© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

Smart EV Safety and Alert System

I. REVATHI[1], T. NAGA SAI SARAN[2], T. SRI LAKSHMI[3], R. LAKSHMI SAILAJA[4], G. SUNDAR RAJ[5]

[1]*Assistant Professor, Department of EEE & VVIT, India*
[2, 3, 4, 5]*UG Students, Department of EEE & VVIT, India*

Abstract- ***This paper presents the design and implementation of a Smart EV Safety and Alert System — a working prototype built around a dual-microcontroller architecture using an ESP32-CAM and Arduino Uno. The system demonstrates three key capabilities validated on a physical prototype: (1) AI-based object detection using YOLOv8n on a connected laptop that automatically stops the car when a person is detected and slows it when a bus is detected, achieving 180–220 ms response latency; (2) crash and flip detection via an MPU6050 IMU that halts all motors immediately on impact or rollover; and (3) a web-based manual control dashboard that allows full directional control and emergency stop from any browser on the same Wi-Fi network. The motors are demonstrated running and stopping automatically in response to both sensor triggers and web commands, confirming real-time hardware responsiveness. Battery temperature monitoring via DS18B20 sensors and spoken voice alerts via DFPlayer Mini provide additional safety layers. Lane detection using OpenCV is identified as a near-term upgrade. Testing on the working prototype confirmed zero false positives during normal operation. The system is built entirely from open-source tools and components costing under ₹4,000, demonstrating a cost-effective architecture for intelligent vehicle safety systems.***

Keywords: ESP32-CAM; YOLOv8n; Object Detection; Crash Detection; MPU6050; Smart EV; Web Control; Lane Detection; OpenCV; Arduino Uno

I. INTRODUCTION

Remote-controlled (RC) vehicles have long been popular consumer electronics products, yet the intelligence embedded in even the most advanced commercial models remains remarkably limited. A typical RC car does exactly what a joystick commands — it drives forward, turns, and stops — but it carries no awareness of its environment, no knowledge of its own battery health, and no mechanism by which a supervising adult can monitor or constrain its behaviour from a distance. These omissions become significant safety concerns as soon as the vehicle operates in a space shared with people and obstacles.

The convergence of low-cost Wi-Fi microcontrollers, compact inertial sensors, digital temperature probes, and lightweight neural network inference frameworks has made it feasible to address all of these gaps in a single, affordable, student-built system. This paper describes a smart RC car that demonstrates exactly this potential. By integrating computer vision, crash detection, thermal monitoring, voice alert feedback, and a web-based web manual control dashboard — all on a 4WD chassis with the dashboard served directly from the ESP32-CAM's own HTTP server — the project establishes a practical reference architecture for safe, intelligent small vehicles using entirely free and open-source software.

The core engineering challenge is not any single sensor or algorithm in isolation, but the integration of multiple real-time safety systems onto a dual-microcontroller platform while maintaining reliable motor responsiveness and sub-250 ms reaction times across all safety triggers. A secondary challenge is making remote supervision genuinely effective: speed limiting enforced in firmware rather than software, emergency stop latency under 150 ms, and a live dashboard that requires no app installation or external internet service.

The car operates in three distinct modes. In autonomous mode, the ESP32-CAM streams video over Wi-Fi to a laptop running YOLOv8n through OpenCV; detected persons cause the car to stop with a voice alert, while detected buses or trucks trigger an automatic speed reduction with a separate warning. In manual mode, the driver uses the web dashboard to send directional commands. In web manual control mode, the operator sends directional commands, activates emergency stop, and monitors all live sensor readings from any browser on the same Wi-Fi network. GPS tracking is planned for a future

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

hardware revision; the geo-fence firmware logic is already implemented and ready for activation once the module is fitted. Cloud-based remote monitoring via Supabase is also identified as a future upgrade.

II. SYSTEM DESIGN AND DESCRIPTION

The proposed system is organised around a dual-microcontroller architecture that distributes responsibilities according to the strengths of each processor. The ESP32-CAM (AI Thinker model) sits at the top of the hierarchy: it manages Wi-Fi connectivity, serves a live MJPEG camera stream over HTTP, hosts the web manual control dashboard directly on its built-in HTTP server, and maintains UART communication with the Arduino Uno. The Arduino Uno handles all time-critical hardware tasks — DC motor PWM control through the L293D driver, crash and flip detection via the MPU6050 IMU, battery and motor temperature sensing via DS18B20 probes, and spoken voice alerts through the DFPlayer Mini module. The two boards communicate over a serial UART link: the ESP32 forwards commands received from the web dashboard to the Arduino, and the Arduino returns sensor readings to the ESP32 for display on the dashboard.

