# Chapter 1

# Introduction

RoboSentinel is an intelligent real-time surveillance system designed to enhance modern security using embedded hardware and artificial intelligence. Traditional CCTV systems function as passive monitoring tools, relying heavily on human supervision and lacking the ability to automatically detect or respond to threats. RoboSentinel overcomes these limitations by integrating live video streaming, low-latency image processing, and automated object/person detection into a single compact platform.

The system uses an ESP32-CAM module to capture and transmit video over Wi-Fi, where lightweight AI models analyze each frame in real time. By combining efficient streaming protocols with optimized detection algorithms, RoboSentinel can identify security-relevant events—such as human presence or suspicious activity—and generate timely alerts without manual intervention. Its modular design, affordability, and scalability make it suitable for homes, industries, campuses, and other environments requiring continuous, intelligent monitoring.

## 1.1 Background of Surveillance Systems

Surveillance systems have evolved significantly over the past few decades, transitioning from simple analog video recorders to sophisticated, AI-driven monitoring solutions. Early surveillance relied on basic closed-circuit television (CCTV) setups, where cameras transmitted footage to a central display monitored manually by security personnel. These systems were limited by low video quality, storage constraints, and the inability to automatically identify or respond to security threats.

Today's intelligent surveillance systems can analyze live video, detect objects or people, track movements, and generate alerts automatically. This evolution forms the foundation for innovative projects like RoboSentinel, which combines embedded hardware, wireless communication, and AI-powered analytics to deliver a proactive, low-cost, and efficient security monitoring solution.

As global security needs intensified, the limitations of traditional surveillance—such as lack of automation, high labour requirements, and inability to perform real-time threat detection—became more evident. This led to the integration of artificial intelligence, machine learning, and computer vision into modern surveillance systems. Emerging technologies enabled automated analysis of video streams, allowing systems to detect human presence, classify objects, track movements, and trigger alerts without manual intervention.

These technological evolutions form the backdrop for the development of RoboSentinel, an intelligent real-time surveillance system that leverages embedded hardware and AI-driven analytics to provide proactive, automated, and reliable security monitoring. RoboSentinel represents the next step in transforming traditional surveillance into a smart, autonomous, and scalable solution.

## 1.2 Problem Statement

Traditional surveillance systems, although widely used, suffer from several inherent limitations that reduce their overall effectiveness in modern security environments. Most conventional CCTV setups function as passive monitoring tools and rely completely on human supervision to identify unusual events. This dependency increases the likelihood of human error, delayed response times, and missed critical incidents. Additionally, these systems lack automation and intelligence, making them incapable of independently detecting, analyzing, or responding to potential threats.

Another major concern is latency in video transmission, especially in IP-based streaming systems. High latency affects real-time monitoring and reduces the accuracy of incident detection. Moreover, existing systems often involve expensive hardware, require high maintenance costs, and lack scalability, making them unsuitable for flexible deployment in diverse environments such as homes, industries, and institutions.

Thus, there is a strong need for an intelligent, low-cost, real-time surveillance system that integrates embedded hardware, AI-driven detection, and optimized communication protocols to address these limitations. RoboSentinel is designed specifically to overcome these challenges by providing automated monitoring, efficient streaming, and accurate, real-time threat detection.

## 1.3 Objectives of RoboSentinel

**1. Implement real-time video capturing and streaming**

To establish a stable and continuous IP-based video feed using ESP32-CAM or Raspberry Pi modules, ensuring smooth frame rates and minimal transmission delays.

**2. Enable object and human detection**

To incorporate optimized computer vision models and capable of running efficiently on edge devices for identifying people and detecting objects.

**3. Reduce latency and enhance streaming performance**

To apply resolution tuning, bitrate optimization, and efficient protocol handling to achieve real-time responsiveness suitable for security applications.

**4. Provide automated alerts and event logging**

To notify users instantly whenever unusual activity or human presence is detected, enabling timely response and improved situational awareness.

**5. Ensure scalability, modularity, and low hardware cost**

To design a flexible system architecture that can accommodate additional sensors, cloud integration, and improved AI models without significant redesign or cost increase.

## 1.4 Applications of RoboSentinel

**1. Residential Surveillance**

Monitoring entrances, hallways, parking areas, and private property for intrusions, unauthorized access, or suspicious activity.

**2. Institutional and Campus Security**

Providing real-time surveillance of classrooms, laboratories, administrative blocks, and campus pathways to ensure safety and prevent unauthorized access.

**3. Industrial and Warehouse Monitoring**

Detecting movement in restricted areas, monitoring workers' safety, tracking equipment, and preventing theft or accidents.

**4. Commercial & Retail Surveillance**

Monitoring customer flow, preventing shoplifting, and ensuring the safety of assets and employees in malls, shops, and business centers.

**5. Remote and High-Risk Zone Monitoring**

Useful in environments where constant human presence is difficult or unsafe, such as construction sites, power stations, outskirts, or wildlife monitoring zones.

**6. IoT-Based Smart Security Systems**

It can be integrated with cloud services, mobile apps, alarms, and automation systems for complete smart-security deployment.

# Chapter 2

## Literature Survey

A comprehensive review of existing research in surveillance systems, computer vision, robotics, and intelligent monitoring technologies was conducted to understand current advancements and identify gaps that the RoboSentinel project aims to address

Mohana et al. [1] developed a web-controlled surveillance robot integrated with computer vision for remote monitoring applications. Their work highlights the importance of real-time video processing, wireless control, and autonomous navigation to enhance situational awareness. The system relied heavily on image-processing algorithms to detect objects during patrol. This research provides a strong foundation for the present project, emphasizing the need for reliable video streaming and vision-based analysis similar to the methods used in RoboSentinel.

Patil et al. [2] proposed an IoT and GSM-based robotic system for hazardous gas detection in coal mines. Their work demonstrates how embedded sensors, wireless communication, and autonomous operation can improve safety in dangerous environments. Although their focus was environmental monitoring rather than intrusion detection, the integration of multi-sensor data and automated alert mechanisms is directly relevant. The sensor-fusion concepts applied in their research inspired the inclusion of PIR and ultrasonic sensors in RoboSentinel to strengthen reliability in intrusion confirmation.

Boufares et al. [3] introduced a hybrid approach to moving object detection by combining background subtraction, fuzzy entropy thresholding, and differential evolution optimization. Their method significantly improved detection accuracy under complex backgrounds and dynamic lighting conditions. This paper is particularly relevant, as moving object detection is a core function of the RoboSentinel system. Their findings underline the need for robust preprocessing and threshold control, which guided the inclusion of image enhancement and optimized detection algorithms in the present study.

He et al. [4] presented an advanced learning-based technique for 3D video object detection using object-centric global optimization. Although highly sophisticated and intended for large-scale computer vision applications such as autonomous driving, this research illustrates the efficiency of deep-learning models in improving detection robustness. While RoboSentinel uses classical techniques such as Haar cascades due to resource limitations, the insights from this work indicate strong potential for upgrading the system to deep-learning frameworks (YOLO, SSD, MobileNet) in future implementations.

Liang and Baker [5] investigated real-time background subtraction techniques that adapt to fluctuating lighting conditions—one of the most common challenges in computer vision systems. Their findings demonstrate methods to minimize false detections caused by shadows, bright spots, or sudden illumination changes. This was particularly useful for the RoboSentinel project, where reliable motion detection is essential for triggering alerts. Their research motivated the adoption of preprocessing steps and lighting-independent thresholding in the system workflow.

The reviewed literature demonstrates significant progress in the fields of autonomous surveillance, sensor integration, object detection, and adaptive machine vision. These research insights collectively informed the design decisions of the RoboSentinel system, enabling the development of a more accurate, intelligent, and efficient surveillance solution.

