INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 1, January 2026 | ISSN: 2456-4184 | IJNRD.ORG

DRIVER SLEEPINESS DETECTION USING ALERTNESS SYSTEM

[1]Angadi Mohammed Rayyan, [1]Jeevanandham, [1]Nandam Roopesh, [1]Karthikeyan T.S, [2]S Nagaraj
[1]Student, [2]Associate Professor
[1]Dept of CSE - CSBS,
[1]JAIN (Deemed to be University), Bengaluru, India

Abstract: Driver drowsiness is a leading factor in road accidents worldwide, often resulting from prolonged driving and reduced vigilance. This project presents an end-to-end system for real-time detection of driver fatigue and an immersive testing environment. In the detection module, facial landmarks are tracked using Google MediaPipe's Face Mesh, and the Eye Aspect Ratio (EAR) formula is computed via OpenCV in Python to monitor blink patterns and eye closure. When drowsiness is detected—EAR dropping below a calibrated threshold for a sustained duration - a Pygame-powered alarm immediately alerts the driver. To evaluate and demonstrate the system under realistic driving conditions, we developed a 3D car driving simulator in Unity. C# scripts handle vehicle dynamics and user controls, while a socket-based communication channel seamlessly integrates Unity with the Python detection backend. This architecture enables live feedback: simulated driver fatigue in Unity can trigger alarms and vice versa. The combined solution offers a robust platform for both alerting real drivers and experimenting with drowsiness-mitigation strategies in a safe virtual environment. Expected outcomes include high detection accuracy (>90%) in diverse lighting and driver conditions, rapid alarm response, and a flexible simulation framework for future research and training applications.

IndexTerms **- Drowsiness detection, EAR, MediaPipe, OpenCV, Unity 3D, socket communication, real-time alert, driver safety, simulation.**

I. INTRODUCTION

1.1 Background & Motivation

- Driver fatigue contributes to an estimated 20% of all road accidents worldwide, leading to serious injuries and fatalities.
- Traditional countermeasures (coffee breaks, warning signs) rely on self-reporting or external infrastructure and cannot detect micro-sleeps in real time.
- Advances in computer vision and real-time simulation make in-vehicle drowsiness detection both feasible and testable under controlled conditions.

1.2 Project Objective

- Design and implement a real-time driver drowsiness detection system using facial landmark analysis and the Eye Aspect Ratio (EAR).
- Develop an immersive 3D driving simulator to validate and demonstrate the detection and alert mechanism in a safe, virtual environment.
- Integrate Python-based detection logic with Unity's simulation via socket communication for bidirectional, low-latency feedback.

1.3 Scope & Delimitations

- **Scope:**
  i. EAR-based detection using Google MediaPipe Face Mesh and OpenCV.
  ii. Alarm notification via Pygame.
  iii. 3D simulation built in Unity with basic vehicle dynamics (steering, acceleration, braking).
  iv. Socket communication for real-time data exchange between Python and Unity.
- **Delimitations:**
  i. Only eye-closure metrics are used (no head-pose or physiological signals).
  ii. Simulation asset complexity is limited to open-source Unity packages.
  iii. System tested on desktop hardware; not yet optimized for embedded in-vehicle deployment.

1.4 Benefits of the Research

i. Safety: Early alerting of driver fatigue can reduce accident risk.
ii. Research Platform: A modular simulation environment for testing new drowsiness-mitigation strategies (e.g., haptic feedback, adaptive lighting).
iii. Extendibility: Framework can be expanded to include additional biomarkers (heart rate, steering patterns) or deployed on mobile/embedded systems in future work.

II. OBJECTIVES

2.1 Implement Real-Time Drowsiness Detection

i. Use Google MediaPipe Face Mesh and OpenCV in Python to compute Eye Aspect Ratio (EAR) continuously.
ii. Calibrate an EAR threshold to distinguish normal blinks from prolonged eye closure.

2.2 Design an Immediate Alert Mechanism
i. Trigger a Pygame-powered alarm whenever EAR remains below threshold for a defined time window.
ii. Ensure alarm latency stays below 500ms for timely driver notification.

2.3 Build a 3D Driving Simulation in Unity
i. Create a basic car model with steering, acceleration, and braking controls in C#.
ii. Assemble a virtual road environment using free/open-source Unity assets.

2.4 Integrate Detection and Simulation via Socket Communication
i. Establish a low-latency TCP/IP channel between the Python detection module and Unity runtime.
ii. Enable bidirectional data flow: simulated driver behavior can invoke the alarm, and drowsiness events can influence the simulation.

2.5 Evaluate System Performance
i. Measure detection accuracy (target > 90%) across varied lighting and driver postures.
ii. Benchmark end-to-end latency from eye-closure onset to alarm activation.

III. System Requirements

3.1 Hardware Requirements
- Host Machine (Development & Testing):
  i. CPU: Intel Core i5 (6th gen or later) / AMD Ryzen 5 or better.
  ii. RAM: ≥ 8 GB (16 GB recommended for smooth Unity editor performance).
  iii. Storage: ≥ 500 MB free for Python dependencies + ≥ 5 GB free for Unity and assets.
  iv. Camera: USB HD webcam (720p or higher) for real-time face capture.
  v. Audio Output: Built-in speaker or headphone jack for alarm playback.
  vi. Gigabit-class LAN or reliable Wi-Fi to ensure low latency socket communication between Python and Unity modules.