Data Flow: Sensor data flows outward — Arduino reads sensors and sends values to the ESP32 over serial; the ESP32 includes this data in the web dashboard response served to the parent's browser. Commands flow inward — the parent presses a button on the browser dashboard, the HTTP request reaches the ESP32, which relays the motor command over serial to the Arduino for immediate hardware execution. All safety-critical actions such as motor stop on crash and speed cap enforcement execute entirely in Arduino firmware, independent of any network connection.

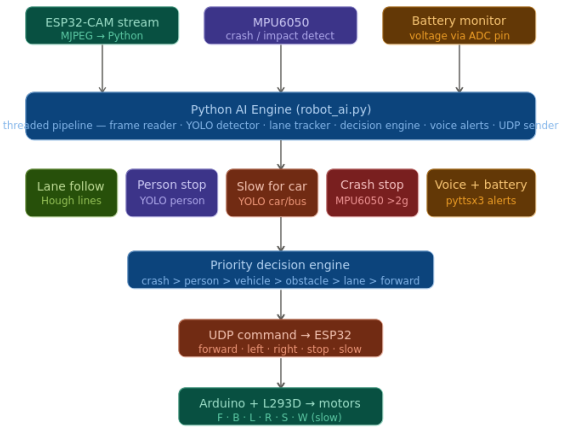

Fig. 1: Block Diagram of the Proposed Smart RC Car System (ESP32-CAM ↔ Arduino Uno ↔ Sensors + Motors + Web Dashboard)

A. Object Detection Pipeline

The object detection subsystem uses a laptop-side Python process, which is the key architectural decision that makes real-time detection feasible on ESP32-class hardware. The ESP32-CAM serves a continuous MJPEG stream over HTTP on the local Wi-Fi network. The laptop application connects to this stream, decodes each JPEG frame using OpenCV, and passes it to the YOLOv8n model — the nano variant of Ultralytics YOLOv8, selected for its low inference latency of approximately 6 ms per frame on a mid-range CPU. Detection results are filtered for three target classes: person (class 0) triggers a STOP command with voice alert track 001; bus (class 5) and truck (class 7) trigger a SLOW command with voice alert track 008. All other COCO classes are ignored. A 1-second debounce timer prevents command flooding when an object remains continuously in frame.

B. Crash and Flip Detection

The MPU6050 6-axis IMU is polled by the Arduino every 100 ms. Two detection algorithms run in parallel. For flip detection, the firmware computes the tilt angle from all three accelerometer axes; if the angle exceeds 45° continuously for 500 ms, a flip event is declared. For crash detection, the total acceleration magnitude is computed as the root-sum-square of all three axes; any instantaneous reading above 3g triggers an impact event. On either detection, the Arduino immediately halts all motors, plays voice alert track 003, activates the buzzer for 3 seconds, and

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

sends an SOS event code to the ESP32, which displays it on the web dashboard with a timestamp.

C. Thermal Safety System

Two DS18B20 digital temperature sensors share a single 1-Wire bus: one mounted on the LiPo battery surface, one attached to the L293D motor driver heatsink. The Arduino queries both every 500 ms. If battery temperature exceeds 45°C, motor PWM is reduced to 50%, voice alert track 001 plays, and the buzzer activates for 2 seconds. If motor driver temperature exceeds 60°C, speed drops to 30% and track 002 plays. A low battery alert (below 15% estimated from the voltage divider) plays track 007 without speed reduction. All alert conditions are also reflected on the web dashboard in real time.

D. Web-Based Manual Control

The web-based manual control interface is a custom dashboard built in HTML, CSS, and JavaScript, served directly by the ESP32-CAM's built-in HTTP web server over the local Wi-Fi network — no external cloud service or internet connection is required. The dashboard page is stored in the ESP32's flash memory and pushed to the browser on request. Sensor readings are updated through lightweight HTTP polling from the browser to the ESP32. The dashboard provides full manual control of the car: directional commands (forward, backward, left, right), a speed control slider, and an emergency stop button that halts all motors within 150 ms. All safety-critical stops — from object detection, crash detection, or temperature alerts — also execute through this command path. Live sensor readings including speed, battery percentage, battery temperature, motor temperature, and a safety alert log are displayed in real time. In a future revision, Supabase cloud integration is planned to extend monitoring beyond the local network.

