## ABSTRACT

Drivers view the most common road accident as a result of not being able to see, if they are behind a vehicle such as a truck. The highway mechanism that can be especially dangerous to attempt is overtaking. And poor visibility coupled with blind spots means drivers cannot judge if it's safe to overtake, raising the risk of serious collisions. In answer to this problem, Vision-Guard can help. With the help of cameras and other devices, the system gives a real-time picture of what is happening in front on its screen at the rear of any truck. It measures distance to vehicles nearby using sensors, and alerts when they're too close. The device objective is to do so prevent collisions, improve visibility and also safer overtaking decisions. Devoted to creating an even more secure driving environment for all the citizens of any country, this project is really a helpful to what technology can offer.

## 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 **Dr Umema, Miss Hira Anwar, Sir Usman, 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

[ABSTRACT i](#_bookmark0)

[ACKNOWLEDGEMENT ii](#_bookmark1)

[DEDICATION iii](#_bookmark2)

[LIST OF FIGUERS vii](#_bookmark3)

[LIST OF TABLES viii](#_bookmark4)

[CHAPTER 1 1](#_bookmark5)

1. [Introduction 1](#_bookmark6)
   1. [Motivations 1](#_bookmark7)
   2. [Problem Statement 1](#_bookmark8)
   3. [Objectives and Contributions 1](#_bookmark9)
   4. [Project Scope 2](#_bookmark10)
   5. [Organization of the Report 2](#_bookmark11)

[CHAPTER 2 3](#_bookmark12)

1. [Literature Review/Process Review 3](#_bookmark13)
   1. [Introduction 3](#_bookmark14)
   2. [Literature Review 3](#_bookmark15)
   3. [Functional and Non Functional Requirements 3](#_bookmark16)
      1. [Functional Requirements 3](#_bookmark17)
      2. [Non-Functional Requirements 4](#_bookmark18)
   4. [Project Significance 4](#_bookmark19)
   5. [Software Platform 4](#_bookmark20)
   6. [Scalability 4](#_bookmark21)
   7. [Services 4](#_bookmark22)

[CHAPTER 3 5](#_bookmark23)

* 1. [Projects diagrams 5](#_bookmark24)
     1. [System Block Diagram: 5](#_bookmark26)
     2. [Flowchart 6](#_bookmark28)
     3. [ESP32 CAMERA FTDI programmer: 7](#_bookmark29)
     4. [Pin-Diagram ESP32: 7](#_bookmark30)
     5. [Sensor node MCU pin: 8](#_bookmark31)
     6. [Activity Diagram: 9](#_bookmark32)
     7. [Collaboration Diagram: 9](#_bookmark33)
     8. [Swimlane Diagram 10](#_bookmark34)
     9. [Dataflow Diagram 10](#_bookmark35)
     10. [Usecase Diagram: 11](#_bookmark36)
  2. [Inside Project 11](#_bookmark37)
     1. [Frontend on html css 11](#_bookmark38)
     2. [Development 11](#_bookmark39)
     3. [Model Training 11](#_bookmark40)
     4. [Flask Dashboard 11](#_bookmark41)
  3. [Technologies Used 12](#_bookmark42)
     1. [Development 12](#_bookmark43)
     2. [Trained Model 12](#_bookmark44)
     3. [Development and Hosting 12](#_bookmark45)

[Chapter NO 4 13](#_bookmark46)

1. [Project Planning 13](#_bookmark47)
   1. [Project Timeline Summary 13](#_bookmark48)
   2. [Project Timeline Details 13](#_bookmark49)
   3. [Black-box Testing 13](#_bookmark50)
   4. [Test Cases 15](#_bookmark51)

[CHAPTER NO 5 17](#_bookmark52)

1. [GUI of Android Application 17](#_bookmark53)
   1. [Login Interface 17](#_bookmark54)
   2. [Sign up Interface 17](#_bookmark55)
   3. [Student Panel 17](#_bookmark56)
   4. [Teacher Panel 17](#_bookmark57)

[CHAPTER 6 18](#_bookmark58)

1. [Conclusion and Future Work 18](#_bookmark59)
   1. [Limitation 18](#_bookmark60)
   2. [Conclusion 18](#_bookmark61)
   3. [Future Works 18](#_bookmark62)