# Chapter 3

# Methodology

## 3.1 Hardware Methodology

### • Hardware Components:

1. **Microcontroller (ESP32):** The central brain, processing sensor data and handling Wi-Fi communication.
2. **ESP32-CAM:** Captures and streams live video for visual monitoring.
3. **L293D Motor Driver:** Controls the speed and direction of the rover's gear motors.
4. **Gear Motors & Chassis:** Provides mobility for the rover.
5. **MQ-135 Gas Sensor:** Detects harmful gases (e.g., smoke, CO2, ammonia).
6. **Temperature and humidity (DHT 11) Sensor:** Senses the temperature and humidity and provides the real-time data.
7. **Power Supply Unit:** Rechargeable batteries and voltage regulators.

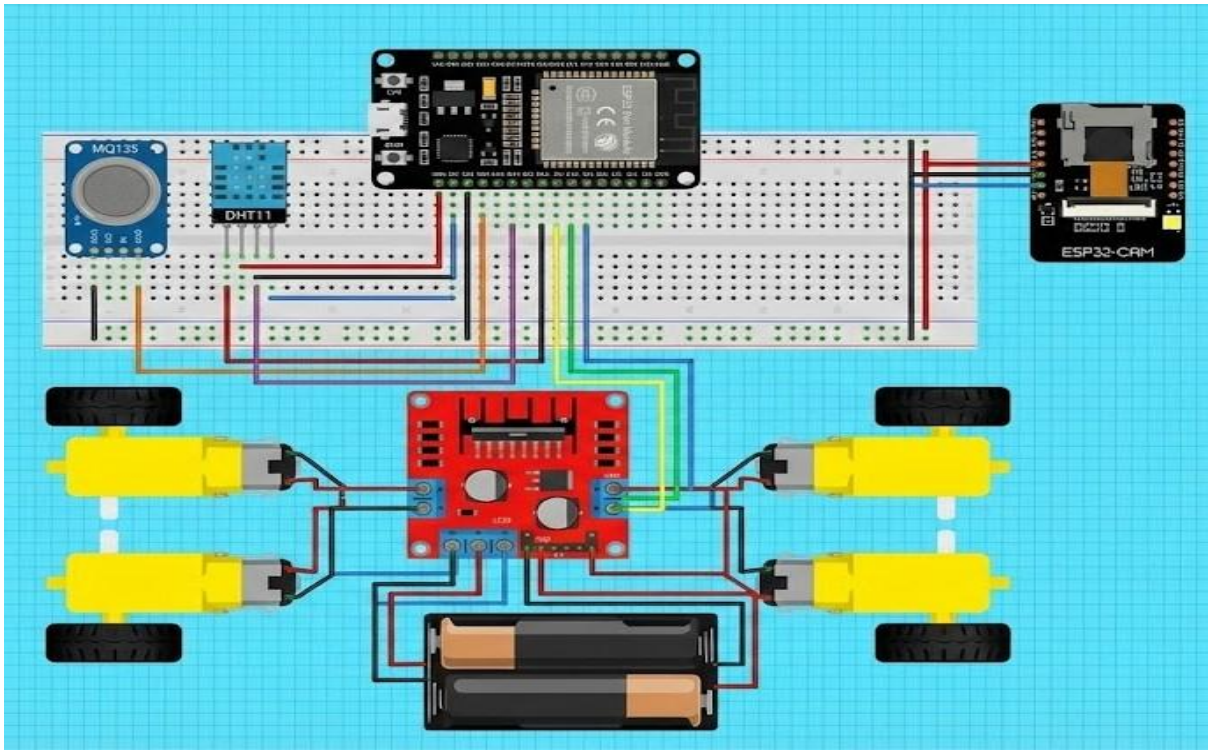


**Fig 3.1** Block Diagram explaining Hardware Methodology

### 3.1.1 Explanation

The system architecture of RoboSentinel is designed to integrate real-time video monitoring, environmental sensing, and automated processing into a unified surveillance platform. The architecture begins by identifying the surveillance requirements, which determine the sensors and components needed for effective monitoring. Based on these requirements, key modules such as the ESP32-CAM for video capture, MQ-135 for gas detection, DHT11 for temperature and humidity sensing, and the L293D motor driver for rover mobility are selected.

The ESP32 microcontroller acts as the central unit that collects data from all sensors and manages communication. The complete system is then assembled onto a mobile rover platform, allowing manual navigation for area inspection. Once operational, the ESP32 streams sensor readings and live video over Wi-Fi to a web server, enabling remote monitoring.

Finally, the streamed data is processed in real time using MATLAB or similar tools, where camera frames and sensor values are synchronized, analyzed, and used for intelligent decision-making. This multi-layered architecture ensures efficient data collection, reliable transmission, and intelligent processing, making RoboSentinel a capable and autonomous surveillance solution.

## 3.2 Software Methodology

### • Software Components:

**MATLAB:**

Used for high-level image processing.

Acquires the live video stream.

Runs feature matching algorithms to compare live video with a reference image.

Triggers alerts based on detection.

**Web Dashboard:**

Provides a user-friendly interface for remote manual control of the rover.

Displays real-time sensor data (e.g., gas levels).

**Arduino IDE:**

Used to program the ESP32 microcontroller.

Manages sensor reading, motor control, and communication with MATLAB.

