

Project Report
On
Embedded Vehicle Black Box for Telematics and
Parameter Logging



Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in Embedded Systems and Design
(PG-DESD)

C-DAC, ACTS (Pune)

Guided By:

Mr. Rhugved Rane

Submitted By:

Kolamala Srivikas	250240130018
Rahul Avhad	250240130027
Abhishek Kalyankar	250240130015
Dipendra Bodke	250240130008

Centre for Development of Advanced Computing(C-DAC), ACTS

(Pune- 411008)

Acknowledgement

We would like to express our sincere gratitude to everyone who contributed to the successful completion of our project, **“Embedded Vehicle Black Box for Telematics and Parameter Logging”**

First and foremost, we are extremely thankful to our project guide, Mr. Rhugved Rane, for their valuable guidance, encouragement, and constant support throughout the project. Their expertise and constructive feedback played a crucial role in shaping the direction and outcome of our work. We extend our heartfelt thanks to the faculty and staff of the PG-DESD CDAC ACTS Pune, for providing the infrastructure and resources necessary for carrying out this project. Their technical support and encouragement were instrumental during the development and testing phases. We are also grateful to our peers and classmates for their feedback, collaboration, and motivating discussions. Their suggestions helped us refine our ideas and improve the functionality of the system. Lastly, we thank our families for their unwavering support and understanding during this journey. Their patience and encouragement inspired us to persevere through challenges and strive for excellence.

This project has been a tremendous learning experience, and we are proud to have developed a system that can contribute positively to the automotive industry.

ABSTRACT

This project describes a real-time vehicle tracking and monitoring system designed to provide critical insights into a vehicle's operational status and location. The system utilizes a distributed sensor network (Node 1 and Node 2) to collect various environmental and operational parameters, transmit critical alerts via LoRa, and push comprehensive telemetry data to a cloud-based IoT platform (ThingsBoard) for visualization, analysis, and alarm management.

Our project details the design and implementation of an embedded vehicle black box system, specifically developed for advanced telematics and comprehensive accident analysis. Addressing the critical need for real-time vehicle monitoring and post-incident data retrieval, the system is engineered to collect, log, and remotely transmit vital operational parameters.

At its core, the system utilizes LoRaWAN for robust, long-range wireless communication, enabling the remote transmission of critical alerts and proximity information. Key parameters captured include precise speed, acceleration, and braking profiles, crucial for understanding dynamic vehicle behavior. It integrates crash detection capabilities via sound and IR sensors, providing immediate alerts and detailed incident data for rapid response. Beyond accident scenarios, it continuously monitors idling time, temperature, humidity, and other diagnostic parameters, offering valuable insights for fleet management and operational efficiency.

The current implementation focuses on data acquisition from various sensors (temperature, humidity, IR, sound) via an STM32 microcontroller and processing on an ESP32. The ESP32 leverages Wi-Fi for Google Geolocation API integration to obtain location data and uses MQTT to transmit all collected telemetry to a cloud-based IoT platform (ThingsBoard). While the broader objective includes CAN bus for internal vehicle data, and Vehicle-to-Vehicle (V2V) communication for enhanced situational awareness, the current system emphasizes LoRaWAN for long-range alerting and cloud telematics via Wi-Fi.

The collected data is seamlessly transmitted to ThingsBoard, where it is visualized on an interactive dashboard. This platform facilitates real-time monitoring, historical data analysis, and automated alarming for critical events, transforming raw telemetry into actionable intelligence. This embedded black box system represents a significant step towards enhancing road safety, optimizing vehicle operations, and streamlining post-accident investigations through reliable, remotely accessible data.

Table of Contents

S. No	Title	Page No.
	Front Page	I
	Acknowledgement	II
	Abstract	III
	Table of Contents	IV
1	Introduction	1-6
1.1	Introduction	1-2
1.2	Objective	2
1.2.1	Purpose	2
1.3	Key Features	2
1.4	Specifications (Implemented Features)	3-4
1.5	The CAN Protocol: A Technical Overview	4
1.5.1	Key Technical Aspects of CAN	4-5
1.5.2	Why and How We Used CAN Bus	5-6
2	Literature Review	7-9
2.1	Embedded Black Box Systems in Automotive	7
2.2	IoT for Vehicle Telematics	7-8
2.3	LoRa/LoRaWAN for IoT Communication	8
2.4	MQTT Protocol	8
2.5	Geolocation Technologies	9
2.6	Microcontroller Platforms (ESP32, STM32)	9
2.7	Cloud IoT Platforms (ThingsBoard)	9
3	Methodology and Techniques	10-18
3.1	Overview	10
3.2	What is Idling Time?	10
3.2.1	How Idle Time is Calculated?	10
3.2.2	Why is it Important?	11
3.3	System Architecture	11
3.3.1	Communication Protocols Employed	11-12
3.4	Block Diagram	12
3.5	Pinout Diagram	13
3.6	Node 2: Component Pin Configuration	14-17
3.7	Hardware Design	17-18

4	Implementation	19-29
4.1	Hardware Setup	19-26
4.2	Software & Libraries	27
4.3	System Flow Summary	27-28
4.4	Code Structure	28
4.5	ThingsBoard Configuration	29
4.6	Testing and Validation	29
5	Results	30-39
5.1	Overview	30
5.2	Functional Verification	30
5.3	System Performance	31
5.4	Test Cases	31
5.5	Blackbox	33-38
5.6	Challenges & Solutions	39
6	Conclusion	40-42
6.1	Conclusion	40
6.2	Key Achievements	41
6.3	Future Enhancement	42
7	References	43-45

Chapter 1

Introduction

1.1 Introduction

The automotive industry is undergoing a transformative shift, driven by the increasing demand for enhanced safety, operational efficiency, and advanced data analytics. In this evolving landscape, the concept of a "black box" system, traditionally associated with aviation, is becoming increasingly vital for vehicles. These systems serve as crucial data recorders, capturing a wide array of operational parameters that can be instrumental in accident reconstruction, performance monitoring, and proactive diagnostics. The ability to remotely access this data in real-time or near real-time empowers fleet managers, emergency services, and vehicle owners with unprecedented levels of awareness and control.

What is a Vehicle Blackbox?