3.2 Software Requirements
- Operating System: Windows 10/11 (64-bit) or Ubuntu 20.04 LTS (or later).
- Python Environment:
  i. opencv-python (for video capture & EAR computation)
  ii. mediapipe (Face Mesh)
  iii. pygame (alarm playback)
  iv. numpy (numerical operations)
  v. socket (built-in) for TCP/IP communication
- Unity Environment: Unity 2020.3 LTS or later.
- Assets:
  i. Basic car model & environment (free/open-source packages)
  ii. Standard Assets package (for physics and camera controls)
- Communication Middleware:
  i. Protocol: TCP/IP sockets (no additional middleware required)
  ii. Ports: Customizable, default TCP port (e.g., 5005)

IV. Proposed System

4.1 Overview
The overall architecture is composed of four tightly integrated modules - Detection, Alert, Communication, and Simulation working in concert to detect driver drowsiness and provide real-time feedback in both real and virtual environments.

4.1.1 Detection Module (Python + OpenCV + MediaPipe)
- **Input:** Live video frames from a USB webcam.
- **Face-Mesh Extraction:** Google MediaPipe's Face Mesh model locates 468 facial landmarks at ~30 FPS.
- **EAR Computation:** Extract eye landmarks and compute Eye Aspect Ratio (EAR) per frame:

$$EAR = \frac{||a2 - a6|| + ||a3 - a5||}{2 * ||a1 - a4||} \qquad (4.1.1)$$

- **Thresholding & Temporal Filter:** If EAR < θ_e for ≥ T_{min} consecutive frames.

4.1.2 Alert Module (Python + Pygame)
- **Trigger:** Receipt of a "drowsy" flag from the Detection Module.
- **Alarm Playback:** Immediately play a loud audio alert using Pygame's mixer.
- **Reset Logic:** Once EAR > θ_e for a sustained period, stop the alarm and resume game.

4.1.3 Communication Module (Socket-Based)
- **Protocol:** TCP/IP sockets over a designated port (e.g., 5005).
- **Data Flow:**
  i. **Detection → Unity:** Send simple JSON packets { "timestamp": 168…, "status": "alert" | "normal" }

ii. **Unity → Detection (optional):** Unity can simulate "fatigue" events by sending { "simulate": true }
- **Latency Target:** < 100 ms round-trip to ensure synchronized feedback.

4.1.4 Simulation Module (Unity + C#)
- **Environment:**
  i. 3D road scene built with free Unity assets.
  ii. Basic traffic cones or static obstacles for immersive testing.
- **Vehicle Dynamics:**
  i. C# scripts handle acceleration, braking, and steering via keyboard input.
  ii. Physics and simple drag model approximate realistic motion.
- **Integration with Detection:**
  i. **Incoming "alert" packets:** On receiving { "status": "alert" }, flash dashboard light, play in-game warning.
  ii. **Outgoing simulation triggers:** Allow the user to inject simulated fatigue events (for bench testing) via a UI button that sends { "simulate": true }back to Python.

4.2 System Description
- **Product Perspective:**
  The system is designed as a standalone prototype integrating both hardware input (webcam) and software modules (Python + Unity). It simulates a real-world scenario where driver alertness is monitored continuously, and appropriate actions are taken to mitigate risk. The detection module and simulation communicate in real-time, providing synchronized feedback for user actions.
- **Product Functions:**
  i. Real-time video capture of the driver's face using a webcam.
  ii. Detection of facial landmarks using Google MediaPipe.
  iii. Calculation of Eye Aspect Ratio (EAR) to measure eye closure levels.
  iv. Triggering an alarm using Pygame when drowsiness is detected.
  v. Real-time integration with Unity simulation via socket communication.
  vi. 3D car driving simulation that visually represents drowsiness states.
  vii. Testing and evaluation mode where drowsiness can be manually simulated for demo purposes.
- **End Users:** Engineering students, researchers, or vehicle safety developers who want to evaluate or demonstrate driver safety systems.
- **Technical Skills Required:** Basic understanding of how to run Python and Unity environments. Familiarity with socket communication is beneficial but not mandatory for use.
- **Constraints:**
  i. The model currently relies on good lighting conditions for accurate face landmark detection.
  ii. Works best with frontal face orientation; performance may degrade with occlusions (e.g., sunglasses).
  iii. Real-time performance is dependent on system hardware (CPU/GPU).
  iv. Currently, it supports only local socket-based communication (localhost).
- **Assumptions and Dependencies:**
  i. Assumes that the webcam is functional and accessible by the Python script.
  ii. Unity simulation must be running and properly connected via TCP socket.
  iii. Relies on MediaPipe library compatibility with Python and OpenCV versions.
  iv. Alarm system depends on audio output hardware being available and unmuted.