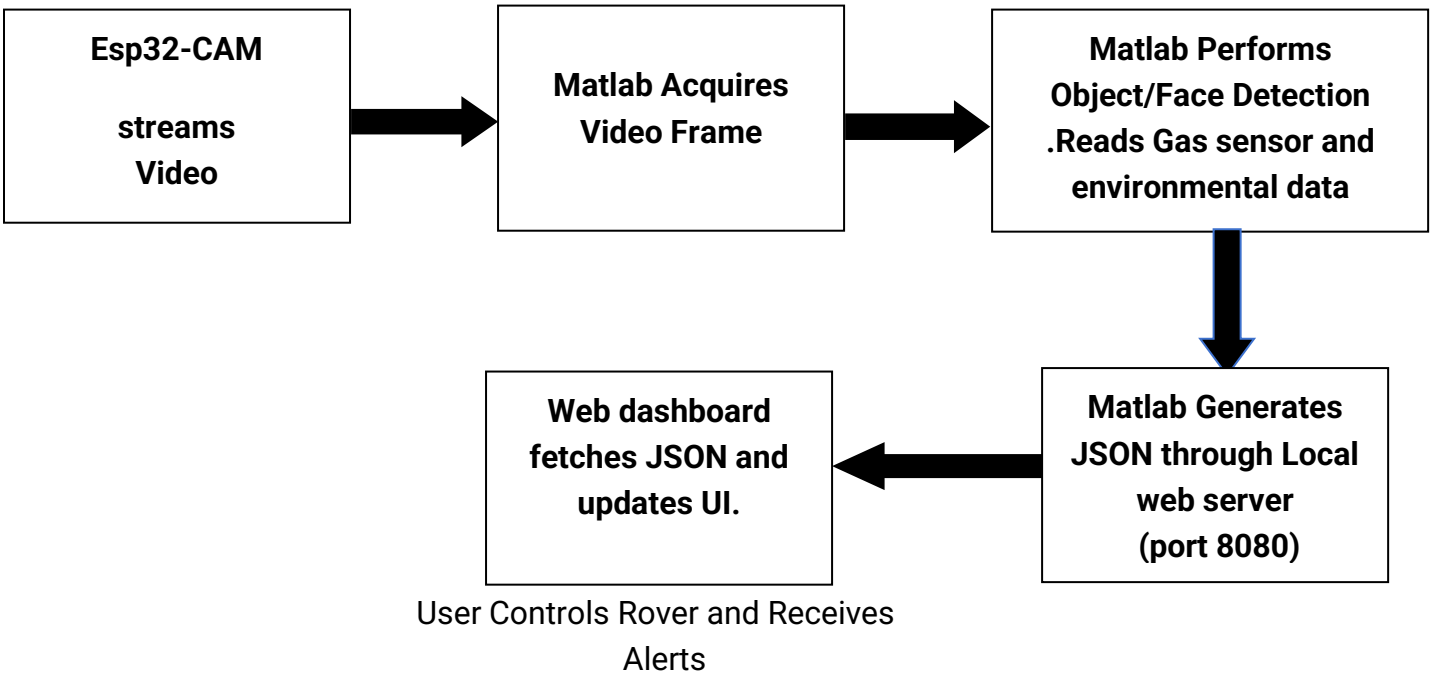


**Fig 3.2** Block Diagram explaining Software Methodology

### 3.2.1 Steps

1. **Power up & connect hardware**
   Power the ESP32-CAM, sensors (MQ-135, DHT11), motor driver (L293D) and ensure the Wi-Fi router is running.
2. **Configure ESP32 network**
   Program the ESP32 (Arduino IDE) with Wi-Fi credentials so it joins the LAN and exposes endpoints (video stream and sensor HTTP endpoints).
3. **Start camera stream**
   Launch the ESP32-CAM streaming server (MJPEG/HTTP or RTSP). Confirm the video URL (e.g. http://<ESP_IP>:8080/video) is accessible in a browser.
4. **Expose sensor & control endpoints**
   Implement HTTP endpoints on ESP32 for sensor readings and control command, returning JSON for sensor data.
5. **Verify endpoints**
   Test the camera stream and sensor JSON endpoints from a PC to ensure reliable connectivity and correct data format.
6. **Connect MATLAB to stream and sensors**
   In MATLAB create an ipcam object for the stream and use webread to poll the /sensor endpoint; verify frames and sensor values are received.
7. **Frame acquisition loop**
   Implement a continuous loop in MATLAB: snapshot(ipcam) → timestamp → webread(sensor) → send frame+sensor to processing pipeline.

8. **Preprocess frames**
   Enhance each frame (resize, convert to grayscale if needed, histogram equalization / CLAHE, denoise) to improve detection robustness.

9. **Run object/face detection**
   Apply detectors: face cascade for faces and either feature-matching (SURF/ORB) or a lightweight CNN (MobileNet/YOLO-tiny) for objects; obtain bounding boxes & confidence scores.

10. **Fuse sensor data with detections**
    Combine detection results with current sensor readings (gas, temp, humidity). Apply decision rules (e.g., person detected AND gas > threshold).

11. **Generate alerts**
    If decision rules are met, trigger alerts: POST to dashboard webhook, send email/SMS, or call ESP32 control endpoint to stop/move rover — include image and sensor data.

12. **Dashboard update & manual control**
    Ensure the web dashboard fetches the live stream and sensor JSON frequently; provide UI controls to remotely drive the rover and acknowledge

# Chapter 4

# Implementation

## 4.1 Working of the RoboSentinel System

**Step 1:** Real-Time Data Acquisition

**Step 2:** Wireless Transmission of Data

**Step 3:** MATLAB Processing and Analysis

**Step 4:** Decision-Making & Event Triggering

**Step 5:** Visualization and User Interaction

**Step 6**: Logging, Storage, and Alerts

## 4.2 Key Hardware and Software Configurations

### 4.2.1 Hardware Configurations

**ESP32-CAM Setup**

- Configured with OV2640 camera parameters
- Connected to 5V stable power supply
- Wi-Fi credentials programmed via Arduino IDE
- Streaming server enabled

**Sensors**

- MQ-135 connected to analog pin A0 for air quality monitoring
- DHT11 connected to digital pin for temperature and humidity
- Periodic sampling configured in firmware

**Motor Driver (L293D) and Rover Base**

- Controls directional movement: forward, reverse, left, right
- PWM signals used for speed control
- Motors powered separately to avoid ESP32 resets.

### 4.2.2 Software Configurations

**Arduino IDE (ESP32 Firmware)**

- Wi-Fi connection setup
- HTTP server for data streaming and sensor API
- Motor control logic
- JSON-formatted sensor output for MATLAB and dashboard

**MATLAB**

- Ipcam object for video acquisition
- Image enhancement pipeline
- Feature matching / face detection algorithms
- Decision-making rules
- Alert and event logging modules

**Web Dashboard**

- Fetches sensor data from ESP32 regularly
- Displays the camera feed directly
- Provides manual rover controls
- Shows alerts and system status

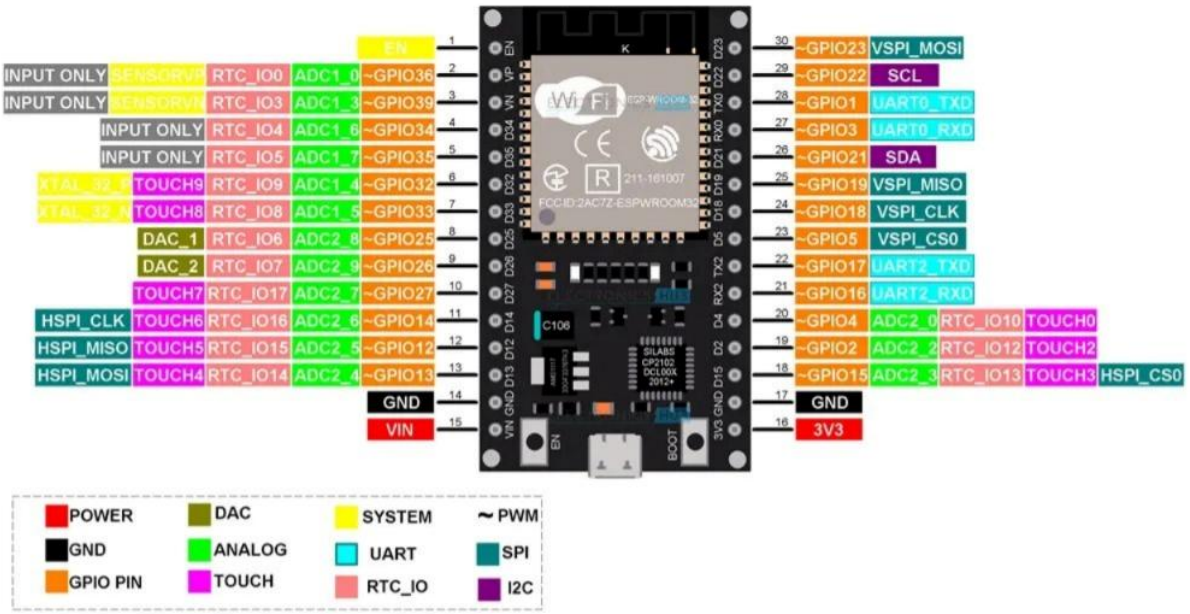


**fig 4.1 Key Configuration of RoboSentinel**

## 4.3 Features Implemented

**1. Real-Time Video Streaming**

ESP32-CAM streams high-frame-rate live video over Wi-Fi for real-time monitoring.

**2. Environmental Monitoring**

MQ-135 and DHT11 sensors continuously provide gas, temperature, and humidity data.

**3. MATLAB-Based Object & Face Detection**

- Feature matching for object recognition
- Bounding boxes & confidence scores

**4. Image Enhancement**

Enhanced clarity improves detection accuracy under varying lighting conditions.

**5. Intelligent Alert System**

Alerts are triggered when:

- A person or object is detected,
- Gas levels exceed safe limits,
- Sensor anomalies occur.