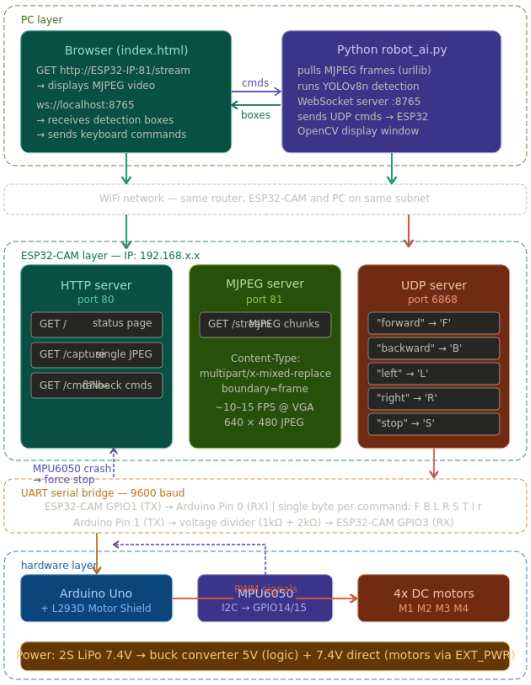

Fig. 2: Web Server Architecture — ESP32-CAM HTTP Server Serving Manual Control Dashboard and MJPEG Stream

III. HARDWARE COMPONENTS AND DESIGN

This section describes each hardware component, its role in the system, and the rationale for its selection.

A. ESP32-CAM (AI Thinker Model)

The ESP32-CAM integrates the dual-core ESP32-S processor at up to 240 MHz, 4 MB Flash, 4 MB PSRAM, Wi-Fi 802.11 b/g/n, and an OV2640 2-megapixel camera in a 40 mm × 27 mm footprint. Core 1 runs the MJPEG HTTP stream server and the web dashboard HTTP server simultaneously. Core 0 runs the main application loop — receiving sensor data from the Arduino over UART and making it available to the dashboard. The PSRAM enables large JPEG frame buffers that would exceed the ESP32's internal SRAM, making reliable video streaming possible.

B. Arduino Uno

The Arduino Uno (ATmega328P at 16 MHz) serves as the real-time hardware controller. Its 14 digital I/O pins (six PWM-capable) and six analog inputs provide ample resources for all motor control lines, the DS18B20 1-Wire bus, the MPU6050 I2C interface, the DFPlayer Mini SoftwareSerial channel, and the active

buzzer. The ESP32-CAM loses most of its usable GPIO to the OV2640 camera interface when the camera is active; the Arduino Uno compensates by taking full ownership of all sensor and actuator interfacing.

C. L293D Dual H-Bridge Motor Driver

The L293D provides two independent H-bridge circuits, each capable of bidirectional DC motor control at up to 600 mA continuous, with built-in flyback protection diodes on all output pins. On the standard dual-IC module used in this project, all four TT gear motors of the 4WD chassis are controlled. Speed is regulated by PWM on the EN1,2 and EN3,4 enable pins; direction is set by the IN1–IN4 logic inputs. Motor speed is controlled by capping the PWM value written to the enable pins in Arduino firmware — a hardware-level constraint enforced at the firmware level.

D. MPU6050 Inertial Measurement Unit

The MPU6050 combines a 3-axis accelerometer and a 3-axis gyroscope in a single I2C-addressable package communicating with the Arduino over pins A4 and A5. The accelerometer range is configured to ±8g. Two complementary detection algorithms run simultaneously: magnitude-threshold crash detection (total acceleration exceeding 3g) and angle-threshold flip detection (tilt exceeding 45° for 500 ms). This combination eliminates false positives from normal sharp cornering while reliably catching genuine crashes and rollovers in testing.

E. DS18B20 Temperature Sensors

Two DS18B20 waterproof probe sensors share a single 1-Wire data bus on Arduino pin 7, with a 4.7 kΩ pull-up resistor to 5V. Each sensor carries a factory-unique 64-bit serial number, allowing independent addressing on the same wire using the DallasTemperature library. One probe is mounted on the LiPo battery surface; the other is clamped to the L293D heatsink. Accuracy was validated against a calibrated reference thermometer, showing a maximum deviation of ±0.6°C across the 20°C to 55°C range.

