VISION GUARD (SMART & SAFE OVERTAKING SOLUTION)

Muhammad Abad Khan

11696

Sadia Naz

11965

Tooba Anwar

13619

Warisha Jamal

64822

Under the supervision of

Umair Qureshi

College Of Computing & Information Science

KIET North Nazimabad, Karachi

ABSTRACT

Road accidents happens because of drivers poor visibility they can't see what's ahead when they are stuck behind large vehicles like trucks. One of the risk on highways is overtaking, especially when visibility is poor due to blind spots. Drivers may not be able to decide whether it is safe to overtake, which can lead to dangerous accidents.

To solve this problem, our project **"Vision-Guard"** introduces a smart safety system for trucks. This system uses a front-facing camera (attached on the front of truck) to capture a live video of front road. This video is displayed on a rear screen placed at the back of the truck. This helps the drivers to clearly see what is in front of the truck and make safe overtaking decisions.

Furthermore, the system uses sensors to detect nearby vehicles and measure their distance. If any vehicle comes too close, the system gives audio and visual alert (buzzer starts buzzing and led blinks), vehicle detection, speed calculation is also added.

The goal of this project is to reduce road accidents by improving visibility, increasing awareness, and helping drivers to make safer overtaking decisions. Vision-Guard is a solution for safer highways and a smart use of technology to protect lives.

ACKNOWLEDGEMENT

In the name of Allah, the most Gracious and the Most Merciful.

Peace and blessing of Allah be upon Prophet Muhammad ﷺ

First, praise of Allah, for giving us this opportunity, the strength and the patience to complete our FYP finally, after the challenges and difficulties. We would like to thank our supervisor **Sir Muhammad Umair Qureshi** for his guidance, motivation and most his significant contribution in this project, expert **Miss Hira Anwar, Miss Nida, Miss Eman Ayesha, Sir Maqsood Razi** for giving us the opportunity to work on this project. We would also like to thanks our parents for financial and moral support and our friends who have helped and motivated us throughout. May Allah reward them all abundantly. Ameen.

DEDICATION

This report is dedicated to KIET University, our Teacher, our Supervisor, our Parents, our fellow colleagues and the hard-working students of KIET, with a hope that they will succeed in every aspect of their Academic Career and this project may help them in any aspect of their life.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. Introduction

Road safety is a big issue, especially on busy roads when following large vehicles like trucks. These vehicles often block the driver's view, making it hard to see if the road ahead is clear for overtaking. This becomes even more dangerous on highways, where quick and accurate decisions are needed. Blind spots around trucks can lead to drivers misjudging distances or missing oncoming traffic, increasing accident risks. Many accidents happen because of unsafe overtaking or not keeping a safe distance from trucks. Such accidents can cause serious injuries or even death.

Vision-Guard is an affordable solution that combines camera, ultrasonic sensors, and a Node MCU to improve road safety. It is designed to help reduce accidents, making roads safer and saving lives.

1.1. Motivation

Vision-Guard is designed to solve the serious traffic safety problems caused by blind spots and poor visibility when driving behind large vehicles like trucks. These issues make overtaking dangerous and often lead to accidents, especially on highways. Vision-Guard helps by giving drivers a live video view of the road ahead, making it easier to pass safely. It also reduces overtaking accidents by helping drivers make better decisions. To prevent rear-end crashes, Vision-Guard includes sensors and warnings to encourage safe distances from trucks. It is designed to be an affordable safety solution so more people can use it. The main goal of Vision Guard is to save lives, reduce accidents, and make roads safer for everyone.

1.1. Problem Statement

Driving behind large vehicles like trucks often blocks the view of the road ahead. This lack of visibility makes overtaking dangerous, especially on highways where quick decisions are needed. Blind spots around these vehicles increase the risk of misjudging distances or missing oncoming traffic. Unsafe overtaking and not keeping enough distance behind trucks are major causes of accidents, leading to serious injuries or deaths. There is a need for a practical and affordable solution to make roads safer and reduce these risks.

1.2. Objectives and Contributions

The features of the Vision-Guard system aim to enhance road safety by addressing visibility challenges and eliminating blind spots caused by large vehicles. Its objectives focus on improving driver awareness, encouraging safe driving behaviors, and minimizing the risks associated with overtaking. Vision-Guard offers an affordable, user-friendly solution that delivers real-time road visibility, promotes safe distances using proximity sensors, and empowers drivers to make better overtaking decisions. By integrating innovative yet practical technology, Vision-Guard seeks to prevent road accidents and protect lives, making driving safer for all road users.

1.3. Project Scope

The goal of the Vision-Guard project is to create an advanced safety system designed to enhance road awareness and minimize accident risks. This system utilizes real-time video streaming, ultrasonic proximity sensors, and user-friendly alerts to provide drivers with clear visibility of the road ahead, safe distance monitoring, and guidance for safer overtaking decisions. By integrating advanced technologies such as live video feeds, sensor-based distance tracking, and real-time alerts, Vision-Guard aims to promote safer driving practices. The system enables users to monitor road conditions, receive timely safety notifications, and make informed decisions while driving, creating a safer environment for all road users.

1.4. Organization of the Report

The report covers a comprehensive understanding of the Vision Guard. It starts by giving a basic introduction, defining a problem statement, and outlining our objectives. Moreover, the report talks about exploring the technologies utilized in the project. The methodology section highlights the steps taken in developing , including the system architecture and technologies utilized. The results findings section will discuss the project performance. In the end, the conclusion Sums up key insights, Appendices include technical details, and references cite the sources that influenced the project's development.

CHAPTER 2

2. Literature Review

2.1. Introduction

This section provides a detailed literature review to examine existing technologies and approaches related to road safety, visibility enhancement, and blind spot reduction. By analyzing current advancements in driver assistance systems and accident prevention technologies, we aim to identify gaps, challenges, and opportunities that will inform and guide the development of the Vision-Guard system.

2.2. Literature Review

In this literature review, we conduct a comparative analysis between well-established Advanced Driver Assistance Systems (ADAS) and our innovative Vision-Guard concept, each offering unique solutions to improving road safety. ADAS, widely known for features like adaptive cruise control, lane-keeping assistance, and collision warnings, focuses on enhancing the overall driving experience through advanced automation. In contrast, Vision-Guard provides a more targeted approach by addressing specific issues such as blind spots, overtaking risks, and maintaining safe distances from large vehicles. While ADAS excels in offering comprehensive driver assistance features, Vision-Guard bridges the gap by introducing affordable, user-friendly solutions to improve visibility and decision-making in real-time. ADAS systems often come with high costs, making them inaccessible to many drivers, whereas Vision-Guard emphasizes simplicity and cost-effectiveness without compromising safety. When it comes to usability, ADAS has received positive feedback for its advanced automation and reliability. However, Vision-Guard prioritizes accessibility by combining real-time road views, proximity alerts, and intuitive safety features, ensuring that its success relies on clear video feeds, accurate sensor data, and ease of integration with different vehicles. This comparative analysis highlights Vision-Guard's potential contribution to the road safety ecosystem, offering a practical, innovative, and inclusive approach to reducing accidents and enhancing driver awareness.

2.3. Functional and Non-Functional Requirements

2.3.1. Functional Requirements

- The system should feature a user friendly interface for seamless navigation and interaction.
- A real-time video streaming feature should be implemented to provide drivers with a clear view of the road ahead.
- Ultrasonic proximity sensors and led lights should detect the distance between vehicles and provide alerts when the following distance is unsafe.
- The web app should allow users to review driving data.
- The system should provide visual and audio alerts for unsafe driving conditions.

2.3.2. Non-Functional Requirements

- The system should process video feeds and sensor data in real-time with minimal latency to ensure timely alerts.
- The web app should be compatible with various devices and browsers, ensuring accessibility.
- The system should maintain high accuracy in distance detection and road visibility enhancement to build driver trust and satisfaction.

2.4. Project Significance

The Vision-Guard system focuses on traditional safety tools. With its advanced features, it not only enhances road safety but also fosters a community of responsible and informed drivers.

2.5. Software Platform

Vision Guard project is a web based application for monitoring real time vehicle streaming and displaying driving data.

2.6. Scalability

The Vision-Guard system is built to grow with the increasing number of users and data. Its design allows it to handle large amounts of video and sensor data without slowing down. As more users join or more vehicles are connected, the system can still provide fast and accurate alerts, ensuring safety at all times.

The system is flexible and can easily add new features as technology improves. This means it can keep up with changes, whether it's upgrading sensors or adding new safety tools. The backend is built to grow, so as more data is generated, the system can handle it efficiently without affecting performance.

Vision-Guard's web app is also designed to work on various devices, making it easy for more users to access. This ensures that even as the number of users grows, the system remains responsive and reliable.

2.7. Services

To enhance the user experience, the Vision-Guard system offers several key features. First, the system provides a real time video stream to help drivers see the road ahead, especially when following large vehicles like trucks. This allows drivers to make better decisions while overtaking, improving safety. The system also uses ultrasonic sensors to monitor the distance between vehicles and provides alerts when the following distance is unsafe. Additionally, Vision-Guard has a simple and user-friendly interface, making it easy for drivers to navigate and access the system's features. Finally, it offers a web application where users can review their driving data and safety alerts, ensuring a seamless experience for all drivers.

CHAPTER 3

3.1. Projects diagrams

Based on the above literature review and project scope here are some diagrams, which illustrates that what will be our project or the system is capable to reach the desired results.

3.1.1. System Architecture Diagram:

This diagram shows the big picture of how all the parts of Vision Guard (camera, sensors, Arduino, display, etc.) work together. It helps understand the overall design and data flow between modules.

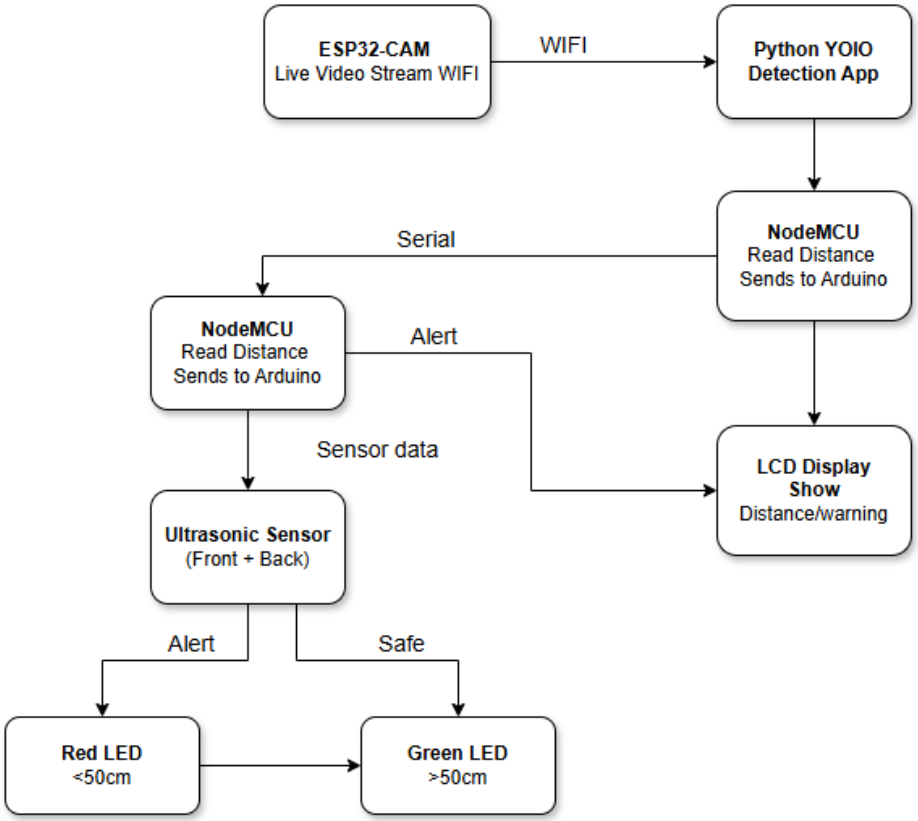


Fig 1

3.1.2. System Block Diagram:

This diagram shows how system divided into blocks (like input, processing, output). Each block represents a main part such as the ESP32-CAM, Node MCU, Arduino, and LCD, and how they are connected.