[References 19](#_bookmark63)

[Appendix 20](#_bookmark64)

1. [Coding – Server Side Coding 20](#_bookmark65)
2. [Coding – JASON Package 20](#_bookmark66)
3. [Coding – Login File 21](#_bookmark67)
4. [Gantt chart 21](#_bookmark68)

## LIST OF FIGUERS

|  |  |  |  |
| --- | --- | --- | --- |
| **SERIAL NO.** | **FIGURE** | **TITLE** | **PAGE NO.** |
| 01 | 3.1.1 | System block diagram | 05 |
| 02 | 3.1.2 | System architecture diagram | 05 |
| 03 | 3.1.3 | Flowchart | 06 |
| 04 | 3.1.4 | ESP 32 camera FTDI programmer | 07 |
| 05 | 3.1.5 | Pin diagram ESP 32 | 07 |
| 06 | 3.1.6 | Sensor node MCU pin | 08 |
| 07 | 3.1.7 | Activity diagram | 09 |
| 08 | 3.1.8 | Collaboration diagram | 09 |
| 09 | 3.1.9 | Swimlane diagram | 10 |
| 10 | 3.1.10 | Data flow diagram | 10 |
| 11 | 3.1.11 | Use case diagram | 11 |

## LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **TABLE NO.** | **TABLE NAME** | **PAGE NO.** |
| **Table No. 4.1.1.** | **Project Timeline Summary** | **14** |

# CHAPTER 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.

#### Motivations

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.

#### 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.

#### 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.

#### 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.

#### 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

### LITERATURE REVIEW

#### 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.

#### 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.

#### Functional and Non-Functional Requirements

##### 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.

##### 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.

#### 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.

#### Software Platform

Web application

#### 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.

#### Services

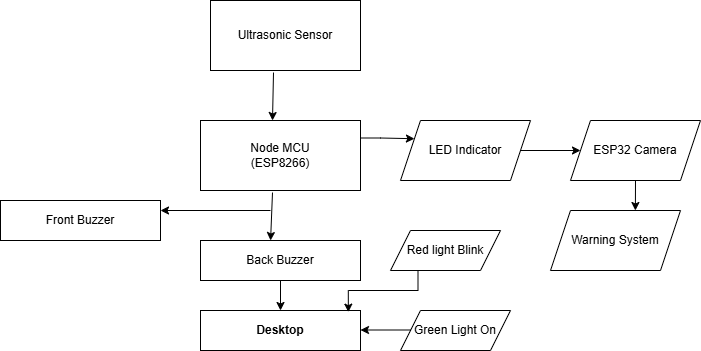
To enhance the user experience, the Vision-Guard system offers several key features. First, the system provides a real time video stream to help drivers see the road ahead, especially when following large vehicles like trucks. This allows drivers to make better decisions while overtaking, improving safety. 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

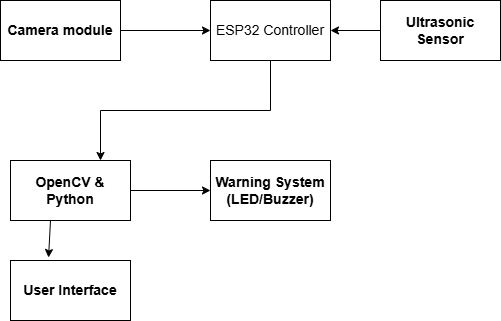
#### Projects diagrams

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.

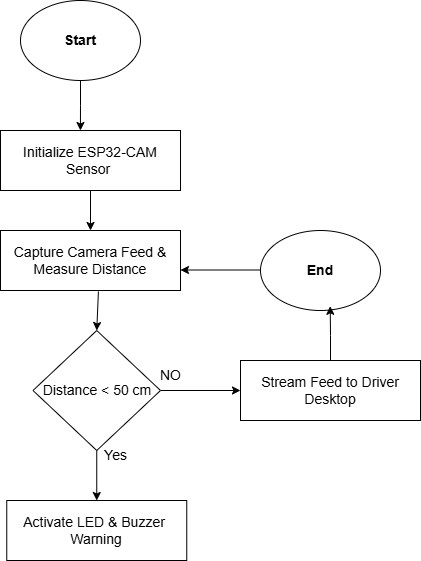
#### System Architecture Diagram:

****

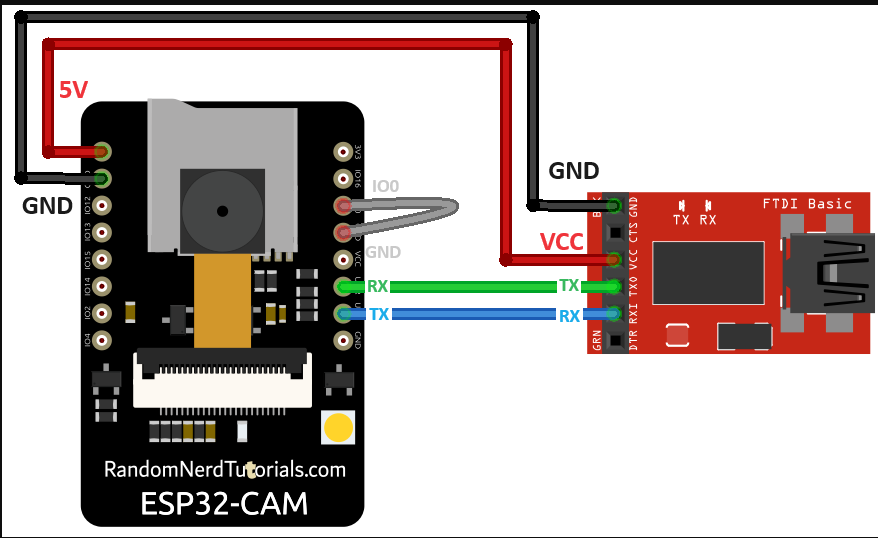
* + 1. **System Block Diagram:**

****

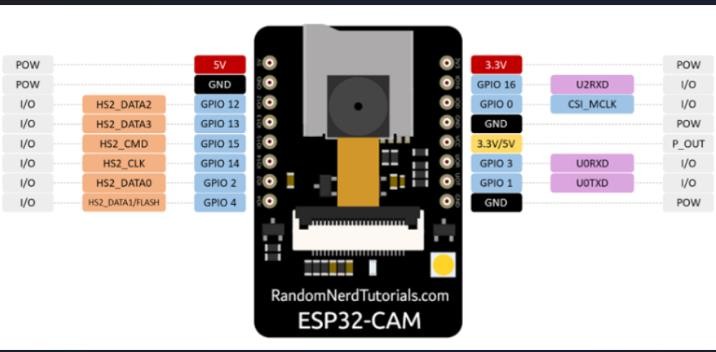
#### Flowchart

****

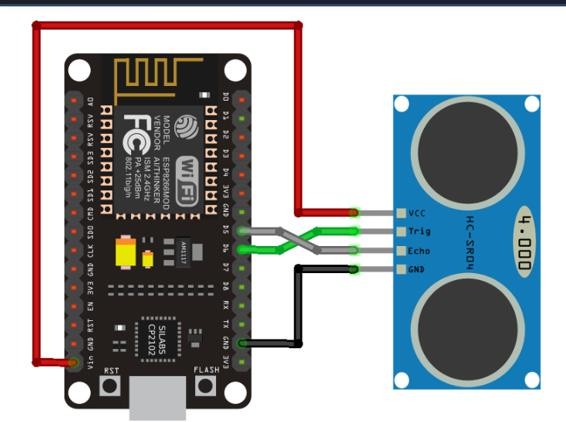
#### ESP32 CAMERA FTDI programmer:



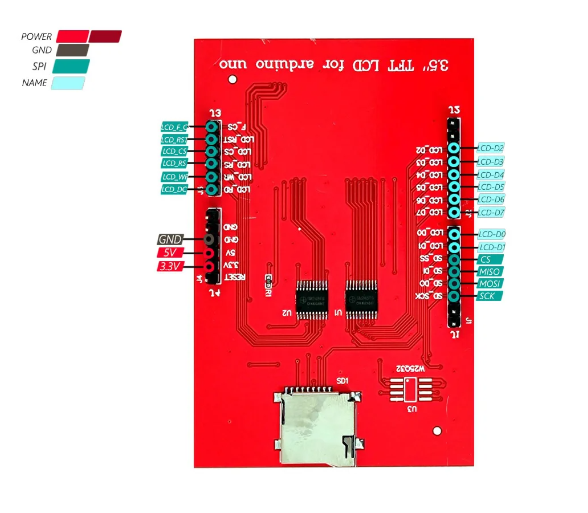
#### Pin-Diagram ESP32:

****

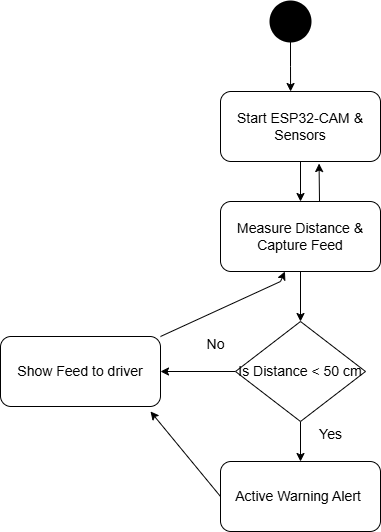
#### Sensor node MCU pin:

****

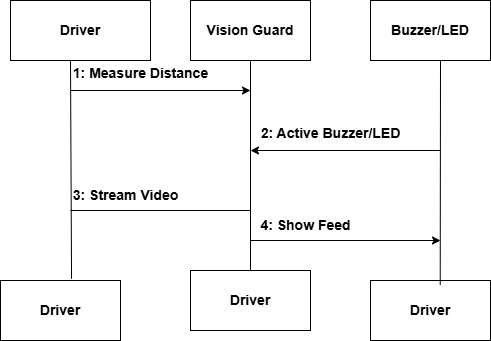
* + 1. **SPI TFT LCD:**

****

#### Activity Diagram:

****

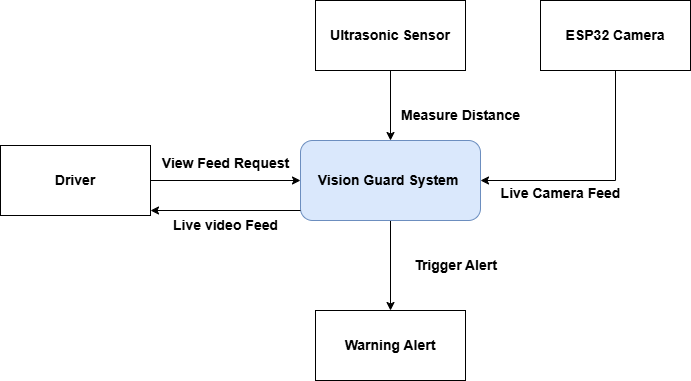
#### Collaboration Diagram:

****

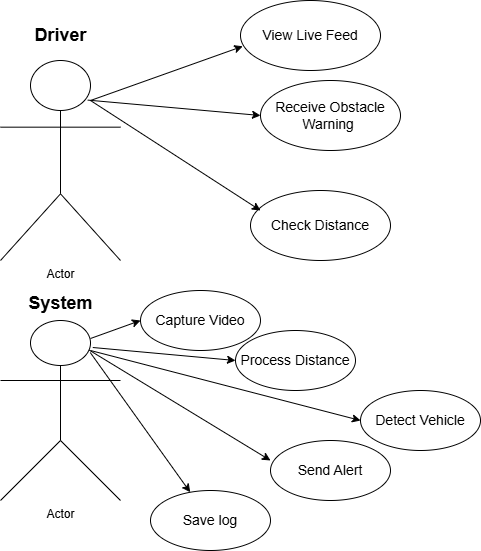
#### Swimlane Diagram:

#### 

#### Dataflow Diagram:

****

#### Use case Diagram:

****

#### Inside Project

#### Frontend on html css

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

#### Development

Utilized Visual Studio Code for coding and debugging.

#### Model Training

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

#### Flask Dashboard

Developed a Flask-based dashboard to view.