Alerts may include:

- MATLAB pop-ups
- Web dashboard notifications
- Event logs with timestamped images

**6. Manual Rover Control**

The web dashboard allows manual operation such as stop or redirect the rover based on command.

# CHAPTER 5

## Results

### 5.1 Hardware Output

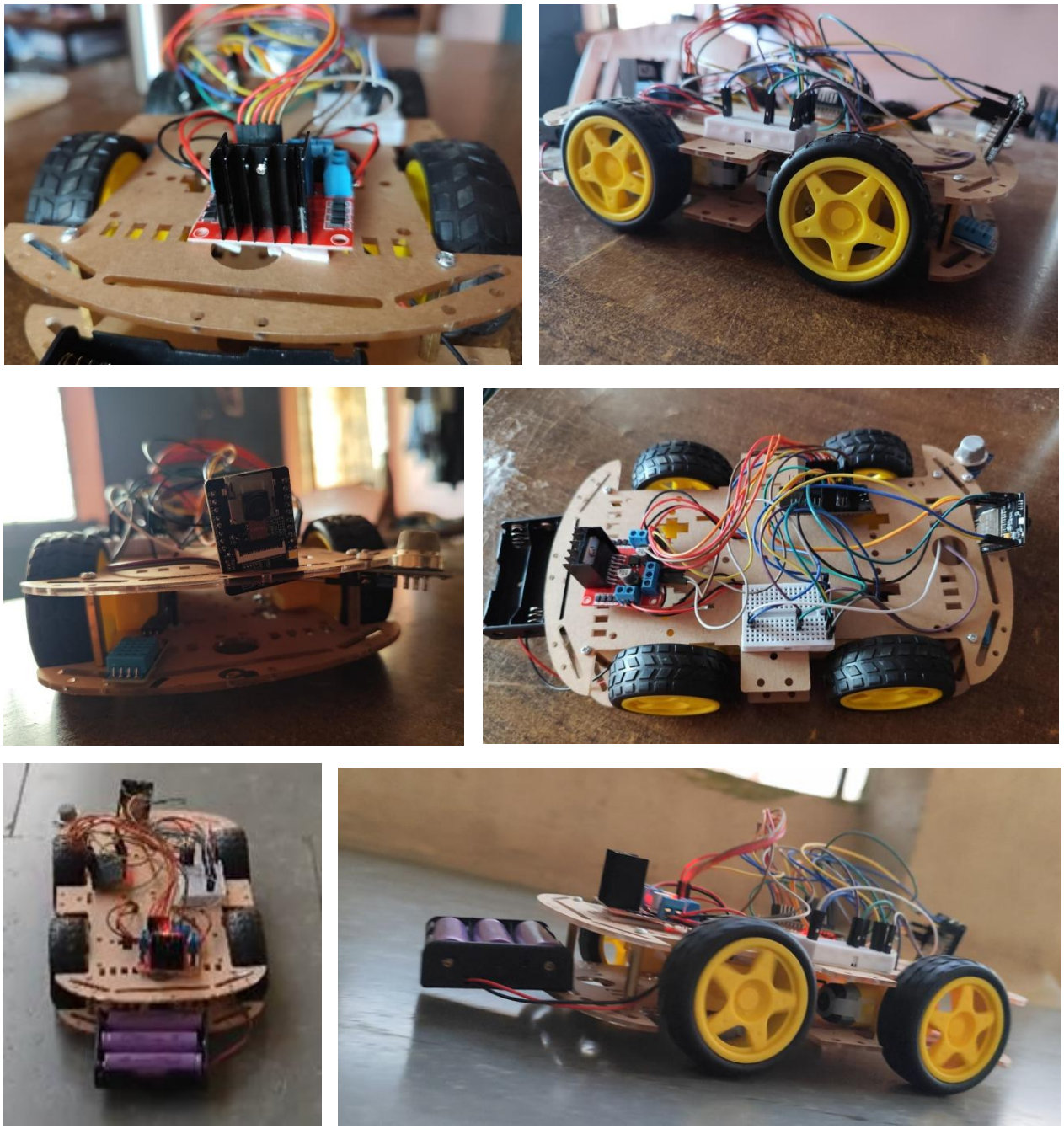

**fig 5.1.1 Hardware-working model of rover**

## 5.2 Detection and Streaming results

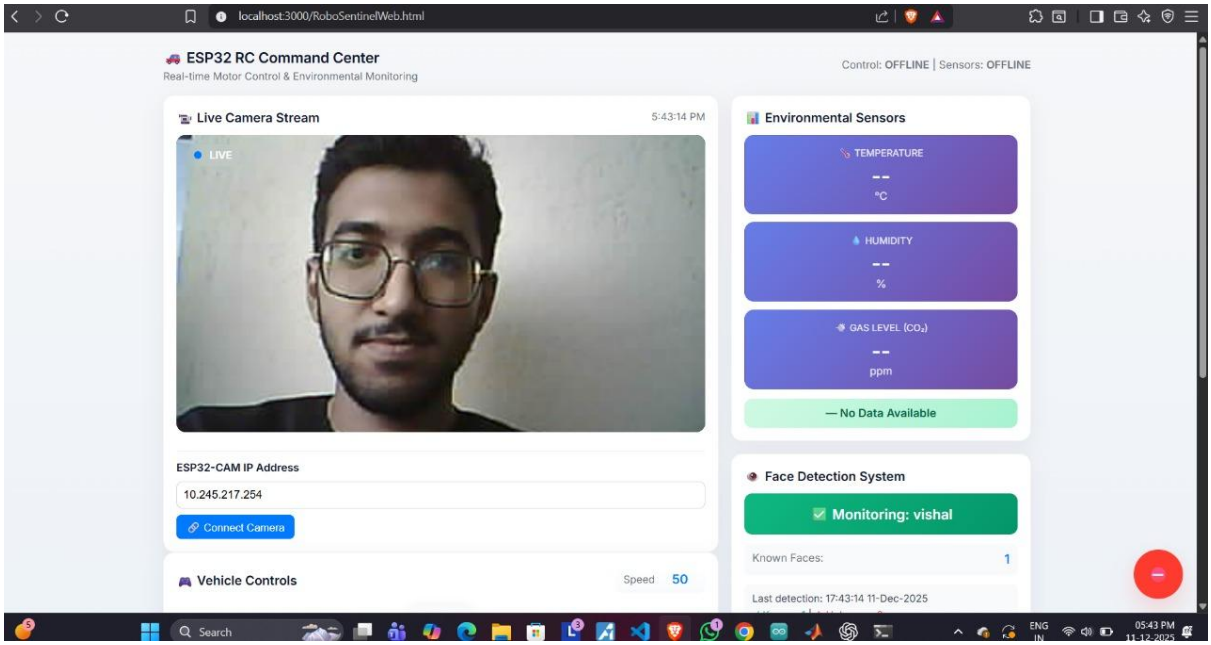


**fig 5.2.1 Face detection in web interface**

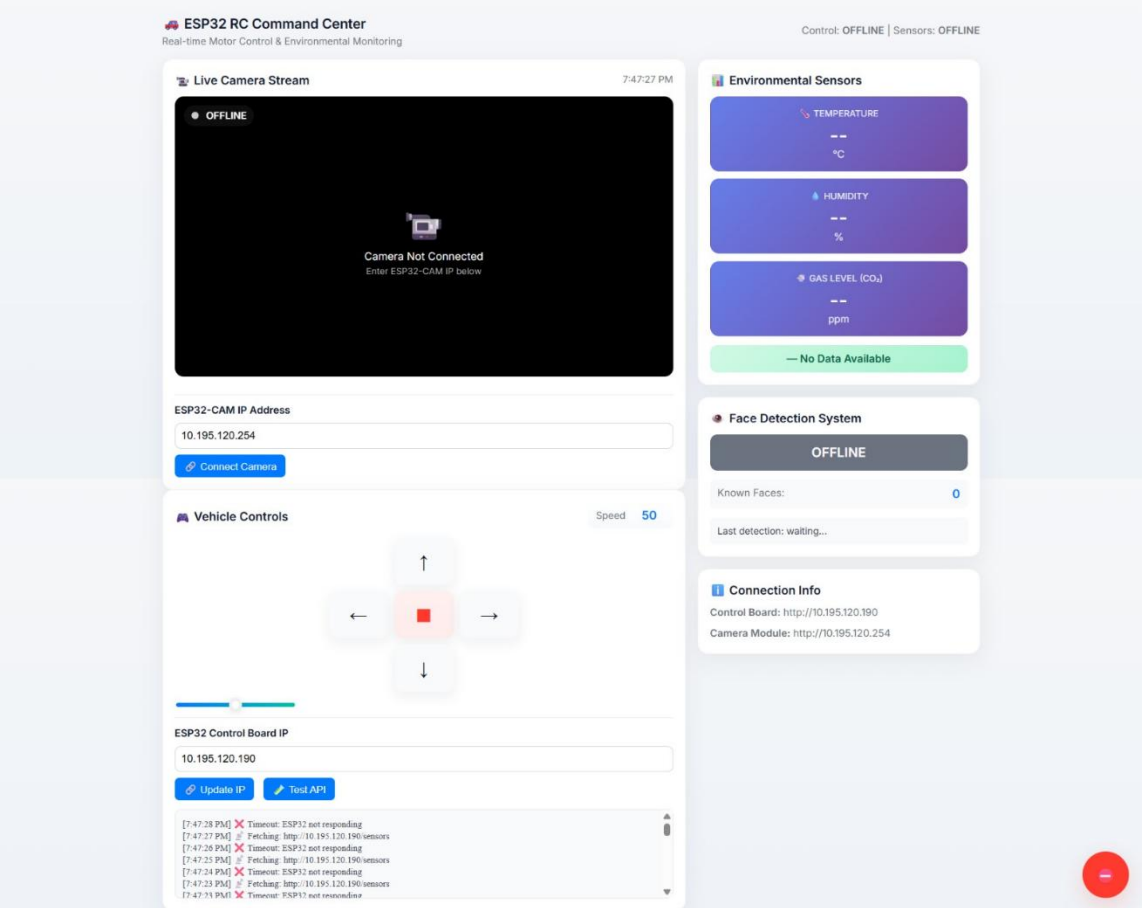


**fig 5.2.2 web interface ui/ux**

## 5.3 Challenges Faced While Implementing RoboSentinel

**1. Integration of Hardware and Software**
Ensuring smooth communication between the ESP32-CAM, sensors, and MATLAB was challenging. Compatibility issues, incorrect configurations, and synchronization delays required repeated debugging and calibration.

**2. Network and Connectivity Issues**
The system relied heavily on Wi-Fi, leading to unstable video streams, high latency, and occasional disconnections. Maintaining a consistent and reliable network connection was a major difficulty.

**3. Real-Time Processing Limitations**
Achieving real-time object and face detection in MATLAB demanded optimization. High-resolution frames caused slow processing, leading to delays in detection and alert generation.

**4. Sensor Calibration and Accuracy Problems**
PIR and ultrasonic sensors produced inconsistent results due to environmental factors such as heat, reflections, and noise. Proper calibration and positioning were essential to ensure accurate readings.

**5. Managing Power Stability for Components**
The ESP32-CAM and sensors required stable power. Voltage drops or fluctuations caused unexpected resets and malfunctioning, making power management a significant practical challenge.

# Chapter 6

## Conclusions

The RoboSentinel project successfully designed and implemented a smart real-time surveillance system capable of:

- Streaming live video through an ESP32-CAM.
- Processing video in MATLAB for enhancement and analysis.
- Detecting faces, people, objects, and motion.
- Collecting data from sensors such as gas, humidity and temperature sensor.
- Analysing and fusing video and sensor data to confirm intrusions.
