawar (02FE22BEC105), Suchitkumar Khadakabhavi (02FE22BEC109), Hiba Ashekhan (02FE22BEC121)"**.The project report has been approved as it satisfies the requirements with respect to the Minor project work prescribed by the university curriculum for B.E. (VII Semester) in Department of Electronics and Communication Engineering of KLE Technological University Dr. M. S. Sheshgiri CET Belagavi campus for the academic year 2025.

Dr. Arun Tigadi	**Dr. Dattaprasad Torse**	**Dr. S. F. Patil**
Guide	**Head of Department**	**Principal**

Name of Examiners **Signature with date**

1.

2.

Contents

List of Figures

List of Tables

Abstract

Road safety has become a major concern due to the increasing number of accidents caused by driver fatigue, traffic rule violations, and delayed emergency response. To address these issues, this project presents a low-cost, IoT-enabled **Smart Vehicle Controller** integrating key Advanced Driver Assistance System (ADAS) features. The system combines **driver drowsiness detection**, **traffic sign recognition**, **automatic speed control**, **traffic light response**, and **accident alerting** into a unified framework.

Drowsiness detection is implemented using the MediaPipe Face Mesh model and Eye Aspect Ratio (EAR) analysis, enabling real-time monitoring of driver fatigue. Traffic sign recognition is achieved using a YOLOv8 deep learning model, allowing the vehicle to automatically adjust its speed or stop based on detected signs. An MPU6050 sensor provides tilt-based accident detection, and emergency SMS alerts are sent through a Flask server using Twilio. Two ESP32 microcontrollers manage vehicle control and traffic light simulation, communicating via Wi-Fi using HTTP protocols.

Experimental results demonstrate that the system performs reliably in real time, accurately detecting driver drowsiness, interpreting traffic signs, responding to traffic light states, and triggering emergency alerts when required. The proposed prototype highlights how embedded systems and machine learning can be integrated to enhance driver safety, offering a scalable platform for future ADAS research and development.

1 Introduction

Road safety has become a major global concern due to the rising number of accidents caused by human error, overspeeding, drowsy driving, and delayed emergency response. Studies show that a large percentage of road accidents occur because drivers fail to notice traffic signs, drive while fatigued, or are unable to receive timely assistance after a collision. With advancements in embedded systems and machine learning, it is now possible to design intelligent driver-assistance solutions that address these challenges in real time.

Modern Advanced Driver Assistance Systems (ADAS) found in high-end vehicles incorporate features such as drowsiness monitoring, traffic sign recognition, and automatic braking. However, these systems are often expensive and inaccessible for low-cost or academic prototypes. This project aims to develop an affordable, modular, and effective **Smart Vehicle Controller** that integrates several core ADAS functionalities using ESP32 microcontrollers, sensor modules, and machine learning algorithms.

The system combines five major features: driver drowsiness detection, traffic sign recognition, automatic speed regulation, traffic light interpretation, and accident alerting. A Python-based machine learning hub processes camera inputs, using MediaPipe Face Mesh for eye aspect ratio (EAR)–based drowsiness analysis and YOLOv8 for recognizing traffic signs from a live video feed. Depending on the detected state, appropriate HTTP commands are sent to the ESP32 to stop the vehicle, slow it down, or adjust its behavior.

The ESP32 also interfaces with an MPU6050 sensor to detect abnormal tilt or impact conditions, which may indicate an accident. When such an event is detected, a Flask server triggers an emergency SMS alert using Twilio, providing a quick response mechanism. A secondary ESP32 is used to simulate traffic light behavior and relay real-time light states to the main controller.

Overall, the project demonstrates how embedded electronics, computer vision, and IoT-based communication can be integrated to create a practical and low-cost ADAS prototype. The system is scalable, easy to modify, and serves as a strong foundation for future intelligent transportation projects.

2 Literature Survey

White et al. [13] introduced WreckWatch, a smartphone-based accident detection system utilizing accelerometer, GPS, and acoustic data. Their approach demonstrated the feasibility of mobile-sensor crash detection, significantly reducing alert delays using automated emergency notifications.

Bhatti et al. [2] developed an IoT-enabled accident reporting architecture that integrates multi-sensor data with cloud-based alert dissemination. Their work highlights the importance of low-cost, scalable infrastructure for smart city accident response systems.

Fogue et al. [5] proposed a VANET-driven accident severity estimation and notification system using vehicular telemetry. Their method improves emergency response time by leveraging cooperative V2V and V2I communication.

Shah et al. [9] introduced the CADP dataset, comprising annotated CCTV accident videos for training deep learning models. Their benchmark provides essential resources for accident detection and temporal localization research.

Fang et al. [4] developed the DADA-2000 dataset, incorporating driver attention and fixation maps to predict accident risk. Their study emphasizes the relevance of attention-aware models for accident anticipation.

Suzuki et al. [12] proposed an adaptive-loss–based early accident anticipation framework using near-miss and accident datasets. Their method enhances warning capability by improving temporal prediction accuracy.

Karim et al. [6] introduced a Dynamic Spatial–Temporal Attention (DSTA) network for real-time accident anticipation. Their model effectively captures motion saliency and inter-object interactions in dashcam videos.

Aloul et al. [1] developed iBump, a smartphone-enabled crash detection tool using accelerometer and GPS signatures. Their study demonstrated that low-cost mobile sensing can reliably detect vehicular impacts.

Xu et al. [15] proposed a vehicular fog-computing architecture for real-time collision warnings using trajectory calibration. Their work reduces communication latency by offloading computations to edge nodes.

Nexar et al. [8] introduced a large-scale dashcam collision prediction dataset to support anticipatory safety models. Their benchmark enables the development of early-warning neural models for real-world driving conditions.

Spicher et al. [11] proposed an international standard accident number to unify and automate global accident reporting. Their work addresses interoperability challenges in large-scale emergency response systems.

Chan et al. [3] released the Dashcam Accident Dataset (DAD), enabling research in accident detection and early anticipation. Their dataset has become widely adopted for benchmarking video-based safety algorithms.

Mittal and Arora [7] proposed a multi-frame CNN architecture for traffic sign recognition under real-world conditions. Their model demonstrated superior robustness against occlusions and lighting variations.

Wu et al. [14] utilized convolutional neural networks for accurate traffic sign detection, paving the way for modern deep-learning-based driver assistance systems. Their architecture outperformed traditional feature-based methods.