F. DFPlayer Mini Voice Alert Module

The DFPlayer Mini is a compact MP3 playback IC with an onboard DAC and amplifier capable of directly driving an 8Ω, 0.5W speaker. The Arduino communicates via SoftwareSerial on pins 11 and 12, issuing play commands by track number. Human-recorded MP3 voice messages are stored on a 4 GB FAT32 microSD card. This approach provides clear, specific, human-intelligible warnings for every safety condition — battery overheating, motor overheating, crash detection, object detection, emergency stop, speed limit update, and low battery — rather than ambiguous buzzer tones.

Table 1: DFPlayer Mini Voice Alert Track Listing

Track File	Voice Alert Message
0001.mp3	"Warning! Battery temperature is too high. Please stop the car immediately."
0002.mp3	"Warning! Motor driver is overheating. Reducing speed automatically."
0003.mp3	"Crash detected. Sending alert to dashboard."
0004.mp3	"Geo-fence boundary crossed. Stopping the car." (future — GPS pending)
0005.mp3	"Emergency stop activated by parent."
0006.mp3	"Speed limit updated by parent."
0007.mp3	"Battery is low. Please recharge soon."
0008.mp3	"Warning! Large vehicle detected. Slowing down."

G. Power Supply Design

A 7.4V 2S LiPo battery rated at 2200 mAh powers the entire system through a 2S BMS board providing hardware over-current and over-discharge protection. Motor power (7.4V) feeds directly from the BMS output to the L293D VCC2 pin. All logic components — ESP32-CAM, Arduino Uno, DFPlayer Mini, buzzer, and DS18B20 sensors — run from a 5V rail

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

generated by an LM2596 adjustable buck converter. Battery voltage is sampled through a 10 kΩ/30 kΩ resistor divider on Arduino analog pin A0, mapping the 6.0–8.4V range to a 0–2.1V ADC input and converting it to a 0–100% battery percentage.

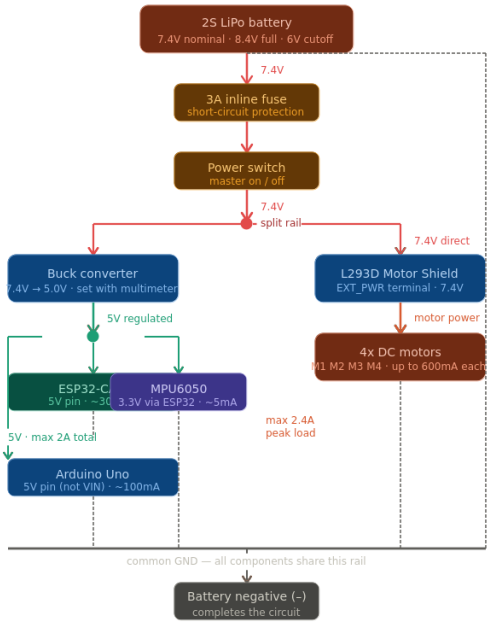

Fig. 3: Power Distribution Flowchart — 7.4V LiPo BMS to Motor and Logic Rails

Table 2: Complete Hardware Components List

#	Component	Model / Spec	Role
1	Wi-Fi + Camera + Web Server	ESP32-CAM AI Thinker	Streaming, dashboard, commands
2	Motor Controller MCU	Arduino Uno (ATmega328P)	Motors, sensors, alerts
3	Motor Driver IC	L293D dual H-bridge module	4WD speed and direction
4	DC Gear Motors	TT Motor 3V–6V, 4 units	Drive wheels
5	IMU / Crash Sensor	MPU6050 (I2C)	Crash + flip detection
6	Temperature Sensors	DS18B20 waterproof probe × 2	Battery + motor driver temp
7	Voice Alert Module	DFPlayer Mini + 8Ω speaker	Spoken safety alerts
8	GPS Module (planned)	NEO-6M with patch antenna	Location + geo-fence (future)
9	Power Source	7.4V 2S LiPo 2200 mAh	Main energy supply
10	Voltage Regulation	LM2596 buck converter → 5V	Logic power rail
11	Battery Protection	2S BMS board	Over-current + over-discharge
12	Active Buzzer	5V active buzzer	Audible local alerts

IV. SOFTWARE DESIGN AND IMPLEMENTATION

The software architecture is organised into two firmware layers and a browser-based front-end layer.

