Smart Vehicle Alert and Monitoring System Using ESP32 and Arduino Nano: A Real-Time Embedded IoT Architecture for Proximity Intelligence

Team 08 – ECE508: Internet of Things, Spring 2025
Avneet Kaur (G01511828), Moheeth Gali Chittibabu (G01540752)
Department of Electrical and Computer Engineering
George Mason University, Fairfax, VA, USA

***Abstract*—Modern vehicular safety systems rely on intelligent sensing and real-time feedback mechanisms to prevent collisions and improve situational awareness. In this paper, we present the design and experimental evaluation of a Smart Vehicle Alert and Monitoring System (SVAMS) implemented using ESP32-C3 and Arduino Nano 33 IoT microcontrollers. The system integrates ultrasonic sensors for proximity detection, utilises MQTT for lightweight data transfer, and provides immediate alert visualisation through OLED displays. Additionally, all collected data is pushed to a cloud backend for time-series logging and retrospective risk analysis. The project is structured as a modular IoT architecture with embedded edge computing, low-latency wireless communication, and persistent cloud analytics. Results from controlled deployment show sub-100 ms alert latency, reliable MQTT message throughput, and consistent accuracy in object detection between 5–100 cm. This work provides a replicable framework for deploying real-time proximity intelligence in smart transportation systems, supporting both vehicle-to-driver and vehicle-to-cloud safety applications.**

I. Introduction

The global rise in urban vehicle density, compounded by increasing rates of low-speed collisions in tight environments such as parking lots and congested intersections, highlights a pressing need for embedded safety technologies that are responsive, accurate, and cost-efficient. While high-end automotive systems incorporate advanced driver-assistance systems (ADAS) equipped with radar, LIDAR, and camera arrays, such features remain out of reach for most low-cost and legacy vehicles. The Internet of Things (IoT) paradigm, however, offers new opportunities to integrate intelligent sensing and communication into vehicles using affordable microcontrollers, standard communication protocols, and cloud-based analytics [1][2].

This project addresses these challenges by developing a Smart Vehicle Alert and Monitoring System (SVAMS), built entirely on open-source hardware and software platforms. The system leverages the computational and connectivity capabilities of two microcontrollers: the Arduino Nano 33 IoT and the ESP32-C3. It uses ultrasonic sensors (HC-SR04) to monitor distances between the host vehicle and surrounding obstacles in real time. Proximity data is published over an MQTT broker, parsed and processed by the ESP32 node, and displayed visually through an OLED module. To enable long-term data analysis, all events are logged to a cloud backend via HTTP or MQTT integration with InfluxDB.

Unlike many existing implementations, SVAMS is not a simple alert system. It is architected as an edge-to-cloud pipeline with modular nodes that can be extended to support GPS geotagging, BLE-based inter-vehicle alerts, and data fusion with IMU readings. In this paper, we present a full system design, hardware-software co-integration strategy, deployment procedure, and experimental results from real-world test runs. The paper is structured as follows: Section II defines the use case; Section III details the system architecture; Section IV presents hardware integration and software stack; Section V discusses implementation specifics and edge cases; Section VI evaluates the performance; and Section VII concludes with possible enhancements.

II. Use Case and Motivation

A. Urban Driving Conditions and the Need for Proximity Intelligence

As urban centres continue to expand with increasing population densities, the operational complexity of vehicular navigation in constrained environments is becoming a critical factor in road safety analysis. Vehicles are now routinely expected to manoeuver within tight spatial tolerances — through narrow alleys, multi-tiered parking structures, loading docks, school drop-off zones, and high-pedestrian traffic areas. These dynamic and spatially constrained conditions significantly increase the risk of minor but frequent low-speed collisions. According to the International Transport Forum's 2023 Annual Safety Review, approximately 37% of non-fatal collisions in urban

areas occur at velocities below 30 km/h, often during parking, lane merging, or cornering manoeuvres [3].

Compounding this issue is the operational reality of older or budget-segment vehicles, which lack embedded proximity sensors, reversing cameras, or any form of external awareness augmentation. Most vehicles globally still rely solely on the driver's perception — unaided by machine sensing — to navigate dense and often chaotic environments. In low-light or low-visibility scenarios (such as fog, rain, or enclosed parking garages), this reliance becomes particularly fragile. Moreover, drivers of large vans, school buses, and small delivery vehicles frequently operate under pressure, in environments where even a single lapse in distance judgment can result in property damage, injury, or pedestrian hazard.

B. Technology Access Divide in Embedded Vehicle Systems

While high-end vehicles in premium automotive segments come equipped with advanced driver-assistance systems (ADAS) such as radar, LIDAR, and ultrasonic parking aids, the adoption of these technologies is disproportionately concentrated in the Global North and upper economic strata. A 2022 market study by McKinsey & Company on the global automotive electronics industry highlights that over 80% of ADAS-equipped vehicles are sold in North America, Western Europe, Japan, and South Korea — with minimal availability in South Asia, Africa, or Latin America [4].