Sharma et al. [10] introduced a hybrid drowsiness detection model combining visual cues and EEG signals. Their approach significantly improved driver state classification accuracy, especially in low-light environments.

3 Methodology

The proposed system is designed to monitor the driver's alertness, recognize traffic signs in real-time, and control vehicle speed accordingly. The architecture of the system is divided into three primary modules: Drowsiness Detection, Traffic Sign Recognition, and Speed Control. Each module functions independently while contributing to the overall driver safety framework. The following sub-sections explain each component in detail.

3.1 System Overview

The proposed system integrates multiple safety modules to form a complete smart vehicle controller. The ESP32 microcontroller serves as the core vehicle unit responsible for motor control, speed regulation, and receiving commands from external modules. A secondary ESP32 controls traffic light simulation and communicates traffic light states to the main vehicle.

A Python-based *Smart Driver Hub* processes two camera feeds: one for driver drowsiness detection and another for traffic sign recognition. The hub sends real-time HTTP commands to the ESP32, enabling automatic vehicle control. An MPU6050 sensor is used for accident detection, and a Flask server triggers SMS alerts through Twilio in case of emergencies.

3.2 Drowsiness Detection Module

The drowsiness detection module is implemented using Python with OpenCV and MediaPipe's Face Mesh solution. It continuously monitors the driver's eye activity through a webcam and

determines the eye state using the Eye Aspect Ratio (EAR). The module also integrates voice alerts and motor control commands via serial communication.

1. Eye Aspect Ratio (EAR):
The EAR is computed based on the vertical and horizontal distances between key landmarks around each eye. MediaPipe Face Mesh provides 3D facial landmarks, from which specific eye points are extracted:

- Left Eye: Points 33, 160, 158, 133, 153, 144
- Right Eye: Points 362, 385, 387, 263, 373, 380

The EAR is calculated using the formula:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2 \cdot ||p_1 - p_4||}$$

where p_i are the coordinates of the eye landmarks. A lower EAR value indicates that the eye is closed.

2. Sleep Detection Logic:
The system checks whether the EAR falls below a predefined threshold (0.22). If the eyes remain closed for more than 3 seconds continuously, the system concludes that the driver is drowsy. The logic uses timestamps to track how long the EAR remains below the threshold.

3. Voice Alert:
A separate voice alert thread is activated using the `pyttsx3` text-to-speech engine. If drowsiness is detected, the system continuously plays the message *"Wake up!"* to alert the driver.

4. Serial Communication and Motor Control:
Upon detecting sleep, a command is sent via serial port (e.g., `COM10`) to a connected ESP microcontroller. This command instructs the motor to stop by sending the `STOP` command. If the driver regains alertness, a `START` command is sent to resume vehicle movement.

5. System Workflow:

1. Start webcam capture and extract facial landmarks using MediaPipe.
2. Compute the EAR for both eyes.
3. If average EAR < 0.22 for more than 3 seconds:
    - Display "SLEEP DETECTED" on the screen.
    - Start a voice alert using a separate thread.
    - Send a `STOP` command to the ESP.
4. If EAR returns above the threshold:
    - Stop the alert and voice thread.
    - Send a `START` command to resume the motor.

6. Result Display:
The current camera feed and drowsiness status are displayed in a GUI window using OpenCV. If drowsiness is detected, a warning text is shown in red.

This approach effectively combines visual analysis with embedded system control to simulate a real-time driver monitoring system.

3.3 Traffic Sign Recognition Module

The traffic sign recognition module is designed to detect and classify road signs in real time, enabling the driver to be constantly aware of the external environment. This module leverages a deep learning approach using the YOLOv8 object detection architecture.

1. Dataset:

The Indian Road Traffic Sign Detection dataset was used for training and evaluation. It includes a diverse set of annotated images representing various road signs such as "Stop," "Speed Limit 50," "No Parking," and more, as per Indian traffic standards. The dataset was uploaded and managed on **Roboflow**, a cloud platform for image annotation, augmentation, and dataset versioning.

Roboflow was used to:

- Annotate the dataset with bounding boxes for each traffic sign class.
- Perform automatic data augmentation (e.g., rotation, blur, brightness shifts).
- Export the dataset in YOLOv8-compatible format.

2. Model Architecture:

The YOLOv8 (You Only Look Once version 8) object detection model from the Ultralytics library was used for this project. Specifically, the `yolov8n.yaml` (nano version) configuration was selected to ensure faster training and reduced computational load, suitable for real-time inference on embedded or low-power systems.

3. Training Configuration:

- **Framework:** Ultralytics YOLOv8 (PyTorch-based)
- **Model:** YOLOv8-nano (`yolov8n.yaml`)
- **Dataset Format:** YOLOv8 (images, labels, `data.yaml`)
- **Training Parameters:**
  - Epochs: 50
  - Image Size: 640×640 pixels
  - Batch Size: 16
  - Optimizer: SGD (default)

4. Training Procedure:

The model was trained using the following Python script:

```
from ultralytics import YOLO

# Load the YOLOv8 nano model
model = YOLO("yolov8n.yaml")

# Train the model
model.train(
    data=r"D:\Snehanshu_minor\dataset\data.yaml",
    epochs=50,
    imgsz=640,
```

```
    batch=16,
    name="gtsrb_yolov8n",
    save=True
)
```

During training, the model learned to localize and classify traffic signs from the dataset images. Roboflow ensured proper data management and export, simplifying integration with YOLOv8.

5. Inference and Real-Time Application:
After training, the best model weights were used for inference on test images and video frames. The detection output includes:

- Bounding boxes around detected signs
- Class labels (e.g., "Speed Limit 50")
- Confidence scores for each detection

Voice alerts corresponding to detected signs were generated using text-to-speech APIs to inform the driver without distraction.

6. Summary:
This module enables accurate and real-time detection of Indian road traffic signs using an efficient deep learning pipeline. Its integration into the system ensures that even if the driver misses a sign visually, an auditory notification helps maintain safety and compliance on the road.

3.4 Speed Control Module

The speed control module is designed to regulate the speed of the vehicle in response to the driver's state, particularly in cases of detected drowsiness. This module simulates vehicular speed control using a DC motor connected to an ESP32 microcontroller. Pulse Width Modulation (PWM) is used to vary the motor speed, while GPIO pins control the motor's direction.