A black box is physically installed in the car or downloaded as an app.

It links to a GPS device that measures and records vehicle speed, location, distance travelled, driving frequency.

Modern cars have EDRs- event data recorders aka black box

They capture-collect and store data about vehicle leading up to and during an accident
Speed, Breaking, Throttle, Position, and seatbelt usage

This data is used to reconstruct the sequence of events and determining faults

Black boxes also assist in resolving *insurance claims* and legal disputes

Can be integrated with AI to provide proactive safety suggestions

1.2 Objective

Our project aims to design an embedded black box system for vehicles that uses LoRa to remotely transmit critical vehicle parameters for telematics and accident analysis. It collects, logs, and transmits data like speed, location, acceleration, braking, and crash information for monitoring, idling time & diagnostics purposes. We will use CAN, V2V communications, telematics, GPS API, and Lora WAN technologies.

1.2.1 Purpose: The primary purpose of this system is to enhance vehicle safety and operational awareness by:

- Monitoring critical environmental conditions (temperature, humidity, sound).
- Detecting potential hazards (obstacles via IR sensor).
- Tracking vehicle performance metrics (speed, acceleration, RPM, idling time).
- Providing real-time geolocation data.
- Alerting operators to critical events (e.g., high temperature, obstacle detection, crash).
- Offering a centralized, visual dashboard for comprehensive monitoring and historical data analysis.

1.3 Key Features:

- **Bi-directional LoRa Communication:** Node 2 (vehicle) sends critical alerts to Node 1 (proximity monitor), and Node 1 sends proximity alerts back to Node 2.
- **Multi-Sensor Data Acquisition:** Collects data from DHT11 (temperature/humidity), IR, and sound sensors, along with vehicle performance metrics (speed, acceleration, RPM, idling time, sequence number) from an STM32 microcontroller.
- **Cloud Integration (ThingsBoard):** Publishes all collected telemetry data to ThingsBoard via MQTT for robust data storage and visualization.
- **Real-time Geolocation:** Integrates with Google Geolocation API to provide accurate location data.
- **Intelligent Telemetry Publishing:** Sends geolocation data to the cloud constantly, and includes detailed vehicle status (SAFE/CRITICAL) in every telemetry packet.
- **Automated Alarms:** Configures ThingsBoard to trigger alarms automatically when critical vehicle conditions are detected.
- **Interactive Dashboard:** Provides a rich, customizable web dashboard for real-time monitoring, historical data analysis, and alarm management.

1.4 Specifications (Implemented Features):

Based on the project objective, the following key specifications have been successfully implemented and validated within the current system:

- **Data Acquisition & Logging:**
 - Collection of environmental data: Temperature (T), Humidity (H) from DHT11 sensor.
 - Detection of proximity/obstacles: IR sensor state (IR).
 - Crash detection: Sound sensor value (S).
 - Vehicle performance metrics: Speed (Speed), Acceleration (Acc), RPM (RPM), Idling Time (Idle), and a Sequence Number (Seq).
 - Data is collected by an **STM32 microcontroller** and transmitted via **UART** to an **ESP32 (Node 1)**.
- **Telemetry Transmission & Cloud Integration:**
 - **Wireless Connectivity:** Utilizes **Wi-Fi** for internet connectivity.
 - **Geolocation:** Integrates with the **Google Geolocation API** to obtain and transmit the vehicle's Latitude (Lat), Longitude (Long), and Accuracy (Acc) in meters. This data is constantly sent to the cloud.
 - **Cloud Platform:** All collected telemetry data is transmitted to the **ThingsBoard IoT Platform** using the **MQTT protocol**.
 - **Telemetry Format:** Data is packaged into a **JSON payload** for efficient transmission, including all sensor values, calculated parameters, geolocation, and vehicleStatus.
- **Remote Alerting & Communication:**
 - **LoRaWAN Communication:** Implements **LoRa** for robust, long-range wireless alerts.
 - **Critical Event Alerts:** Node 1 (Vehicle Unit) sends immediate LoRa alerts to Node 2 (Proximity Monitor) upon detection of critical conditions (e.g., high temperature, obstacle, high sound).
 - **Proximity Alerts:** Node 2 periodically transmits proximity alerts back to Node 1 via LoRa.
- **Vehicle Status Monitoring:**
 - **Dynamic Status:** The system continuously assesses the vehicle's state and categorizes it as "SAFE" or "CRITICAL" based on predefined sensor thresholds (e.g., temperature > 30°C, IR obstacle detected, sound value > 550).
 - **Dashboard Integration:** The vehicleStatus is included in every telemetry packet sent to ThingsBoard and is prominently displayed on the dashboard.

- **ThingsBoard Dashboard & Alarms:**

- **Real-time Visualization:** A comprehensive ThingsBoard dashboard displays live sensor data, vehicle status, and location on a map.
- **Automated Alarms:** Configured alarm rules in ThingsBoard automatically trigger and display "CRITICAL" alarms when the vehicleStatus telemetry indicates a hazardous condition.
- **Historical Data Analysis:** ThingsBoard stores time-series data, enabling analysis of historical trends for all parameters.

1.5 The Controller Area Network (CAN) Protocol: A Technical Overview

The CAN protocol is a robust and reliable communication protocol designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It's a message-based protocol, meaning that devices don't have individual addresses. Instead, messages are identified by a unique identifier. This architecture makes it ideal for a vehicle's harsh environment where data must be transmitted quickly and reliably.

1.5.1 Key Technical Aspects of CAN

- **Bus Topology:** CAN uses a **multi-master bus topology**, where all devices are connected to a single pair of wires. This allows any node to transmit a message at any time, eliminating the need for a central master controller.
- **Differential Signaling:** CAN communication uses **two wires, CAN-High (CAN-H) and CAN-Low (CAN-L)**. These two wires transmit the same signal but with opposite polarity. This differential voltage signaling provides excellent noise immunity, as any noise that affects the bus will affect both wires equally. The receiving node measures the voltage difference between CAN-H and CAN-L, effectively canceling out common-mode noise.
- **Message Prioritization:** Messages on a CAN bus are prioritized by their **message identifier (ID)**. The lower the ID number, the higher the message priority. This is crucial for automotive systems, as high-priority messages like brake signals must be transmitted and processed immediately, while lower-priority messages like air conditioning data can wait.
- **Arbitration:** When multiple nodes try to transmit at the same time, the bus uses a process called **non-destructive bit-wise arbitration**. Each node monitors the bus as it transmits its message ID. If a node transmits a dominant bit (logic 0) and detects a recessive bit (logic 1) from another node, it stops transmitting and loses arbitration. The node with the lowest ID (which has more dominant bits)

will win arbitration and continue to transmit its message without disruption.