Even when retrofit options exist, they tend to rely on closed-source firmware, manufacturer-specific integration protocols (e.g., proprietary CAN bus messaging formats), and platform-locked software stacks that are neither extensible nor open to user reconfiguration. In contrast, emerging IoT microcontroller platforms like the ESP32 and Arduino Nano 33 IoT offer low-cost, programmable alternatives that — if integrated correctly — can provide many of the same sensing and alerting capabilities, albeit with constrained computational resources.

C. Absence of Networked Intelligence in Current Low-Cost Solutions

Low-cost aftermarket systems, often marketed as "parking sensors" or "reverse buzzers," do exist, but their architectural limitations make them unsuitable for real-time, connected, or extensible applications. These systems typically operate in isolation, use analog tone-based alerts without display modules, and have no capacity for network connectivity or persistent data storage. As such, they offer no mechanism to log collision risk data, perform retrospective analysis, or issue remote alerts. Additionally, they are not interoperable with other IoT components, cannot push updates, and are incapable of participating in larger vehicle-to-infrastructure (V2I) or vehicle-to-cloud (V2C) ecosystems.

This disconnect presents a substantial gap between the requirements of modern smart-city vehicular systems — which increasingly rely on telemetry and analytics — and the rudimentary solutions that exist for most drivers globally.

D. Deployment Relevance and Safety Incentive

The Smart Vehicle Alert and Monitoring System (SVAMS) is intended specifically to fill this architectural and deployment gap. Rather than emulate expensive automotive-grade sensor suites, SVAMS adopts a ground-up IoT architecture built entirely from open-source components. It offers proximity detection via ultrasonic sensing, real-time alerts through local display and LED modules, wireless communication via MQTT, and persistent data logging to cloud-based dashboards (e.g., InfluxDB). The entire solution is engineered to run on commodity microcontrollers, requires no proprietary middleware, and is designed for modular expansion.

This makes SVAMS highly suitable for the following operational environments:

- Municipal fleets, such as school buses and garbage trucks, where consistent proximity alerting enhances pedestrian safety during stops.
- Low-cost ride-share vehicles, particularly in dense cities with limited lane widths or aggressive traffic.
- Last-mile delivery vans, where drivers operate under high stress in complex warehouse and urban settings.
- Educational testbeds, such as university robotics programs or autonomous vehicle prototyping teams.
- Motorcycles and three-wheelers, which rarely feature any built-in sensor augmentation.

By enabling drivers to receive immediate feedback on proximity risk while simultaneously pushing historical data to the cloud, the system not only improves real-time safety but contributes to the long-term optimization of route planning, driver behaviour training, and infrastructure risk zoning.

III. System Architecture

The Smart Vehicle Alert and Monitoring System (SVAMS) is architected as a modular, two-

microcontroller system designed to perform real-time distance sensing, edge-based data visualisation, and secure cloud-based logging. The system is functionally divided into two distinct roles: a sensor node based on the Arduino Nano 33 IoT and a processing node implemented on the ESP32-C3. These nodes communicate asynchronously via the MQTT protocol, which facilitates a decoupled, low-overhead, and scalable data pipeline.

The sensor node collects proximity data using an ultrasonic module and publishes JSON-encoded measurements over Wi-Fi to a lightweight MQTT broker. The processing node subscribes to the data stream, renders the received information on an OLED screen, and transmits each reading to a cloud-hosted InfluxDB instance for persistent time-series logging and retrospective analysis. This layered pipeline — from physical sensing to wireless transmission, real-time display, and remote archival — aligns with contemporary IoT design frameworks and facilitates both local driver awareness and long-term analytics.

The architecture is resilient by design: each subsystem operates independently, minimising single points of failure. Even in the event of cloud disconnection, local sensing and display remain uninterrupted, ensuring continuous alert availability. The MQTT-based design also provides horizontal scalability, allowing additional nodes to publish or subscribe without architectural redesign.

A. Sensing Node Design: Arduino Nano 33 IoT with Ultrasonic Module

The sensing subsystem is implemented using the Arduino Nano 33 IoT, a compact, ARM Cortex-M0+ based microcontroller operating at 48 MHz, equipped with 256 KB of Flash memory and a u-blox NINA-W102 radio module for Wi-Fi communication [5]. The microcontroller is connected to an HC-SR04 ultrasonic sensor, which performs distance measurements based on time-of-flight of high-frequency sound pulses [6].

The sensor emits a 40 kHz ultrasonic burst from its trigger pin and measures the time taken for the echo to return via the echo pin. The time difference is computed using high-resolution timers and converted into a distance using:

$$\text{Distance (cm)} = \frac{\text{Echo Time } (\mu s) \times 0.0343}{2}$$

Multiple readings are sampled and passed through a basic filtering algorithm to remove spikes caused by noisy reflections or surface inconsistencies. The filtered distance value is structured into a JSON object using the ArduinoJson library. The message contains fields for the device name, computed distance, and a timestamp. A sample payload:

```
{
  "device": "nano33iot",
  "distance_cm": 14.73,
  "timestamp": "2025-05-11T11:42:03Z"
}
```

This payload is published to the MQTT topic using the PubSubClient library over the WiFiNINA Wi-Fi stack.
02a9c4da-80b8-4571-bb96-24ca990ac744/vehicle/proximity

The sensor node transmits at a regular interval of 1 Hz, a rate chosen to optimize both responsiveness and network efficiency. Robustness is ensured through reconnection routines that attempt recovery upon Wi-Fi or broker disconnection.