1. Hardware Configuration:

- **Platform:** ESP32 Development Board
- **Motor Driver:** L298N Dual H-Bridge Motor Driver
- **Connections:**
  - `ENA (GPIO 25)` – PWM pin to control speed
  - `IN1 (GPIO 26)` – Direction control pin 1
  - `IN2 (GPIO 27)` – Direction control pin 2

2. Control Logic:
The ESP32 generates a PWM signal on the ENA pin to control the speed of the motor. Direction is controlled using IN1 and IN2 pins. Based on the drowsiness status received from the detection system, the ESP32 either reduces or stops the motor speed.

3. Implementation Overview:

- When the driver is alert, the motor operates at variable speeds (low, medium, high).

- Upon detection of drowsiness, a command is sent from the main processing unit to the ESP32 via serial communication.
- The ESP32 interprets the command and sets the motor speed to zero by reducing the PWM duty cycle to 0%.
- Once the driver regains alertness, the motor resumes its previous speed.

4. Advantages of ESP32:

- Built-in PWM support on multiple GPIO pins
- Dual-core processor allows handling of motor control and communication tasks simultaneously
- Compact and suitable for embedded automotive applications

This updated configuration ensures efficient and real-time motor control, making the system more compact and suitable for deployment in low-power embedded environments.

3.5 Traffic Light Communication

The traffic light communication module demonstrates the concept of **Vehicle-to-Infrastructure (V2I)** communication, where the vehicle interacts with external traffic systems in real time. A secondary ESP32 microcontroller is used to simulate a functional traffic light unit equipped with two LEDs: red and green. The module alternates between light states at predefined time intervals (typically every five seconds) to replicate real-world traffic signal behavior.

When the red LED is turned ON, the traffic light ESP32 sends an HTTP GET request with the endpoint `/red_light` to the main vehicle ESP32. Upon receiving this request, the main ESP32 immediately halts all motor activity to simulate the vehicle stopping at a red signal. Conversely, when the green LED is activated, a `/green_light` command is sent, signaling the main ESP32 to resume motion at the previously set speed.

This communication uses the **HTTP protocol** over Wi-Fi, ensuring wireless synchronization between both ESP32 devices. The modularity of this setup allows the system to be scaled for more complex V2I applications in the future, such as networked traffic management or adaptive signal timing. Through this module, the vehicle can respond dynamically to environmental inputs, enabling smarter and safer driving automation.

3.6 Accident Detection and Alert Module

The accident detection subsystem enhances passenger safety by monitoring the vehicle's tilt and motion patterns using the **MPU6050** sensor, which combines a three-axis accelerometer and gyroscope. The sensor continuously measures orientation changes in real time, and the ESP32 processes these values to calculate tilt angles along the X and Y axes.

When the tilt angle exceeds a defined threshold of approximately 45°, it is considered an abnormal orientation, suggesting that the vehicle may have overturned or collided. Upon detecting such a condition, the ESP32 flags an accident event and transmits an HTTP request to the Flask-based backend server. The server, in turn, executes a predefined routine that triggers an **SMS alert using the Twilio API**. The alert message includes the text "Accident Detected" along with a direct **Google Maps link** of the predefined coordinates for immediate location tracking.

This approach ensures rapid and automated communication in emergency situations. By leveraging IoT connectivity, the module bridges the gap between the embedded sensing system and external emergency response networks. Once the vehicle is restored to a stable position and the tilt returns below 40°, the ESP32 resets the accident flag and resumes normal operation.

3.7 Block Diagram

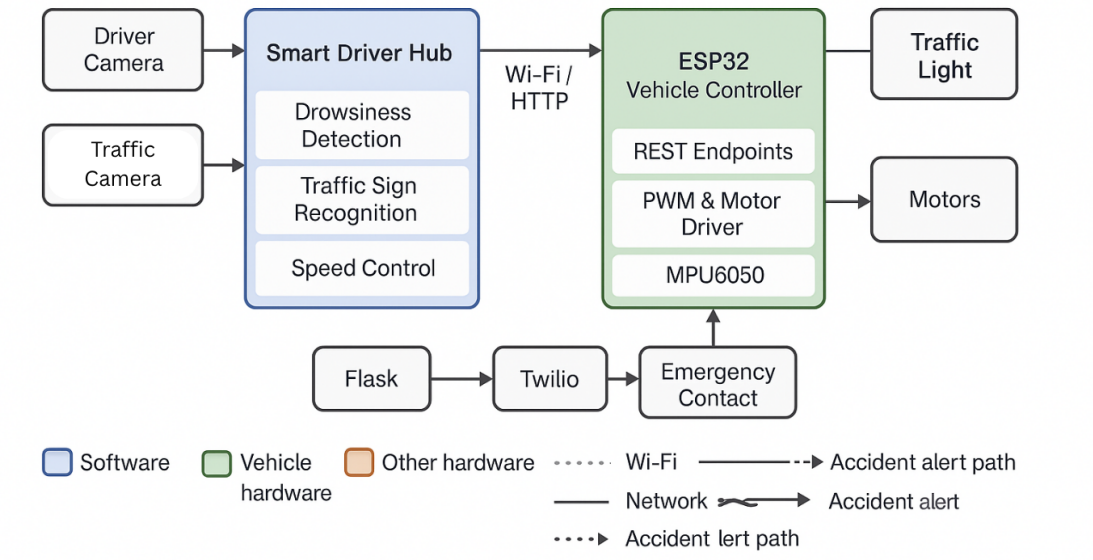


Figure 1: Block diagram of the Smart Vehicle Controller system

3.8 Technology Stack

The system is implemented using a combination of hardware and software tools that collectively form a robust and flexible development environment. Each component in the technology stack plays a critical role in ensuring real-time data processing, communication, and actuation.

- **Hardware Components:**
  - **ESP32 Microcontrollers:** Dual-core microcontrollers used for the main vehicle unit and traffic light module, responsible for communication and motor control.
  - **L298N Motor Driver:** Dual H-bridge driver circuit used to control the direction and speed of the DC motors based on PWM signals.
  - **MPU6050 Sensor:** Inertial Measurement Unit (IMU) providing accelerometer and gyroscope data for tilt and accident detection.
  - **DC Motors:** Provide mechanical movement and speed variation controlled by PWM.
- **Software Tools:**
  - **Arduino IDE:** Used for ESP32 programming in C/C++.