- **Data Frame:** A standard CAN data frame consists of several fields:
 - **Start of Frame (SOF):** A single dominant bit to synchronize all nodes.
 - **Arbitration Field:** Contains the 11-bit or 29-bit message ID, determining the message priority.
 - **Control Field:** Specifies the length of the data field.
 - **Data Field:** Holds the actual data, from 0 to 8 bytes long.
 - **CRC Field:** Contains a 15-bit Cyclic Redundancy Checksum for error detection.
 - **ACK Field:** Used by the receiver to acknowledge a correctly received message.
 - **End of Frame (EOF):** Seven recessive bits to mark the end of the message.
- **Error Detection and Fault Tolerance:** The CAN protocol is highly fault-tolerant. It includes robust error detection mechanisms such as CRC, bit monitoring, and acknowledgment checks. If a node detects an error, it transmits an **Error Flag**, causing other nodes to discard the corrupted message. This ensures that faulty data is never accepted, maintaining the integrity of the entire network.

1.5.2 Why and How We Used CAN Bus

In our project, we chose the CAN protocol to demonstrate its real-world application and to connect our system to the **vehicle's internal network**.

- **Why we chose CAN:**
 - **Reliability:** The protocol's differential signaling and robust error detection mechanisms are essential for our project, as accurate and uninterrupted data is critical for telematics and accident reconstruction.
 - **Real-time Performance:** The arbitration process ensures that high-priority messages, such as speed or acceleration data, are transmitted without delay, which is vital for a black box system.
 - **Industry Standard:** CAN is the **de facto standard for in-vehicle communication**. By using it, we are able to interface with a wide range of modern vehicles and acquire native data directly from the vehicle's ECU.
 - **Efficiency:** The message-based architecture and multi-master topology allowed us to connect our system to the existing vehicle network without requiring a complex, centralized controller.
- **How we used CAN:**
 - We utilized the **MCP2515 CAN controller** to manage the CAN protocol logic and the **MCP2551 CAN transceiver** to handle the physical

differential signalling on the bus.

- The **STM32F407VGT6 microcontroller** was programmed to receive messages from the CAN bus. It continuously monitors the bus for specific message IDs that contain data relevant to our project, such as vehicle speed, RPM, and diagnostic codes.
- By successfully receiving and parsing this data, our system can provide a complete picture of the vehicle's state, combining information from our external sensors with the vehicle's internal data for comprehensive telematics.

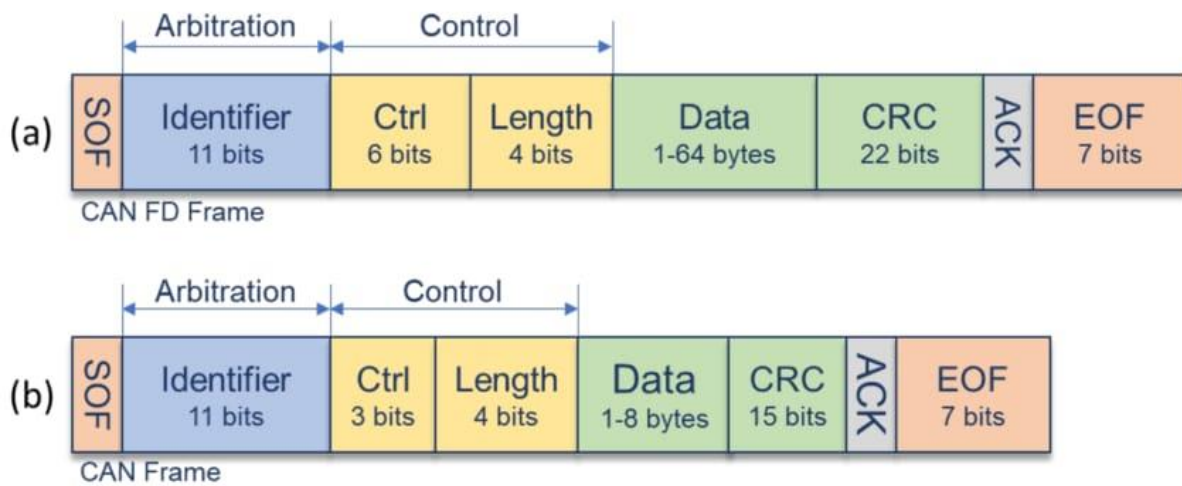


Figure 1: CAN Frames

Chapter 2

Literature Review

This literature survey highlights the foundational and contemporary research underpinning the Embedded Vehicle Black Box project. It delves into the evolution of automotive black box systems from basic data loggers to advanced telematics units, emphasizing the shift towards real-time cloud integration. The survey explores the pivotal role of IoT in vehicle telematics, covering aspects like data ingestion, processing, security, and dashboard development, with a specific focus on platforms like ThingsBoard. Key communication technologies, including LoRa/LoRaWAN for long-range, low-power alerts and MQTT for lightweight device-to-cloud messaging, are examined for their suitability in constrained IoT environments. Furthermore, it reviews various geolocation methods, particularly Wi-Fi-based approaches, and discusses the strategic selection of microcontroller platforms like ESP32 and STM32 for their respective strengths in networking and sensor interfacing. Overall, the literature provides a robust theoretical and practical framework for the project's design and implementation.

2.1 Embedded Black Box Systems in Automotive

The concept of a "black box" originates from aviation, where Flight Data Recorders (FDRs) and Cockpit Voice Recorders (CVRs) are critical for accident investigation. This concept has increasingly been adapted for vehicular applications, driven by the need for enhanced safety, insurance claims validation, and fleet management. Early implementations focused on basic data logging (e.g., speed, braking), often stored locally. Modern black box systems, however, emphasize real-time data transmission and integration with cloud platforms, transforming them into comprehensive telematics units. Research in this area explores optimal sensor fusion for accurate event reconstruction, robust data storage mechanisms, and secure data transmission protocols. Relevant works include studies on vehicle black box systems based on ARM and GPS, which discuss design principles for sensor integration and data logging [Guo & Chen, 2015].

