International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

SNAPS - SmartDrive: Drowsiness & Accident Detection

Shivam Kumar Mishra[1], Priyosmit Polley[2], Dr. Rakhi Bhattacharjee[3]

[1,2]Student, Computer Science and Engineering, Calcutta Institute of Engineering and Management
[3]Associate Professor, Electronics and Communications, Calcutta Institute of Engineering and Management

Abstract

India has one of the highest rates of road accidents globally, with a large share involving two-wheelers. Many of these accidents occur late at night or on isolated roads where immediate help is unavailable. In such scenarios, timely detection and automated emergency communication can mean the difference between life and death. To address this issue, we present SNAPS (SmartDrive), a real-time embedded safety system that integrates AI, sensors, and communication technologies to improve vehicular safety. Crucially, SNAPS is tailored to vehicle-specific needs: accident detection for two-wheelers and both accident + drowsiness detection for four-wheelers.

For two-wheelers, SNAPS focuses solely on accident detection. The system employs microcontrollers such as the ESP32,MPU6050 gyroscope, and ultrasonic sensors. These sensors continuously monitor motion patterns, impact forces, and sudden decelerations. In the event of a crash or fall—such as a skid on an empty road—the system immediately triggers an emergency response using a GSM module (SIM800L/SIM900). The alert, containing location and event details, is sent to pre-registered family contacts, local police stations, ambulances, and optionally nearby petrol pumps. This rapid alert mechanism ensures timely intervention, even when the rider is unconscious or unable to seek help.

For four-wheelers, SNAPS provides a dual-layer safety mechanism. In addition to the accident detection system described above, a Python-based AI module (running on a laptop or embedded board) uses OpenCV, MediaPipe, and Dlib to perform real-time monitoring of the driver's face and eye behavior. The system can identify signs of fatigue or micro-sleeps, which are leading causes of highway accidents. If drowsiness is detected, the system issues immediate visual/audio alerts and may initiate safety protocols to reduce the risk of a crash. If an accident still occurs, the GSM-based emergency alert mechanism activates as described for two-wheelers.

SNAPS supports communication via serial, Wi-Fi, and Bluetooth protocols to interface cloud services. Alerts and system status are displayed on a 16x2 LCD in the vehicle. Optionally, Firebase or SQL-based databases can log incident history and driver behavior for post-event analysis.

By distinguishing between the needs of two-wheelers and four-wheelers, SNAPS offers a scalable and cost-effective safety platform. In high-risk, low-visibility conditions—especially common on Indian roads—this technology can play a vital role. For example, a late-night bike skid on a deserted road may otherwise go unnoticed for hours; with SNAPS, emergency services are notified in real time, significantly improving survival chances.

In conclusion, SNAPS demonstrates how AI and embedded systems can work together to address India's urgent road safety challenges, reducing fatalities from both unreported accidents and drowsy driving.

Keywords: Accident Detection; Drowsiness Detection; AI-Embedded Safety System; ESP32; GSM Emergency Alerts;

1. Introduction

India's road infrastructure supports the movement of over 295 million registered vehicles as of 2023, with two-wheelers accounting for nearly 75% of this traffic volume (MoRTH, 2022). While two-wheelers offer economical and flexible mobility, they also represent a disproportionately high share of road fatalities. According to the National Crime Records Bureau (NCRB) and the World Health Organization (WHO), more than 50,000 motorcyclists lose their lives annually in India, many due to delayed emergency response in the aftermath of accidents. A significant number of these accidents occur on under-lit, isolated roads, especially during late-night hours when traffic is sparse and human intervention is minimal. These stark figures underline a critical gap in the country's road safety and emergency response infrastructure—one that disproportionately affects two-wheeler users.

Despite increasing awareness and technological advancement in automobile safety systems for cars and heavy vehicles, two-wheelers have largely been left behind in the adoption of life-saving technologies. Existing smart safety systems, such as Advanced Driver Assistance Systems (ADAS), often focus on four-wheelers and above, excluding the vast majority of vehicles populating Indian roads. This technological neglect, combined with the inherent physical vulnerability of bike riders, necessitates a dedicated system that can autonomously detect accidents and immediately notify emergency services and contacts.