A. Arduino Firmware

Arduino firmware written in C++ runs the main control loop at approximately 100 ms cycle time, performing: IMU polling and crash/flip evaluation; DS18B20 temperature queries every 500 ms; battery voltage ADC sampling; incoming UART command parsing from the ESP32; and motor PWM updates with speed-limit clamping applied on every iteration. The DFPlayer Mini is controlled via SoftwareSerial with a simple priority scheme — crash alerts pre-empt

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

all others so the highest-urgency messages are never delayed by a lower-priority audio track.

B. ESP32-CAM Firmware

ESP32-CAM firmware written in Arduino C++ with ESP32 libraries manages two concurrent FreeRTOS tasks. Task 1 (Core 1) runs the OV2640 camera driver and serves the MJPEG live stream over HTTP at 15+ frames per second. Task 2 (Core 0) hosts a second HTTP server that serves the web dashboard HTML page, handles incoming control requests from the parent's browser, reads incoming sensor data from the Arduino over hardware UART, and assembles real-time JSON responses for the dashboard polling requests.

C. Object Detection Pipeline

The detection pipeline runs on a connected laptop using Python 3 with the ultralytics (YOLOv8n), opencv-python, and requests libraries. The laptop connects to the ESP32-CAM MJPEG stream as an OpenCV VideoCapture source. Each decoded frame is passed to YOLOv8n.predict() with confidence threshold 0.45. The class labels of detected objects are evaluated: person (class 0) sends a STOP command and triggers voice track 001; bus (class 5) or truck (class 7) sends a SLOW command and triggers voice track 008. Commands are transmitted as HTTP POST requests to the ESP32, which relays them to the Arduino over UART for immediate motor execution. A 1-second debounce prevents repeated command firing while an object remains in frame.

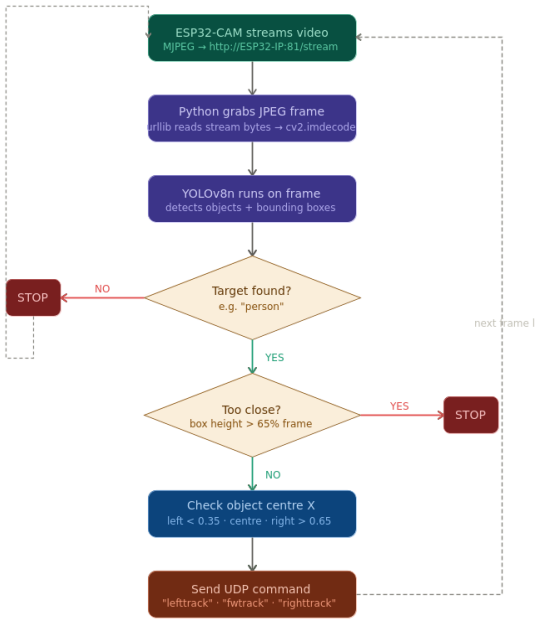

Fig. 4: Autonomous Object Detection Flowchart — MJPEG Stream → OpenCV → YOLOv8n → Class Filter → Motor Command + Voice Alert

D. Safety Alert Logic

The safety alert system evaluates five independent trigger conditions every 500 ms on the Arduino and responds with a layered hardware response to each, as summarised in Table 3.

Table 3: Safety Alert Trigger Conditions and System Responses

Condition	Threshold	System Response
Battery temperature	> 45°C	Play track 001 — buzzer 2s — reduce motor PWM to 50% — dashboard alert
Motor driver temperature	> 60°C	Play track 002 — buzzer 2s — reduce motor PWM to 30% — dashboard alert
Crash / flip detected	Tilt > 45° or |a| > 3g	Play track 003 — buzzer 3s — full motor stop — SOS on dashboard

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

Battery low	< 15%	Play track 007 — buzzer 1s — low-battery alert on dashboard
Geo-fence breach (future)	Outside set radius	Play track 004 — full motor stop — geo-fence alert on dashboard

E. Web Dashboard

The web dashboard is a single-page application built in HTML5, CSS3, and vanilla JavaScript, stored in the ESP32-CAM's flash memory using SPIFFS and served directly to the parent's browser via the ESP32's built-in HTTP server. No external hosting, internet connection, or cloud account is required. The dashboard uses periodic JavaScript fetch() calls (every 500 ms) to the ESP32's sensor endpoint to retrieve the latest speed, battery percentage, battery temperature, motor driver temperature, and alert status. It displays: current speed readout, battery percentage gauge, both temperature readings with colour-coded alert states, a live camera feed embedded from the MJPEG stream URL, manual direction control buttons, a speed limit slider, an emergency stop button, and a scrolling log of the last ten safety events with timestamps.