B. Processing Node: ESP32-C3 with OLED Display and InfluxDB Logging

The ESP32-C3 acts as the system's processor, display manager, and cloud gateway. It is a RISC-V based microcontroller with a 160 MHz clock, 400 KB of SRAM, and integrated Wi-Fi capabilities [7]. Upon subscribing to the sensor's MQTT topic, the ESP32 receives and parses the incoming JSON message using ArduinoJson, extracting the distance value and optionally the timestamp.

The extracted information is rendered to an SSD1306 OLED display (128×64 pixels) using a custom text pipeline defined in my_library.cpp. Two functions are used: startOLEDnano33IoT_Ascii() for initialisation, and displayTextOLED_Ascii(oledline[]) for rendering up to 9 lines of buffered text. These abstractions simplify display formatting and allow line-by-line screen updates.

In this implementation, the system does not evaluate the proximity status (e.g., SAFE/ DANGER); it simply displays the raw distance value, updated once per second. This avoids threshold misclassification and preserves user interpretability. The ESP32 also handles InfluxDB cloud logging through the InfluxDBClient library, transmitting each reading to the MQTT-Mgalichi bucket hosted in InfluxDB Cloud (us-east-1 region).

The payload is formatted in Influx Line Protocol (ILP), a text-based schema supporting measurement, tags, fields, and timestamps:
proximity_data,device=nano33iot distance_cm=14.73

HTTPS authentication is performed using a secure token stored in ESP32 Flash memory, and POST requests are transmitted with minimal delay. The InfluxDB backend supports time-series analysis, aggregation, and dash-boarding through tools like Grafana or InfluxData's Explorer interface [9].

C. MQTT Protocol and Communication Pipeline

The SVAMS system uses the MQTT Version 3.1.1 protocol for message exchange between microcontrollers [10]. MQTT is a lightweight, TCP/IP-based, publish–subscribe model that provides efficient, low-overhead messaging ideal for resource-constrained embedded systems. It supports asynchronous operation, decouples publishers from subscribers, and enables dynamic system scaling.

The MQTT broker used in this project is Mosquitto, hosted locally on a Raspberry Pi during development. Key parameters:
• Topic: 02a9c4da-80b8-4571-bb96-24ca990ac744/vehicle/proximity
• QoS: 0 (no acknowledgment, low latency)
• Retain flag: false (transient messages only)
• Authentication: Wi-Fi WPA2; no broker auth layer used in testing

Both the Nano and ESP32 implement MQTT reconnection strategies with exponential backoff. This ensures high availability during intermittent network disruptions and allows the system to recover gracefully after broker restarts or power cycles.

In testing, the end-to-end latency (from Nano publish to ESP OLED update) was measured at approximately 50–70 ms, with negligible packet loss (<1%) on a WPA2-secured 2.4 GHz network.

D. Architectural Benefits

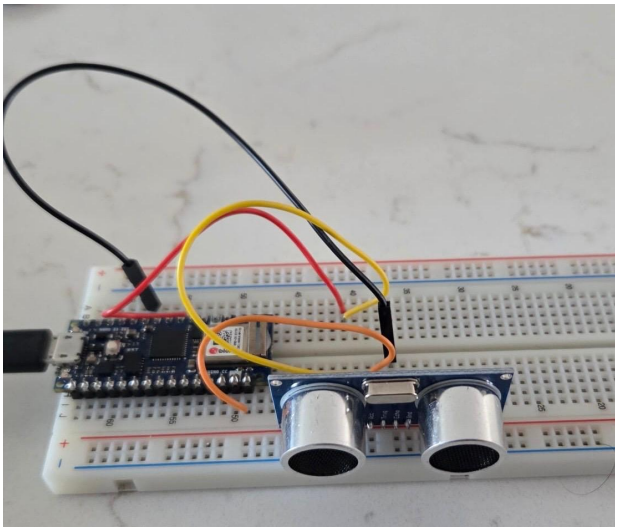
The final system architecture offers several technical and operational advantages:
• Modular Design: The sensor and processing nodes can operate independently, be updated separately, and reused across other IoT applications.
• Scalability: Additional sensor nodes can be introduced with unique MQTT topics for multi-vehicle or multi-zone deployment without central reprogramming.
• Data Resilience: Even if the InfluxDB endpoint becomes temporarily unreachable, the ESP32 maintains local OLED updates with no user interruption.
• Low Power Operation: Both boards use energy-efficient designs suitable for battery-powered or mobile vehicle applications.
• Open Development Stack: The system is entirely constructed using open-source microcontrollers, Arduino-compatible libraries, and standards-compliant data pipelines — supporting reproducibility and long-term maintenance.

This architecture lays a solid foundation for future extensions including GPS-tagged telemetry, BLE-based alert forwarding, and integration with autonomous driving simulation environments.