- Automatically generating alerts (email, buzzer, LED indicators)

RoboSentinel proves to be a cost-effective, intelligent, and scalable solution for modern surveillance applications such as home security, industrial monitoring, and restricted-area protection.

## Future Scope

The future scope of the RoboSentinel system is highly promising, as it can be expanded into a fully automated and intelligent security solution. With advancements in AI and embedded systems, the model can be upgraded to incorporate deep-learning–based detection for improved accuracy in identifying people, objects, and suspicious activities. The integration of cloud platforms and mobile applications would enable remote monitoring, data storage, and real-time notifications from any location. The system can also be enhanced with night-vision cameras, GPS tracking, and servo-based motion control to actively follow intruders. Additionally, by using low-power edge devices like Raspberry Pi or NVIDIA Jetson, the entire framework can be made more efficient, scalable, and suitable for smart home, industrial, and public surveillance applications.

# References

[1] M. Mohana, R. Adhyapak, N. Yadav, and M. P. Madhumitha, "Design and development of web-controlled surveillance robot using computer vision," in *Proc. 5th Int. Conf. Trends Mater. Sci. Inventive Mater. (ICTMIM)*, Bengaluru, India, Apr. 2025, pp. 1418–1423, doi: 10.1109/ICTMIM65579.2025.10988169.

[2] S. Patil, V. Desai, and R. Kulkarni, "Hazardous gas detection and environmental monitoring in coal mines using IoT and GSM-based autonomous robot," *Int. J. Emerg. Trends Eng. Res.*, vol. 13, no. 4, pp. 220–228, Apr. 2025.

[3] O. Boufares *et al.*, "Moving object detection: A new method combining background subtraction, fuzzy entropy thresholding and differential evolution optimization," *Acta Mechanica et Automatica*, vol. 19, no. 1, pp. 106–116, Mar. 2025, doi: 10.2478/ama-2025-0013.

[4] J. He, Y. Chen, N. Wang, and Z. Zhang, "3D video object detection with learnable object-centric global optimization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 5106–5115.

[5] S. Liang and D. Baker, "Real-time background subtraction under varying lighting conditions," in *Proc. 2023 IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, doi: 10.1109/ICRA48891.2023.10160223.