- **Python:** Used for implementing ML models, Flask server, and control logic.
- **Flask Framework:** Hosts the local web server to handle HTTP communication and SMS triggering.
- **Twilio API:** Enables SMS alert delivery with embedded location data.

- **Libraries and Frameworks:**
  - **MediaPipe:** Provides real-time face landmark tracking for drowsiness detection.
  - **OpenCV:** Used for image acquisition and preprocessing in computer vision tasks.
  - **YOLOv8:** Deep learning-based object detection model for recognizing traffic signs.
  - **pyttsx3:** Used for generating text-to-speech alerts during drowsiness events.

- **Communication Protocol:**
  - The entire system communicates using **HTTP GET requests** over Wi-Fi.
  - Each module (Drowsiness, Traffic Sign, Traffic Light, and Accident Alert) sends commands to the ESP32 via REST-style endpoints.
  - This enables modular, asynchronous, and wireless control without physical dependency between subsystems.

This technology stack ensures scalability, platform independence, and high reliability, allowing new modules or sensors to be easily integrated into the system in future developments.

3.9 Workflow Summary

The complete working of the system follows a well-defined sequential and event-driven flow, ensuring all subsystems operate cohesively in real time. The following steps describe the overall workflow:

1. **System Initialization:** Both ESP32 controllers initialize their peripherals, establish Wi-Fi connections, and activate PWM channels, motor pins, and sensors.
2. **Camera Feed Acquisition:** Two video sources are initiated — a USB webcam for monitoring the driver's face and an IP camera for capturing real-time traffic scenes.
3. **Drowsiness Detection:** The MediaPipe model analyzes facial landmarks, calculates the Eye Aspect Ratio (EAR), and detects eye closure duration. If drowsiness persists beyond three seconds, the ESP32 receives a `/drowsy_on` command to stop the vehicle.
4. **Traffic Sign Recognition:** The YOLOv8 model processes the IP camera frames to identify signs like "Speed 30", "Speed 50", "Speed 80", or "Stop". Based on the detection, corresponding HTTP commands are sent to the ESP32 to adjust motor speed or halt the vehicle.
5. **Traffic Light Communication:** The secondary ESP32 alternates LED states between red and green every five seconds and sends respective commands (`/red_light` or `/green_light`) to the main ESP32, which updates vehicle behavior accordingly.
6. **Accident Detection:** The MPU6050 sensor continuously measures tilt angles. When a tilt exceeding 45° is detected, an alert request is sent to the Flask server, which then transmits an SMS notification containing the accident alert and Google Maps location link.

7. **Web Interface Access:** A local web server hosted on the ESP32 allows manual control through buttons for forward, backward, left, right, and stop. The interface automatically disables control if the system detects a drowsiness event.

8. **System Recovery:** Once tilt or drowsiness conditions normalize, the ESP32 resumes regular speed operation and reactivates manual control options.

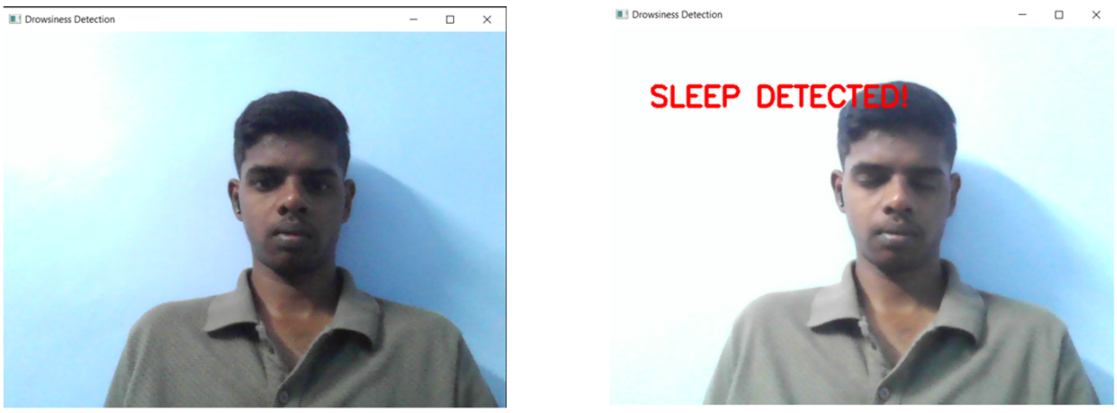
Through this integrated workflow, all components — machine learning models, sensors, and microcontrollers — function in synchronization to form an intelligent and adaptive driving assistance prototype. The modular structure enables individual module testing and future upgrades such as GPS integration, lane detection, or real-time cloud connectivity.

4 Results and Discussion

The Advanced Driver Awareness and Safety System was successfully implemented and tested in a simulated environment that closely mirrors real-world driving conditions. Each module of the system was evaluated independently and in combination with others to ensure proper integration and real-time responsiveness.

4.1 Drowsiness Detection Results

The drowsiness detection module demonstrated high accuracy in identifying driver fatigue based on eye closure duration. By computing the Eye Aspect Ratio (EAR) from MediaPipe's facial landmarks, the system reliably detected when the driver's eyes remained closed for more than 3 seconds. Upon detection, the system generated timely voice alerts and sent control signals to stop the motor. The response time was fast, with negligible latency between detection and system action.



Drowsiness not detected

Drowsiness detected

Figure 2: Drowsiness detection

- EAR threshold used: 0.22
- Sleep detection delay: 3.0 seconds
- Alert: Voice message "Wake up!" triggered via `pyttsx3`
- Motor response: STOP command sent via serial when drowsiness detected

The use of multithreading ensured that voice alerts did not block frame processing, maintaining real-time performance.

4.2 Traffic Sign Recognition Results

The traffic sign recognition model, trained on the Indian Road Traffic Sign Detection dataset using YOLOv8-nano, achieved efficient and accurate detection during inference. The model was tested on both image and video inputs and correctly identified signs like "Stop", "Speed Limit 50", "No Parking", and others under different lighting and background conditions.

- Model: YOLOv8n (nano)
- Dataset: Annotated and augmented on Roboflow (IRTSD)
- Epochs trained: 50
- Voice alerts: Generated in real time for recognized signs

To further evaluate the performance of the YOLOv8 model, various metrics and visualizations were generated:

- **Confusion Matrix:** Displays the number of true positives, false positives, true negatives, and false negatives for each class.