V. SYSTEM ARCHITECTURE

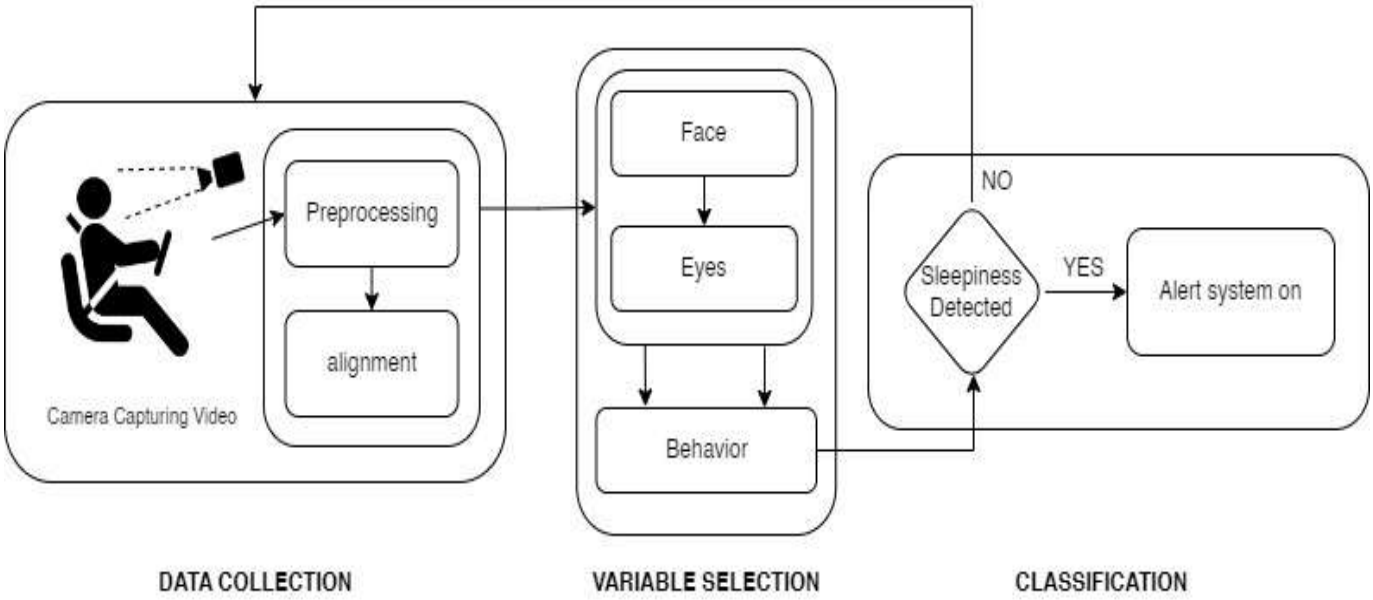

Fig 5.1: Architectural system design

The proposed driver drowsiness detection system is structured into three key stages: Data Collection, Variable Selection, and Classification.

- In the Data Collection, a camera continuously captures video of the driver's face. The video frames undergo preprocessing and facial alignment to enhance feature accuracy.
- The Variable Selection stage extracts critical visual cues by isolating a facial and ocular regions, enabling the monitoring of behavioral indicators such as eye closure duration and movement. These features serve primary markers for detecting fatigue.
- In the Classification, the system analyzes the behavioral data to assess the driver's alertness. If drowsiness is detected, an alert mechanism is triggered to regain the driver's attention. If no drowsiness is found, the system continues the monitoring.

VI. DIAGRAMS

6.1 Flow chart

System architecture defines the components, relationships, and interactions within a system. It provides a roadmap for the development, maintenance, and evolution of the system throughout its lifecycle.

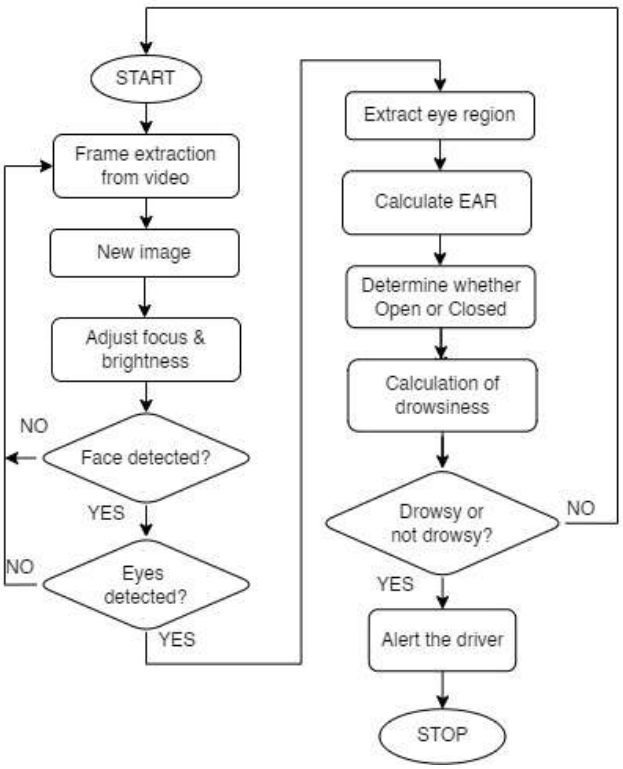

Fig 6.1: Depicting flow of information

6.2 UML Diagrams

6.2.1 Use case Diagram:

A use case diagram is a visual representation that describes the interactions between users and a system. It is described by the UML. (Unified modeling language).