# Appendix

## Matlab :

### i]Face Detection:

```
%%
================================================================================
==
%% ROBOSENTINEL  - QUICK START
%%
================================================================================
==

clear; clc; close all;

fprintf('\n');
fprintf('╔═══════════════════════════════════════════════════════════════════════
═══════════╗\n');
fprintf('║      ROBOSENTINEL v2.0 - FACE DETECTION SYSTEM            ║\n');
fprintf('║      Improved Recognition with HOG + LBP Features         ║\n');
fprintf('╚═══════════════════════════════════════════════════════════════════════
═══════════╝\n\n');

%%
================================================================================
==
%% STEP 1: CONFIGURATION
%%
================================================================================
==
fprintf('STEP 1: Configuration\n');
fprintf('──────────────────────────────────────\n');

default_cam_ip = '192.168.1.100';

fprintf('Enter ESP32-CAM IP address\n');
fprintf('(Press Enter for default: %s): ', default_cam_ip);
user_cam_ip = input('', 's');

if isempty(user_cam_ip)
   ESP32_CAM_IP = default_cam_ip;
else
   ESP32_CAM_IP = user_cam_ip;
end
```

```
fprintf('\n✓ Using ESP32-CAM IP: %s\n\n', ESP32_CAM_IP);

%%
================================================================================
%% STEP 2: TEST CONNECTION
%%
================================================================================
fprintf('STEP 2: Testing Connection\n');
fprintf('──────────────────────────────────────────────────\n');

try
    testURL = ['http://' ESP32_CAM_IP '/capture'];
    fprintf('Testing: %s\n', testURL);
    testImg = webread(testURL, weboptions('Timeout', 8));
    fprintf('✓ Connection successful!\n');
    fprintf('✓ Image size: %dx%d\n\n', size(testImg, 2), size(testImg, 1));
catch ME
    fprintf('✗ Connection FAILED!\n');
    fprintf('  Error: %s\n\n', ME.message);
    fprintf('Troubleshooting:\n');
    fprintf('  1. Check ESP32-CAM power (needs 5V, 2A+)\n');
    fprintf('  2. Verify IP address matches ESP32-CAM\n');
    fprintf('  3. Ensure both on same WiFi network\n');
    fprintf('  4. Try ping: ping %s\n', ESP32_CAM_IP);
    fprintf('  5. Check ESP32 serial monitor for errors\n\n');
    return;
end

%%
================================================================================
%% STEP 3: CREATE SYSTEM
%%
================================================================================
fprintf('STEP 3: Initializing System\n');
fprintf('──────────────────────────────────────────────────\n');

try
    bridge = RoboSentinelWebBridge(ESP32_CAM_IP, 'ProcessInterval', 0.4);
    fprintf('✓ RoboSentinel v2.0 initialized!\n');
    fprintf('✓ Detection features: HOG + LBP + Pixel\n\n');
catch ME
    fprintf('✗ Initialization failed: %s\n', ME.message);
    fprintf('  Make sure RoboSentinel.m and RoboSentinelWebBridge.m exist\n\n');
    return;
end
```

```
%%
==========================================================================
==
%% STEP 4: ADD KNOWN FACES (CRITICAL FOR RECOGNITION)
%%
==========================================================================
==
fprintf('╔═══════════════════════════════════════════════════════════════
═══════════╗\n');
fprintf('║  STEP 4: Add Known Faces (IMPORTANT!)                    ║\n');
fprintf('╚═══════════════════════════════════════════════════════════════
═══════════╝\n\n');

fprintf('⚠️  WARNING: Without known faces, everyone will be "Unknown"!\n\n');
fprintf('Options:\n');
fprintf('  1 = Add from image files (recommended)\n');
fprintf('  2 = Capture from ESP32-CAM live\n');
fprintf('  3 = Skip (test detection only)\n\n');

choice = input('Select option (1/2/3): ');

switch choice
    case 1
        %% Add from files
        fprintf('\n--- ADD FROM IMAGE FILES ---\n');
        fprintf('Tips for best results:\n');
        fprintf('  • Use well-lit photos\n');
        fprintf('  • Face should be clearly visible\n');
        fprintf('  • Frontal view works best\n');
        fprintf('  • Avoid sunglasses/hats\n\n');

        numPeople = input('How many people to add? ');

        for i = 1:numPeople
            fprintf('\n┌─ Person %d of %d ─────────────────────────┐\n', i, numPeople);

            while true
                imagePath = input('│  Image file path: ', 's');
                if exist(imagePath, 'file')
                    break;
                else
                    fprintf('│  ✗ File not found. Try again.\n');
                end
            end

            personName = input('│  Person name: ', 's');

            try
                % Show image preview
```

```
            img = imread(imagePath);
            figure('Name', 'Preview');
            imshow(img);
            title(sprintf('Adding: %s', personName));
            pause(0.5);
            close(gcf);

            bridge.addKnownFace(imagePath, personName);
            fprintf('  └─ ✓ Added %s successfully!\n', personName);
        catch ME
            fprintf('  └─ ✗ Error: %s\n', ME.message);
        end
    end

case 2
    %% Capture from camera
    fprintf('\n--- CAPTURE FROM CAMERA ---\n');

    fprintf('Testing camera...\n');
    try
        testImg = bridge.sentinel.captureFrame();
        fprintf('✓ Camera working!\n\n');

        % Show test frame
        figure('Name', 'Camera Test');
        imshow(testImg);
        title('Camera Feed OK');
        pause(1);
        close(gcf);

    catch ME
        fprintf('✗ Camera error: %s\n', ME.message);
        return;
    end

    numPeople = input('How many people to add? ');

    fprintf('\nCapture Tips:\n');
    fprintf('  • Position face centered\n');
    fprintf('  • Ensure good lighting\n');
    fprintf('  • Look at camera\n');
    fprintf('  • Stay still when capturing\n\n');

    for i = 1:numPeople
        fprintf('\n ┌─ Person %d of %d ──────────────────────────┐ \n', i, numPeople);
        personName = input(' │  Person name: ', 's');

        fprintf(' │  Position %s in front of camera\n', personName);
        fprintf(' │  Capturing in: ');
```

```
            for j = 3:-1:1
                fprintf('%d... ', j);
                pause(1);
            end
            fprintf('NOW!\n');

            % Capture
            img = bridge.sentinel.captureFrame();

            % Show captured image
            fig = figure('Name', sprintf('Captured: %s', personName));
            imshow(img);
            title(sprintf('Is this good for %s?', personName));

            useThis = input('  Use this image? (y/n): ', 's');
            close(fig);

            if strcmpi(useThis, 'y')
                filename = sprintf('known_%s_%d.jpg', strrep(personName, ' ', '_'), i);
                imwrite(img, filename);

                bridge.addKnownFace(filename, personName);
                fprintf('  ✓ Added %s!\n', personName);
            else
                fprintf('  Skipped. Retrying...\n');
                i = i - 1;
            end
        end

    case 3
        fprintf('\n⚠️  SKIPPING face database!\n');
        fprintf('   All faces will be detected as UNKNOWN.\n');
        fprintf('   This is only useful for testing detection.\n\n');

    otherwise
        fprintf('Invalid option. Skipping.\n\n');
end

fprintf('\n');
fprintf('═══════════════════════════════════════════════════════════════
═══════════\n');
fprintf('Known Faces Database: %d people\n', length(bridge.sentinel.knownNames));

if ~isempty(bridge.sentinel.knownNames)
    fprintf('Registered:\n');
    for i = 1:length(bridge.sentinel.knownNames)
        fprintf('  %d. %s\n', i, bridge.sentinel.knownNames{i});
    end
end
```

```
fprintf('═══════════════════════════════════════════════════════════
═══════════\n\n');

%%
═══════════════════════════════════════════════════════════════════
==
%% STEP 5: WEB INTERFACE INSTRUCTIONS
%%
═══════════════════════════════════════════════════════════════════
==
fprintf('STEP 5: Web Dashboard Setup\n');
fprintf('──────────────────────────────────────────────\n');
fprintf('To view detection results on your HTML dashboard:\n\n');
fprintf(' 1  Status data will be written to: sentinel_data/status.json\n');
fprintf(' 2  Your HTML file should read this JSON file\n');
fprintf(' 3  Start a local web server:\n');
fprintf('    - Open terminal in this folder\n');
fprintf('    - Run: python -m http.server 3000\n');
fprintf('    - Or: python3 -m http.server 3000\n');
fprintf(' 4  Open browser: http://localhost:3000/your_page.html\n\n');

fprintf('JSON data structure:\n');
fprintf('{\n');
fprintf('  "currentKnownFaces": 2,\n');
fprintf('  "currentUnknownFaces": 0,\n');
fprintf('  "knownFaceNames": ["Alice", "Bob"],\n');
fprintf('  "alert": false,\n');
fprintf('  "message": "✓ Recognized: Alice, Bob",\n');
fprintf('  "history": [...]\n');
fprintf('}\n\n');

%%
═══════════════════════════════════════════════════════════════════
==
%% STEP 6: START DETECTION
%%
═══════════════════════════════════════════════════════════════════
==
fprintf('STEP 6: Start Face Detection\n');
fprintf('──────────────────────────────────────────────\n');

startNow = input('Start detection system now? (y/n): ', 's');

if strcmpi(startNow, 'y')
    fprintf('\n');

fprintf('╔═══════════════════════════════════════════════════════════
═══════════╗\n');
    fprintf('║            STARTING DETECTION SYSTEM...                  ║\n');
```

```
fprintf('╚═══════════════════════════════════════════════════════════════════════════════╝\n\n');

    fprintf('System Info:\n');
    fprintf('  Camera: %s\n', ESP32_CAM_IP);
    fprintf('  Known: %d people\n', length(bridge.sentinel.knownNames));
    fprintf('  Data: sentinel_data/status.json\n');
    fprintf('  Features: HOG + LBP + Pixel matching\n\n');

    fprintf('Controls:\n');
    fprintf('  • Video window shows live detection\n');
    fprintf('  • Green box = Recognized person (with name)\n');
    fprintf('  • Red box = Unknown person (alert triggered)\n');
    fprintf('  • Close window or Ctrl+C to stop\n\n');

    fprintf('Starting in 3 seconds...\n');
    pause(3);

    try
        bridge.start(true);
    catch ME
        fprintf('\n✗ Detection error: %s\n', ME.message);
    end

else
    fprintf('\n System ready but not started.\n');
    fprintf('   To start manually:\n');
    fprintf('   >> bridge.start(true);\n\n');
end

%%
%% HELPFUL COMMANDS
%%
fprintf('\n');
fprintf('╔═══════════════════════════════════════════════════════════════════════════════╗\n');
fprintf('║                    USEFUL COMMANDS                                            ║\n');
fprintf('╚═══════════════════════════════════════════════════════════════════════════════╝\n');
fprintf('bridge.start(true)                - Start with video\n');
fprintf('bridge.stop()                     - Stop detection\n');
fprintf('bridge.addKnownFace(path, name)      - Add more faces\n');
fprintf('bridge.sentinel.captureFrame()       - Test camera\n');
fprintf('bridge.sentinel.minFaceQuality=0.2   - Lower quality threshold\n\n');
```

```
fprintf('Detection not working? Try:\n');
fprintf('1. Add more photos of each person (different angles)\n');
fprintf('2. Use better lit photos\n');
fprintf('3. Check camera focus\n');
fprintf('4. Lower threshold: bridge.sentinel.minFaceQuality = 0.2\n\n');

fprintf('✅ Setup complete!\n\n');
```