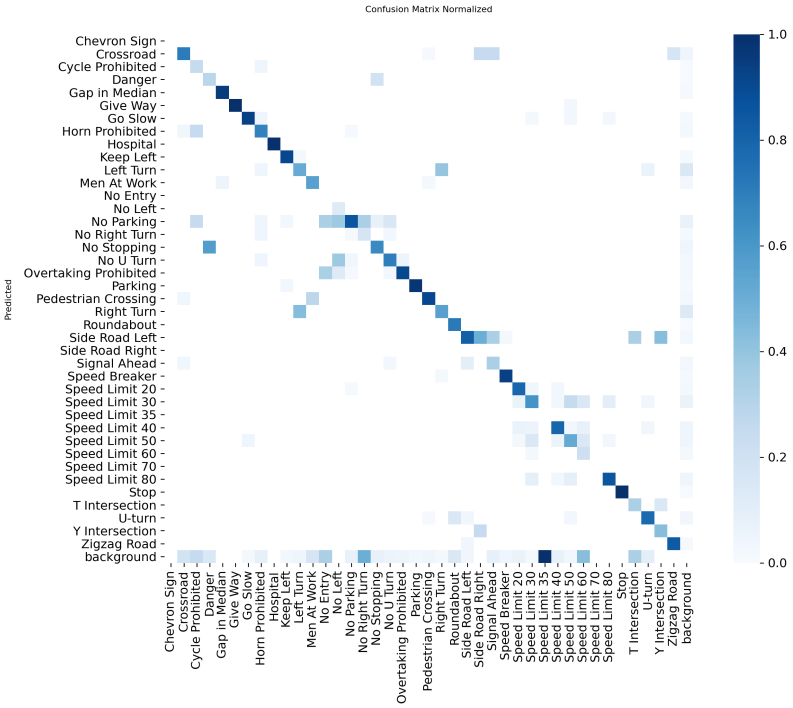


Figure 3: Confusion matrix

- **Precision-Recall Curve (PR Curve):** Illustrates the trade-off between precision and recall for different threshold values.

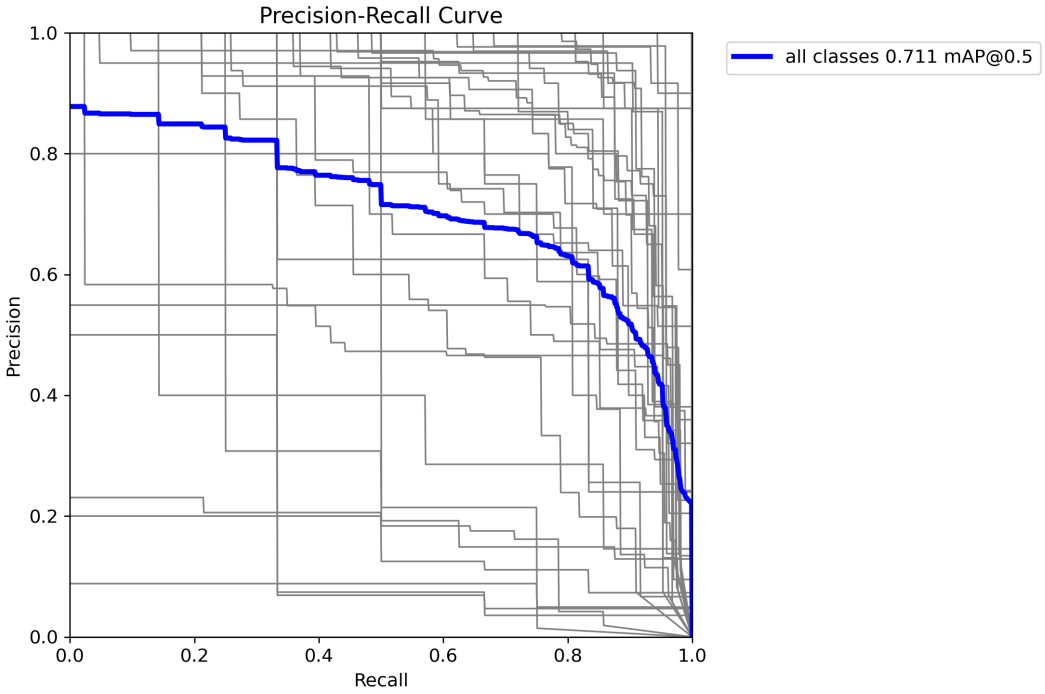


Figure 4: Precision - Recall curve

- **Recall Curve (R Curve):** Shows recall variation across classes.

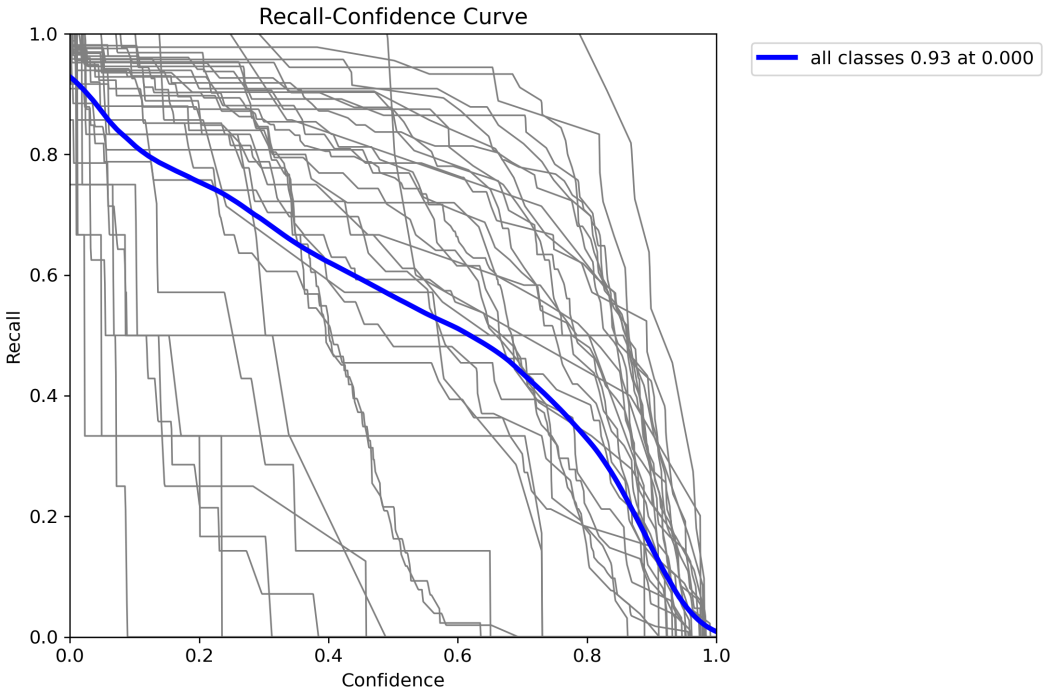


Figure 5: Recall curve

.