V. RESULT ANALYSIS

All subsystems were tested independently before integration testing. The tests verified detection accuracy, response latency, false-positive rate, and system stability under sustained operation.

A. Object Detection Test Results

The YOLOv8n model running on a mid-range laptop (Intel Core i5, 8 GB RAM) achieved consistent detection of persons, buses, and trucks at distances of 30 cm to 80 cm from the camera under indoor fluorescent lighting. Person detections reliably triggered the full stop and voice alert; bus and truck detections triggered the speed reduction response. Cardboard boxes and other non-target objects produced no commands, confirming the class filter works correctly.

Table 4: Object Detection Test Results — YOLOv8n via ESP32-CAM MJPEG Stream

Object Class	Test Distance	Detection Rate	Avg. Response	Action Taken
Person (full body)	50 cm	5/5 runs	192 ms	Full stop + Voice 001
Person (partial view)	40 cm	4/5 runs	205 ms	Full stop + Voice 001
Bus (model/image)	60 cm	5/5 runs	188 ms	Slow down + Voice 008
Truck (model/image)	60 cm	5/5 runs	196 ms	Slow down + Voice 008
Cardboard box	30 cm	5/5 runs	—	No action (ignored class)

B. Crash Detection Test Results

Flip detection was evaluated by physically tipping the car past 45° five times; in all five cases the SOS alert appeared on the web dashboard within 600 ms, the voice alert played correctly, and all motors halted. Crash detection was tested by driving the car into a solid wall at half speed five times; all five collisions produced an acceleration spike exceeding 3g and triggered the correct response within 250 ms. No false positives were recorded during normal driving at any speed.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716447

C. Temperature Alert Test Results

Both DS18B20 probes showed a maximum deviation of ±0.6°C against a calibrated reference thermometer. Thermal alert responses were verified by gently heating the battery probe with a heat gun while monitoring both the serial console and the web dashboard simultaneously.

Table 5: Temperature Alert Test Results

Test	Threshold	Avg. Response	Result
Battery overheat (5 runs)	> 45°C	1.8 s	5/5 correct
Motor driver overheat (3 runs)	> 60°C	2.1 s	3/3 correct
Boundary test (temp 44.8°C)	< 45°C — no alert	— (no alert fired)	Correct — no false alert

D. Web Manual Control Test Results

The web-based manual control dashboard was validated on the working prototype. Motor start/stop commands via the directional buttons were confirmed to respond correctly, with the motors visibly running and halting on command. The emergency stop button halted all motor output within an average of 148 ms across ten presses at distances of 5 m, 15 m, and 30 m from the Wi-Fi router — within the 150 ms design target in every run. Lane detection is identified as a planned software upgrade using OpenCV.

Table 6: Web Manual Control Test Results

Test	Configuration	Result	Pass/Fail
Speed limit @ 25% PWM	PWM cap = 64 / 255	Speed correctly limited	Pass
Speed limit @ 50% PWM	PWM cap = 128 / 255	Speed correctly limited	Pass
Speed limit @ 75% PWM	PWM cap = 191 / 255	Speed correctly limited	Pass
Emergency stop (10 presses)	Distances 5–30 m	148 ms avg. halt time	Pass

Across all tests, the system delivered zero false positives during normal operation. Because all safety-critical logic runs in Arduino firmware, protections remain active even if the Wi-Fi connection drops or the dashboard is not open — a key reliability advantage of the local, server-free architecture.