**ii] ESP32 :**

```
/*
 * ESP32 Dev Module - Complete Smart Car System
 * Motors + Sensors + Web Interface with Camera Stream
 */

#include <Arduino.h>
#include <WiFi.h>
#include <WebServer.h>
#include <math.h>
#include "DHT.h"

// ========== CONFIGURE IPs HERE ==========
const char* ssid = "Redmi note 13 pro 5g ";
const char* password = "prajwal123";
const char* ESP32_CAM_IP = "192.168.178.254";

// ========== Motor Pins ==========
const int IN1 = 25;
const int IN2 = 26;
const int ENA = 33;
const int IN3 = 27;
const int IN4 = 14;
const int ENB = 32;

// ========== PWM Settings ==========
const int pwmFreq = 1000;
const int pwmResolution = 8;
int currentSpeed = 128;

// ========== DHT11 Sensor ==========
#define DHTPIN 5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// ========== MQ135 Gas Sensor ==========
#define RL_MQ135 10.0
#define Vc       5.0
#define ADC_REF  5.0
```

```
#define ADC_MAX  4095.0
#define Ro       76.5
#define MQ135_A 116.6020682
#define MQ135_B -2.769034857

int pinMQ135 = 35;

// ========== Sensor Data Variables ==========
float temperature = 0.0;
float humidity = 0.0;
float gasLevel = 0.0;
unsigned long lastSensorRead = 0;
const unsigned long sensorInterval = 2000;

WebServer server(80);

// ========== Function Prototypes ==========
float readVoltage(int pin);
float calcRs(float Vout, float RL);
float getPPM(float ratio, float a, float b);
void readSensors();

void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("\n========================================");
  Serial.println("ESP32 Dev - Smart Car System");
  Serial.println("========================================");

  // ========== Motor Pin Setup ==========
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  ledcAttach(ENA, pwmFreq, pwmResolution);
  ledcAttach(ENB, pwmFreq, pwmResolution);

  stopMotors();
  Serial.println("✅ Motors initialized");

  // ========== Initialize Sensors ==========
  dht.begin();
  Serial.println("✅ DHT11 sensor initialized");
  Serial.println("✅ MQ135 sensor initialized");
  Serial.printf("Using Ro = %.2f kΩ\n", Ro);

  // ========== Connect to WiFi ==========
  WiFi.begin(ssid, password);
```

```
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\n✅ WiFi connected!");
  Serial.println("==========================================");
  Serial.print("📡 ESP32 Dev IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("📷 ESP32-CAM IP: ");
  Serial.println(ESP32_CAM_IP);
  Serial.println("==========================================");
  Serial.println("🌐 Open this URL in browser:");
  Serial.print("   http://");
  Serial.println(WiFi.localIP());
  Serial.println("==========================================\n");

  // ========== Web Server Routes ==========
  server.on("/", handleRoot);
  server.on("/control", HTTP_OPTIONS, handleCORS);
  server.on("/control", HTTP_GET, handleControl);
  server.on("/sensors", HTTP_OPTIONS, handleCORS);
  server.on("/sensors", HTTP_GET, handleSensors);

  // Legacy endpoints
  server.on("/forward", []{ moveForward(); server.send(200, "text/plain", "Forward"); });
  server.on("/backward", []{ moveBackward(); server.send(200, "text/plain", "Backward");
});
  server.on("/left", []{ turnLeft(); server.send(200, "text/plain", "Left"); });
  server.on("/right", []{ turnRight(); server.send(200, "text/plain", "Right"); });
  server.on("/stop", []{ stopMotors(); server.send(200, "text/plain", "Stop"); });

  server.begin();
  Serial.println("✅ HTTP server started");
  Serial.println("==========================================");
  Serial.println("✅ SYSTEM READY!");
  Serial.println("==========================================\n");
}

void loop() {
  server.handleClient();

  if (millis() - lastSensorRead >= sensorInterval) {
    lastSensorRead = millis();
    readSensors();
  }
}

// ========== CORS Handler ==========
```

```
void handleCORS() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "Content-Type");
  server.send(204);
}

// ========== Sensor Reading Function ==========
void readSensors() {
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();

  float Vout = readVoltage(pinMQ135);
  float Rs = calcRs(Vout, RL_MQ135);

  if (Rs > 0) {
    float ratio = Rs / Ro;
    gasLevel = getPPM(ratio, MQ135_A, MQ135_B);
  } else {
    gasLevel = -1;
  }

  Serial.println("========== Sensor Readings ==========");

  if (!isnan(temperature) && !isnan(humidity)) {
    Serial.printf("Temperature: %.1f°C | Humidity: %.1f%%\n", temperature, humidity);
  } else {
    Serial.println("⚠ DHT11 read failed!");
  }

  if (gasLevel > 0) {
    Serial.printf("Gas Level (CO₂ approx): %.0f ppm\n", gasLevel);

    if (gasLevel > 3500) {
      Serial.println("⚠ ALERT: Poor Air Quality!");
    } else if (gasLevel > 1800) {
      Serial.println(" 🟠 WARNING: Moderate Air Quality");
    } else {
      Serial.println(" ✅ Air Quality is Good");
    }
  } else {
    Serial.println("⚠ MQ135 read failed!");
  }

  Serial.println("=====================================\n");
}

// ========== MQ135 Helper Functions ==========
float readVoltage(int pin) {
```

```
  return (analogRead(pin) * ADC_REF / ADC_MAX);
}

float calcRs(float Vout, float RL) {
  if (Vout >= Vc || Vout <= 0) {
    return -1;
  }
  return RL * (Vc - Vout) / Vout;
}

float getPPM(float ratio, float a, float b) {
  if (ratio <= 0) return -1;
  return a * pow(ratio, b);
}

// ========== Web Server Handlers ==========
void handleControl() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "Content-Type");

  if (server.hasArg("speed")) {
    int speedPercent = server.arg("speed").toInt();
    currentSpeed = map(constrain(speedPercent, 0, 100), 0, 100, 0, 255);
  }

  if (server.hasArg("move")) {
    String move = server.arg("move");

    if (move == "forward") moveForward();
    else if (move == "backward") moveBackward();
    else if (move == "left") turnLeft();
    else if (move == "right") turnRight();
    else if (move == "stop") stopMotors();
  }