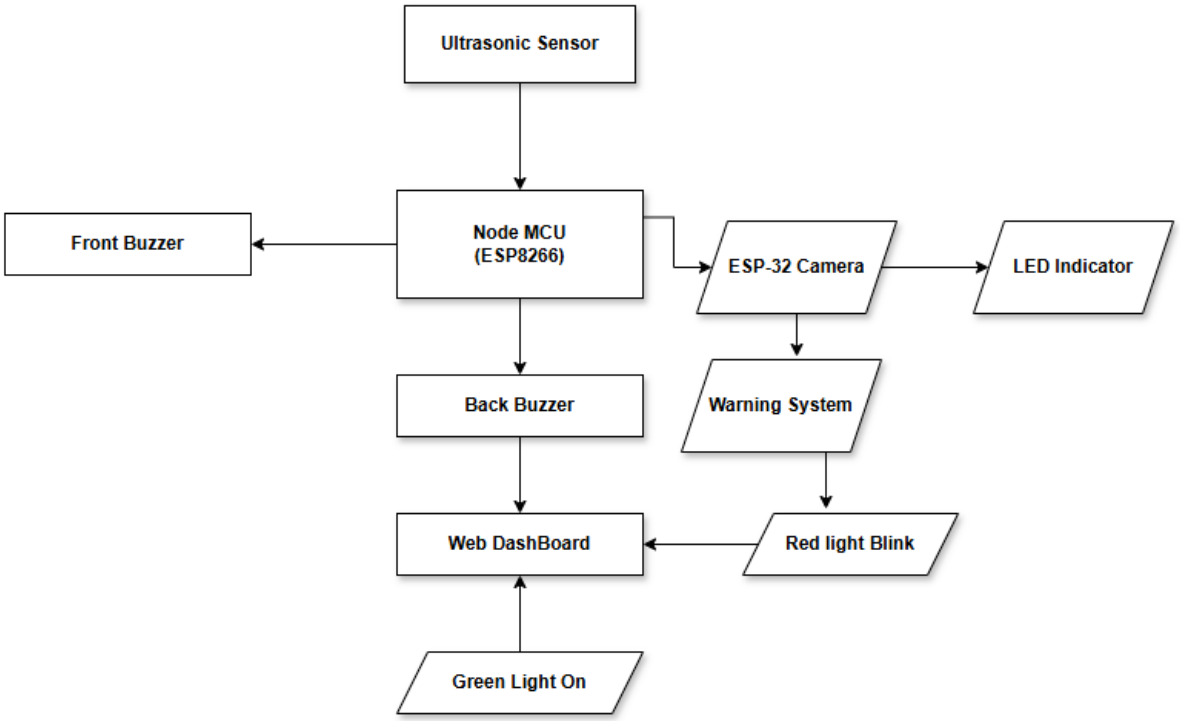


Fig 2

3.1.3. Flowchart:

This is a step by step diagram that explains how the vision guard system works. It starts from capturing video, checking distance, sending data, and giving alerts or displaying results.

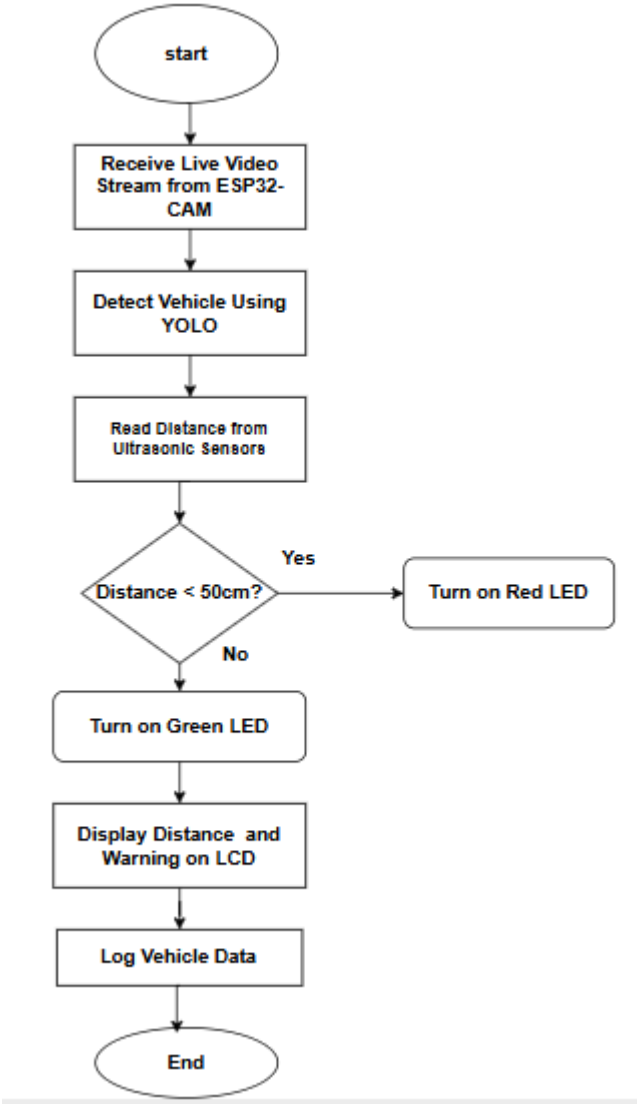


Fig 3

3.1.4. ESP32 CAMERA FTDI programmer:

This FTDI programmer helps to upload code on the ESP32-CAM using a USB cable. This setup is used to program and power the ESP32-CAM for capturing video and processing.

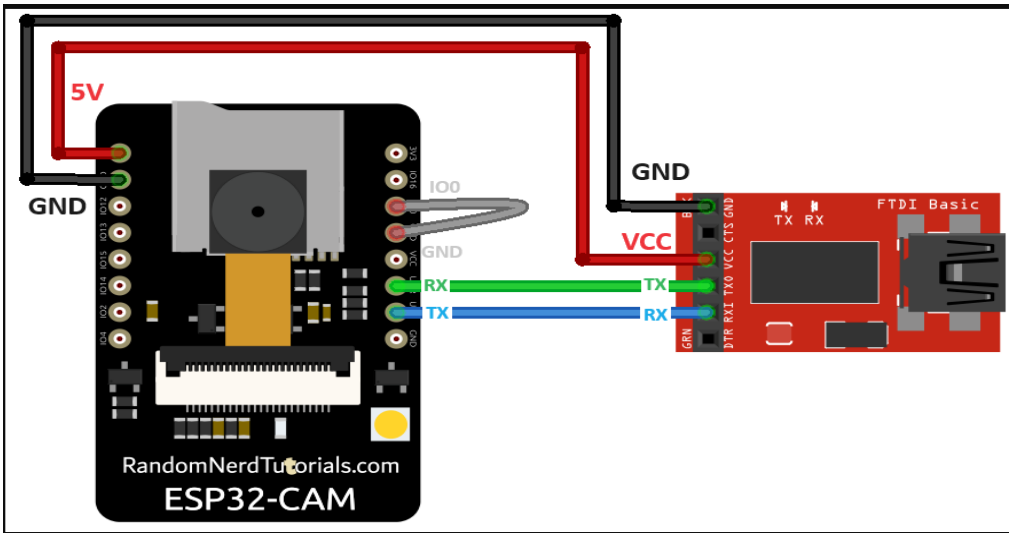


Fig 4

3.1.5. Pin-Diagram ESP32:

It shows all the pins on ESP-32 CAM. Each pin has a specific job like taking input sending output or connecting to other modules.

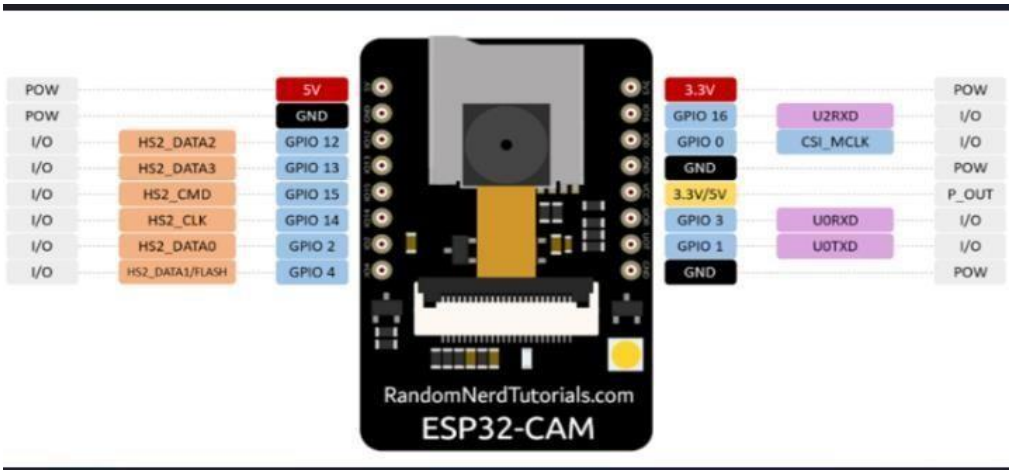


Fig 5

3.1.6. Sensor node MCU pin:

This diagram shows which pins of the Node MCU are used to connect ultrasonic sensors It helps for smooth communication wiring in the components.

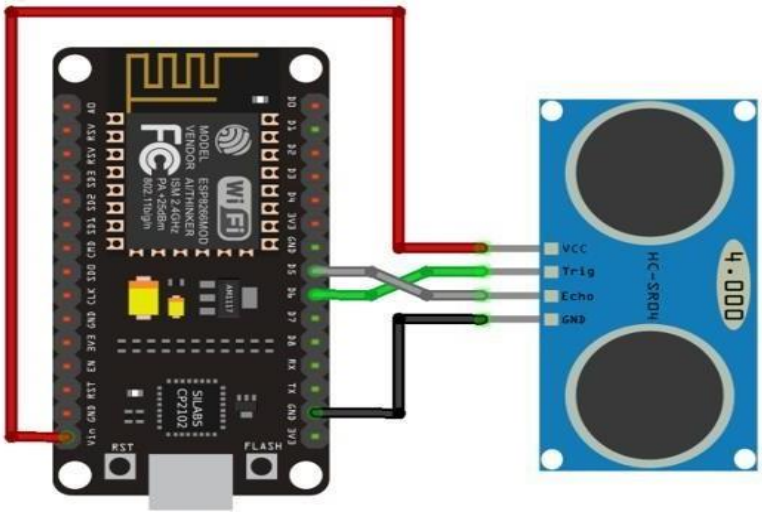


Fig 6

3.1.7. SPI TFT LCD:

This is a display screen that uses SPI (Serial Peripheral Interface) to connect with Arduino. It shows the live camera feed or any alerts for the vehicle behind the truck.

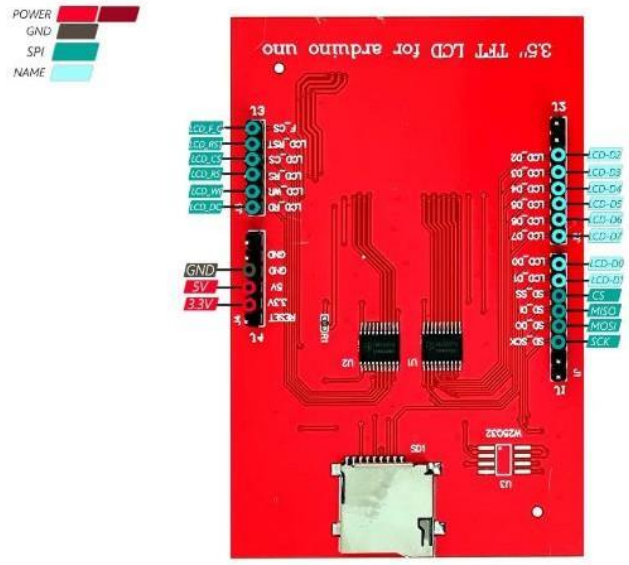


Fig 7

3.1.8. Activity Diagram:

This diagram is for understanding the system's behavior. It shows the sequence of actions the system takes, like detecting a vehicle, calculating distance, triggering alerts, and displaying info.