In response to this glaring need, we propose SNAPS (SmartDrive)—a novel, embedded, and scalable driver safety platform specifically designed to address India's road safety challenges. SNAPS operates on a dual-mode architecture: for two-wheelers, it provides a dedicated accident detection and alert mechanism; for four-wheelers, it incorporates both accident detection and driver drowsiness monitoring using AI-based vision systems. This bifurcated architecture ensures that each vehicle type receives a tailored safety protocol, optimized for its specific vulnerabilities and usage conditions.

The motivation behind SNAPS stems from real-world scenarios frequently witnessed across Indian cities and rural stretches alike. Consider a common situation: a motorcyclist traveling alone late at night on a remote stretch of highway. The rider may lose control due to poor lighting, a pothole, or even sudden mechanical failure. Upon skidding and falling, there is often no immediate passerby to witness the crash or render aid. In such a scenario, the victim may lie injured or unconscious for hours until discovered, significantly reducing their chances of survival. This is where SNAPS intervenes. Through onboard sensors such as accelerometers (ADXL345), gyroscopes (MPU6050), and ultrasonic modules, SNAPS can detect the abrupt deceleration, orientation shift, or impact patterns associated with an accident. Upon identification of such anomalies, the system activates a GSM module (SIM800L or SIM900) to instantly dispatch an alert via SMS or call to emergency services, registered family members, and nearby response units like police stations or petrol pumps. This immediate relay of critical information—including GPS coordinates—bridges the life-threatening time gap between accident occurrence and emergency response.

While the two-wheeler use case forms the core of SNAPS's innovation in the Indian context, its four-wheeler integration adds an additional layer of safety through drowsiness detection. Studies show that driver fatigue contributes to over 20% of road accidents on highways (WHO, 2021). Using AI-driven computer vision algorithms (implemented in Python with OpenCV, MediaPipe, and Dlib), SNAPS monitors a driver's facial features and eye behavior to detect early signs of drowsiness or unconsciousness. If fatigue is detected, the system issues auditory and visual alerts to regain the driver's attention. Should

the drowsy state persist or an accident still occur, the embedded GSM system ensures that emergency messages are sent in real time, just as in the bike-oriented deployment.

SNAPS is built upon accessible, low-cost microcontroller platforms such as ESP32, ensuring that the technology remains both affordable and scalable. Data transfer is handled via serial communication, Bluetooth, and Wi-Fi, allowing seamless integration with cloud infrastructure. An in-vehicle 16x2 LCD display provides real-time feedback to the rider or driver, offering system status updates and confirmation of emergency alert dispatch. For backend storage and analysis, SNAPS optionally integrates with Firebase or SQL databases to log sensor readings, alert history, and driver behavior patterns for future risk assessment or legal evidence.

The system's modularity makes it adaptable to a variety of vehicle types—personal motorcycles, commercial fleet bikes, taxis, and private cars alike. This adaptability ensures wide deployment potential across both urban and rural India, a feature critical for national-scale impact. From a policy perspective, integrating SNAPS into transport licensing norms or vehicle manufacturing standards could dramatically shift the safety landscape for millions of Indian road users.

It is also worth noting that SNAPS's reliance on GSM-based communication enables cloud independence, a major advantage in regions with unreliable internet connectivity. By leveraging SMS and cellular networks—available even in many low-bandwidth zones—the system maintains robust functionality regardless of digital infrastructure limitations.

In terms of future directions, SNAPS's framework is open to enhancements such as health vitals monitoring (pulse rate, oxygen level), fall detection, unconsciousness verification, and AI-based driver profiling using machine learning models. These enhancements would enable even finer-grained intervention and predictive risk assessment, pushing the boundaries of automated vehicular safety.