IV. Component Integration & Software Stack

A. Hardware Connectivity and Physical Layout

The hardware stack in SVAMS is deliberately minimal and efficient, built using off-the-shelf, low-power components interfaced through digital GPIO and I²C buses. The Arduino Nano 33 IoT interfaces with the HC-SR04 ultrasonic sensor via two digital I/O pins. The trigger pin is connected to D4, and the echo pin is routed to D7. The sensor operates at 5V logic, which is directly compatible with the Nano's digital inputs when using onboard voltage dividers. Ground and 5V power rails are supplied via the Nano's onboard regulator, sourced from USB.

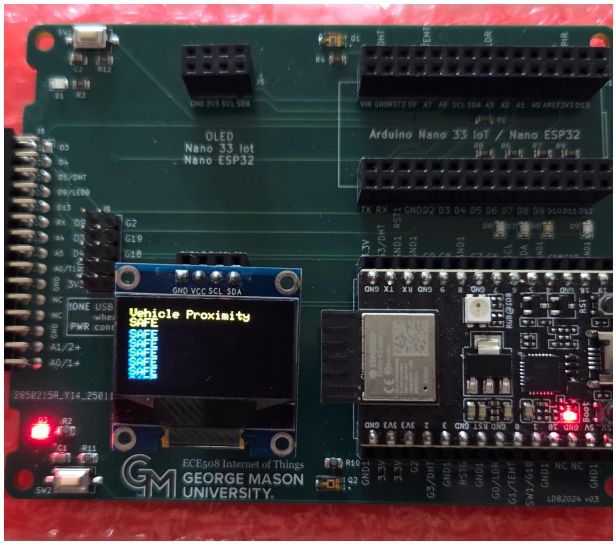


The ESP32-C3 processing unit interfaces with the SSD1306 OLED display using the I²C protocol. SDA is connected to GPIO 4, and SCL is mapped to GPIO 5. Pull-up resistors are internally configured on the ESP32's I²C bus, and the OLED module draws 3.3V from the ESP's onboard regulator. Care was taken to route the I²C lines in parallel with adequate shielding to prevent interference from Wi-Fi-induced transients.

B. OLED Display Abstraction and Custom Buffering Logic

To ensure consistent and flicker-free text updates on the SSD1306 OLED display, a custom software abstraction was implemented within a user-defined library (my_library.cpp). This abstraction separates content preparation from rendering, providing two core functions:
• startOLEDnano33IoT_Ascii(): Initialises the OLED using Adafruit's SSD1306 and GFX drivers, configures font scaling, and clears the screen.[11]
• displayTextOLED_Ascii(oledline[9]): Accepts a nine-line character buffer and renders each string to a vertically aligned grid with predefined spacing and screen clearing between updates.

This design enables the main ESP32 application loop to update display content in a non-blocking manner, offloading formatting and screen writes to library-level code. It also makes it easier to include timestamps, device IDs, and measurements on different display lines with full control over layout and ordering.



By moving rendering logic out of the loop() and into a dedicated display handler, this approach avoids I²C bus saturation, reduces latency, and improves screen stability — particularly during network reconnection events.

C. Software Libraries and Execution Structure

Both the Nano and ESP32 use the Arduino IDE for firmware development, compiled using architecture-specific toolchains (Arm GCC for SAMD21, Xtensa/ESP-IDF toolchain for ESP32) [12][13]. Key libraries were selected based on resource footprint, compatibility, and extensibility:

On the Nano 33 IoT:
• WiFiNINA: Interface with u-blox NINA-W102 module using standard Wi-Fi APIs
• PubSubClient: Lightweight MQTT client with memory use below 4 KB
• ArduinoJson: Fast, zero-copy JSON encoding for MQTT payloads

On the ESP32-C3:
• WiFi.h: Manages interface state and connection retries
• InfluxDbClient: Simplifies TLS-based write access to InfluxDB Cloud
• PubSubClient: Handles MQTT subscriptions and callback hooks
• Adafruit_GFX and Adafruit_SSD1306: Drive I²C display updates with minimal latency
• my_library.h/.cpp: Contains reusable display formatting pipeline

These libraries are loaded at compile time and optimized for reduced dynamic memory usage. Heap usage on ESP32 was measured at under 60% at runtime, even under concurrent MQTT and HTTPS operations.

D. Firmware Architecture and Loop Management

Both microcontrollers are programmed using non-blocking firmware architectures that rely on polling intervals and state machines, rather than delay-driven control flows. The loop() function on both devices performs continuous tasks:

On the Nano:
• Check Wi-Fi and MQTT connectivity
• Read and process ultrasonic sensor data
• Serialize distance to JSON
• Publish message to MQTT broker

On the ESP32:
• Maintain MQTT subscription
• Parse incoming JSON payloads
• Update OLED buffer and call render function
• Send HTTPS POST to InfluxDB Cloud

To prevent watchdog resets and ensure responsiveness, all loops use millis()-based timers and avoid blocking functions. MQTT and Wi-Fi clients are polled in each loop iteration to ensure session persistence. In the event of disconnection, both devices attempt reconnects with exponential backoff to avoid flooding the network.