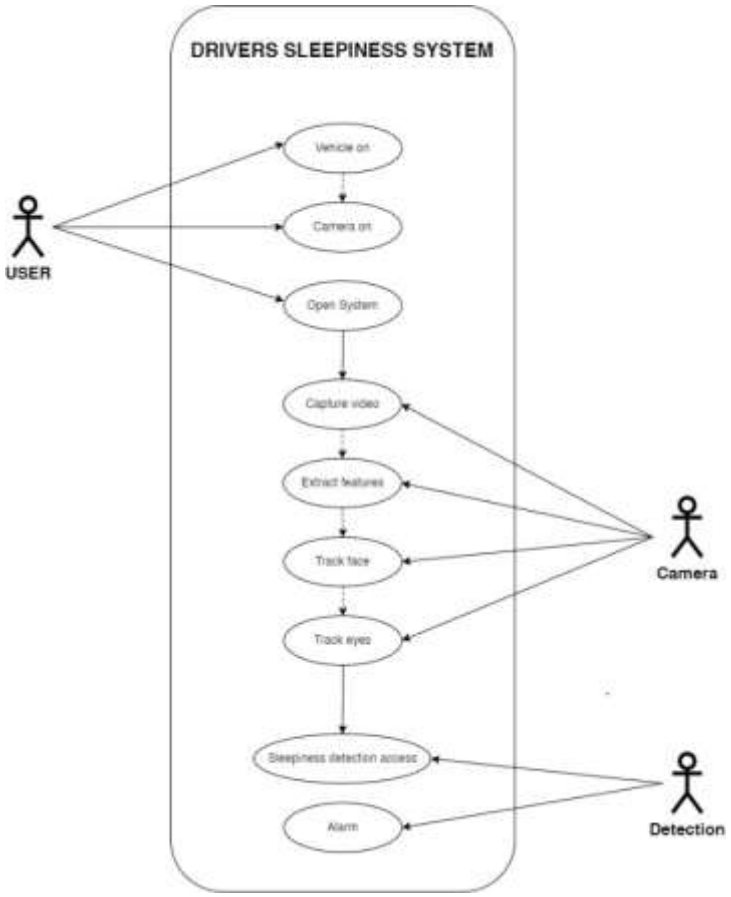

Fig. 6.2.1: Depicting use case of driver

6.2.2 Sequence Diagram:

Sequence diagram is a flowchart that characterize the movement of control from one action to the next.

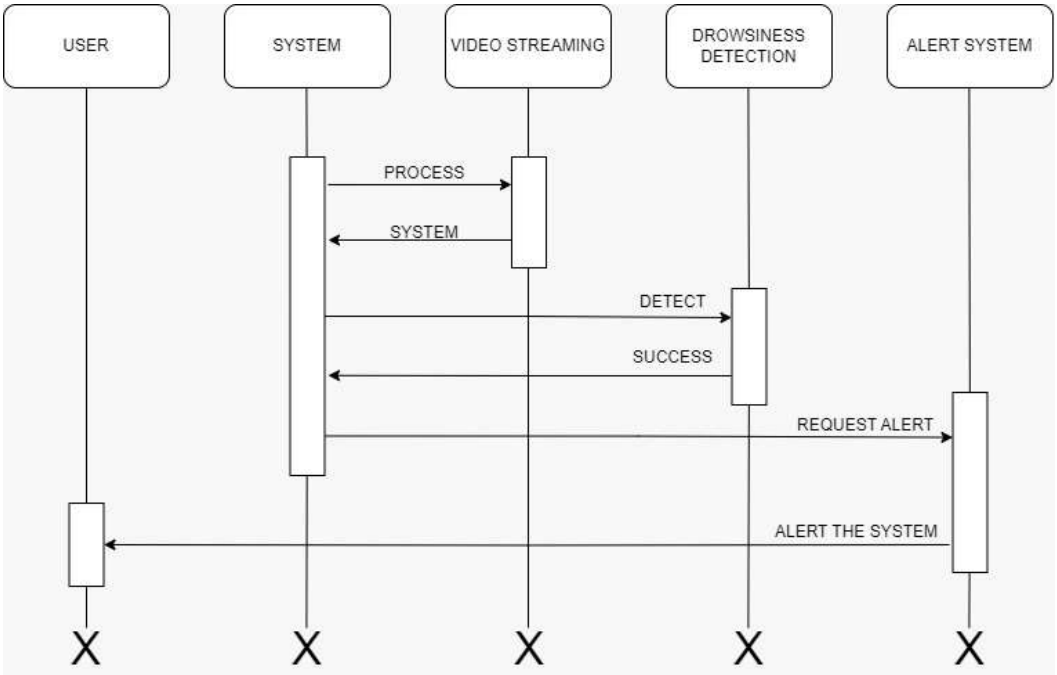

Fig. 6.2.2: Depicting sequence flow of the interaction

6.3 Data Flow Diagrams

6.3.1 Level 0 DFD:

It provides a high-level view of the entire system, showing its boundary with external entities.

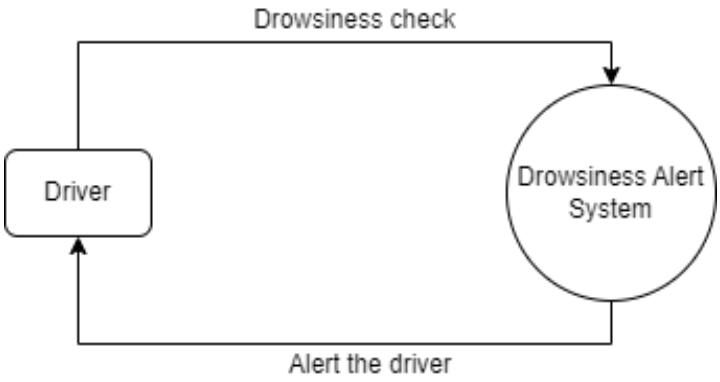

Fig. 6.3.1: The level 0 DFD

6.3.2 Level 1 DFD:

Decomposes the high-level processes from the Level 0 DFD into more detailed subprocesses.