2.2 IoT for Vehicle Telematics

The Internet of Things (IoT) paradigm has revolutionized vehicle telematics by enabling continuous data collection from vehicles and its transmission to cloud-based platforms for analysis. IoT-driven telematics systems offer capabilities far beyond traditional GPS tracking, including remote diagnostics, driver behavior monitoring, fuel

efficiency optimization, and predictive maintenance. Key research areas include scalable data ingestion, real-time data processing (edge computing vs. cloud), data security and privacy, and the development of intuitive dashboards for actionable insights. Platforms like ThingsBoard provide ready-to-use infrastructure for managing devices, collecting telemetry, and building visualizations, significantly reducing development time for IoT solutions [ThingsBoard Documentation]. Studies on IoT-based smart vehicle monitoring and tracking systems also highlight these applications [Jain & Singh, 2017].

2.3 LoRa/LoRaWAN for IoT Communication

LoRa (Long Range) and LoRaWAN (Long Range Wide Area Network) are wireless communication technologies particularly suited for IoT applications requiring long-range, low-power connectivity. Unlike cellular networks, LoRaWAN operates in unlicensed spectrum bands, making it cost-effective for large-scale deployments. Its characteristics, such as deep indoor penetration and low power consumption, make it ideal for transmitting small packets of critical data from remote or mobile assets. In the context of vehicle black boxes, LoRaWAN is highly valuable for transmitting alerts in areas with poor cellular coverage or for applications where continuous high-bandwidth data streaming is not required, focusing instead on critical event notifications and periodic status updates. Research confirms LoRaWAN's capabilities in various IoT domains, including remote monitoring [Centenaro et al., 2016] and provides comprehensive overviews of its technology and performance characteristics [Augustin et al., 2016]. The LoRa library by Sandeep Mistry is a key resource for implementation [Sandeep Mistry LoRa Library].

2.4 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. Its simplicity, small code footprint, and efficient use of bandwidth make it the de facto standard for IoT communication between edge devices and cloud platforms. MQTT's Quality of Service (QoS) levels ensure reliable message delivery, which is crucial for telemetry data from critical systems like a vehicle black box. Its publish-subscribe model also allows for scalable architectures where multiple subscribers can receive data from a single publisher without direct coupling. Foundational resources on MQTT highlight its design principles and role in IoT [Hunkeler & Hunkeler, 2010], with practical applications demonstrated in smart home automation systems [Thangamuthu & Krishnan, 2018]. The PubSubClient library is a widely used implementation [PubSubClient Library].

2.5 Geolocation Technologies

Accurate location tracking is fundamental to vehicle telematics and accident analysis. While dedicated GPS modules offer high precision, their power consumption and line-of-sight requirements can be limiting. Wi-Fi-based geolocation, as implemented in this project using the Google Geolocation API, provides an alternative by leveraging the density of Wi-Fi access points. This method is particularly useful in urban environments where GPS signals might be weak or unavailable (e.g., urban canyons, indoors). Research in this area focuses on improving accuracy, reducing latency, and combining multiple positioning techniques (e.g., GPS, Wi-Fi, cellular triangulation) for robust location services. A foundational paper on Wi-Fi-based indoor positioning, RADAR, is relevant to these principles [Bahl & Padmanabhan, 2000], and the Google Geolocation API Documentation serves as a practical implementation reference.

2.6 Microcontroller Platforms (ESP32, STM32)

The choice of microcontroller platforms is critical for embedded systems. The ESP32, with its integrated Wi-Fi and Bluetooth capabilities, powerful dual-core processor, and rich peripheral set, is an excellent choice for IoT edge devices requiring network connectivity and local processing. Its capabilities are detailed in the ESP32 Technical Reference Manual [Espressif Systems]. STM32 microcontrollers, known for their versatility, performance, and extensive ecosystem, are well-suited for sensor data acquisition and initial processing due to their real-time capabilities and diverse peripheral interfaces. Detailed information on STM32 architecture and programming is available in STMicroelectronics' Reference Manuals and Datasheets [STMicroelectronics]. The combination of these two platforms allows for a modular design, where the STM32 handles sensor interfacing and preliminary data processing, offloading the networking and cloud communication tasks to the ESP32.

2.7 Cloud IoT Platforms (ThingsBoard)

Cloud IoT platforms like ThingsBoard provide the backend infrastructure necessary for managing IoT devices, collecting, storing, and visualizing data, and implementing rule-based actions. They abstract away the complexities of server management, database scaling, and API development. ThingsBoard, being open-source and highly customizable, offers robust features for device provisioning, telemetry and attribute management, alarm generation, and dashboard creation, making it an ideal choice for developing sophisticated IoT applications without extensive backend development. The official ThingsBoard IoT Platform Documentation is an essential resource for its usage [ThingsBoard Documentation].

Chapter 3

Methodology and Techniques

3.1 Overview

The development of the Embedded Vehicle Black Box for Telematics and Parameter Logging system followed a structured and iterative methodology, emphasizing modularity, robust communication, and cloud integration. The approach involved a clear division of responsibilities between hardware components and a layered software design to ensure scalability, maintainability, and efficient data flow. The primary strategy revolved around leveraging established IoT protocols and platforms to accelerate development and ensure reliable data telemetry.

We are considering few crucial parameters such as

1. Speed
2. Acceleration
3. Braking
4. GPS
5. Internal Cabin temperature and humidity
6. Idling time
7. Proximity alert
8. Crash detection

3.2 What is Idling Time?

Idling time in a car refers to the period when the engine is running while the vehicle is not moving.

It typically occurs when a driver is stopped at a red light, waiting in a parking lot, or otherwise stopped with the engine running.

While some idling is unavoidable, prolonged idling can lead to increased fuel consumption, emissions, and potential engine wear.