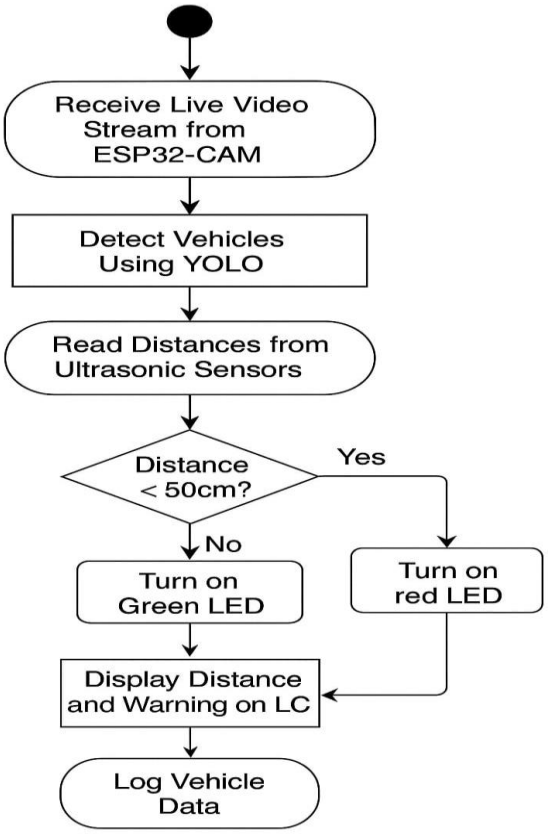


Fig 8

3.1.9. Collaboration Diagram:

This diagram shows how different parts of the system (camera, sensors, display) communicate and work together to make decisions and show results.

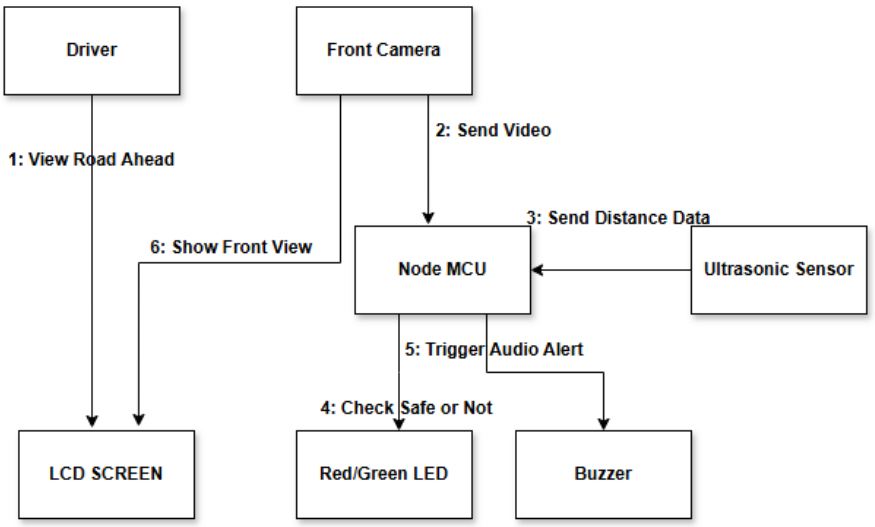


Fig 9

3.1.10. Swimlane Diagram:

It divides among different parts of the system in separate lanes. Like one for hardware flow one for streaming module, one for python program. It helps to understand what it actually does.

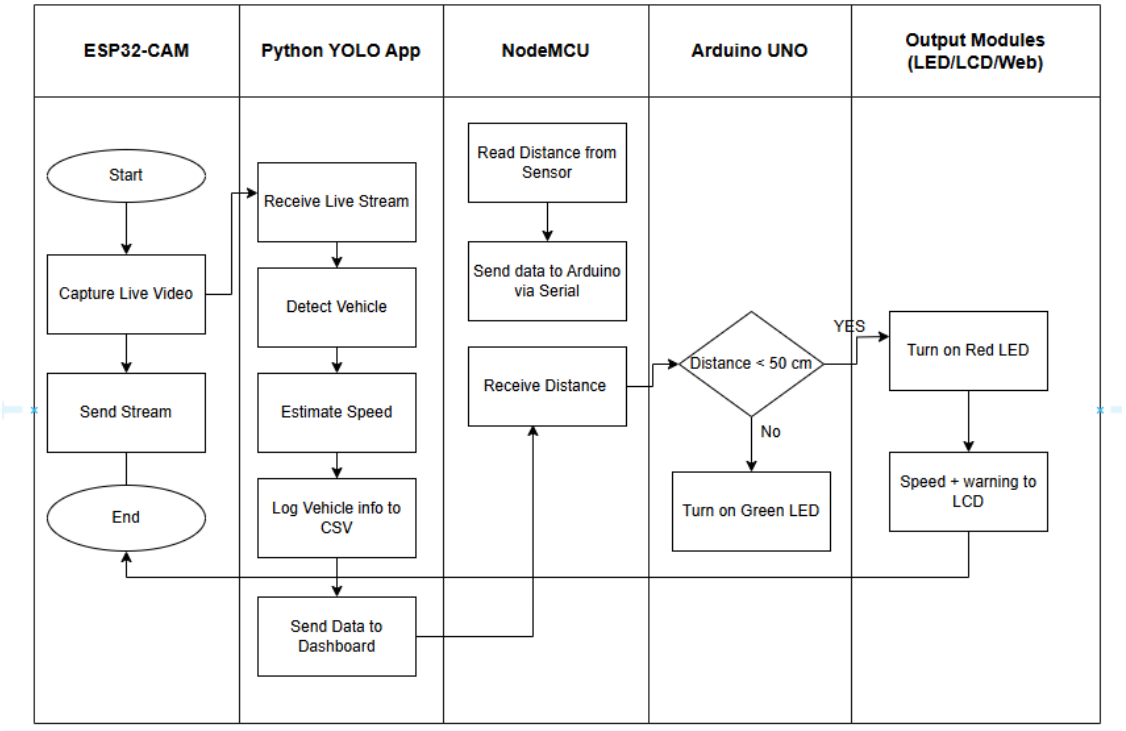


Fig 10

3.1.11. Sequence Diagram:

It shows the sequence of operations performed. Like which component perform what kind of operation. It helps to understand what it actually does.

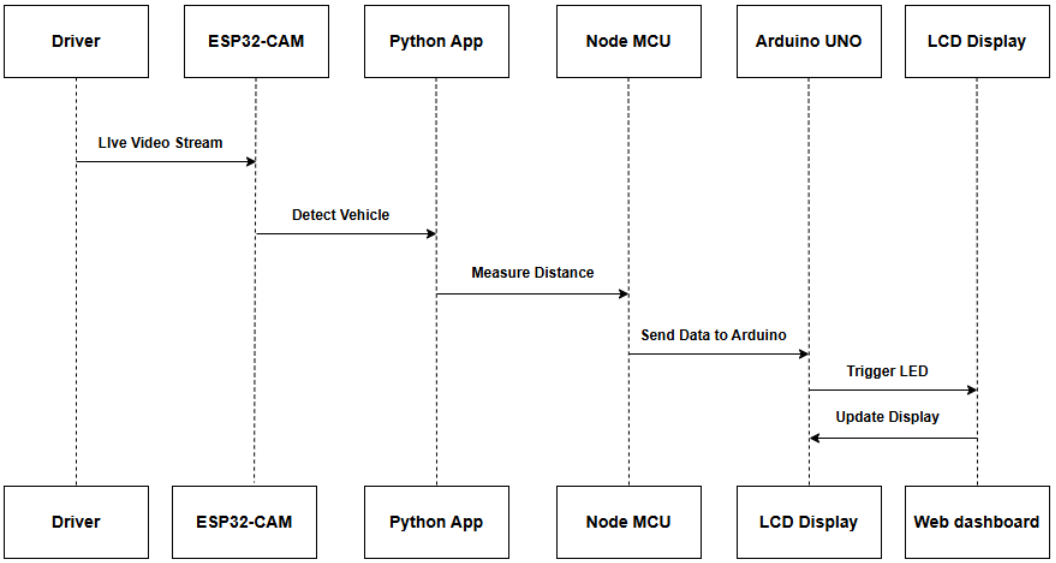


Fig 11

3.1.12. Dataflow Diagram:

It shows how data (like distance or video feed) moves from one part to another. It includes input, processing, and output parts of the system in an easy way.

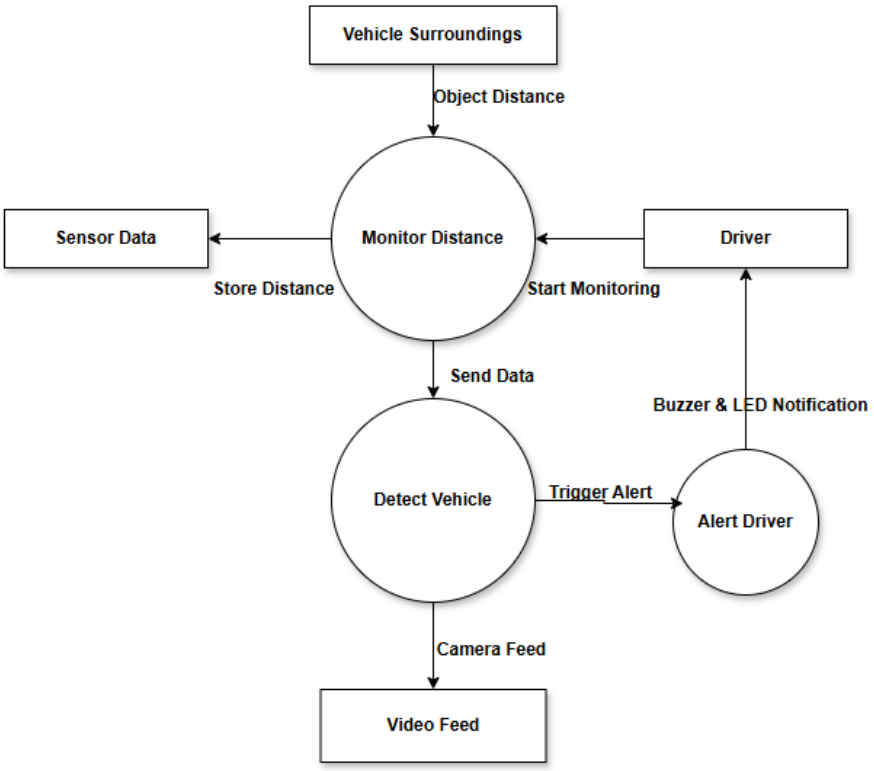


Fig 12

3.1.13. Use case Diagram:

This explains what the user can do with the system like watching live video, checking warnings, and viewing the dashboard. Shows user interaction in a simple way.

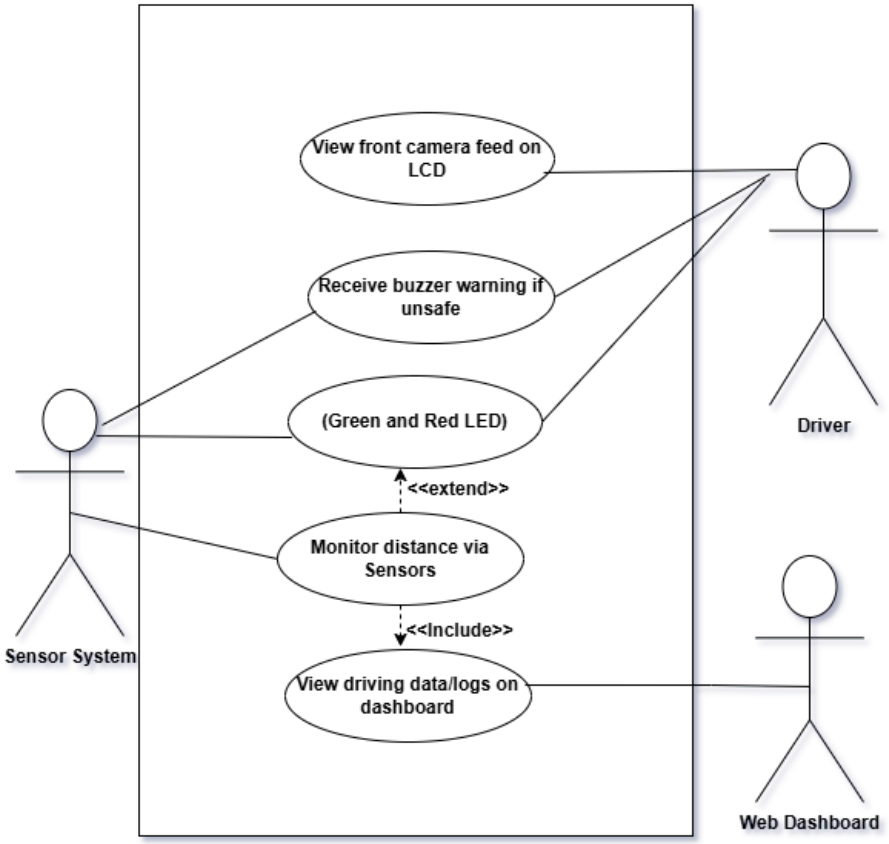


Fig 13

3.12. Inside Project

3.12.1. Frontend on HTML CSS

- Created intuitive and user-friendly interfaces.
- Focused on seamless user experience.