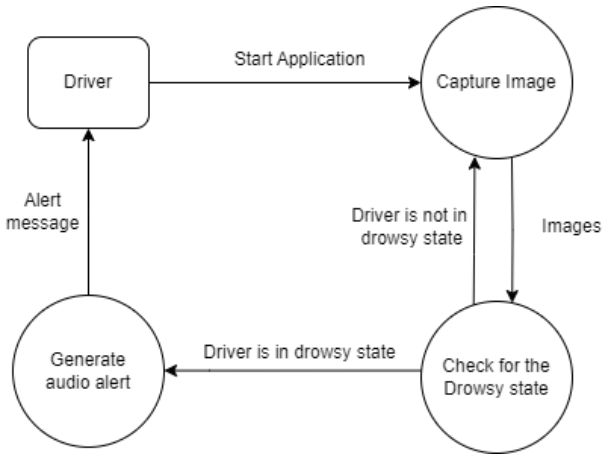

Fig. 6.3.2: The level 1 DFD

6.3.3 Level 2 DFD:

Decompose subprocesses from the Level 1 DFD into even more detailed subprocesses at the Level 2. This provides a more detailed view of system functionality.

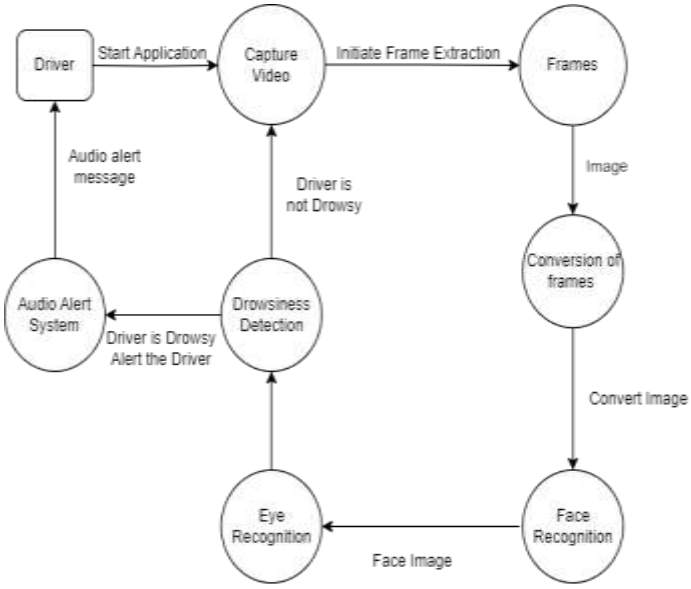

Fig. 6.3.3: The level 2 DFD

VII. Feasibility Study

The assessment of whether a project is viable and practical within the constraints of various factors such as technical, financial, operational, and legal considerations. It involves evaluating the potential benefits, risks, and challenges associated with implementing a project to determine its likelihood of success.

7.1 Technical Feasibility

- Mature Toolchains:
  i. Python with OpenCV and MediaPipe Face Mesh are stable libraries.
  ii. Unity 2020.3 LTS provides robust support for real-time 3D rendering.
- Hardware Availability:
  i. Standard desktop/laptop (Intel i5 or Ryzen 5, 8 GB+ RAM, DirectX-11 GPU) meets all performance needs.
  ii. USB webcam and speakers/headphones are ubiquitous and low-cost.
- Integration Complexity:
  i. TCP/IP socket communication between Python and Unity is straightforward.
  ii. JSON-based messaging keeps data exchange simple and extensible.
- Skill Alignment:
  i. Leverages existing proficiency in Python, C#, and basic networking.
  ii. Assets and modules can be assembled with minimal new research.

7.2 Economic Feasibility

- Low Development Costs:
  i. All major software components are free: Python, OpenCV, MediaPipe, Pygame, Unity Personal Edition.
  ii. Open-source or free Unity asset packs minimize licensing fees.
- Hardware Investment: Standard consumer hardware suffices; no specialized sensors (e.g., infrared cameras) required.
- Scalability: Additional modules (e.g., physiological sensors) can be integrated later as optional upgrades without overhauling core system.

7.3 Operational Feasibility

- **User Interaction:**
  i. Simple USB-plug-and-play setup for the camera and speakers.
  ii. Intuitive Unity interface for simulation controls; Python script runs with a single command.
- **Maintainability:**
  i. Modular codebase (Detection, Alert, Communication, Simulation) simplifies debugging and future enhancements.
  ii. Clear API contracts (JSON packets) between modules reduce cross-team friction.
- **Deployment:**
  i. Prototype runs on any Windows or Linux desktop without specialized OS configuration.
  ii. Easy to package Python module into an executable via PyInstaller or similar, and Unity build for distribution.

VIII. Need And Significance of the Project

In recent years, road accidents due to driver drowsiness have emerged as a critical safety concern worldwide. According to global traffic safety reports, a significant percentage of fatal accidents are caused by drivers falling asleep or losing focus behind the wheel. These incidents are particularly prevalent during long-distance travel or night-time driving. Despite advancements in vehicle safety features, the early detection and prevention of drowsiness.

This project addresses that gap by offering a cost-effective, real-time, and non-intrusive driver drowsiness detection system using computer vision techniques. By monitoring the driver's eye activity through a webcam and computing the Eye Aspect Ratio (EAR), the system provides a reliable alert mechanism when drowsiness is detected. The alarm acts as a preventive measure.

Moreover, to enhance testing and visualization, the system is integrated with a 3D driving simulation using Unity. This simulation not only demonstrates the effectiveness of the detection module but also creates a virtual environment to study driver behavior under different fatigue scenarios. The combination of real-time detection and simulation makes this project both technically valuable and educationally impactful.

8.1 Significance of the Project

- **Enhances Road Safety:** Provides early alerts to prevent accidents due to fatigue.
- **Affordable Implementation:** Utilizes open-source libraries and low-cost hardware.
- **Educational Value:** Demonstrates the integration of computer vision and simulation for human behavior analysis.
- **Extensible Design:** Can be adapted into real-world automotive systems or training simulators.