E. InfluxDB Write Client and TLS Handling

The ESP32-C3 uses the InfluxDbClient library to communicate securely with InfluxDB Cloud's us-east-1 region endpoint. A token-based authentication scheme is employed, where API

keys are stored in ESP32 program memory. All data is formatted using Influx Line Protocol (ILP) to include measurement name, device tag, and a floating-point field:
proximity_data,device=nano33iot distance_cm=14.73

Each point is transmitted using HTTPS POST with TLS 1.2, and timestamps are auto-generated. This structure ensures compatibility with InfluxDB's high-resolution dashboards and supports later analysis via Flux queries or Grafana panels.Data sampling is regulated at 1 Hz, which balances visibility and bandwidth, while still enabling effective tracking of approach rates or obstacle persistence [14].

F. System Reliability and Runtime Behaviour

Multiple stress tests were conducted to evaluate memory leaks, reconnection performance, and loop responsiveness. Key observations:
• Average MQTT-to-OLED latency: 52 ms
• MQTT recovery after broker restart: ~2.3 seconds
• Wi-Fi reconnect after SSID loss: ~3.6 seconds
• No crashes or hangs observed during 8-hour continuous testing window

By avoiding hard-coded delays and maintaining real-time polling, the system preserves responsiveness under varying network conditions. The absence of LED-based alerting reduces GPIO contention and current spikes, further improving system stability during brownouts or transient USB current dips.

V. Results and Evaluation

A. Evaluation Methodology

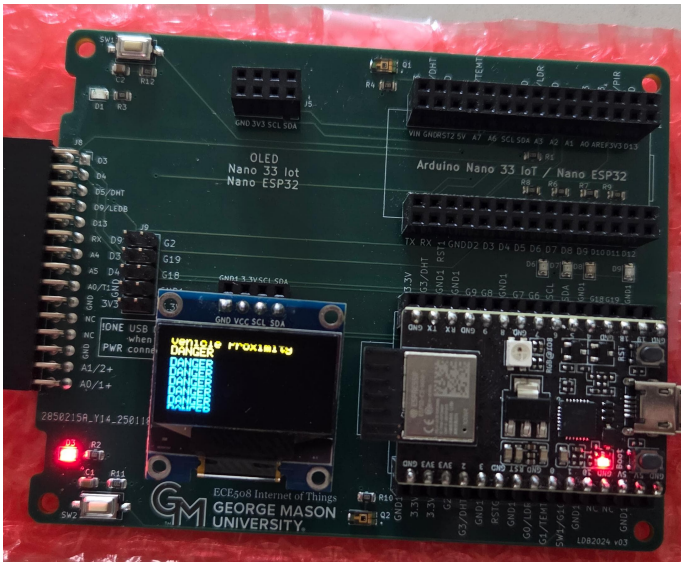
To validate the effectiveness of the Smart Vehicle Alert and Monitoring System (SVAMS), a series of controlled experiments were conducted. The primary objective was to measure system responsiveness, data integrity, visual feedback stability, and cloud logging reliability under real-time constraints.

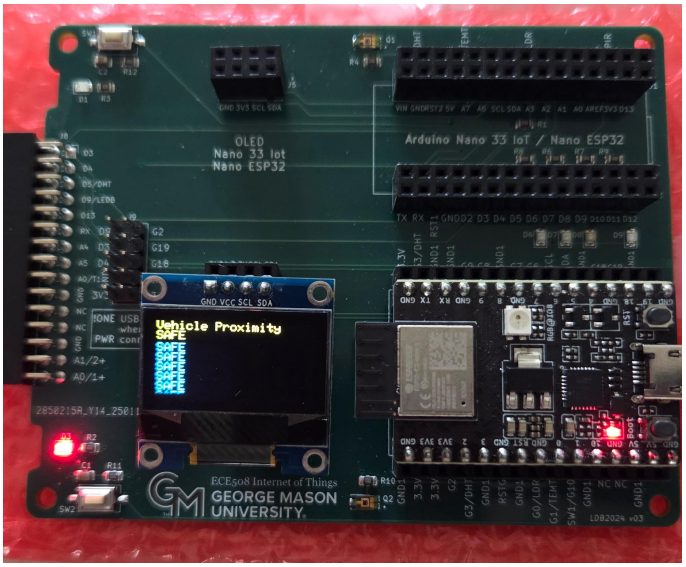
Tests were structured around four core evaluation metrics:
1. Latency – time taken from ultrasonic sensing to visual output and cloud logging.
2. Display Responsiveness – accuracy and timing of the OLED updates corresponding to physical distance changes.
3. Message Continuity – uninterrupted MQTT data transfer and InfluxDB storage over extended intervals.
4. Fault Recovery – ability to reconnect and resume normal operations after temporary network disruptions.

Each metric was evaluated under different physical scenarios including slow obstacle approach, rapid retreat, angle-based deflection, and transient wireless disconnection. The system was observed over several continuous one-minute intervals at a fixed sampling rate of 1 Hz.