3.12.2. Development

Utilized Visual Studio Code for coding and debugging.

3.12.3. Model Training

- Trained the vehicle detection and identification model using YOLO 3.
- Employed image processing and machine learning techniques for vehicle detection.

3.12.4. Flask Dashboard

Developed a Flask-based dashboard to view.

3.13. Technologies Used

3.13.1. Development

- Framework: VS code
- Programming Language: Python
- IDE: Arduino IDE

3.13.2. Trained Model

- Platform: Google COLAB/ Roboflow
- Library: Open CV
- Programming Language: Python

3.13.3. Development and Hosting

- Framework: Flask

CHAPTER 4

4.1. Project Planning

4.1.1. Project Timeline Summary

Phase	**Tasks**	**Estimated Duration**
Project Definition *(Week 1–2)*	● Define the goal: improving road safety behind trucks. ● Identify target users (e.g., highway drivers, logistics companies). ● Study existing solutions like ADAS and Samsung Safety Truck. ● Finalize features: live video, sensors, alerts, and web app.	2 weeks
Hardware Setup & Integration *(Week 3–4)*	● Connect ESP32-CAM for video streaming. ● Set up ultrasonic sensors and Node MCU. ● Test sensor range and camera quality. ● Ensure smooth communication between hardware parts**.**	2 weeks
Software Development *(Week 5–8)*	● Develop video streaming module. ● Implement real-time distance alerts using sensors. ● Build the basic user interface for the web app. ● Program system logic for alerts and responses.	4 weeks
Web Application Features *(Week 9–10)*	● Add driving data storage in web app. ● Show alerts and trip summaries. ● Create user-friendly dashboard with safety info.	2 weeks
Testing and Debugging *(Week 11–13)*	● Test all modules: camera, sensors, web app. ● Check system responsiveness and reliability. ● Fix bugs and optimize alert accuracy.	3 weeks

System Deployment *(Week 14)*	• Deploy complete Vision Guard system on test vehicle. • Ensure real time performance in actual road conditions.	1 week
Documentation *(Week 15)*	• Write a final report (intro, objectives, methods, results).	1 week
Project Presentation *(Week 16)*	• Prepare PowerPoint slides and system demo. • Practice delivery for final presentation. • Highlight problem, solution, demo, and impact.	1 week

4.1.2. Project Timeline Details

Vision guard project followed a structured development timeline to ensure consistent progress for real world implementation. The project starts with detailed requirement analysis and research, focusing on vehicle detection technologies and road safety features. Following this, a YOLO based object detection system is implemented using the ESP32-CAM module for real time vehicle detection. Parallel to software planning, the hardware dashboard is develop using Arduino Uno and an 8-bit parallel ILI9486 LCD to display driving logs and alerts. As development phase, modules for vehicle tracking, smooth flow for speed estimation, and overtaking logic with LED lights which is proper integrated and tested. A web dashboard is created to visualize logs of driving data, jumper wire connectivity is enhanced to reduce signal errors. The prototype is test using toy cars in a simulated environment, ensuring proper synchronization between detection, alerts, and UI displays.

4.1.3. Black-box Testing

Black box testing focuses on testing the system without looking at the internal code. In this type of testing, we only care about what the system does, not how it does it.

For our vision guard project, black box testing involves giving inputs (like a toy vehicle appearing in front of the camera) and checking if the expected outputs are shown (such as the vehicle being detected, LED indicators turning on, or logs being updated on the dashboard). For example, if a car comes too close, the system should trigger a distance alert or activate an LED. Here, we does not need to know how YOLO works internally we just check if the detection and response work correctly.

This helps verify if the system behaves correctly from an external user's point of view.

Black Box testing method is applicable to the following levels of software testing:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

4.1.4. Unit Testing

Unit testing is all about testing each & every individual components and functions of system are work correctly on their own.
In our project, we test:

Software side includes:

- A function that calculates vehicle speed using frame differences.
- A function that detects vehicles.
- A code module that triggers the LEDs when overtaking conditions are meet.
- A code module that generates alerts when distance is less than 50 cm and ultrasonic sensors starts buzzing sound.
- A function that logs vehicle data on the web dashboard.

Hardware side includes:

- Test the ESP32-CAM captures and streams video correctly.
- Check the ILI9486 LCD displays output properly.
- Verify the LED lights glow when powered through Arduino commands.
- Test ultrasonic sensors works properly and sound of buzzer is audible.
- Ensuring jumper wires and connections are giving proper signals.

Each component is tested individually to catch problems early before combining them.
called End-to-End testing scenario. Verify thorough testing of every input in the application to check for desired outputs.

4.1.5. Integration Testing

Integration testing checks how different components work together as a team both software modules and hardware components.

In our project, we test:

Software side includes:

- Checking the vehicle detection correctly works.
- Verifying the speed calculating module.
- Ensuring the overtaking logic is correctly sending output to the LEDs & Ultrasonic sensors.
- Testing the web dashboard is receiving data from the ESP32-CAM or Arduino.

Hardware side includes:

- Arduino Uno correctly communicates with the LCD screen using 8-bit parallel communication.
- ESP32-CAM and Arduino modules work together (e.g., detection by ESP32-CAM leads to alerts through Arduino and LEDs).
- Data captured in real time through the camera works accurately on both the LCD dashboard and web interface.

This ensures all parts of the system (software + hardware) are working smoothly when integrated.

4.1.6. System Testing

System testing involves checking the full project as a complete working system, for it would be used in the real world.

In our project, we tests:

- Mounting the hardware (ESP32-CAM, Arduino, LCD, LEDs) on a toy truck.
- Detecting and tracking other vehicles.
- Verifying speed estimation, overtaking logic, and distance alerts.
- Checking that LED signals work during overtaking.
- Viewing real-time data on the LCD dashboard and logs on the web dashboard.

4.1.7. User Acceptance Testing

User Acceptance Testing is done to check if the final system is usable, understandable, and useful from the end user's point of view.

In our project, this involves:

- Letting our teachers, advisor, or friends to test the system.
- Asking them to observe how the system behaves when a toy car is detected or when the truck is overtaking.
- Getting their feedback on the alerts, LEDs, and dashboard make sense and are easy to understand.
- Ensuring the system is reliable, user friendly, and meets the goals of driver safety.

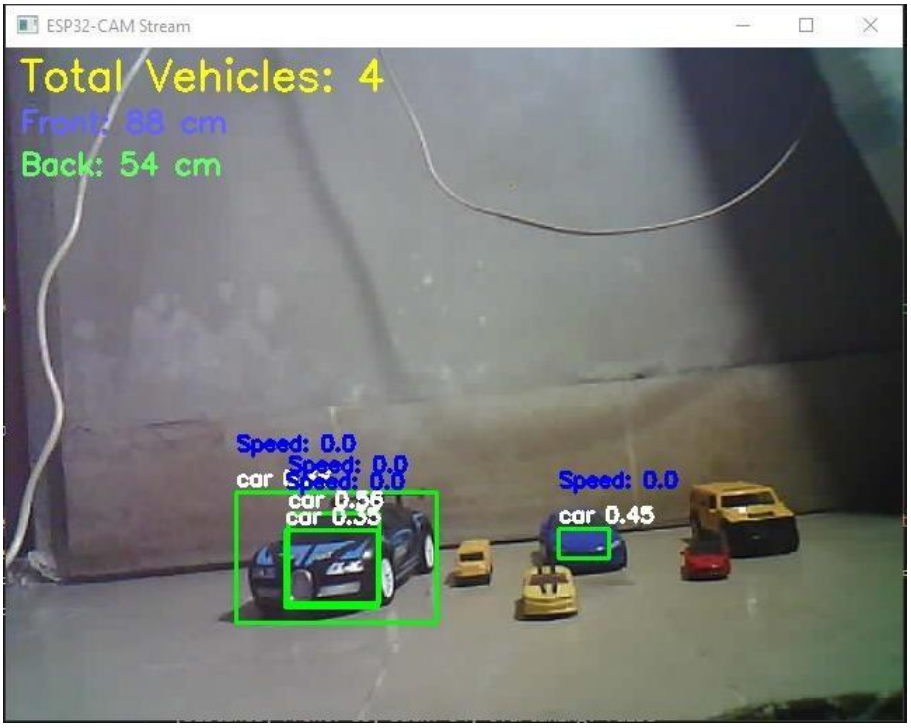
4.2. Test Cases

The purpose of test cases in vision guard is to systematically verify that each software and hardware component functions correctly, reliably, and according to requirements. Test cases help ensure that key features such as vehicle detection & tracking, speed estimation, LED & Ultrasonic sensors activation for overtaking logic perform as expected in real time scenarios. Moreover, test cases serve as documentation for future reference and development, and they increase user and stakeholder confidence by proving that the system has been thoroughly validates for real world use.

4.2.1. TEST CASE # 1

Test Case Title	Live Front Camera Stream
Test Case ID	VG_TC_001
Test Case Objective	Check the live camera shows the road ahead clearly
Pre-Conditions	System and camera must be powered on
Test Steps	• Turn on the Vision-Guard system. • Open the live video stream. • Observe the video feed.
Expected Result	The road ahead is clearly visible with minimum delay.
Actual Result	Road is clearly visible with minimum delay.
Test By	Abad, Sadia, Tooba, Warisha
Status	Pass

These vehicles are detected from front camera. This shows total number of vehicles and distance

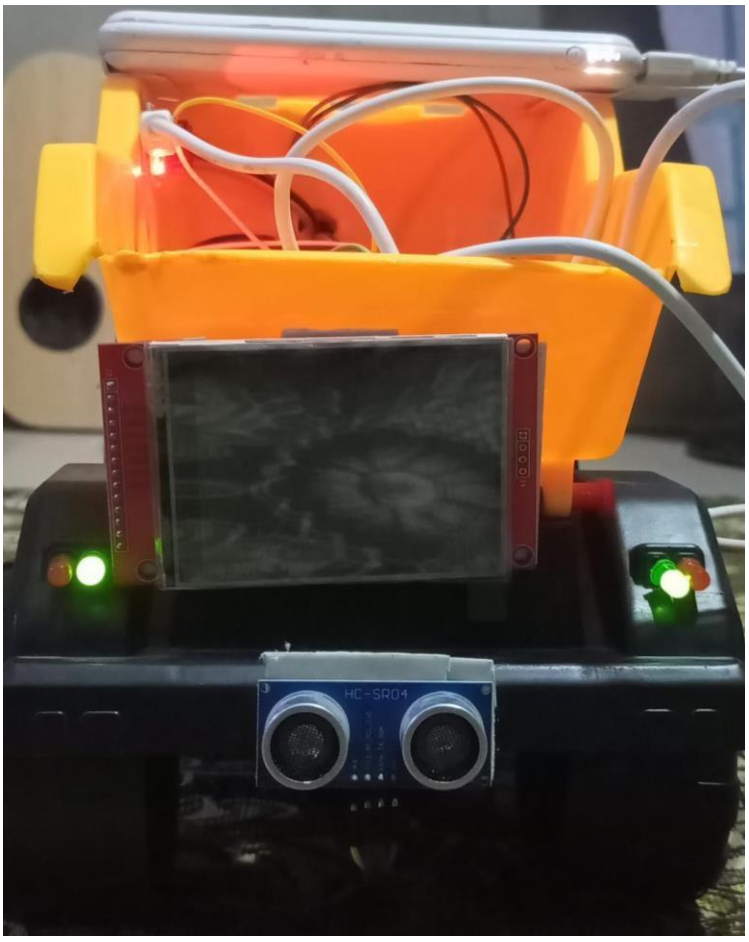


4.2.2. TEST CASE # 2

Test Case Title	Safe Distance Alert
Test Case ID	VG_TC_002
Test Case Objective	Check the led light begin when another vehicle is too close.
Pre-Conditions	Led lights must be connected and active.
Test Steps	• Bring another vehicle closer to the truck. • Monitor the system. • Watch for a visual alert.
Expected Result	• Red light is blink when the other vehicle is less than 50 cm away. • Green light is blink when the vehicle is far then 50 cm away.
Actual Result	• Red light starts blinking when the vehicle distance is 50 cm less. • Green light starts blinking when vehicle is far and green signal for overtaking.
Test By	Abad, Sadia, Tooba, Warisha
Status	Pass

This light blinks when the distance is less than 50 cm and not safe for overtaking.