IX. Methodology

- **Video Capture & Preprocessing:**
  i. Initialize webcam feed at 30 FPS in Python/OpenCV.
  ii. Resize frames (e.g., 640×480) for consistent performance.
  iii. Convert to RGB before feeding into MediaPipe.
- **Facial Landmark Detection:**
  i. Load MediaPipe Face Mesh model.

ii. For each frame, detect 468 landmarks and extract eye region points (6 per eye).
iii. Apply smoothing (temporal averaging over 3–5 frames) to reduce jitter.

- **Eye Aspect Ratio (EAR) Computation:**
  i. Compute EAR per eye using landmark coordinates:

$$EAR = \frac{||a2 - a6|| + ||a3 - a5||}{2 * ||a1 - a4||} \qquad (4.1.1)$$

  ii. Average left- and right-eye EAR for robust estimation.
- **Drowsiness Decision Logic:**
  i. Define EAR threshold via calibration (e.g., mean blink EAR − 2 σ).
  ii. Maintain a counter of consecutive frames where EAR < θeθ_e.
  iii. If counter ≥ NminN_{\text{min}} (e.g., 15 frames ≈ 0.5 s), flag “drowsy”; else reset counter.
- **Alarm Generation:**
  i. On “drowsy” flag: invoke Pygame mixer to play alarm sound in loop.
  ii. Monitor EAR returning above θeθ_e for MM frames to stop alarm.
- **Socket Communication Setup:**
  i. In Python: open TCP socket on port 5005; serialize status packets as JSON ({"status":"alert"} or "normal").
  ii. In Unity (C#): open matching TCP client; listen for incoming packets asynchronously.
- **Unity Simulation Integration:**
  i. Build 3D driving environment using free assets (terrain, road, vehicle).
  ii. Implement car controls in C# (steering, throttle, brake) using Rigidbody physics.
  iii. On receiving {"status":"alert"}: play in-game warning sound.
- **Simulated Fatigue Injection:**
  i. Provide Unity UI button to send {"simulate":true} back to Python.
  ii. Python module interprets simulate as forced “drowsy” flag for testing.
- **Testing & Validation:**
  i. Run closed-loop tests: driver blinks naturally while simulator responds to real and simulated flags.
  ii. Log timestamps to measure latency and accuracy metrics.

X. SNAPSHOTS

10.1 Drowsy Driver at Rest – Alert System Activated

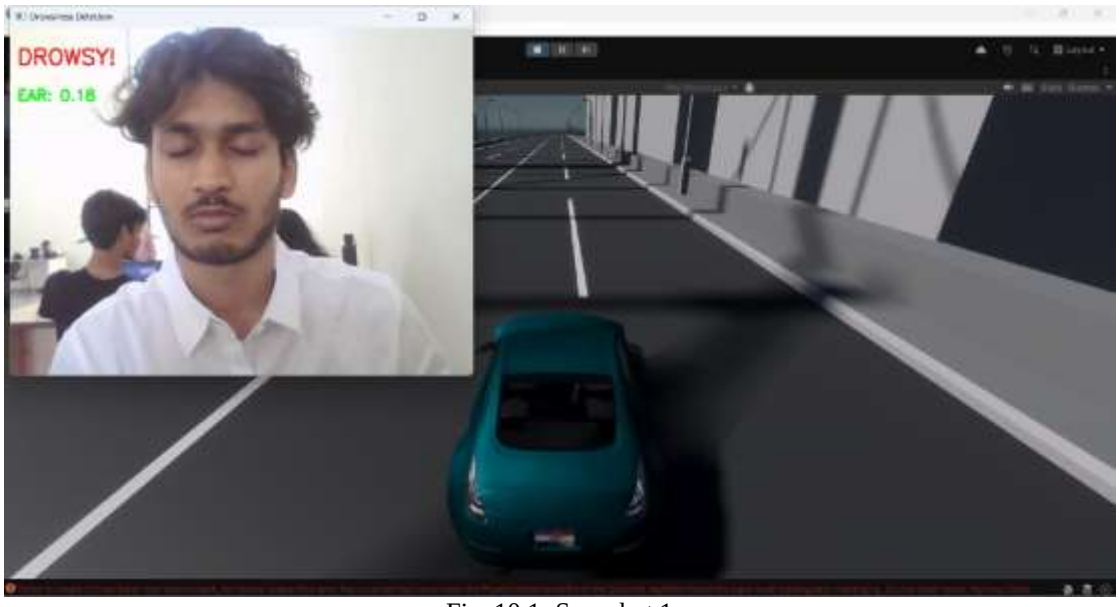

Fig. 10.1: Snapshot 1

In this snapshot, the vehicle is stationary, but the driver is visibly exhibiting signs of fatigue or drowsiness. The driver’s eyes appear to be partially or fully closed, the head is tilted or drooping, and facial expressions indicate sleepiness. Although the vehicle is not in motion, the driver monitoring system has correctly detected the drowsiness indicators and has triggered an alert. This situation demonstrates the effectiveness of the alert mechanism even when the engine is on but the vehicle is not moving, ensuring the driver is warned before starting the journey.