In conclusion, SNAPS represents a crucial step toward democratizing road safety through embedded intelligence. By addressing the most vulnerable segment of Indian road users—two-wheeler riders—with a dedicated accident detection and real-time emergency alert system, while also safeguarding four-wheeler drivers from fatigue-induced crashes, SNAPS offers a comprehensive, scalable, and life-saving solution. Its relevance is not only technological but humanitarian, positioning it as a vital tool in India's pursuit of safer roads and reduced traffic fatalities.

2. Methodology

The SNAPS (Safety Navigation and Predictive System) prototype integrates wireless microcontrollers, sensors, and computer vision to detect driver drowsiness and predict accidents in a 4-wheeler. Figure: **System Architecture:** An ESP32-CAM module is configured in Wi-Fi Access Point (AP) mode, hosting a private network. In this configuration, the ESP32-CAM acts as a soft-AP, creating its own Wi-Fi hotspot to which the driving console (laptop) and a secondary microcontroller (ESP32-WROOM) can connect as clients. The ESP32-CAM runs an HTTP web server on this network to handle incoming requests. Both the laptop dashboard and the ESP32-WROOM join the ESP32-CAM's SSID using preconfigured credentials, obtaining IP addresses (e.g. 192.168.4.x) on the same subnet. This setup eliminates the need for an external router or Internet access, enabling a self-contained, local IoT network. The ESP32-CAM AP supplies DHCP and listens for HTTP/TCP connections from the clients; in practice, it is programmed to serve status pages and to relay commands as needed. Overall, this architecture places the ESP32-CAM as the central communication hub: it receives sensor data and video from clients, and forwards control commands.

Both the laptop and the ESP32-WROOM communicate via standard Wi-Fi networking (TCP/IP). The Python-based dashboard on the laptop uses network calls (WebSocket messages) to interact with the ESP32-CAM server. For example, the dashboard periodically polls the ESP32-CAM's web API to retrieve sensor readings (such as the MPU-6050 motion data) and video frames. Likewise, when control actions are needed, the dashboard sends command messages to the ESP32-CAM server, which in turn relays them to the ESP32-WROOM. The communication protocol is built custom binary or UDP schemes is used. The ESP32 modules use the Arduino WiFi API for AP mode and client mode. Both devices configure WiFi.mode(WIFI_AP) (for the CAM) and WiFi.mode(WIFI_STA) (for the WROOM) using the Arduino core libraries. By hosting the ESP32-CAM as an AP, nearby stations (the laptop and the WROOM) connect directly to it. This ensures low-latency, local-area data exchange for real-time control.

On the laptop, a Python dashboard application implements the core data processing and user interface. This dashboard uses OpenCV (Open Source Computer Vision Library) to process the live video stream captured by the ESP32-CAM. The camera module on the ESP32-CAM streams a video feed (e.g. via MJPEG) which the dashboard fetches and decodes. Using OpenCV, the software locates the driver's face and eyes in each frame. Facial landmark techniques (e.g.Dlib models) identify key points such as eye corners and mouth corners. The dashboard computes the Eye Aspect Ratio (EAR) by measuring the distances between eyelid landmarks; a low EAR sustained over several frames indicates closed eyes and drowsiness. Concurrently, a pre-trained convolutional neural network (CNN) model evaluates each frame (or a sequence of frames) to predict collision risk. This model may be a lightweight CNN trained on dashboard camera footage, designed to recognize scenarios like sudden lane departure or approaching obstacles. The vision model is implemented in Python using Keras/TensorFlow or PyTorch. In our prototype, the workflow is analogous to published driver-monitoring systems: an in-car camera captures facial features, OpenCV extracts these features, and a deep-learning classifier infers the driver's alertness. Indeed, Keras is a high-level neural networks API and OpenCV provides the image-processing tools the SNAPS dashboard leverages both libraries for real-time inference. A trained CNN analyzes either facial patterns (for drowsiness) or the scene ahead (for collision cues) to output alert scores. The dashboard logic continuously evaluates these outputs: if the drowsiness score or collision risk exceeds a threshold, it triggers a warning.