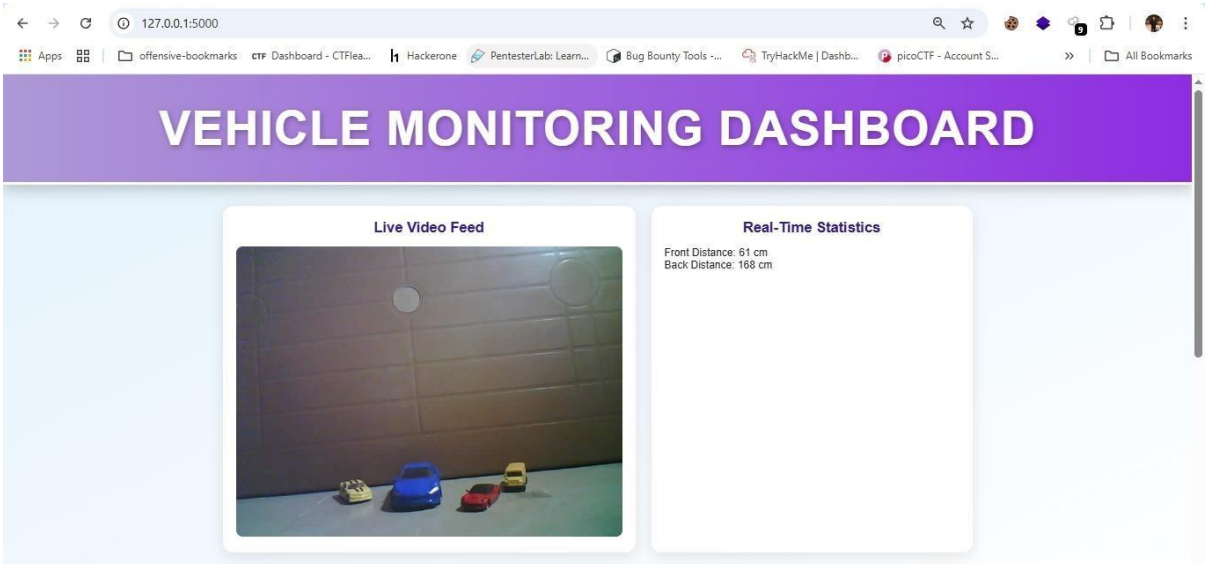
This light blinks when the distance is more than 50 cm and safe for overtaking.

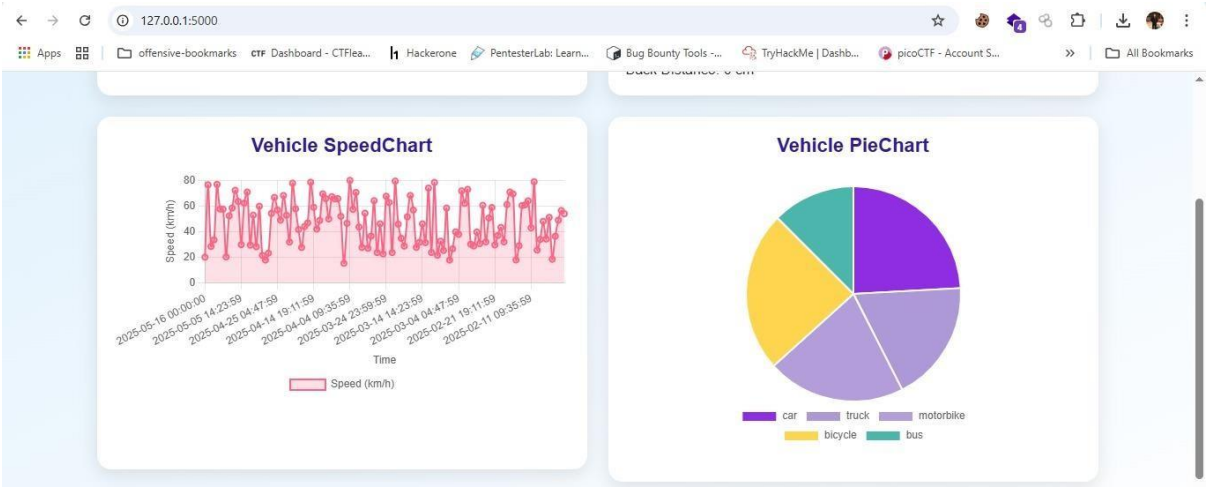


4.2.3. TEST CASE # 3

Test Case Title	Driving History in Web Dashboard.
Test Case ID	VG_TC_003
Test Case Objective	Verify the driving data and logs on web dashboard.
Pre-Conditions	At least one trip must be recorded.
Test Steps	• Open the Vison Guard Web Dashboard. • Go to the driving history section. • Review trip details.
Expected Result	All driving data is shown correctly and clearly.
Actual Result	All driving data & logs are clearly visible.
Test By	Abad, Sadia, Tooba, Warisha
Status	Pass

This is web dashboard that visualizes driving data.





Logs

Timestamp	Vehicle ID	Vehicle Type	Speed	Front Distance	Back Distance	Overtaking Event
2025-05-16 00:00:00	car_001	car	19.89 km/h	114	106	Yes
2025-05-15 03:11:59	truck_001	truck	76.52 km/h	102	87	Yes
2025-05-14 06:23:59	motorbike_001	motorbike	28.28 km/h	155	111	No
2025-05-13 09:35:59	bicycle_001	bicycle	33.4 km/h	132	96	No
2025-05-12 12:47:59	car_002	car	76.81 km/h	147	97	Yes
2025-05-11 15:59:59	truck_002	truck	57.6 km/h	126	103	No

4.2.4. TEST CASE # 4

Test Case Title	Proximity Alert Detection.
Test Case ID	VG_TC_004
Test Case Objective	Verify that the system correctly detects vehicle and triggers visual alert when vehicle distance is less than 50 cm (enters the danger zone) .
Pre-Conditions	System is powered on and all modules (ESP32-CAM, NodeMCU, sensors, LCD, LEDs) are connected properly.
Test Steps	• Start the system. • Place the test vehicle behind the truck at more than 100 cm. • Gradually move the vehicle closer to within 40 cm. • Observe the LED and Ultrasonic sensor response.
Expected Result	Starts generating visual alert when the distance is < 50 cm.
Actual Result	Visual alert is generating when the vehicle distance is < 50 cm.
Test By	Abad, Sadia, Tooba, Warisha
Status	Pass

4.2.5. TEST CASE # 5

Test Case Title	Video Stream Latency.
Test Case ID	VG_TC_005
Test Case Objective	Ensure the live video stream has low delay.
Pre-Conditions	System and camera must be running.
Test Steps	• Start the live camera feed. • Place a timer in front of the camera. • Compare the actual time with what is shown on the screen
Expected Result	Delay should be less than 1 second.
Actual Result	Delay is less than 1 second.
Test By	Abad, Sadia, Tooba, Warisha
Status	Pass

This is the live video stream captures from front camera.

CHAPTER 5

5.1. Components of system

5.1.1. Vision Guard System Prototype

5.1.2. Screen

This screen is for the display of video that streams from front of the truck.

5.1.3. Ultrasonic Sensor

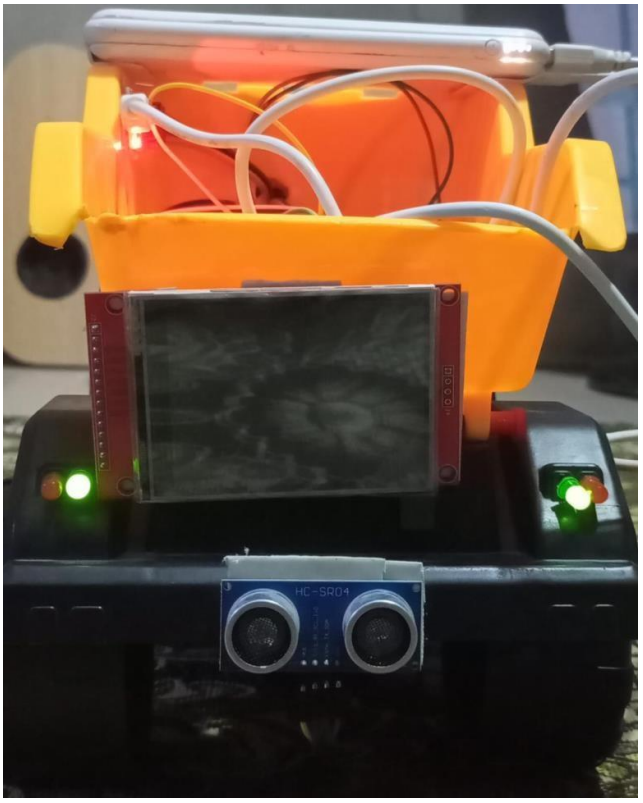
This is the back ultrasonic sensor starts buzzing sound when the distance is less than 50cm.

This is the front ultrasonic sensor starts buzzing sound when the distance is less than 50cm.

5.1.4. LED Indicators

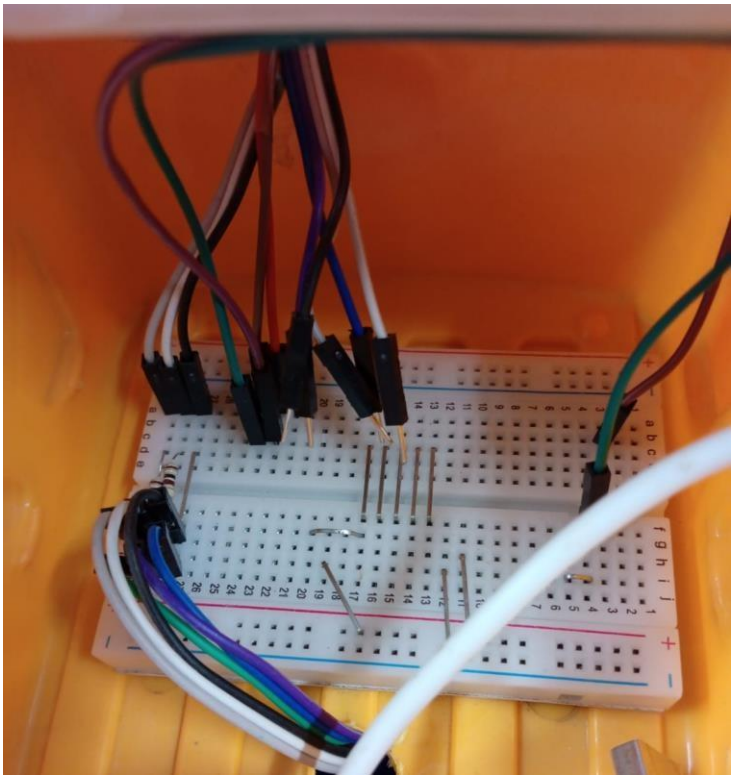
Red led for not safe for overtaking.

Green led is for safe overtaking.