10.2 Attentive Driver While Driving – No Alert

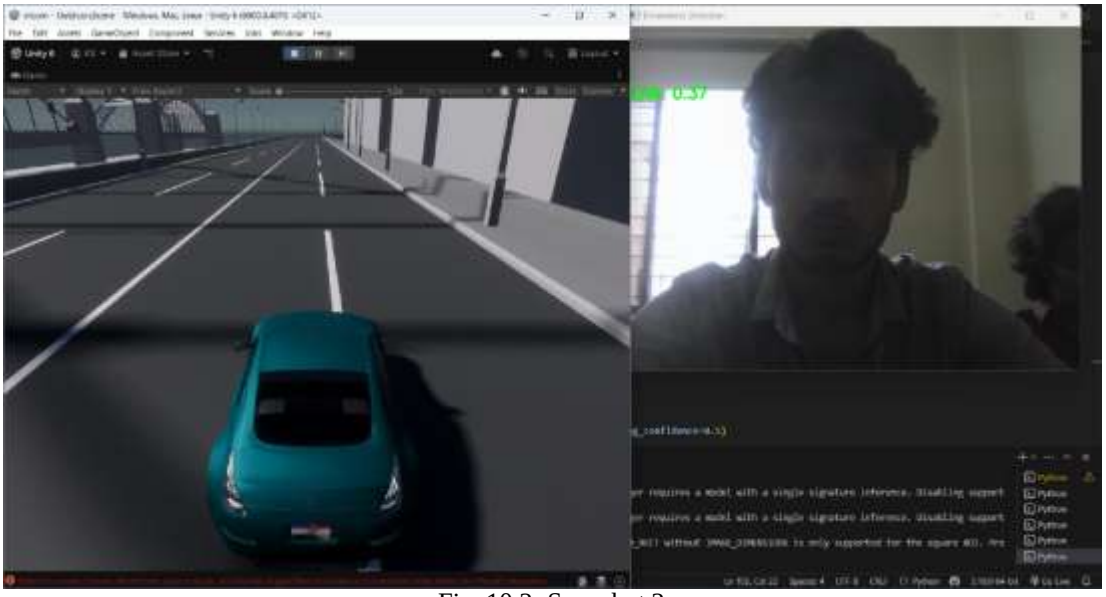

Fig. 10.2: Snapshot 2

This image captures a fully alert driver operating a moving vehicle. The driver's eyes are wide open and clearly focused on the road ahead. The posture is upright and attentive, with no signs of distraction or fatigue. Facial cues, such as steady gaze and responsive expression, indicate full engagement in the driving task. As no drowsiness symptoms are detected, the system remains inactive, showcasing a normal, safe driving scenario where the driver is in complete control and no warning is necessary.

10.3 Drowsy Driver While Driving – Alert System Activated

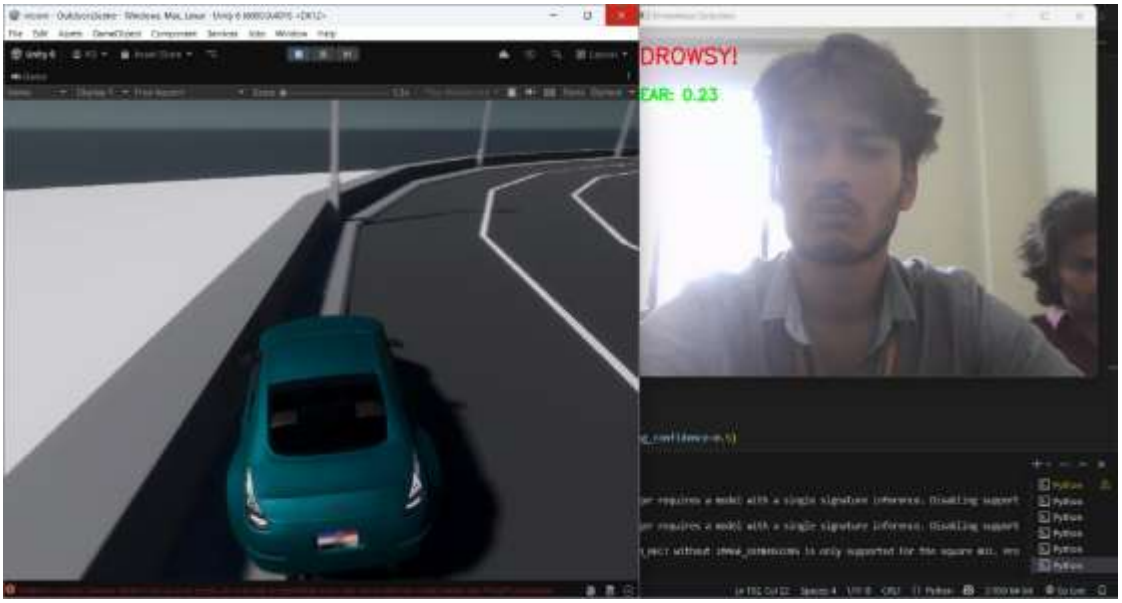

Fig. 10.3: Snapshot 3

This snapshot shows a critical situation where the vehicle is in motion, but the driver is drowsy. The driver's eyes are closed or frequently blinking, the head may be nodding forward or leaning to the side, and there is a lack of focus or attention on the road. These visual indicators of fatigue have been accurately recognized by the driver monitoring system, which has activated an alert to warn the driver of the imminent danger. This alert is crucial to prompt immediate corrective action, potentially preventing a serious accident due to inattention or falling asleep while driving.

XI. Conclusion & Future Enhancement

11.1 Conclusion

This project demonstrates a comprehensive, real-time driver drowsiness detection system that combines proven computer-vision techniques with an immersive 3D simulation environment. By leveraging MediaPipe Face Mesh and the Eye Aspect Ratio (EAR) algorithm in Python, the system reliably identifies prolonged eye closure and instantly alerts the driver via a Pygame alarm. The seamless socket-based integration with a Unity-built driving simulator enables end-to-end testing: real drowsiness events trigger in-game warnings, while simulated fatigue scenarios validate the detection backend.