When an alert condition is detected, the dashboard sends control commands back through the network to the vehicle. Specifically, the dashboard's decision logic issues structured commands (e.g. UDP packets) over Wi-Fi to the ESP32-CAM server. The ESP32-CAM firmware listens for these incoming command requests (for example, via a TCP server or REST endpoint) and then forwards the commands to the ESP32-WROOM, which is also connected as a station on the same Wi-Fi network. (The forwarding may be direct over the Wi-Fi network – i.e., the Python script could also target the WROOM's IP address – or the CAM could act as a proxy.) Typical commands include: motor control signals (such as target speed or braking), requests to read the latest sensor data (MPU-6050 readings), and messages to display on the LCD. On receiving a motor control command, the ESP32-WROOM actuates the L298N dual H-bridge driver to adjust the DC motors 'speed and direction. The L298N module accepts up to 35V and drives two motors at up to 2A each, controlling each motor's direction and PWM speed input. For motion monitoring, the ESP32-WROOM continuously reads the MPU-6050 inertial measurement unit (a 6-axis accelerometer/gyroscope) via I²C. The MPU-6050 is a 3-axis accelerometer plus 3-axis gyroscope chip for motion sensing. Its real-time acceleration and rotation data are sent to the dashboard (on demand or periodically) to complement the vision-based predictions; for example, a sudden spike in linear

acceleration could confirm a collision event. The ESP32-WROOM also drives a 16×2 character LCD to present status and alerts. This LCD uses the standard HD44780 controller and operates in 4- or 8-bit parallel mode. The microcontroller updates the display text (e.g. "Drowsiness Alert", "Braking", or sensor values) in real time. Thus, the control subsystem (ESP32-WROOM + peripherals) implements the actuation (motors) and immediate alerting (display and buzzer) based on commands from the dashboard. The system operation can be summarized in the following procedural flow:

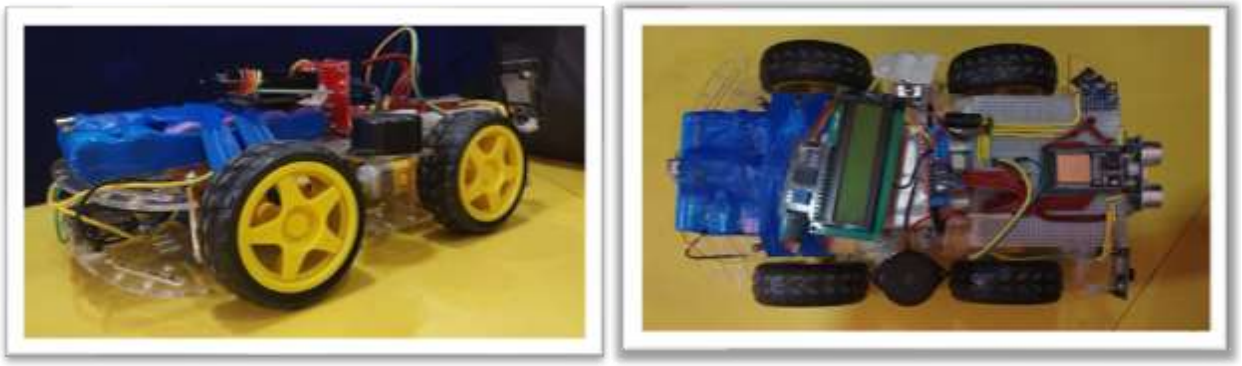
A. Data Acquisition: The ESP32-CAM continuously captures frames from its camera (up to 1–10 fps, depending on resolution) and either buffers them or streams them to the dashboard. The Python dashboard retrieves these frames over HTTP. At the same time, the ESP32-WROOM regularly reads the MPU-6050 to obtain linear acceleration and angular velocity data, making this motion data available on request. The dashboard polls or subscribes to this data via the ESP32-CAM (which may simply forward the current MPU-6050 readings in response to a query).