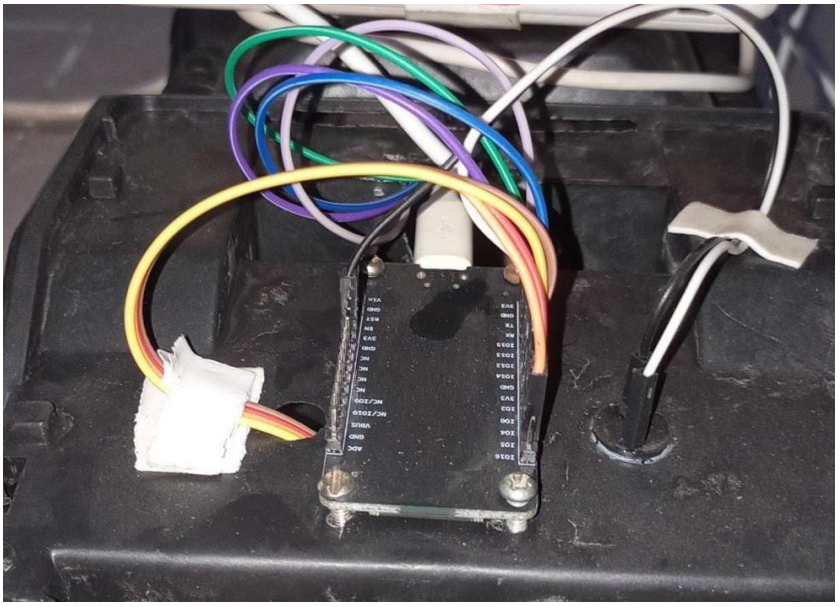
5.1.5. Breadboard Wiring

All wirings are implemented on this.



5.1.6. Node MCU

This handles all sensor data collection and send to their modules.

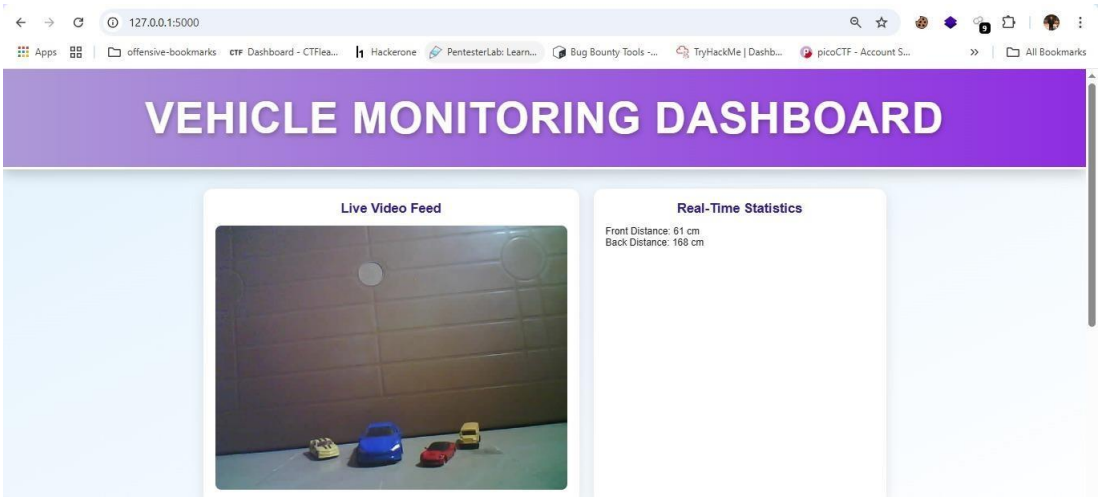


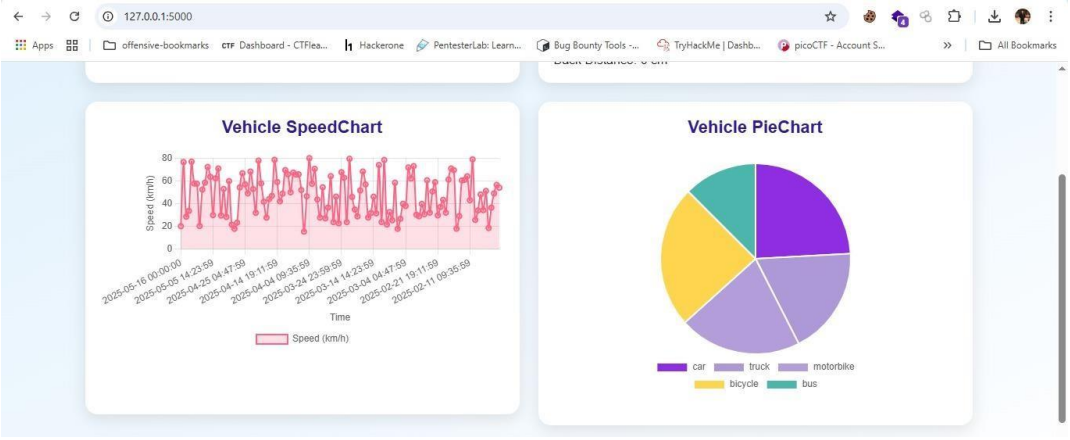
5.1.7. Arduino UNO

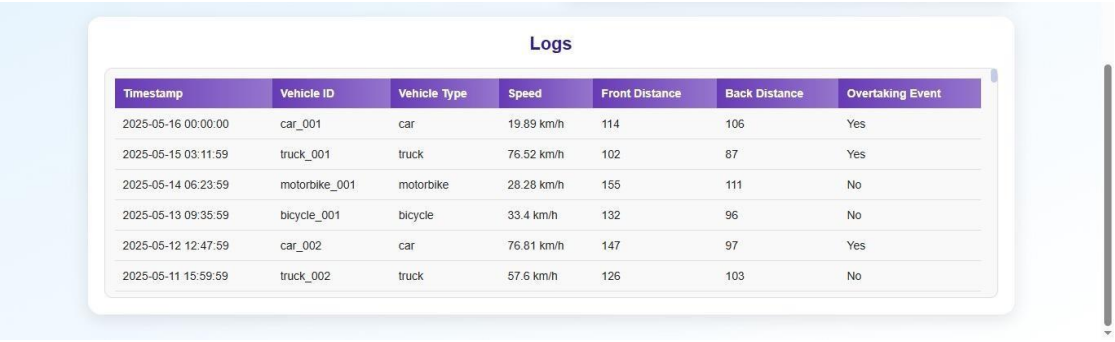
Controls the system for overtaking.

5.1.8. Web Dashboard

This shows the driving data.







CHAPTER 6

6.1. Limitation

- The system is tested in a controlled environment using toy vehicles, so real world traffic behavior and environmental conditions (like night time, rain, or fog) are not fully covered.
- ESP32-CAM has limited resolution and processing power may reduce detection accuracy in fast moving or low light scenarios.
- Arduino Uno has limited memory and computational capacity, so it restrict for addition of more advanced features.
- System depends on wired connections (jumper wires), which may lead to loose connections or errors during movement.
- When multiple modules (LCD, LED detection) are running simultaneously may be affect due to hardware limitations.
- Web dashboard requires a stable connection for continuous data logging and display, and may not function properly if connectivity is lost.

6.2. Conclusion

In conclusion, Vision Guard is a smart safety solution designed to improve driver awareness and reduce road accidents caused by blind spots of trucks and dumpers. Using real time AI and embedded systems. It combines vehicle detection, speed estimation and overtaking alerts with visual indicators and dashboard displays. The system is successfully shows how low cost hardware like ESP32-CAM and Arduino can work together to perform advanced safety tasks. Instead of some limitations in hardware and testing conditions, the prototype proves to be a functional and practical model for future smart vehicle systems. The integration of visual alerts, logging, and real time feedback makes it an effective safety assistant for driving environments.

6.3. Future Work

There is always a chance of improvement, following are the aspects where the system requires some time to be analyzed and modified.

- Implement on real road testing vehicles.
- Upgrading to a high resolution camera and more powerful processor (e.g., Raspberry Pi).
- Adding lane departure warning and night vision support.
- Improving the LED signaling system for clearer overtaking alerts.
- Implementation of driver drowsiness and distraction detection using AI.
- Expand the web dashboard to support mobile alerts and cloud logging.
- Integrate GPS for location based safety purpose.
- Enhancing system stability with wireless communication and better hardware.
- Expand for commercial purpose and public safety vehicle.

REFERENCES

[1] Deriu, Chiara. "Vehicular Communication for Road Safety: Simulation and Analysis of Emergency Vehicles Preemption Using Traffic Light-Based V2I Technology." PhD diss., Politecnico di Torino, 2024

[2] Yu, Xiang, and Linlin Zhang. “Toward an Automated, Proactive Safety Warning System Development for Truck Mounted Attenuators in Mobile Work Zones.” arXiv preprint arXiv:2412.18189 (2024).

[3] MLA Zhang, Zhengming, et al. “Implementation and performance evaluation of in-vehicle highway back-of-queue alerting system using the driving simulator.” 2021 IEEE International Intelligent Transportation Systems Conference (ITSC). IEEE, 2021.

[4] Harvard Barosan, I., Basmenj, A.A., Chouhan, S.G. and Manrique, D., 2020. Development of a virtual simulation environment and a digital twin of an autonomous driving truck for a distribution center. In Software Architecture: 14th European Conference, ECSA 2020 Tracks and Workshops, L’Aquila, Italy, September 14–18, 2020, Proceedings 14 (pp. 542-557). Springer International Publishing.

[5] https://www.ijraset.com/research-paper/vision-guard-smart-surveillance-for-tomorrow

[6] https://arxiv.org/abs/2104.12583

[7] https://www.mdpi.com/2072-4292/13/11/2163

[8] Chee Seng, Kwang & Abdul Razak, Siti Fatimah & Yogarayan, Sumendra. (2025). Enhancing Vehicle Overtaking System via LoRa-Enabled Vehicular Communication Approach. Computer Systems Science and Engineering. 1-10. 10.32604/csse.2024.056582.

[9] https://doi.org/10.48550/arXiv.2306.05203

[10] https://arxiv.org/abs/2311.08491

[11] Huang, J.; Sun, S.; Long, K.; Yin, L.; Zhang, Z. Automatic Overtaking Path Planning and Trajectory Tracking Control Based on Critical Safety Distance. *Electronics* **2024**, *13*, 3698.

[12] Marcano, M.; Tango, F.; Sarabia, J.; Chiesa, S.; Pérez, J.; Díaz, S. Can Shared Control Improve Overtaking Performance? Combining Human and Automation Strengths for a Safer Maneuver. *Sensors* **2022**, *22*, 9093. https://doi.org/10.3390/s22239093

APPENDIX

A. ESP-32 CAM:

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_http_server.h"
// Replace with your network credentials
const char* ssid = "DESKTOP-F614S19";
const char* password = "Error404";
// Function to handle video streaming
esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t *_jpg_buf = NULL;
  char part_buf[64];
  // Set the response type as multipart
  httpd_resp_set_type(req, "multipart/x-mixed-replace; boundary=frame");
  while (true) {
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      httpd_resp_send_500(req);
      return ESP_FAIL;
    }
    if (fb->format != PIXFORMAT_JPEG) {
      // Convert to JPEG if necessary
      bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
      if (!jpeg_converted) {
        Serial.println("JPEG compression failed");
        esp_camera_fb_return(fb);
        return ESP_FAIL;
      }
    } else {
      _jpg_buf_len = fb->len;
      _jpg_buf = fb->buf;
    }
    // Send the frame header
    size_t hlen = snprintf(part_buf, 64, "--frame\r\nContent-Type:
   image/jpeg\r\nContent-Length: %u\r\n\r\n", _jpg_buf_len);
    res = httpd_resp_send_chunk(req, part_buf, hlen);
    if (res == ESP_OK) {
      // Send the actual JPEG image
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf,
  _jpg_buf_len);
    }
    if (res == ESP_OK) {
      // Send the frame boundary
      res = httpd_resp_send_chunk(req, "\r\n", 2);
    }
    if (fb->format != PIXFORMAT_JPEG) {
      free(_jpg_buf);
    }
    esp_camera_fb_return(fb);
    if (res != ESP_OK) {
      break;
    }
  }
```