* 1. **Technologies Used** **3.13.1.Development**
* Framework: VS code
* Programming Language: Python
* IDE: Arduino IDE

#### Trained Model

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

#### Development and Hosting

* + - * Framework: Flask

# CHAPTER NO 4

#### Project Planning

##### 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 |

##### 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.

##### *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

###### 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.

###### 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.

###### 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.

###### 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.

##### 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.

* + 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** | QA |
| **Status** | Pass |

****

****

* + 1. **TEST CASE # 2**

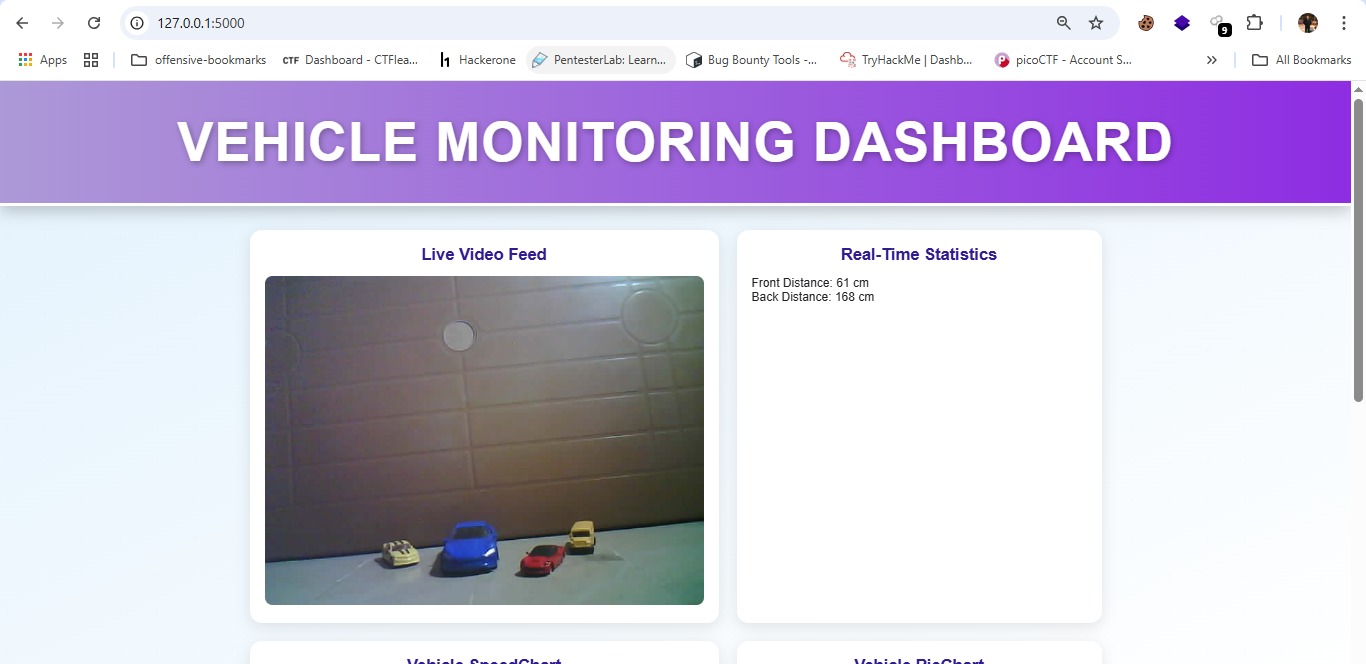
|  |  |
| --- | --- |
| **Test Case Title** | Safe Distance Alert |
| **Test Case ID** | VG\_TC\_002 |
| **Test Case Objective** | Check the led light begin when another vehicle is too close. |
| **Pre-Conditions** | Led lights must be connected and active. |
| **Test Steps** | * Bring another vehicle 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** | QA |
| **Status** | Pass |