  server.send(200, "application/json", "{\"status\":\"ok\",\"speed\":" + String(currentSpeed) +
"}");
}

void handleSensors() {
  // CRITICAL: Send CORS headers FIRST
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "Content-Type");

  // Read fresh sensor data
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
```

```
  // Read MQ135
  float Vout = readVoltage(pinMQ135);
  float Rs = calcRs(Vout, RL_MQ135);
  float gas = 0;

  if (Rs > 0) {
    float ratio = Rs / Ro;
    gas = getPPM(ratio, MQ135_A, MQ135_B);
  }

  // Validate values
  if (isnan(temp) || temp < -50 || temp > 100) temp = 0.0;
  if (isnan(hum) || hum < 0 || hum > 100) hum = 0.0;
  if (gas < 0 || isnan(gas)) gas = 0.0;

  // Air quality status
  String airQuality = "Good";
  if (gas > 3500) {
    airQuality = "Poor";
  } else if (gas > 1800) {
    airQuality = "Moderate";
  }

  // Build compact JSON
  String json = "{";
  json += "\"temperature\":" + String(temp, 1) + ",";
  json += "\"humidity\":" + String(hum, 1) + ",";
  json += "\"gas\":" + String((int)gas) + ",";
  json += "\"airQuality\":\"" + airQuality + "\"";
  json += "}";

  // Debug print
  Serial.println("=== Sending Sensor Data ===");
  Serial.println(json);
  Serial.println("============================");

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

void handleRoot() {
  String html = "<!DOCTYPE html><html><head><meta name='viewport'
content='width=device-width,initial-scale=1'>";
  html += "<title>ESP32 Smart Car</title>";
  html += "<style>";
  html += "* { margin: 0; padding: 0; box-sizing: border-box; }";
  html += "body { font-family: 'Segoe UI', Arial, sans-serif; background: linear-
gradient(135deg, #667eea 0%, #764ba2 100%); min-height: 100vh; padding: 20px; }";
  html += ".container { max-width: 900px; margin: 0 auto; }";
  html += ".card { background: white; border-radius: 20px; padding: 25px; margin-bottom:
20px; box-shadow: 0 10px 40px rgba(0,0,0,0.2); }";
```

```
  html += "h1 { color: #333; font-size: 28px; margin-bottom: 10px; text-align: center; }";
  html += ".subtitle { color: #666; text-align: center; margin-bottom: 20px; font-size: 14px;
}";
  html += ".video-container { background: #000; border-radius: 15px; overflow: hidden;
margin-bottom: 20px; position: relative; min-height: 300px; display: flex; align-items: center;
justify-content: center; }";
  html += ".video-container img { width: 100%; height: auto; display: block; }";
  html += ".sensors { display: grid; grid-template-columns: repeat(auto-fit, minmax(200px,
1fr)); gap: 15px; margin-bottom: 20px; }";
  html += ".sensor { background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
color: white; padding: 20px; border-radius: 15px; text-align: center; }";
  html += ".sensor-value { font-size: 32px; font-weight: bold; }";
  html += ".btn-grid { display: grid; grid-template-columns: repeat(3, 1fr); gap: 10px; max-
width: 400px; margin: 0 auto; }";
  html += ".ctrl-btn { background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
border: none; color: white; font-size: 32px; padding: 20px; border-radius: 15px; cursor:
pointer; }";
  html += "</style>";

  html += "<script>";
  html += "setInterval(() => {";
  html += "  fetch('/sensors').then(r => r.json()).then(d => {";
  html += "    document.getElementById('temp').textContent = d.temperature;";
  html += "    document.getElementById('hum').textContent = d.humidity;";
  html += "    document.getElementById('gas').textContent = d.gas;";
  html += "  }).catch(e => console.error('Sensor fetch error:', e));";
  html += "}, 2000);";
  html += "function sendCmd(cmd) { fetch(/control?move=${cmd}&speed=50).catch(e =>
console.error(e)); }";
  html += "</script></head><body>";

  html += "<div class='container'>";
  html += "<div class='card'><h1>🚗 ESP32 Smart Car</h1>";
  html += "<div class='video-container'><img src='http://" + String(ESP32_CAM_IP) +
":81/stream'></div></div>";

  html += "<div class='card'><h2>📊 Sensors</h2><div class='sensors'>";
  html += "<div class='sensor'><div>🌡️ Temperature</div><div class='sensor-value'
id='temp'>--</div></div>";
  html += "<div class='sensor'><div>💧 Humidity</div><div class='sensor-value'
id='hum'>--</div></div>";
  html += "<div class='sensor'><div>💨 Gas</div><div class='sensor-value' id='gas'>--
</div></div>";
  html += "</div></div>";

  html += "<div class='card'><h2>🎮 Controls</h2><div class='btn-grid'>";
  html += "<div></div><button class='ctrl-btn' onmousedown=\"sendCmd('forward')\"
onmouseup=\"sendCmd('stop')\">↑</button><div></div>";
```

```
  html += "<button class='ctrl-btn' onmousedown=\"sendCmd('left')\"
onmouseup=\"sendCmd('stop')\">←</button>";
  html += "<button class='ctrl-btn' onclick=\"sendCmd('stop')\">■</button>";
  html += "<button class='ctrl-btn' onmousedown=\"sendCmd('right')\"
onmouseup=\"sendCmd('stop')\">→</button>";
  html += "<div></div><button class='ctrl-btn' onmousedown=\"sendCmd('backward')\"
onmouseup=\"sendCmd('stop')\">↓</button><div></div>";
  html += "</div></div></div>";

  html += "</body></html>";

  server.send(200, "text/html", html);
}

// ========== Motor Control Functions ==========
void moveForward() {
  Serial.println("Moving Forward");
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  ledcWrite(ENA, currentSpeed);
  ledcWrite(ENB, currentSpeed);
}

void moveBackward() {
  Serial.println("Moving Backward");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  ledcWrite(ENA, currentSpeed);
  ledcWrite(ENB, currentSpeed);
}

void turnLeft() {
  Serial.println("Turning Left");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  ledcWrite(ENA, 0);
  ledcWrite(ENB, currentSpeed);
}

void turnRight() {
  Serial.println("Turning Right");
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
```

```
  digitalWrite(IN4, LOW);
  ledcWrite(ENA, currentSpeed);
  ledcWrite(ENB, 0);
}

void stopMotors() {
  Serial.println("Stopping");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  ledcWrite(ENA, 0);
  ledcWrite(ENB, 0);
}
```