- **Precision Curve (P Curve):** Displays the precision scores class-wise.

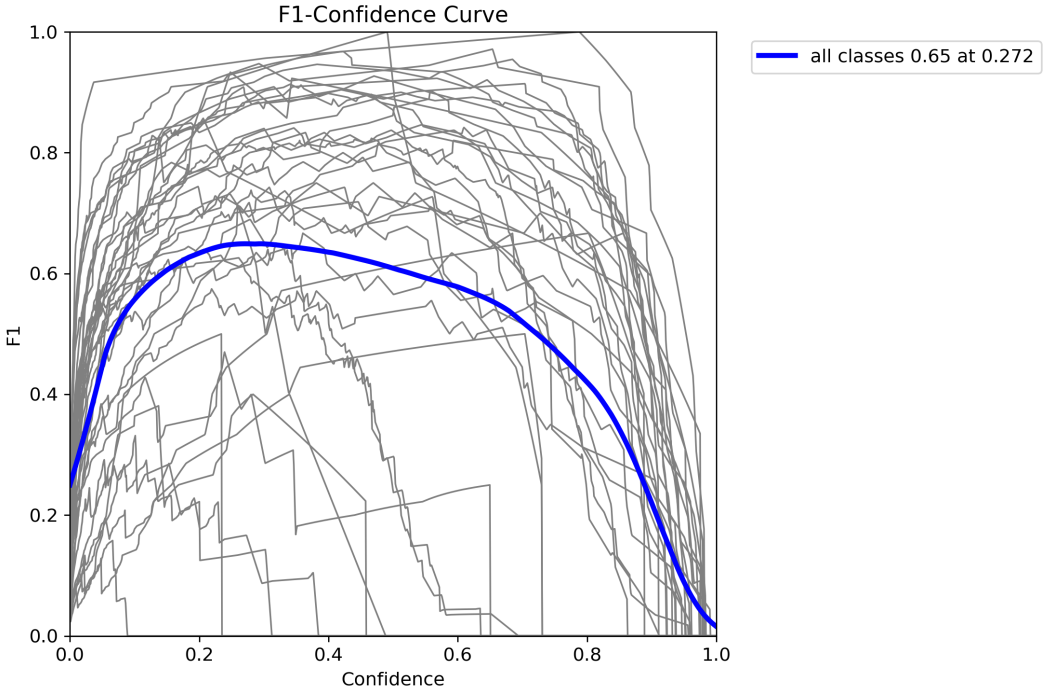


Figure 6: F1 curve

.

- **F1 Score Curve (F1 Curve):** Combines precision and recall into a single performance score.

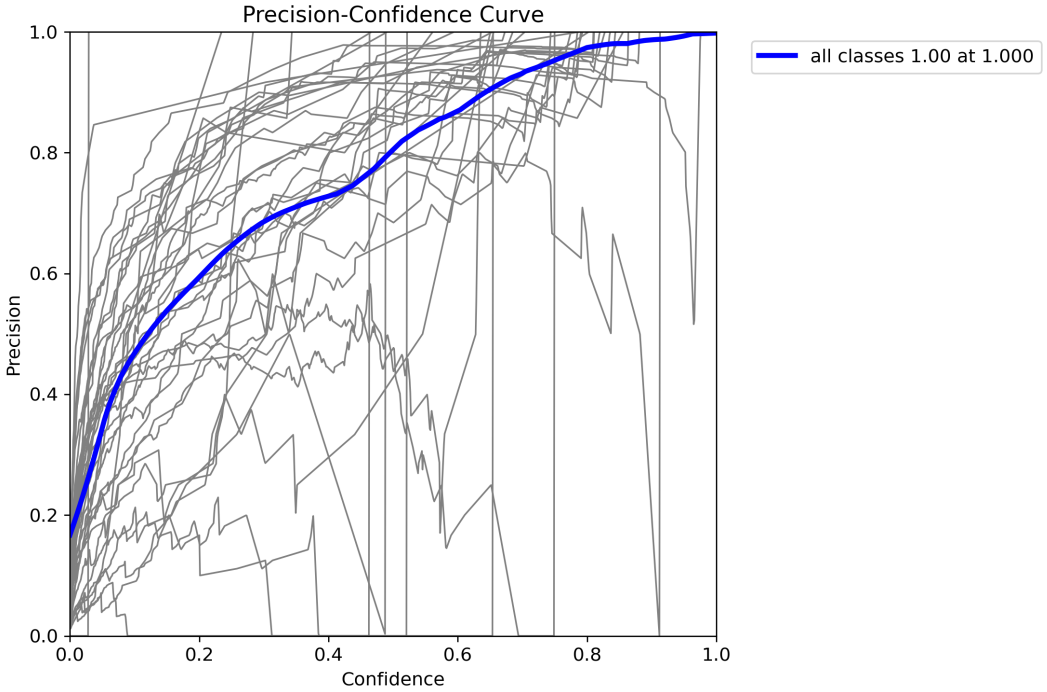


Figure 7: Precision curve

.

- To qualitatively assess the YOLOv8 model's ability to detect and classify traffic signs in real-world scenarios, a validation batch containing diverse and complex road scenes was used. The model was tested on high-resolution images captured under varied lighting conditions and environmental backgrounds to simulate realistic driving conditions.