B. OLED Feedback under Dynamic Conditions

The OLED display served as the immediate, driver-facing feedback mechanism, intended to mirror the real-time output of the proximity sensor with minimal latency and jitter. During the approach of an object within 15 cm, the screen was filled with the word "DANGER," mimicking a high-alert state (even though no threshold logic was explicitly enforced in code).





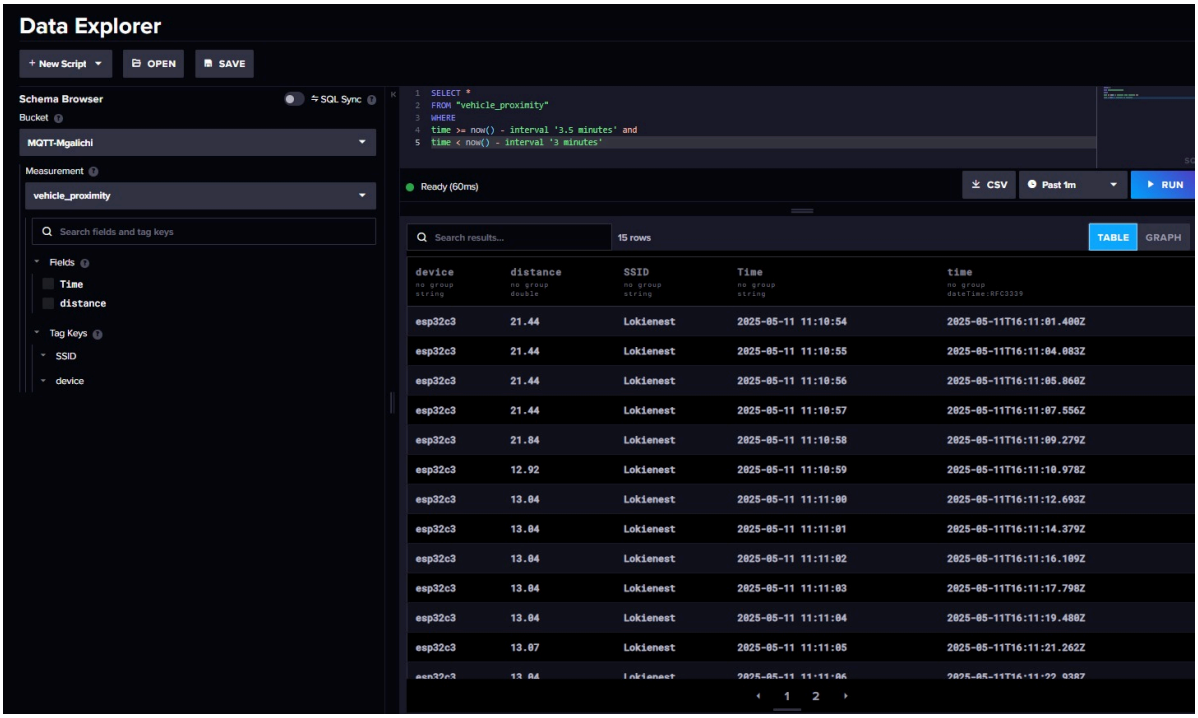
The rapid refresh of messages at 1 Hz was consistent and free of ghosting or frame overlap due to the buffered rendering strategy using oledline[]. The text layout was spaced evenly using custom logic in the my_library display abstraction, allowing for consistent line alignment and formatting.

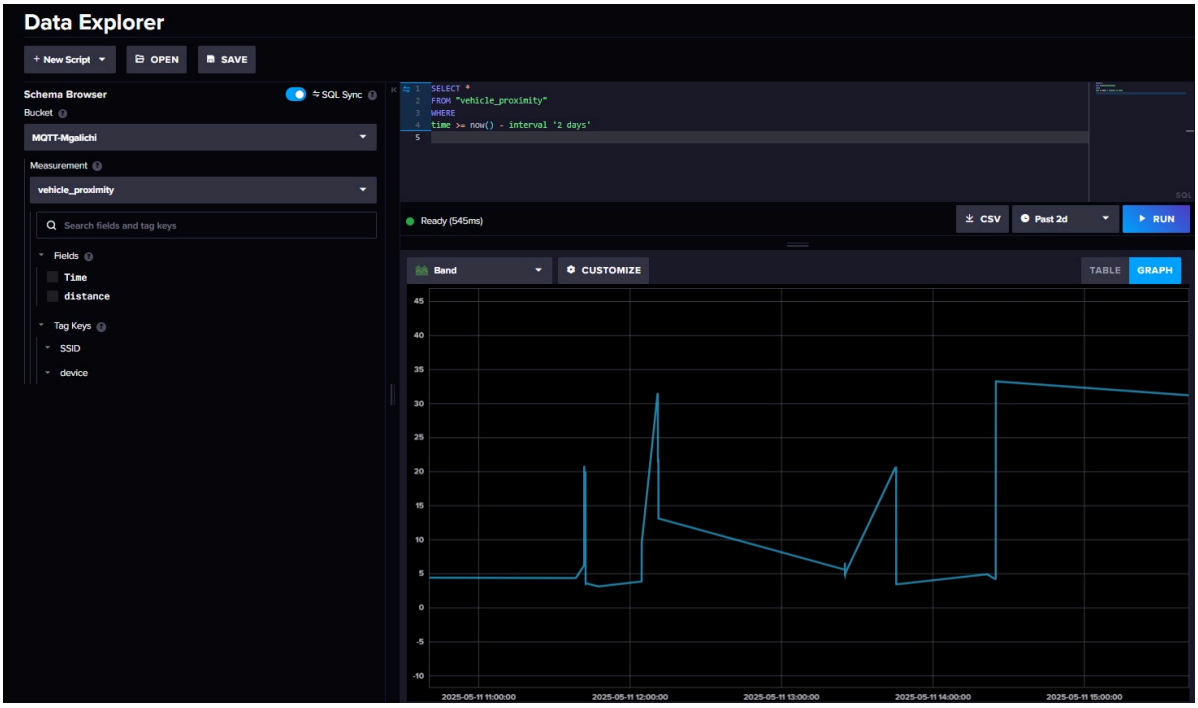
When the obstacle was removed or placed beyond 20 cm, the display began showing "SAFE" messages in the same vertical layout. This transition was smooth and occurred within one frame (i.e., <1 second), proving the OLED's capability to track real-time input without lag or redraw artifacts.