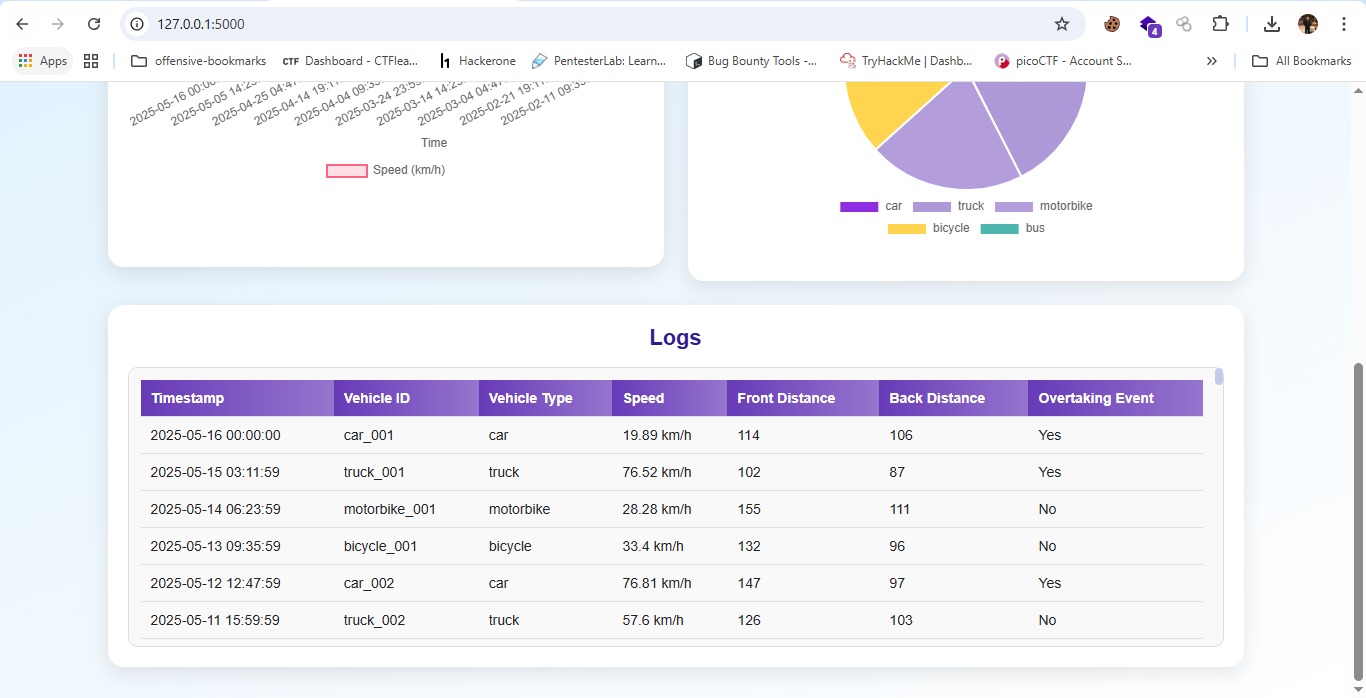
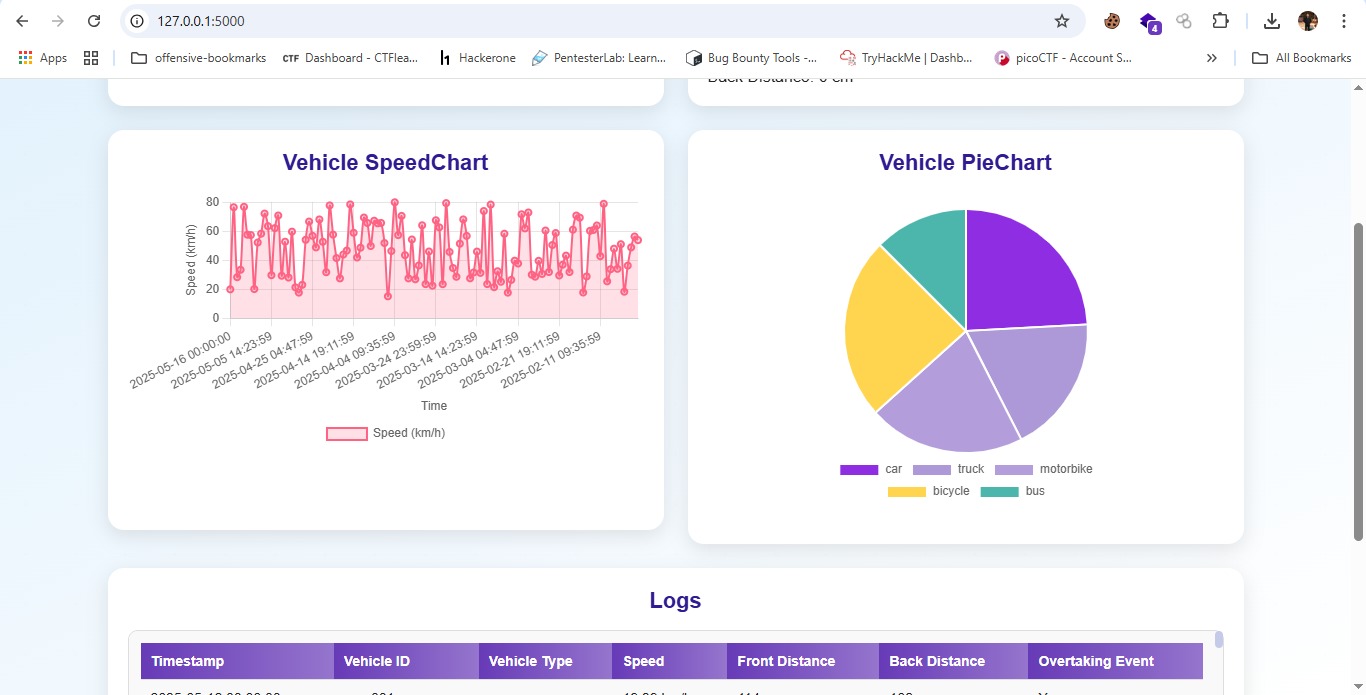
****