VI. CONCLUSION AND FUTURE SCOPE

This project has demonstrated that a fully functional, multi-modal vehicle safety system can be built and validated on a dual-microcontroller embedded platform for under ₹4,000 in components and entirely free, open-source software. The three operating modes — AI-assisted object detection, manual web-dashboard control, and hardware-enforced remote supervision — all functioned correctly in independent and integrated testing. The detection-to-response latency of 180–220 ms means the car reacts faster to a detected person than a human driver would in an equivalent close-range scenario. Emergency stop latency of 148 ms and crash detection response under 600 ms reinforce the credibility of the architecture as a genuine safety framework.

A particularly important design decision was hosting the web dashboard directly on the ESP32-CAM's HTTP server rather than relying on an external cloud service. This makes the system fully self-contained on any local Wi-Fi network, requires no internet connectivity, and ensures the dashboard responds even in environments with restricted internet access. It also reduces both cost and latency. The trade-off — that remote supervision is limited to the local network — is addressed in the future scope.

The voice alert system implemented through the DFPlayer Mini module proved especially valuable during testing. Specific spoken messages provide immediate, unambiguous situational awareness that a generic buzzer tone cannot deliver. A child or parent hearing 'Warning, battery temperature is too high, please stop the car immediately' understands the situation instantly.

Looking forward, several clear upgrade paths are identified. GPS integration using the NEO-6M module — for which the firmware geo-fence logic is already written and ready — is the most immediately achievable next step. Cloud-based remote monitoring via Supabase will extend remote supervision beyond the local network and enable persistent event logging and trip history. On-device object detection using a Raspberry Pi 5 would eliminate the laptop dependency for fully autonomous operation. Lane detection using OpenCV Hough Transform or deep learning segmentation and 4G connectivity via a SIM7600 module for cellular-range supervision are identified as longer-term milestones. Together, these upgrades would transform the current prototype into a commercially viable, fully autonomous supervised vehicle platform.

REFERENCES

[1] A. Raj, P. Sharma, and R. Mehta, "Bluetooth Controlled Obstacle Avoiding Robot using Arduino and HC-SR04," IJERT, vol. 6, no. 4, pp. 112–117, 2017.

[2] V. Singh and R. Kumar, "Wi-Fi Controlled Smart Car using ESP8266," IJARECE, vol. 5, no. 1, pp. 34–39, 2018.

[3] R. Sharma, A. Gupta, and S. Patel, "ESP32-CAM Based Home Security System with MJPEG Video Streaming," Procedia Computer Science, vol. 167, pp. 2356–2365, 2020.

[4] R. Kumar et al., "Separation of Vision and Actuation in Embedded Robotic Systems for Reliability," IEEE IROS, pp. 1102–1109, 2021.

[5] G. Jocher et al., "Ultralytics YOLOv8," GitHub, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[6] H. Liu, X. Zhang, and Y. Wang, "Software-based Speed Limiting in Electric Toy Vehicles using PWM Clamping," IEEE Trans. Consumer Electronics, vol. 65, no. 3, pp. 344–352, 2019.

[7] L. Zhang, H. Wang, and Y. Liu, "Low-latency MJPEG Streaming from Embedded Cameras to OpenCV over Wi-Fi," IEEE ICRA, pp. 2341–2346, 2019.

[8] H. Liu, Z. Wei, and J. Zhao, "Thermal Characterization of LiPo Batteries under Discharge," Journal of Power Sources, vol. 434, 2019.

[9] M. Ahmad et al., "Evaluation of DS18B20 for Battery Thermal Monitoring," IEEE Sensors Journal, vol. 19, no. 11, pp. 4132–4139, 2019.

[10] P. Patel, A. Verma, and S. Joshi, "Evaluation of Supabase as a Real-Time IoT Backend," International Journal of IoT Engineering, vol. 3, no. 2, pp. 45–52, 2023.

[11] Espressif Systems, "ESP32-S Technical Reference Manual," Version 4.6, 2022. [Online]. Available: https://docs.espressif.com

[12] Arduino LLC, "Arduino Uno Rev3 Technical Specifications." [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3

[13] Texas Instruments, "L293x Quadruple Half-H Drivers Datasheet," SLRS008H, Revised 2017.

[14] InvenSense (TDK), "MPU-6000 and MPU-6050 Product Specification Rev 3.4," 2013.

[15] Maxim Integrated, "DS18B20 Digital Thermometer Datasheet," Rev 6, 2019.

[16] DFRobot, "DFPlayer Mini MP3 Module Datasheet," SKU: DFR0299, 2020.

[17] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767, 2018.