```
  return res;
}
// Function to start the web server
void startCameraServer() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  httpd_handle_t server = NULL;

  if (httpd_start(&server, &config) == ESP_OK) {
    // Register URI handler for the video stream
    httpd_uri_t uri_stream = {
      .uri       = "/stream",
      .method    = HTTP_GET,
      .handler   = stream_handler,
      .user_ctx  = NULL
    };
    httpd_register_uri_handler(server, &uri_stream);
  }
}
void setup() {
  Serial.begin(115200);
  // Disable WiFi power saving mode for reduced latency
  WiFi.setSleep(false);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("WiFi connected");
  Serial.print("Camera Stream Ready! Go to: http://");
  Serial.println(WiFi.localIP());
  // Camera configuration
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = 5;
  config.pin_d1 = 18;
  config.pin_d2 = 19;
  config.pin_d3 = 21;
  config.pin_d4 = 36;
  config.pin_d5 = 39;
  config.pin_d6 = 34;
  config.pin_d7 = 35;
  config.pin_xclk = 0;
  config.pin_pclk = 22;
  config.pin_vsync = 25;
  config.pin_href = 23;
  config.pin_sscb_sda = 26;
  config.pin_sscb_scl = 27;
  config.pin_pwdn = 32;
  config.pin_reset = -1;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA; // Lower resolution for faster stream
  config.jpeg_quality = 12;          // Reduce image quality to decrease
   delay
  config.fb_count = 1;               // Use single frame buffer for lower
   latency
  // Initialize the camera
  esp_err_t err = esp_camera_init(&config);
```

```
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  // Start the web server
  startCameraServer();
}
void loop() {
  delay(1);
}
```

B. ARDUINO UNO:

```
#include <MCUFRIEND_kbv.h>
MCUFRIEND_kbv  tft;
#define WIDTH 480
#define  HEIGHT  320
uint16_t lineBuffer[WIDTH];
void setup() {
  Serial.begin(2000000);
  uint16_t ID = tft.readID();
  tft.begin(ID);
  tft.setRotation(1);
  tft.fillScreen(0x0000); // Clear screen to black
}
void loop() {
  if (Serial.available() >= 6) {
    if (Serial.read() == 0xFF && Serial.read() == 0xD8) {
      uint16_t w = (Serial.read() << 8) | Serial.read();
      uint16_t h = (Serial.read() << 8) | Serial.read();
      if (w != WIDTH || h != HEIGHT) return; // ensure correct size
      for (uint16_t y = 0; y < HEIGHT; y++) {
        uint16_t i = 0;
        while (i < WIDTH) {
          if (Serial.available() >= 2) {
            uint8_t hi = Serial.read();
            uint8_t lo = Serial.read();
            lineBuffer[i++] = (hi << 8) | lo;
          }
        }
        // Only draw after full line is ready
        tft.setAddrWindow(0, y, WIDTH - 1, y);
        tft.pushColors(lineBuffer, WIDTH, 1);
      }
    }
  }
}
```

C. NODE MCU:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <NewPing.h>

// Pin definitions
#define FRONT_TRIGGER_PIN 5    // GPIO5 (D1)
#define FRONT_ECHO_PIN 4       // GPIO4 (D2)
#define BACK_TRIGGER_PIN 14    // GPIO14 (D5)
#define BACK_ECHO_PIN 12       // GPIO12 (D6)
#define BUZZER_PIN 13          // GPIO13 (D7)
#define GREEN_LED 15           // GPIO15 (D8)
#define RED_LED 16             // Define only if GPIO16 used externally
#define DISTANCE_SELECT_PIN A0

#define MAX_DISTANCE 200

NewPing frontSonar(FRONT_TRIGGER_PIN, FRONT_ECHO_PIN, MAX_DISTANCE);
NewPing backSonar(BACK_TRIGGER_PIN, BACK_ECHO_PIN, MAX_DISTANCE);

const char* ssid = "DESKTOP-F614S19";
const char* password = "Error404";

ESP8266WebServer server(80);

int warningDistance = 150;       // Default warning distance
const int safeDistance = 50;     // Critical warning if below

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);
  digitalWrite(GREEN_LED, LOW);
  digitalWrite(RED_LED, HIGH); // Off (if onboard LED is active low)

  server.on("/", []() {
    int frontDistance = frontSonar.ping_cm();
    int backDistance = backSonar.ping_cm();

    updateWarningDistance();

    Serial.print("Front: "); Serial.print(frontDistance);
    Serial.print(" | Back: "); Serial.println(backDistance);

    if ((frontDistance > 0 && frontDistance < safeDistance) ||
        (backDistance > 0 && backDistance < safeDistance)) {
```

```
      int dangerDistance = min(frontDistance, backDistance);
      warnProximity(dangerDistance);
    } else {
      digitalWrite(RED_LED, LOW);
      digitalWrite(GREEN_LED, HIGH);
      digitalWrite(BUZZER_PIN, LOW);
    }

    String jsonResponse = "{";
    jsonResponse += "\"front\":" + String(frontDistance) + ",";
    jsonResponse += "\"back\":" + String(backDistance);
    jsonResponse += "}";

    server.send(200, "application/json", jsonResponse);
    Serial.println(jsonResponse);
  });

  server.begin();
  Serial.println("Server started");
}

void loop() {
  server.handleClient();
}

// Beep and blink red LED if too close
void warnProximity(int distance) {
  Serial.print("Proximity Warning! Distance: ");
  Serial.println(distance);

  int beepDelay = (distance <= 100) ? 100 : 300;

  for (int i = 0; i < 3; i++) {
    digitalWrite(BUZZER_PIN, HIGH);
    digitalWrite(RED_LED, HIGH);
    digitalWrite(GREEN_LED, LOW);
    delay(beepDelay);
    digitalWrite(BUZZER_PIN, LOW);
    digitalWrite(RED_LED, LOW);
    delay(beepDelay);
  }
}

// Adjust warning distance using analog input
void updateWarningDistance() {
  int sensorValue = analogRead(DISTANCE_SELECT_PIN);
  if (sensorValue < 341) {
    warningDistance = 100;
  } else if (sensorValue < 682) {
    warningDistance = 150;
  } else {
    warningDistance = 200;
  }
  Serial.print("Warning Distance Set To: ");
  Serial.println(warningDistance);
}
```

D. LCD:

```
import requests
import threading
import serial
import struct
from PIL import Image
import io
import time

ESP32_URL = 'http://192.168.137.34/stream'
SERIAL_PORT = 'COM8'
BAUD = 2000000
WIDTH, HEIGHT = 480, 320

latest_jpg = None
decoded_frame  =  None
jpg_lock = threading.Lock()
frame_lock = threading.Lock()

def rgb888_to_rgb565(r, g, b):
    return ((r & 0xF8) << 8 | (g & 0xFC) << 3 | b >> 3).to_bytes(2, 'big')

def stream_reader():
    global latest_jpg
    stream = requests.get(ESP32_URL, stream=True)
    buf = b''
    for chunk in stream.iter_content(chunk_size=4096):
        buf += chunk
        while True:
            start = buf.find(b'\xff\xd8')
            end = buf.find(b'\xff\xd9')
            if start != -1 and end != -1 and end > start:
                frame = buf[start:end+2]
                with jpg_lock:
                    latest_jpg = frame
                buf = buf[end+2:]
            else:
                break

def frame_decoder():
    global latest_jpg, decoded_frame
    while True:
        current_jpg = None
        with jpg_lock:
            if latest_jpg:
                current_jpg = latest_jpg
                latest_jpg = None

        if current_jpg:
            try:
                img = Image.open(io.BytesIO(current_jpg))
                img = img.resize((WIDTH, HEIGHT)).convert('RGB')
                with frame_lock:
                    decoded_frame = img
            except Exception as e:
                print(f"[ERROR] Decode failed: {e}")

        time.sleep(0.005)
```

```
def frame_sender(ser):
    global decoded_frame
    while True:
        frame = None
        with frame_lock:
            if decoded_frame:
                frame = decoded_frame.copy()  # Shallow copy for thread
   safety

        if frame:
            try:
                ser.flushOutput()
                ser.write(b'\xFF\xD8')
                ser.write(struct.pack('>H', WIDTH))
                ser.write(struct.pack('>H', HEIGHT))

                pixels = frame.load()
                for y in range(HEIGHT):
                    for x in range(WIDTH):
                        r, g, b = pixels[x, y]
                        ser.write(rgb888_to_rgb565(r, g, b))

            except Exception as e:
                print(f"[ERROR] Send failed: {e}")

        time.sleep(0.01)

def main():
    ser = serial.Serial(SERIAL_PORT, BAUD)
    time.sleep(2)

    threading.Thread(target=stream_reader, daemon=True).start()
    threading.Thread(target=frame_decoder, daemon=True).start()
    frame_sender(ser)  # Run in main thread

if __name__== "__main__":
    main()
```

E. DASHBOARD:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vehicle Monitoring Dashboard</title>
    <link rel="stylesheet" href="/static/css/styles.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>
<body>
    <header>
        <h1>Vehicle Monitoring Dashboard</h1>
    </header>
```

```
        <main>

            <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
            <script>
            const ctx = document.getElementById('myPieChart').getContext('2d');
            new Chart(ctx, {
                type: 'pie',
                data: {
                    labels: ['Search', 'Direct', 'Referral'],
                    datasets: [{
                        label: 'Traffic Sources',
                        data: [55, 25, 20],
                        backgroundColor: ['#8e2de2', '#ad99d7', '#b39ddb'],
                        borderWidth: 1
                    }]
                },
                options: {
                    responsive: true,
                    plugins: {
                        legend: {
                            position: 'bottom'
                        }
                    }
                }
            });
            </script>
            <div class="container">
                <div class="video-container">
                    <h2>Live Video Feed</h2>
                    <!-- Add buttons to switch video mode
                    <button id="raw-stream-btn">Raw Stream</button>
                    <button id="opencv-stream-btn">OpenCV Stream</button> -->
                    <img id="video-feed" src="/video_feed" alt="Video Feed">
                </div>
                <div class="stats-container">
                    <h2>Real-Time Statistics</h2>
                    <div id="stats">
                        <!-- <p>Total Vehicles: <span id="total-
vehicles">0</span></p> -->
                        <p>Front Distance: <span id="front-distance">0</span>
cm</p>
                        <p>Back Distance: <span id="back-distance">0</span>
cm</p>
                    </div>
                </div>
                <div class="chart-container">
                    <h2>Vehicle SpeedChart</h2>
                    <canvas id="speedChart"></canvas>
                    <!-- Chart will be dynamically inserted here -->
                </div>
                <div class="extra-container">
                    <h2>Vehicle PieChart</h2>
                    <div class="piechart-container">

                        <canvas id="myPieChart"></canvas>
                      </div>
                    </div>

                <div id="logs-container">
                    <h2>Logs</h2>
```

```
                    <div id="logs-content">
                        <table id="logs-table">
                            <thead>
                                <tr>
                                    <th>Timestamp</th>
                                    <th>Vehicle ID</th>
                                    <th>Vehicle Type</th>
                                    <th>Speed</th>
                                    <th>Front Distance</th>
                                    <th>Back Distance</th>
                                    <th>Overtaking Event</th>
                                </tr>
                            </thead>
                            <tbody id="logs-list">
                                <!-- Logs will be dynamically inserted here -->
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </main>
        <script src="/static/js/dashboard.js"></script>
    </body>
    </html>
```

F. DASHBOARD.JS:

```
    let speedChart = null;
    let pieChart = null;
    document.addEventListener('DOMContentLoaded', () => {
        initDashboard();
        setInterval(initDashboard, 5000); // auto refresh
    });

    async function initDashboard() {
        const logs = await fetchLogs();
        updateLogsTable(logs);
        // updateStats(logs);
        updateSpeedChart(logs);
        updatePieChart(logs);
        // await updateLiveStats();
    }

    async function fetchMetrics() {
        try {
            const response = await fetch('/metrics');
            const data = await response.json();

            // Update DOM elements
            // document.getElementById('total-vehicles').textContent =
       data.total_vehicle_count || 0;
            document.getElementById('front-distance').textContent =
       data.front_distance || 0;
            document.getElementById('back-distance').textContent =
       data.back_distance || 0;
        } catch (error) {
```

```
            console.error('Error fetching metrics:', error);
        }
    }
    setInterval(fetchMetrics, 500);

    async function fetchLogs() {
        try {
            const response = await fetch('/get_logs');
            return await response.json();
        } catch (error) {
            console.error('Failed to fetch logs:', error);
            return [];
        }
    }

    function updateLogsTable(logs) {
        const logsList = document.getElementById('logs-list');
        logsList.innerHTML = ''; // Clear old logs

        logs.forEach(log => {
            const row = document.createElement('tr');
            row.innerHTML = `
                <td>${log.timestamp}</td>
                <td>${log.vehicle_id}</td>
                <td>${log.vehicle_type}</td>
                <td>${log.speed} km/h</td>
                <td>${log.front}</td>
                <td>${log.back}</td>
                <td>${log.overtaking === 'True' ? 'Yes' : 'No'}</td>
            `;
            logsList.appendChild(row);
        });
    }