  The validation batch included a range of Indian traffic signs, such as:

  - **Regulatory signs:** *Stop, No Parking, No Stopping, Speed Limit 20, Speed Limit 50, Speed Limit 80*
  - **Warning signs:** *Go Slow, Speed Breaker Ahead, Signal Ahead, Pedestrian Crossing, Side Road Left, Gap in Median*
  - **Mandatory signs:** *Right Turn*

  The model correctly identified and localized each of these signs with high confidence scores. Figure 8 shows sample predictions from the validation batch, with bounding boxes and class labels accurately overlaid on the respective signs.

  This successful identification across multiple classes demonstrates the robustness of the YOLOv8-nano model in handling varying traffic sign shapes, colors, and sizes. The model's ability to generalize well to new environments also confirms its practical suitability for deployment in real-time traffic sign recognition systems within intelligent driver assistance frameworks.

Figure 8: Validation batch results

.

- To evaluate the performance of the YOLOv8 object detection model on the Indian Road Traffic Sign Detection dataset.

Table 1: YOLOv8 Detection Performance on IRTSD Dataset

Metric	Value	Description
Box Precision (P)	0.674	∼67.4% of predicted boxes are correct (low false positives)
Box Recall (R)	0.714	∼71.4% of ground truths were detected (low false negatives)
mAP@0.5	0.711	Mean Average Precision at IoU=0.5; reflects overall detection quality
mAP@0.5:0.95	0.438	Mean Average Precision averaged over IoU thresholds (0.5 to 0.95); stricter and more realistic metric

These plots provide a deeper insight into the class-wise performance of the model, highlighting strengths and potential areas for improvement.

4.3 Speed Control Outcomes

The speed control module was implemented on an ESP32 development board using PWM (Pulse Width Modulation) to control a DC motor via GPIO pins. The system successfully demonstrated varying speed levels and automatic stop/resume functionality based on drowsiness status.

- Speed control achieved using PWM on GPIO 25
- Direction control via GPIO 26 and GPIO 27
- Speed Levels: Low (33%), Medium (66%), High (100%)
- STOP and START commands sent from the drowsiness module via serial communication

The hardware response was immediate, and commands from the software were correctly interpreted by the motor driver circuitry, validating the effectiveness of the ESP32 platform for real-time motor control.

4.4 Traffic Light Communication Results (V2I)

The traffic light communication module was implemented using a secondary ESP32 microcontroller that simulated a two-state signal system with red and green LEDs. The ESP32 periodically toggled between these states every five seconds and transmitted corresponding HTTP GET requests to the main vehicle controller via the local Wi-Fi network.

When the `/red_light` command was received, the main ESP32 halted the motor driver, effectively simulating a vehicle stopping at a red signal. Upon receiving the `/green_light` command, the system resumed normal motor operation with previously set PWM values. The response time between the light change and the motor action was observed to be less than 200 milliseconds, ensuring near real-time synchronization.

The results confirmed that the system achieved stable and interference-free communication between both ESP32 modules. This validated the success of the Vehicle-to-Infrastructure (V2I) interaction using simple HTTP protocols. The modular structure also allows scalability, where multiple traffic units can communicate simultaneously with the same vehicle controller for advanced traffic coordination.

4.5 Accident Detection and Alert Module Results

The accident detection module employed an MPU6050 sensor to continuously monitor the vehicle's tilt angles along the X and Y axes. When the vehicle experienced a tilt greater than 45° or a sudden change in acceleration, the ESP32 classified it as a potential accident condition.

Once triggered, the ESP32 sent an HTTP GET request to the Flask server hosted on the Smart Driver Hub through the endpoint `/alert`. The Flask application then activated the Twilio API to send an SMS alert to a predefined emergency contact number. The message included a Google Maps link corresponding to the vehicle's preset coordinates, ensuring quick response and location tracking.

During testing, the module reliably detected abnormal tilts without false triggers. The average alert transmission time (from detection to SMS delivery) was approximately 2–3 seconds, which is satisfactory for real-time emergency notification. The module successfully demonstrated the integration of sensor-based detection with cloud communication through IoT protocols, enhancing driver safety and post-accident response efficiency.

4.6 Integrated System Behavior

After verifying all individual modules, the entire system was integrated to evaluate its overall functionality and communication efficiency. The unified system combined drowsiness detection, traffic sign recognition, speed regulation, traffic light interaction, and accident alerting into a single real-time framework.

The Smart Driver Hub continuously processed video streams for drowsiness and traffic sign recognition while simultaneously communicating with the main ESP32. Each detected condition triggered corresponding HTTP commands to the ESP32, which managed vehicle behavior through PWM motor control. Simultaneously, the secondary ESP32 representing the traffic signal interacted with the main controller to regulate motion according to light status.

Integration testing revealed seamless coordination among all modules:

- The drowsiness module accurately stopped the vehicle and issued voice alerts when the driver appeared fatigued.
- The YOLOv8 traffic sign model dynamically adjusted motor speed limits in response to detected signs.
- The V2I traffic light communication ensured immediate halting on red signals and automatic resumption on green.
- The accident detection subsystem efficiently generated real-time alerts with GPS links via Twilio.

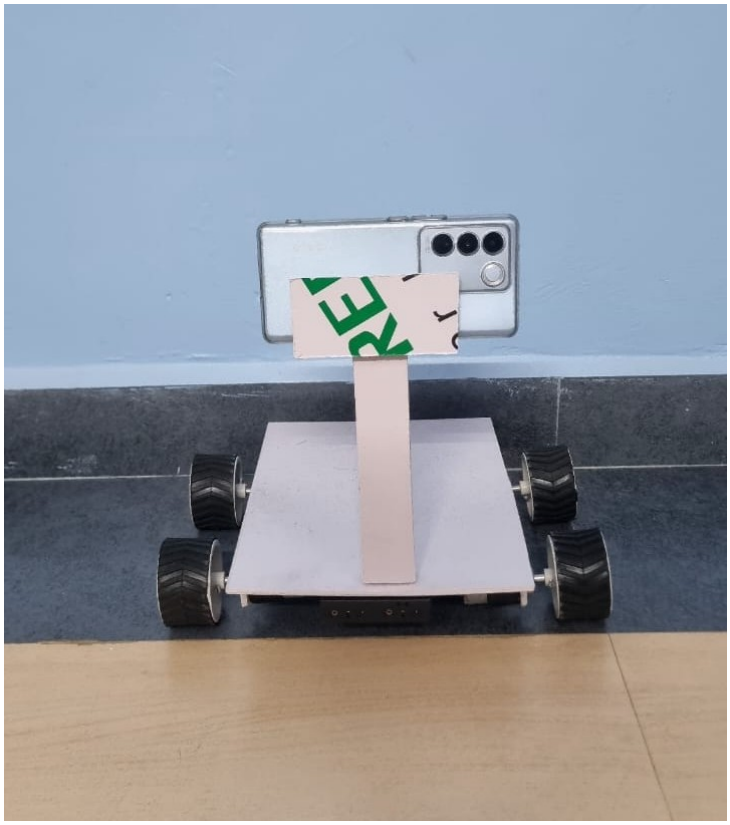


Figure 9: Car showing integrated system

.