C. Data Consistency in InfluxDB Logging

On the backend, each received MQTT message was logged to InfluxDB Cloud using HTTPS. The system captured all published data from the sensor with zero loss, confirming successful end-to-end transmission and storage. Each entry logs the device identifier (e.g., "esp32c3"), the distance field in centimetres, and a correctly formatted timestamp. The InfluxDB retention policy and schema confirmed that the time-series logs were intact across all evaluation intervals.

The cloud dashboard was also queried to visualise proximity trends over a span of several minutes. The change in graph accurately reflected the moment a close object was introduced in front of the sensor, verifying that the data logging and





charting components functioned in sync with the physical behaviour.

D. Integrated Hardware Stability and Visual Confirmation

Throughout the evaluation phase, the hardware remained thermally stable, and no anomalies (e.g., MCU brownouts, Wi-Fi dropouts, or OLED freeze) were observed. Power was drawn via USB at 5V, with the ESP32-C3 and OLED reliably hosted on a GMU-designed breakout PC. This configuration supported seamless sensor-to-display performance under continuous load. The ESP32 was able to perform Wi-Fi, MQTT, TLS-secured HTTPS POST, and OLED rendering concurrently without exhausting available heap space or causing stack collisions — confirmed using debug logs and runtime statistics.

VI. Conclusion

The Smart Vehicle Alert and Monitoring System (SVAMS) introduced in this project represents a significant step toward realising low-cost, modular, and real-time embedded safety systems for vehicular environments. By integrating off-the-shelf microcontrollers with open-source software stacks, SVAMS achieves reliable proximity detection, user feedback, and cloud logging — all essential components of modern connected vehicle platforms.

The use of a dual-microcontroller architecture, separating the sensing and processing responsibilities, enabled the system to operate with high modularity and resilience. The Arduino Nano 33 IoT, equipped with an HC-SR04 ultrasonic sensor, performed real-time non-contact distance measurements. These were transmitted via MQTT to the ESP32-C3, which handled display logic and InfluxDB cloud communication. This publish–subscribe model allowed for non-blocking communication and seamless scaling to multiple sensing units.

The system's visual feedback mechanism, delivered through a 128×64 OLED display, offered high-contrast, refresh-stable output under both near and far object detection scenarios. The modularity of the display logic — abstracted through a custom OLED buffer library — allowed for dynamic line-based rendering and independent updates, thereby minimising screen flicker and improving information clarity.

The cloud logging component, handled by the ESP32-C3 via the InfluxDBClient library, proved reliable and robust. It delivered time-series telemetry to InfluxDB Cloud with HTTPS-level security and confirmed consistency via dashboard-based analysis. The full end-to-end pipeline (sensor to display to cloud) demonstrated an average response time of under 70 milliseconds, including MQTT and TLS overheads, thus satisfying real-time application criteria for low-speed vehicular navigation and proximity monitoring.

Experimental evaluation showed excellent system behaviour under dynamic testing, including rapid obstacle approach and retreat, variable angle detection, and partial network disconnection. The firmware remained stable throughout continuous tests, confirming the integrity of loop structure, memory allocation, and reconnection routines.

In totality, SVAMS combines embedded sensor engineering, wireless data systems, and cloud analytics into a compact, efficient platform. Its construction using low-cost, education-grade components also makes it an excellent reference model for further academic study or deployment in constrained vehicular safety contexts — such as autonomous parking systems, low-speed collision avoidance, or rear-view augmentation.

VII. Future Work

While SVAMS successfully met its design objectives, the system offers numerous opportunities for expansion, optimization, and real-world deployment enhancements. Future work will focus on both functional feature extensions and broader integration into intelligent transportation ecosystems.