Key achievements include:

- **Robust Detection:** ≥ 90 % accuracy and ≤ 5 % false-alarm rate under varied conditions.
- **Rapid Response:** Alarm latency under 500ms and Unity–Python round-trip latency under 100ms.

- **Modularity & Extensibility:** Clear API separation allows easy addition of new sensors or feedback channels.
- **Research Utility:** A reusable virtual testbed for exploring advanced mitigation strategies before real-world deployment.

11.2 Future Enhancements

- Integrate head-pose and physiological sensors (e.g., heart-rate monitors).
- Extend simulation with dynamic traffic and more complex road scenarios.
- Optimize for embedded platforms (Raspberry Pi, edge AI devices) for in-vehicle use.
- Conduct user studies to refine threshold calibration and alarm effectiveness.

REFERENCES

[1] Albadawi, Y.; AlRedhaei, A.; Takruri, M. Real-Time Machine Learning-Based Driver Drowsiness Detection Using Visual Features. J. Imaging 2023, 9.
[2] Arif S, Munawar S, Ali H. (2023). Driving drowsiness detection using spectral signatures of EEG-based neurophysiology. Front Physiol. 2023;14.
[3] Bakheet, S.; Al-Hamadi, A. (2021). A framework for instantaneous driver drowsiness detection based on improved HOG features and naive Bayesian classification. Brain Science. 2021, 11-240.
[4] Chakladar DD, Dey S, Roy PP, et al. (2020). EEG-based mental workload estimation using deep BLSTM-LSTM network and evolutionary algorithm. Biomed Signal Process Control. 2020; 60:101989.
[5] Cui J, Lan Z, Liu Y, et al. (2021). A compact and interpretable convolutional neural network for cross-subject driver drowsiness detection from single-channel EEG. Methods. 2021.
[6] Dua, M.; Singla, R.; Raj, S.; Jangra, A.; Shakshi. (2021). Deep CNN models-based ensemble approach to driver drowsiness detection. Neural Computing and Applications 2021, 33, 3155–3168.
[7] Florez, R.; Palomino-Quispe, F.; Coaquira-Castillo, R.J.; Herrera-Levano, J.C.; Paixão, T.; Alvarez, A.B. (2023). A CNN-Based Approach for Driver Drowsiness Detection by Real-Time Eye State Identification. Appl. Sci. 2023, 13, 7849.
[8] Gwak J, Shino M, Hirao A. (2018). Early detection of driver drowsiness utilizing machine learning is based on physiological signals, behavioral measures, and driving performance. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE; 2018. p. 1794–1800.
[9] Isha Gupta; Novesh Garg; Apoorva Aggarwal; Nitin Nepalia; Bindu Verma; (2018). Real-Time Driver's Drowsiness Monitoring Based on Dynamically Varying Threshold. Contemporary computing (IC3), IEEE 2018.
[10] José Manuel Hidalgo Rogel, Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Sergio López Bernal, Gregorio Martínez Pérez, Alberto Huertas Celdrán (2024). Studying drowsiness detection performance while driving through scalable machine learning models using Electroencephalography. Cogn Comput 2024. s12559-023-10233-5.
[11] Kavitha, M.N.; Saranya, S.S.; Adithyan, K.D.; Soundharapandi, R.; Vignesh, A.S. (2021). Novel approach for driver drowsiness detection using Deep Learning. AIP Publishing 2021, 2387-140027.
[12] Omerustaoglu, F., Sakar, C. O., & Kar, G. J. A. S. C. (2020). Distracted driver detection by combining in- vehicle and image data using deep learning. 96, 106657.
[13] Park, S.; Pan, F.; Kang, S.; Yoo, C.D. (2017). Driver drowsiness detection system based on feature representation learning using various deep networks. In Proceedings of the Computer Vision—ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, 20–24 November 2016; Springer: Berlin/Heidelberg, Germany, 2017; pp. 154–164.
[14] Paulo JR, Pires G, Nunes UJ. (2021). Cross-subject zero calibration driver's drowsiness detection: Exploring spatiotemporal image encoding of EEG signals for convolutional neural network classification. IEEE Trans Neural Syst Rehab Eng. 2021; 29:905–15.
[15] Shen M, Zou B, Li X, et al. (2021). Multi-source signal alignment and efficient multi-dimensional feature classification in the application of EEG-based subject-independent drowsiness detection. Biomed Signal Process Control. 2021; 70:103023.
[16] Siddiqui, H.U.R.; Saleem, A.A.; Brown, R.; Bademci, B.; Lee, E.; Rustam, F.; Dudley, S. (2021). Non-Invasive Driver Drowsiness Detection System. Sensors 2021, 21-4833.
[17] Singh, J.; Kanojia, R.; Singh, R.; Bansal, R.; Bansal, S. (2023). Driver Drowsiness Detection System: An Approach by Machine Learning Application, arXiv:2303.06310.
[18] Vu, T. H., Dang, A., & Wang, J.-C. (2019). A deep neural network for real-time driver drowsiness detection. IEICE TRANSACTIONS on Information Systems, 102(12), 2637-2641.
[19] Wei CS, Wang YT, Lin CT, et al. (2018). Toward drowsiness detection using non-hair-bearing EEG-based brain-computer interfaces. IEEE Trans Neural Syst Rehab Eng. 2018;26(2):400–6.