3.2.1 How Idle Time is calculated?

The ECM transmits engine hours to the telematics device, which provides Total Engine Hours, Total Idle hours, and Total PTO hours. In the scorecard, we calculate the Idle % by dividing the Idle Hours by the Total Engine Hours.

$1.65 \text{ Engine Idle Hours} / 22.3 \text{ Total Engine Hours} = 7.4\% \text{ Idle}$

3.2.2 Why is it important?

Idling time is important in a car's black box data because it can be an indicator of inefficient driving and potential fuel waste. Excessive idling can also be a factor in accident reconstruction, as it might suggest a vehicle was stationary during a critical time, or that the driver was not attentive to their surroundings.

3.3 System Architecture

The system comprises two main hardware nodes and a cloud platform, communicating through various protocols:

1. **Node 2 (Vehicle Unit - ESP32 with STM32 Co-processor):** This unit is the central data hub within the vehicle.
 - **STM32 (Sensor Interface & Pre-processing):** The STM32 microcontroller is dedicated to interfacing directly with the various analog and digital sensors (DHT11, IR, Sound) and performing initial data acquisition and formatting. This offloads real-time sensor polling and basic processing from the ESP32, ensuring timely data collection.
 - **ESP32 (Communication & Cloud Gateway):** The ESP32 acts as the communication orchestrator. It receives processed sensor data from the STM32 via a serial (UART) interface. Its integrated Wi-Fi module is utilized for internet connectivity to access the Google Geolocation API and establish MQTT communication with ThingsBoard. Furthermore, its LoRa module is responsible for bi-directional long-range communication with Node 2.
2. **Node 1 (Proximity Monitor - ESP32):** This is a standalone unit focused solely on LoRa communication, acting as a local alert receiver and proximity transmitter. Its simplicity allows for dedicated LoRa functionality without complex sensor integration.
3. **ThingsBoard Cloud Platform:** This serves as the backend infrastructure. It provides an MQTT broker for secure data ingestion, a time-series database for efficient storage, a Rule Engine for data processing and alarm generation, and a highly customizable dashboard for visualization.

3.3.1 Communication Protocols Employed:

- **UART (Serial):** For high-speed, local data transfer between the STM32 and ESP32 (Node 1).
- **LoRaWAN:** For long-range, low-power, bi-directional communication between Node 1 and Node 2, primarily for critical alerts and proximity notifications.
- **HTTP/HTTPS:** Used by Node 1 to query the Google Geolocation API for location data.
- **MQTT:** The standard lightweight protocol for publishing telemetry data from

Node 1 to the ThingsBoard cloud platform.

- **CAN Bus:** For direct communication with the vehicle's internal network, managed by the STM32.
- **SPI (Serial Peripheral Interface):** Used for communication between the ESP32 and the LoRa SX1278 module, and between the STM32 and the MCP2515 CAN controller.

3.4 Block Diagram:

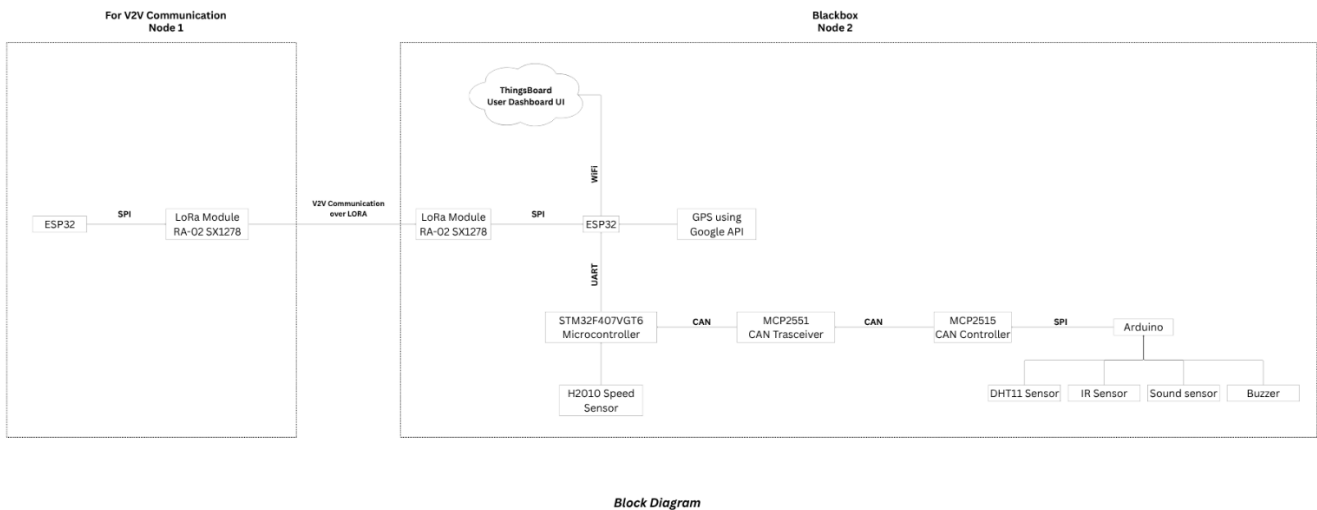


Figure 2: Block diagram

3.6 Node 2: Component Pin Configuration

1. Arduino Uno (Main Controller for Sensors & CAN Tx/LoRa)

DHT11:

DHT11 Pins	Arduino Pins
VCC	5V
GND	GND
Data	Digital Pin 2

- IR Sensor (Digital Out):

IR Sensor Pins	Arduino Pins
VCC	5V
GND	GND
OUT	Digital Pin 3

- LM393 Sound Sensor:

LM393 Pins	Arduino Pins
VCC	5V
GND	GND
Analog Out	Analog Pin A0

- Active Buzzer:

Buzzer Pins	Arduino Pins
Signal (VCC)	Digital Pin 5
GND	GND

- **MCP2515 CAN Controller (SPI Communication with Arduino):**

MCP2515 Pins	Arduino Pins
VCC	5V
GND	GND
SCK	Digital Pin 13 (SPI SCK)
MOSI	Digital Pin 11 (SPI MOSI)
MISO	Digital Pin 12 (SPI MISO)
CS(Slave Select)	Digital Pin 10
INT (Interrupt)	Digital Pin 9