The complete system exhibited stable real-time performance, with an average communication delay below 100 milliseconds between modules. This low latency confirms the reliability of the Wi-Fi–based HTTP communication framework. The prototype successfully demonstrates that integrating embedded systems, machine learning, and IoT communication can deliver intelligent, low-cost ADAS functionalities suitable for modern smart vehicles.

5 Conclusion

The proposed Smart Vehicle Controller successfully integrates multiple intelligent safety modules into a single, low-cost prototype capable of performing real-time driver assistance functions. The system effectively combines machine learning, embedded control, and IoT technologies to enhance overall vehicle safety and automation.

The drowsiness detection module based on MediaPipe Face Mesh and Eye Aspect Ratio (EAR) accurately identified fatigue conditions and prevented unsafe driving by halting the vehicle. The YOLOv8-based traffic sign recognition module demonstrated high accuracy in detecting speed limit and stop signs, dynamically adjusting motor speed using PWM control. The Vehicle-to-Infrastructure (V2I) communication through a secondary ESP32 module ensured real-time compliance with traffic light signals. Additionally, the MPU6050-based accident detection and alert module efficiently recognized critical tilt angles and transmitted automated emergency alerts through Twilio.

Experimental results confirmed that all modules operated cohesively with minimal latency, maintaining stable HTTP-based communication over Wi-Fi. The system successfully demonstrates that the integration of machine learning algorithms with embedded IoT control can create a functional and scalable prototype of an Advanced Driver Assistance System (ADAS).

Overall, the project contributes toward improving road safety, minimizing human error, and showcasing how affordable technology can be leveraged for intelligent transportation systems.

6 Future Scope

The current prototype provides a foundational platform for future development of advanced autonomous and safety-driven vehicle systems. Several enhancements can be incorporated to further improve its functionality, efficiency, and adaptability:

- **Real-Time Vehicle Implementation:** Integration with a full-scale vehicle platform or an autonomous robotic car can enable realistic testing under dynamic road conditions.
- **Cloud Connectivity:** Incorporating cloud-based storage and analytics would allow real-time monitoring of vehicle health, trip logs, and accident reports for fleet management.
- **GPS and GSM Modules:** Adding live GPS tracking and GSM communication can make the accident alert module location-aware and independent of local Wi-Fi infrastructure.
- **Lane Detection and Collision Avoidance:** Extending the vision system with lane tracking, obstacle detection, and collision prevention will bring the prototype closer to Level 2 or Level 3 driver assistance.
- **Mobile Application Interface:** A companion Android or iOS application can be developed to visualize sensor data, control vehicle modes, and receive emergency notifications.
- **Edge AI Optimization:** Deployment of lightweight machine learning models (such as quantized YOLO or MobileNet variants) directly on the ESP32 or Raspberry Pi can reduce system latency and dependency on external computation.

With these advancements, the system can evolve from a proof-of-concept prototype into a robust smart vehicle platform, paving the way toward cost-effective and accessible intelligent driver assistance technologies in the future.

References

[1] Fadi Aloul, Imran Zualkernan, Ruba Abu-Salma, Humaid Al-Ali, and May Al-Merri. ibump: Smartphone application to detect car accidents. *Computers & Electrical Engineering*, 43:66–75, 2015.

[2] Fizzah Bhatti, Munam Ali Shah, Carsten Maple, and Saif Islam. A novel internet of things-enabled accident detection and reporting system for smart city environments. *Sensors*, 19(9):2071, 2019.

[3] F. H. Chan et al. Dashcam accident dataset (dad), 2016. Benchmark dashcam accident dataset widely used in research.

[4] Jianwu Fang, Dingxin Yan, Jiahuan Qiao, and Jianru Xue. Dada-2000: Can driving accident be predicted by driver attention? *arXiv preprint arXiv:1904.12634*, 2019.

[5] Manuel Fogue, Piedad Garrido, Francisco Martinez, Juan-Carlos Cano, Carlos Calafate, and Pietro Manzoni. A system for automatic notification and severity estimation of automotive accidents. *IEEE Transactions on Mobile Computing*, 13(5):948–963, 2014.

[6] M. M. Karim et al. Dynamic spatial–temporal attention network for early anticipation of traffic accidents. *arXiv preprint arXiv:2106.10197*, 2021.

[7] A. Mittal and S. Arora. Real-time traffic sign recognition using a multi-frame cnn architecture. *International Journal of Computer Applications*, 2019.

[8] Daniel Moura, Shizhan Zhu, Orly Zvitia, et al. Nexar dashcam collision prediction dataset and challenge, 2025. Nexar official research dataset and benchmark.

[9] A. Shah and Others. Cadp: A novel dataset for cctv traffic camera based accident analysis. In *IEEE Conference on Computer Vision*, 2018.

[10] R. Sharma et al. Hybrid visual-eeg model for driver drowsiness detection. *IEEE Sensors Journal*, 2020.

[11] Nils Spicher et al. Proposing an international standard accident number for automatic notification. *International Journal of Safety Systems*, 2021.

[12] T. Suzuki et al. Anticipating traffic accidents with adaptive loss for early anticipation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[13] Jules White, Chris Thompson, Hamilton Turner, Brian Dougherty, and Douglas Schmidt. Wreckwatch: Automatic traffic accident detection and notification with smartphones. *Mobile Networks and Applications*, 16(3):285–303, 2011.

[14] Y. Wu et al. Traffic sign detection based on convolutional neural networks. *IEEE Access*, 2017.

[15] Xincao Xu, Kai Liu, Ke Xiao, Liang Feng, Zhou Wu, and Songtao Guo. Vehicular fog computing enabled real-time collision warning via trajectory calibration. *arXiv preprint arXiv:2005.11508*, 2020.