B. Perception Processing: For each video frame, the dashboard applies computer-vision routines. It uses OpenCV to detect the driver's face and eyes, computing an Eye Aspect Ratio or similar fatigue metric. For example, it marks eye landmarks and computes EAR; if both eyes remain closed (low EAR) for several seconds, the system flags drowsiness. Concurrently, the dashboard passes the frame through a trained neural network model that detects hazardous driving patterns (e.g. sudden obstacle appearance or erratic vehicle motion) to predict accidents. This CNN-based accident model operates on either image sequences or fused image+sensor data. The decision logic in the Python code evaluates the outputs: if either drowsiness or accident likelihood crosses a preset threshold, an alert condition is raised.

C. Command & Control: When an alert is triggered, the dashboard sends a command packet to the ESP32-CAM. For example, a "SLOW_DOWN" command might be issued upon detecting drowsiness, whereas a "BRAKE" command might be sent when an imminent collision is predicted. The ESP32-CAM receives this command (e.g. via an HTTP POST) and relays it to the ESP32-WROOM—either by sending a separate Wi-Fi message to the WROOM's IP or by invoking a callback in the CAM firmware. The ESP32-WROOM interprets each command: it sets PWM outputs on the L298N to adjust motor speed, toggles motor direction pins if needed, and updates the LCD message with a status string.

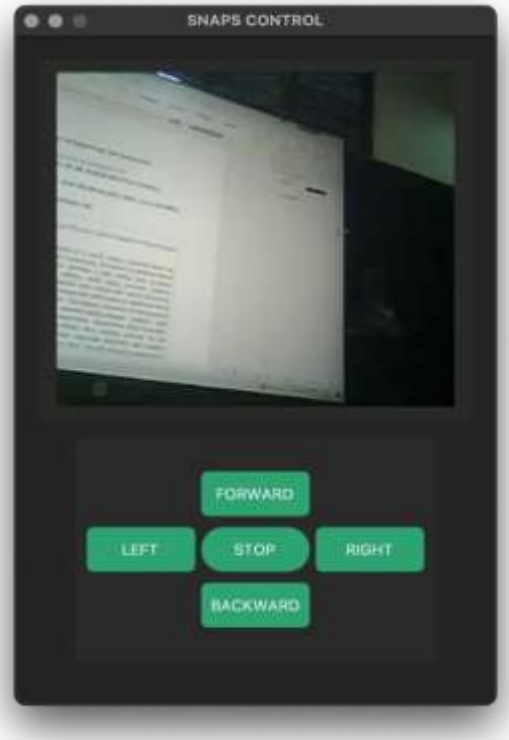
D. Actuation & Alert: The ESP32-WROOM carries out the issued commands. Driving signals from the L298N cause the vehicle's DC motors to slow, stop, or reverse as directed. Meanwhile, the LCD displays real-time alerts (e.g. "Driver Drowsy!", "Emergency Brake!"). Optionally, an audible buzzer may also be driven to warn the driver. If sensor data from the MPU-6050 confirms a severe impact (e.g. acceleration beyond a crash threshold) or if the dashboard's emergency protocol is invoked, the system enters "Emergency Alert" mode. In this mode, the dashboard initiates a higher-level alert: it could send an SMS or email to preconfigured contacts (via a connected smartphone or IoT gateway) with the vehicle's status and location. The ESP32-CAM may further broadcast a distress packet on the Wi-Fi network. The LCD will show an emergency indicator, and the system logs all critical data.

E. Loop Continuation: If no critical event occurs, the system continues monitoring in the loop from step 2. In normal driving, the dashboard and microcontrollers run continuously until the vehicle is shut off or the system is manually reset. In the case of an emergency alert dispatch, the loop may terminate the driving operation until a safe condition is restored.