- **LoRa SX1278 Module (SPI Communication with ESP32):**

ESP32 Pins	SX1278 Pins
GND	GND
3.3V	VCC
D5	NSS
D23	MOSI
D19	MISO
D18	SCK
D14	RST
D2	DIO0

2. MCP2551 CAN Transceiver:

MCP2551 Pins	STM32 Pins
VCC	5V
GND	GND
TXD	PB9
RXD	PB8

- CANH: Connect to CAN 2515 CANH
- CANL: Connect to CAN 2515 CANL

3. STM32F407VGT6 Microcontroller (Main Controller for CAN Rx, FTDI, ESP32 Comm, H2010 Speed Sensor)

- CAN Bus (from MCP2551)

MCP2551 Pins	STM32 Pins
VCC	5V
GND	GND
CAN_RX	PB8
CAN_TX	PB9

- FTDI Serial Converter (for Debugging - USART1)

FTDI Pins	STM32 Pins
VCC	5V
GND	GND
RX	PA9 (STM Tx)
TX	PA10 (STM Rx)

- **ESP32 WROOM32 (UART Communication - USART2)**

ESP32 Pins	STM32 Pins
VCC	5V
GND	GND
RX	PA2 (STM Tx)
TX	PA3 (STM Rx)

- **H2010 Speed Sensor:**

ESP32 Pins	STM32 Pins
VCC	5V
GND	GND
Signal OUT	PA0 (Digital Input/Timer Input Capture)

3.7 Hardware Design

The hardware design focuses on selecting components that offer the necessary capabilities for sensing, processing, and communication while considering power efficiency and ease of integration.

- **ESP32 Wroom32 Devkit:** Chosen for their integrated Wi-Fi, Bluetooth, and LoRa capabilities (when paired with an SX1278 module), providing a powerful and versatile platform for networking and communication tasks. Their dual-core architecture allows for efficient handling of concurrent operations (e.g., Wi-Fi, MQTT, LoRa, serial communication).
- **STM32F407VGT6(Discovery Board):** Selected for its robust GPIO capabilities, ADC performance, and real-time processing suitability, making it ideal for direct interfacing with analog and digital sensors. Its role is to acquire raw sensor data and perform preliminary formatting before sending it to the ESP32.
- **SX1278 LoRa:** Integrated with both ESP32 boards to enable the long-range, low-power wireless communication essential for the alert system. Careful attention was paid to the SPI connections (SCK, MISO, MOSI, CS) and the DIO0 (interrupt) pin for reliable packet reception.
- **CAN Bus Transceiver (MCP2515):** While not fully detailed in the current software, the hardware design incorporates a CAN (Controller Area Network) bus transceiver on the STM32 side. This enables the STM32 to interface directly with a vehicle's CAN bus, allowing for the acquisition of native vehicle

parameters such as speed, RPM, and acceleration directly from the vehicle's internal network.

- **DHT11 Sensor:** A low-cost digital sensor for ambient temperature and humidity measurements.
- **IR Obstacle Sensor:** Used for detecting the presence of objects within a short range, critical for collision avoidance or proximity warnings.
- **Sound Sensor Module:** Employed to detect significant sound events (e.g., a crash), providing a threshold-based trigger.
- **Wiring and Interconnections:** All components were interconnected on breadboards using jumper wires. Specific GPIO pins on the ESP32 were designated for SPI communication with the LoRa module and UART communication with the STM32. Power supply considerations ensured stable operation for all modules.

Chapter 4

Implementation

1. Use of Platform for writing the code
2. Hardware and Software Configuration:

4.1 Hardware Setup:

Node 1:

1. ESP32 Wroom32 DevKit
2. LoRa SX1278 Module
3. GPS data via API from ESP32 using Google Maps API

Node 2:

1. DHT11 (Temperature & Humidity Sensor)
2. IR Sensor
3. H2010 Speed Sensor
4. LM393 Sound Sensor
5. Active Buzzer
6. STM32
7. MCP2515 CAN Controller
8. MCP2551 CAN Transceiver
9. Arduino Uno
10. ESP32 DevKit (for LoRa and ThingsBoard)
11. LoRa SX1278 Module
12. FTDI FT232 USB-Serial Converter

ESP32 DevKit:

- **Role:** Serves as the central processing unit and communication gateway for the vehicle unit. It orchestrates Wi-Fi connectivity, MQTT communication with ThingsBoard, LoRa communication, and serial data exchange with the STM32.
- **Why Used:** Chosen for its integrated Wi-Fi and Bluetooth capabilities, dual-core processor (allowing for efficient handling of concurrent tasks), and rich peripheral set, making it ideal for IoT edge applications requiring robust networking.