****

* + 1. **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** | QA |
| **Status** | Pass |

****

****

* + 1. **TEST CASE # 4**

|  |  |
| --- | --- |
| **Test Case Title** | Easy User Interface. |
| **Test Case ID** | VG\_TC\_004 |
| **Test Case Objective** | Confirm that the system interface is simple to use. |
| **Pre-Conditions** | System must be installed & working. |
| **Test Steps** | * Open the Vision-Guard interface. * Use all main features (video, alerts, data). * Check if everything works easily. |
| **Expected Result** | All features are easy to find and use without needing extra help. |
| **Actual Result** | Every features are easy to use. |
| **Test By** | QA |
| **Status** | Pass |

* + 1. **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** | QA |
| **Status** | Pass |

# CHAPTER NO 5

#### GUI of Android Application

##### Login Interface

##### Sign up Interface

##### Student Panel

##### Teacher Panel

# CHAPTER 6

#### Conclusion and Future Work

##### 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, restricting 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.

##### 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.

* 1. **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 vehicles.

## 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

#### ESP-32 CAM:

|  |
| --- |
| #include "esp\_camera.h"#include <WiFi.h>#include "esp\_http\_server.h"// Replace with your network credentialsconst char\* ssid = "DESKTOP-F614S19";const char\* password = "Error404";// Function to handle video streamingesp\_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 multiparthttpd\_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 necessarybool 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 headersize\_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 imageres = httpd\_resp\_send\_chunk(req, (const char \*)\_jpg\_buf, \_jpg\_buf\_len);}if (res == ESP\_OK) {// Send the frame boundaryres = 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 servervoid 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 streamhttpd\_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 latencyWiFi.setSleep(false);// Connect to Wi-FiWiFi.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 configurationcamera\_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 streamconfig.jpeg\_quality = 12; // Reduce image quality to decrease delayconfig.fb\_count = 1; // Use single frame buffer for lower latency// Initialize the cameraesp\_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 serverstartCameraServer();}void loop() {delay(1);} |

#### ARDUINO UNO:

|  |
| --- |
| #include <MCUFRIEND\_kbv.h>MCUFRIEND\_kbv tft;#define WIDTH 480#define HEIGHT 320uint16\_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 sizefor (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 readytft.setAddrWindow(0, y, WIDTH - 1, y);tft.pushColors(lineBuffer, WIDTH, 1);}}}} |

#### 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 200NewPing 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 distanceconst int safeDistance = 50; // Critical warning if belowvoid 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 closevoid 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 inputvoid 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);} |

1. **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() |

1. **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> |

1. **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();  } |

1. **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) |

1. **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() |