Hardware and Software Components: The SNAPS prototype is built with off-the-shelf hardware and open-source software libraries. Key components include:

- **ESP32-CAM:** An ESP32-based development board with an onboard OV2640 camera. It serves as the Wi-Fi access point and video server.
- **ESP32-WROOM-32:** A second ESP32 module (generic WROOM form factor) acting as a Wi-Fi station for actuator control.
- **MPU-6050 Sensor:** A 6-axis inertial sensor (3-axis accelerometer + 3-axis gyroscope) for motion monitoring.
- **L298N Motor Driver:** A dual H-bridge driver board (ST L298N chip) to control DC motor speed and direction.
- **DC Motors and Vehicle Frame:** Four DC motors (one per wheel) powered by a battery pack, providing motion to the prototype vehicle.
- **16×2 LCD Display:** A character LCD with HD44780-compatible controller for status messages.
- **Power Supply:** A suitable battery or power bank supplying 5V–12V to the ESP32 boards and motors.



Software and firmware include:

- **Arduino/ESP32 SDK:** Both ESP32 modules are programmed via the Arduino IDE (Espressif Arduino core). Libraries used include WiFi.h for networking, WebServer.h (or similar) for HTTP, Wire.h for I²C with the MPU-6050, and a LiquidCrystal library (or direct GPIO code) for the LCD interface.
- **Python 3.x Dashboard:** The laptop runs Python with libraries such as OpenCV (for real-time image processing), NumPy (numerical arrays), and TensorFlow/Keras (for the deep-learning models). The dashboard also uses networking libraries (e.g. requests or socket) to communicate over Wi-Fi, and any GUI toolkit (if a graphical interface is implemented).
- **Machine Learning Models:** A pre-trained CNN model (developed using Keras) is embedded in the dashboard to classify driver state. The model was trained offline on a dataset of labeled driving videos (normal vs. fatigue/accident scenarios) and loaded for inference at runtime.
- **Additional Tools:** Optionally, libraries like Dlib for facial landmark detection or matplotlib for plotting, as needed for development and debugging.

All code was organized to operate in real-time: the ESP32 firmwares loop at up to 100Hz for sensor reading and command execution, while the Python dashboard processes video at ~5–10 frames per second. Through this tightly integrated hardware-software arrangement, the SNAPS system can detect driver drowsiness and impending collisions, respond by controlling the vehicle's actuators, and dispatch alerts autonomously.

Flowchart (System Process): The above steps can be viewed as a text-based flowchart:

- **Startup:** ESP32-CAM sets up Wi-Fi AP & server; ESP32-WROOM and laptop connect.
- **Sensor/Video Stream:** Camera frames and MPU-6050 data stream to dashboard.
- **Analysis:** Dashboard uses OpenCV to detect eyes/facial features and runs CNN-based accident prediction.
- **Decision:** If drowsy or high-risk detected, trigger action.
- **Command:** Dashboard sends control commands (motor/brake/LCD) via ESP32-CAM to ESP32-WROOM.
- **Actuation:** ESP32-WROOM drives motors (via L298N, updates LC, and sounds alarms.
- **Emergency Alert:** On critical conditions, system dispatches emergency notifications.
- **Loop:** Return to step 2 until shutdown or fault.

References

1. A supervised machine learning algorithm, specifically the k-Nearest Neighbors (k-NN) algorithm, is employed to classify the voice commands into predefined categories such as "forward," "backward," "left," "right," and "stop" [1].
2. To process the video feed, the ESP32-CAM module captures real-time video, which is transmitted wirelessly to a computer or smartphone over Wi-Fi using MJPEG streaming protocol [2].
3. The camera's resolution and frame rate are configured to balance clarity and performance, considering the constraints of the ESP32-CAM hardware [3].
4. For wireless communication, ESP-NOW protocol is used between ESP32-CAM and NodeMCU to send data with low latency [4].
5. The system uses serial communication (UART) to transmit instructions from NodeMCU to Arduino Uno, which directly controls the car's motors via the L293D motor driver shield [5].

6. The L293D shield facilitates the control of four DC motors, enabling precise movement and turning actions based on received commands [6].