    // function updateStats(logs) {
    //     const totalVehicles = logs.length;
    //     document.getElementById('total-vehicles').textContent =
       totalVehicles;

    //     if (totalVehicles > 0) {
    //         const latest = logs[logs.length - 1];
    //         document.getElementById('front-distance').textContent =
       latest.front;
    //         document.getElementById('back-distance').textContent =
       latest.back;
    //     } else {
    //         document.getElementById('front-distance').textContent = '0';
    //         document.getElementById('back-distance').textContent = '0';
    //     }
    // }

    function createSpeedChart() {
        const ctx = document.getElementById('speedChart').getContext('2d');
        speedChart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: [],
                datasets: [{
                    label: 'Speed (km/h)',
                    data: [],
```

```
                borderColor: 'rgba(255, 99, 132, 1)',
                backgroundColor: 'rgba(255, 99, 132, 0.2)',
                borderWidth: 2,
                tension: 0.3,
                fill: true
            }]
        },
        options: {
            responsive: true,
            scales: {
                x: {
                    title: { display: true, text: 'Time' },
                    ticks: { maxTicksLimit: 10 }
                },
                y: {
                    title: { display: true, text: 'Speed (km/h)' },
                    beginAtZero: true
                }
            },

            plugins: {
                legend: { position: 'bottom' }
            }
        }
    });
}

function updateSpeedChart(logs) {
    if (!speedChart) createSpeedChart();

    const timestamps = logs.map(log => log.timestamp);
    const speeds = logs.map(log => parseFloat(log.speed));

    speedChart.data.labels = timestamps;
    speedChart.data.datasets[0].data = speeds;
    speedChart.update();
}

function createPieChart() {
    const ctx = document.getElementById('myPieChart').getContext('2d');
    pieChart = new Chart(ctx, {
        type: 'pie',
        data: {
            labels: [],
            datasets: [{
                data: [],
                backgroundColor: ['#8e2de2', '#ad99d7', '#b39ddb',
   '#ffd54f', '#4db6ac']
            }]
        },
        options: {
            responsive: true,
            plugins: {
                legend: {
                    position: 'bottom'
                }
            }
        }
    });
}
```

```
function updatePieChart(logs) {
    if (!pieChart) createPieChart();

    const typeCounts = {};
    logs.forEach(log => {
        const type = log.vehicle_type;
        typeCounts[type] = (typeCounts[type] || 0) + 1;
    });

    const labels = Object.keys(typeCounts);
    const data = Object.values(typeCounts);

    pieChart.data.labels = labels;
    pieChart.data.datasets[0].data = data;
    pieChart.update();
}
```

G. SERVER:

```
from flask import Flask, render_template, Response, jsonify
import cv2
import numpy as np
import urllib.request
import threading
import queue
import time
import requests
import os
import csv
# from detection import detect_vehicles, draw_detections

app = Flask(__name__)
CSV_LOG_FILE = 'vehicle_logs.csv'

# Global variables
frame_queue = queue.Queue(maxsize=5)
current_frame = None
frame_lock = threading.Lock()
stream = None
bytes_data = b''

metrics_data = {
    "front_distance": 0,
    "back_distance": 0,
    "total_vehicle_count": 0,
    "overtaking": False
}

# Function to simulate ESP32 stream
def gen_frames():
    import cv2
    import urllib.request
    stream = urllib.request.urlopen('http://192.168.137.107/stream')
    bytes_data = b''
    while True:
        try:
```

```
                bytes_data += stream.read(4096)
                a = bytes_data.find(b'\xff\xd8')
                b = bytes_data.find(b'\xff\xd9')
                if a != -1 and b != -1:
                    jpg = bytes_data[a:b + 2]
                    bytes_data = bytes_data[b + 2:]
                    frame = cv2.imdecode(np.frombuffer(jpg, dtype=np.uint8),
    cv2.IMREAD_COLOR)
                    _, buffer = cv2.imencode('.jpg', frame)
                    yield (b'--frame\r\n'
                           b'Content-Type: image/jpeg\r\n\r\n' +
    buffer.tobytes() + b'\r\n')
            except Exception as e:
                print(f"Error in video feed: {e}")
                break

    def fetch_real_time_data():
        while True:
            try:
                response = requests.get(f"http://192.168.137.95/", timeout=5)
                if response.status_code == 200:
                    data = response.json()
                    metrics_data.update({
                        "front_distance": data.get("front", 0),
                        "back_distance": data.get("back", 0),
                        "overtaking": data.get("overtaking", False)
                    })
            except Exception as e:
                print(f"Sensor error: {e}")
            time.sleep(1)

    @app.route('/')
    def index():
        return render_template('dashboard.html')

    @app.route('/video_feed')
    def video_feed():
        return Response(gen_frames(), mimetype='multipart/x-mixed-replace;
       boundary=frame')

    @app.route('/metrics')
    def get_metrics():
        return jsonify(metrics_data)

    @app.route('/get_logs')
    def get_logs():
        logs = []
        try:
            with open(CSV_LOG_FILE, mode='r') as file:
                reader = csv.DictReader(file)
                for row in reader:
                    logs.append({
                        'timestamp': row['Timestamp'],
                        'vehicle_id': row['Vehicle ID'],
                        'vehicle_type': row['Vehicle Type'],
                        'speed': row['Speed'],
                        'front': row['Front'],
                        'back': row['Back'],
                        'overtaking': row['Overtaking']
                    })
```

```
        except FileNotFoundError:
            pass  # Return empty list if file not found
        return jsonify(logs)

    if __name__== '__main__':
        threading.Thread(target=fetch_real_time_data, daemon=True).start()
        app.run(debug=True)
```

H. CONFIGUARATION ON PYTHON:

```
    import os
    import cv2
    import numpy as np
    import urllib.request
    import threading
    import queue
    import time
    import csv
    import json
    import requests

    # CONFIGURATION
    ESP32_CAM_STREAM_URL = 'http://192.168.137.16/stream'
    NODEMCU_JSON_URL = 'http://192.168.137.131/'  # Replace with your NodeMCU
       IP
    FRAME_RATE = 15
    CSV_LOG_FILE = "vehicle_log.csv"

    # Queue for frames
    frame_queue = queue.Queue(maxsize=5)

    # Load YOLOv3-tiny
    net = cv2.dnn.readNet('Datasets/yolov3-tiny.weights', 'Datasets/yolov3-
       tiny.cfg')
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

    # Load COCO classes
    with open("Datasets/coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]

    # Vehicle tracking
    vehicle_positions = {}
    tracked_vehicles = set()
    last_detection_time = time.time()

    # Create CSV file and headers
    file_exists = os.path.isfile(CSV_LOG_FILE)
    with open(CSV_LOG_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)
        # Write headers only if the file is new
        if not file_exists:
            writer.writerow(["Timestamp", "Vehicle_ID", "Vehicle_Type",
       "Speed", "Front_Distance", "Back_Distance", "Overtaking"])
```

```
# Logging function
def log_event(vehicle_id, vehicle_type, speed, front, back,
    overtaking=False):
     with open(CSV_LOG_FILE, mode='a', newline='') as file:
         writer = csv.writer(file, quoting=csv.QUOTE_MINIMAL)
         writer.writerow([
             time.strftime("%Y-%m-%d %H:%M:%S"),
             str(vehicle_id),
             str(vehicle_type),
             f"{speed:.2f}",
             str(front),
             str(back),
             bool(overtaking)
         ])
         file.flush()  # Optional: Ensures data is written to disk
    immediately
latest_distance_data = {"front": 0, "back": 0, "overtaking": False}

def distance_updater():
     global latest_distance_data
     while True:
         try:
             response = requests.get(NODEMCU_JSON_URL, timeout=2.0)
             if response.status_code == 200:
                 data = response.json()

                 front_raw = data.get("front")
                 back_raw = data.get("back")

                 front = int(front_raw) if isinstance(front_raw, (int,
    float)) else 0
                 back = int(back_raw) if isinstance(back_raw, (int, float))
    else 0

                 latest_distance_data["front"] = front
                 latest_distance_data["back"] = back

                 # Infer overtaking based on back distance
                 if back >= 80:
                     inferred_overtaking = True
                 elif 70 <= back < 80:
                     inferred_overtaking = True
                 elif 60 <= back < 70:
                     inferred_overtaking = False
                 else:
                     inferred_overtaking = False
                 print(f"[Distance] Front: {front}, Back: {back},
    Overtaking: {inferred_overtaking}")

                 latest_distance_data["overtaking"] = inferred_overtaking
             else:
                 raise ValueError(f"Bad HTTP response:
    {response.status_code}")
         except Exception as e:
             print(f"[Distance Fetch Error] {e}")
             latest_distance_data["front"] = 0
             latest_distance_data["back"] = 0
             latest_distance_data["overtaking"] = False
         time.sleep(1.0)
```

```
    # YOLO Detection
    def detect_vehicles(frame):
        height, width, _ = frame.shape
        blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0),
       True, crop=False)
        net.setInput(blob)
        outs = net.forward(output_layers)
        detections = []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.3 and classes[class_id] in ['car', 'bus',
       'truck', 'bicycle', 'motorbike']:
                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)
                    detections.append((classes[class_id], confidence, x, y, w,
       h, center_x, center_y))
        return detections
    # Speed Estimation
    def estimate_speed(vehicle_id, current_pos):
        if vehicle_id in vehicle_positions:
            prev_pos = vehicle_positions[vehicle_id]
            distance = np.sqrt((current_pos[0] - prev_pos[0]) ** 2 +
       (current_pos[1] - prev_pos[1]) ** 2)
            speed = distance * FRAME_RATE
            vehicle_positions[vehicle_id] = current_pos
        else:
            vehicle_positions[vehicle_id] = current_pos
            speed = 0
        return speed

    # Frame fetching thread
    def frame_fetcher():
        stream = urllib.request.urlopen(ESP32_CAM_STREAM_URL)
        bytes_data = b''
        while True:
            bytes_data += stream.read(4096)
            a = bytes_data.find(b'\xff\xd8')
            b = bytes_data.find(b'\xff\xd9')
            if a != -1 and b != -1:
                jpg = bytes_data[a:b + 2]
                bytes_data = bytes_data[b + 2:]
                frame = cv2.imdecode(np.frombuffer(jpg, dtype=np.uint8),
       cv2.IMREAD_COLOR)
                if not frame_queue.full():
                    frame_queue.put(frame)

    # Start threads
    threading.Thread(target=frame_fetcher, daemon=True).start()
    threading.Thread(target=distance_updater, daemon=True).start()

    # Main loop
    total_vehicle_count = 0
    detections = []
```

```
while True:
    if not frame_queue.empty():
        frame = frame_queue.get()
        current_time = time.time()

        if current_time - last_detection_time >= 0.5:
            detections = detect_vehicles(frame)
            last_detection_time = current_time

        # Get sensor data
        front_dist = latest_distance_data.get("front", 0)
        back_dist = latest_distance_data.get("back", 0)
        overtaking_flag = latest_distance_data.get("overtaking", False)

        for idx, (label, conf, x, y, w, h, cx, cy) in
   enumerate(detections):
            vehicle_id = f"{label}_{idx}"
            if vehicle_id not in tracked_vehicles:
                tracked_vehicles.add(vehicle_id)
                total_vehicle_count += 1

            speed = estimate_speed(vehicle_id, (cx, cy))

            # Annotate
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame, f"{label} {conf:.2f}", (x, y - 5),
   cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, f"Speed: {speed:.1f}", (x, y - 30),
   cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

            # Log
            log_event(vehicle_id, label, speed, front_dist, back_dist,
   overtaking_flag)

        # Display extra info
        cv2.putText(frame, f"Total Vehicles: {total_vehicle_count}", (10,
   30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
        cv2.putText(frame, f"Front: {latest_distance_data['front']} cm",
   (10, 60),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 100, 100), 2)
        cv2.putText(frame, f"Back: {latest_distance_data['back']} cm", (10,
   90),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (100, 255, 100), 2)
        if bool(overtaking_flag):
            cv2.putText(frame, "OVERTAKING DETECTED", (10, 120),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)

        cv2.imshow("ESP32-CAM Stream", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

cv2.destroyAllWindows()
```