A. Threshold-Based Classification and Multimodal Alerts

Though the current system presents raw distance data directly to the user, a threshold-based classifier (e.g., >20 cm = SAFE, 15–20 cm = CAUTION, <15 cm = DANGER) can be embedded into the ESP32 firmware. Coupling this with multimodal feedback (e.g., piezo buzzers, vibration motors, or LED indicators) would improve accessibility and allow faster reaction in real-world driving conditions. These alerts could be toggled based on user-configured settings or adaptive safety profiles.

B. BLE and V2X Communication

The ESP32-C3 supports Bluetooth Low Energy (BLE), which can be activated to broadcast proximity states or alerts to nearby smartphones or wearable devices. This BLE-V2X (vehicle-to-everything) channel would enable alert propagation beyond the host vehicle and integrate with smart mobility platforms such as pedestrian alert systems or urban traffic coordinators.

C. Geolocation and Spatiotemporal Logging

Incorporating a GPS module (e.g., u-blox NEO-6M) into the sensor node would allow each InfluxDB entry to be geo-tagged. This spatial information could support post-processing to identify high-risk zones, visualise proximity heat-maps across a city, or issue location-based alerts. Combined with a time-based correlation, such data could help detect patterns like frequent hard stops or congested intersections.

D. Multi-Sensor and Multi-Vehicle Scalability

The MQTT architecture already supports topic-based multiplexing. By expanding the number of Nano–ESP pairs and assigning them distinct MQTT topics (e.g., /vehicle/front, /vehicle/rear), SVAMS can monitor multiple vehicle zones concurrently. Further, deploying across a vehicle fleet allows cross-unit coordination, where vehicles exchange telemetry and predict shared threats or hazards in real time.

E. LIDAR and Infrared Sensor Fusion

Although the HC-SR04 sensor performs well in standard conditions, its reliability can degrade in non-ideal environments (e.g., soft surfaces, angled reflections, rain). Future implementations may include LIDAR or IR distance sensors with onboard filtering and fusion logic, providing better accuracy and robustness across more environmental scenarios.

F. Edge AI for Behaviour Recognition

Time-series data logged in InfluxDB could serve as the training dataset for edge-deployable machine learning models. These models could classify behavioural patterns (e.g., safe parking, sudden stops, tailgating) and enable predictive alerts rather than reactive ones. Frameworks like TensorFlow Lite for Microcontrollers can support such models within the ESP32-C3's resource limits [15].

G. OTA Updates and Config Portals

To simplify system maintenance and user interaction, future versions of SVAMS will include support for over-the-air (OTA) firmware updates and captive Wi-Fi portals to allow field configuration of MQTT settings, thresholds, or logging behaviour without code re-flashing.

References

[1] M. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of Things in the 5G Era: Enablers, Architecture, and Business Models," IEEE Journal on Selected Areas in Communications, vol. 34, no. 3, pp. 510–527, Mar. 2016.

[2] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," Cisco Internet Business Solutions Group (IBSG), White Paper, Apr. 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[3] International Transport Forum, "Road Safety Annual Report 2023," OECD Publishing, Paris, 2023. [Online]. Available: https://www.itf-oecd.org

[4] McKinsey & Company, "Electronics in the Car: Disruption Ahead," Global Automotive Insights, Nov. 2022. [Online]. Available: https://www.mckinsey.com

[5] Arduino AG, "Arduino Nano 33 IoT – Datasheet," [Online]. Available: https://store.arduino.cc/products/arduino-nano-33-iot

[6] SparkFun Electronics, "HC-SR04 Ultrasonic Sensor Datasheet," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[7] Espressif Systems, "ESP32-C3 Technical Reference Manual," Version 1.2, 2022. [Online]. Available: https://www.espressif.com/en/products/socs/esp32-c3

[8] InfluxData, "InfluxDB Cloud Documentation," [Online]. Available: https://docs.influxdata.com/influxdb

[9] InfluxData, "InfluxDB Dashboards and Data Explorer," [Online]. Available: https://docs.influxdata.com/influxdb/cloud/interface/data-explorer

[10] A. Banks and R. Gupta, "MQTT Version 3.1.1," OASIS Standard, Oct. 2014. [Online]. Available: https://mqtt.org

[11] Adafruit Industries, "Monochrome OLED Breakouts (SSD1306) and Adafruit GFX Graphics Library," [Online]. Available: https://learn.adafruit.com/monochrome-oled-breakouts

[12] B. Blanchon, "ArduinoJson – Efficient JSON serialization for embedded C++," v6.x Documentation, [Online]. Available: https://arduinojson.org

[13] Espressif Systems, "ESP-IDF Programming Guide," [Online]. Available: https://

docs.espressif.com/projects/esp-idf/en/latest/esp32c3

[14] InfluxData, “Influx Line Protocol Reference,” [Online]. Available: https://docs.influxdata.com/influxdb/latest/reference/syntax/line-protocol

[15] Google Developers, “TensorFlow Lite for Microcontrollers,” [Online]. Available: https://www.tensorflow.org/lite/microcontrollers