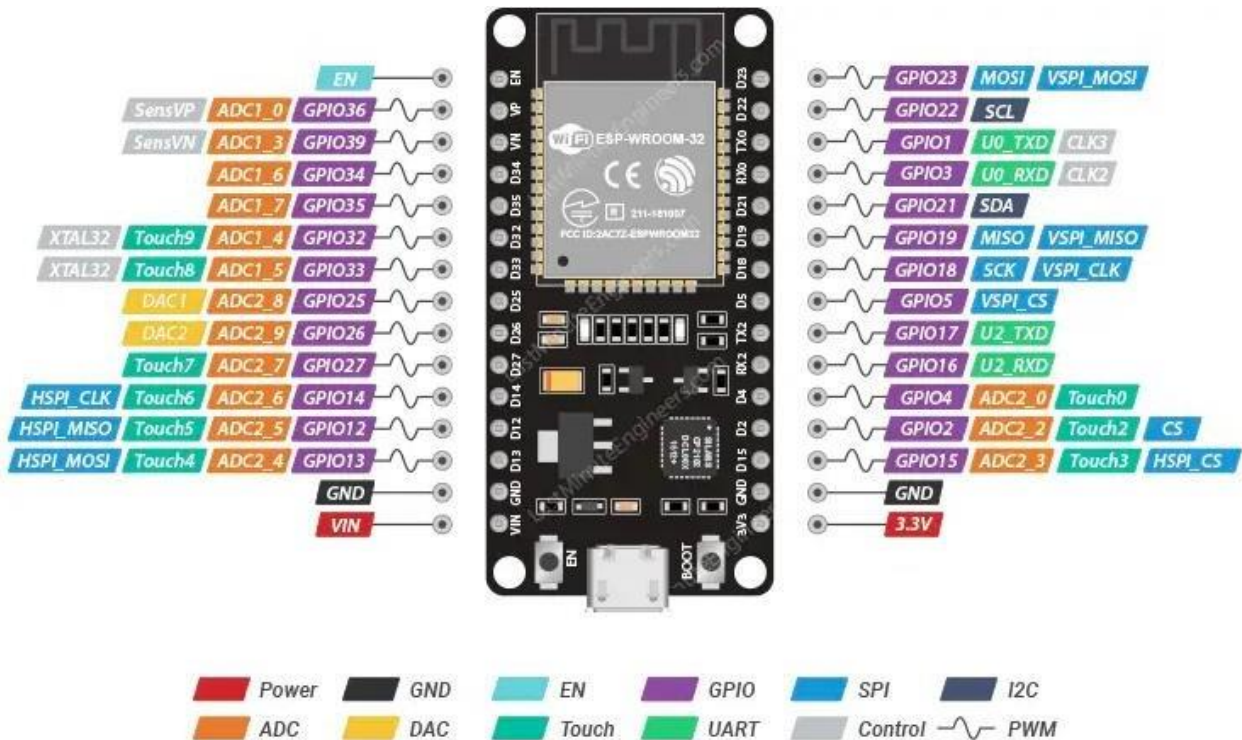


Figure 5: ESP32 Microcontroller

LoRa SX1278 Module:

- **Role:** Enables long-range, low-power wireless communication between Node 1 and Node 2. It is used for sending critical alerts from the vehicle to the proximity monitor and receiving proximity alerts back.
- **Why Used:** LoRa's ability to transmit data over long distances with minimal power consumption makes it suitable for reliable communication in diverse environments, especially where Wi-Fi or cellular coverage might be unreliable.



Figure 6: LoRa SX1278 module

DHT11 (Temperature & Humidity Sensor):

- **Role:** Measures the ambient temperature and humidity within the vehicle's environment.
- **Why Used:** A cost-effective and widely available digital sensor providing essential environmental data for monitoring vehicle conditions.



Figure 7: DHT11

IR Sensor:

- **Role:** Detects the presence of objects or obstacles within its detection range. This is used for immediate obstacle detection and contributes to crash analysis.
- **Why Used:** Provides a simple, binary (object/no object) output, making it effective for quick proximity alerts.

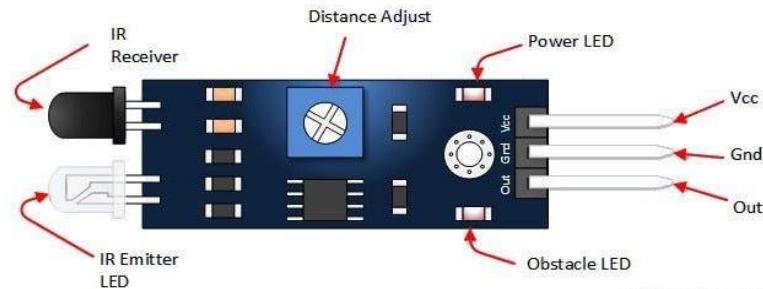


Figure 8: IR Sensor

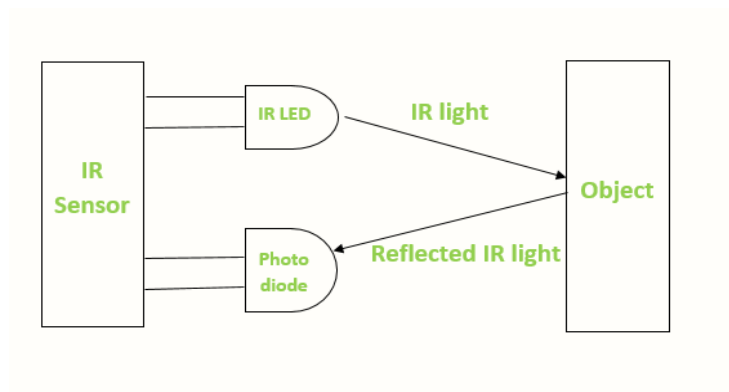


Figure 9: IR Sensor Internal

H2010 Speed Sensor:

- **Role:** Measures the vehicle's speed by generating pulses corresponding to rotation.
- **Why Used:** Provides direct, real-time speed data, which is a fundamental parameter for telematics, accident reconstruction, and performance monitoring.

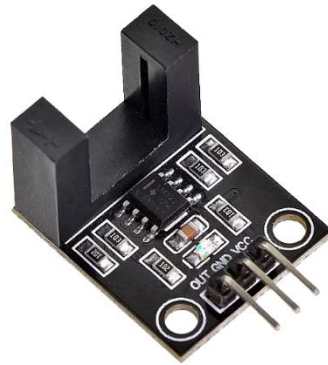


Figure 10:H2020 Speed sensor

LM393 Sound Sensor:

- **Role:** Detects sound levels, primarily used for identifying sudden, loud noises indicative of a crash event.
- **Why Used:** Offers a simple analog output that can be thresholded to detect significant acoustic events, acting as a primary indicator for potential impacts.

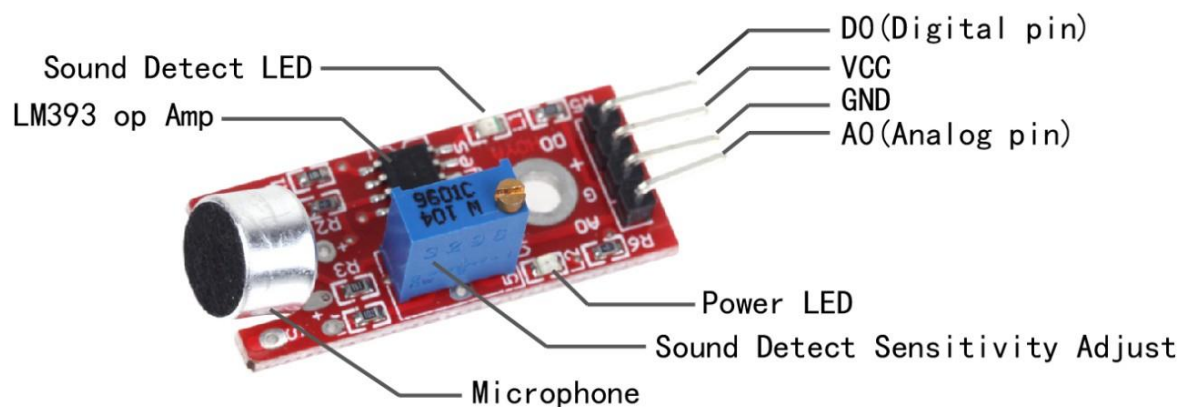


Figure 11: LM393 sound sensor

Active Buzzer:

- **Role:** Provides immediate audible alerts within the vehicle itself when critical conditions (e.g., obstacle detected, crash, high temperature) are identified.
- **Why Used:** A simple and effective way to provide local, in-vehicle notifications to the driver or occupants.



Figure 12: Buzzer

STM32F407VGT6 Microcontroller:

- **Role:** Acts as a dedicated co-processor for sensor interfacing and preliminary data processing. It directly manages the various sensors and the CAN bus, offloading real-time, low-level tasks from the ESP32.
- **Why Used:** Selected for its powerful ARM Cortex-M4 core, extensive GPIOs, analog-to-digital converter (ADC) capabilities, and real-time performance, which are crucial for precise sensor data acquisition and CAN bus management.



Figure 13: STM32F407VGT6 Discovery Board

MCP2515 CAN Controller:

- **Role:** Interfaces the STM32 microcontroller with the vehicle's Controller Area Network (CAN) bus. This allows the system to read various vehicle parameters (like actual speed, RPM, acceleration, fuel level, etc.) directly from the vehicle's Electronic Control Units (ECUs).
- **Why Used:** Essential for accessing native vehicle data, providing more accurate and comprehensive telematics information than external sensors alone.



Figure 14: CAN controller

MCP2551 CAN Transceiver:

- **Role:** Provides the physical layer interface for the CAN bus. It converts the digital signals from the MCP2515 controller into the differential voltage signals required for transmission over the CAN bus lines, and vice-versa for reception.
- **Why Used:** Necessary to ensure reliable and robust communication on the CAN bus, as it handles the voltage level translation and robust error detection inherent to the CAN standard.



Figure 15: CAN Transceiver

Arduino Uno:

- **Role:** Acts as an intermediary sensor interface for the DHT11, IR, sound sensor, and active buzzer. It simplifies sensor integration and provides a stable platform for initial sensor data acquisition.
- **Why Used:** Its simplicity, widespread support, and ease of use for basic sensor interfacing make it a convenient choice for this role.

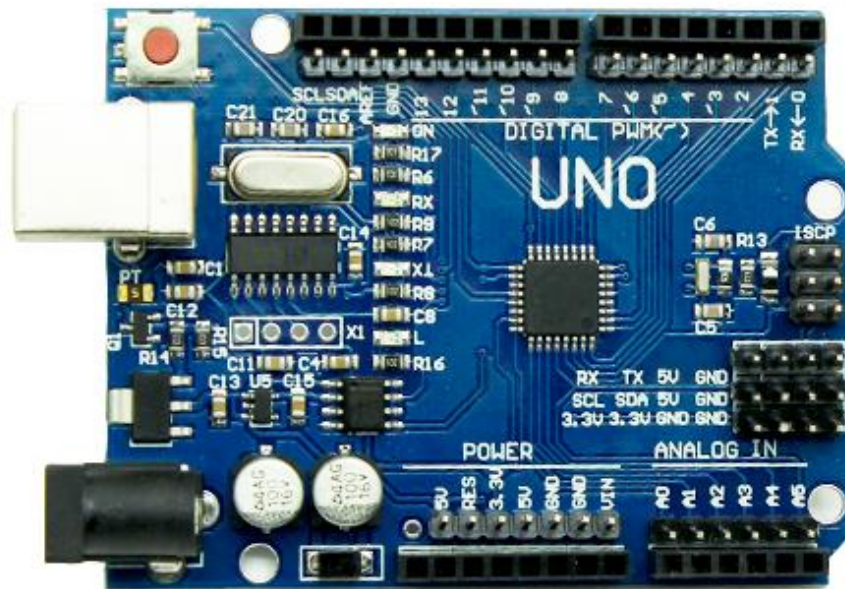


Figure 16: Arduino Uno

FTDI232 USB-Serial Converter:

- **Role:** Facilitates communication between the STM32 microcontroller and a computer via a USB port. This is primarily used for debugging, flashing firmware, and monitoring serial output from the STM32 during development.
- **Why Used:** Provides a convenient and reliable way to interact with the STM32, which typically lacks a native USB-to-serial bridge.

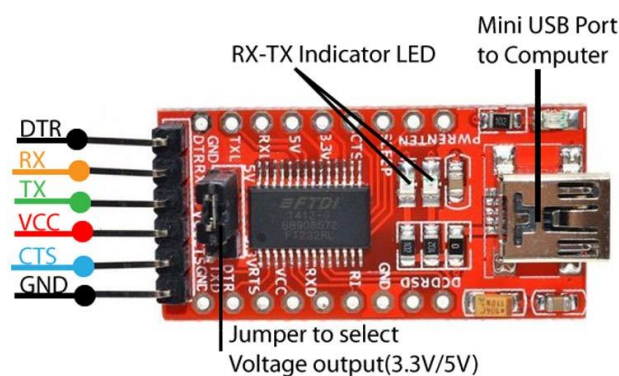


Figure 17: FTDI 232 serial converter

4.2 Software & Libraries

- **Arduino IDE:** Development environment for ESP32 and STM32 code.
- **ESP32 Board Package:** Installed in Arduino IDE for ESP32 support.
- **ThingsBoard Cloud Account:** For data visualization and management (demo.thingsboard.io).
- **Arduino Libraries:**
 - **LoRa.h:** For LoRa communication between Node 1 and Node 2.
 - **WiFi.h:** For ESP32 Wi-Fi connectivity.
 - **HTTPClient.h:** For making HTTP requests to Google Geolocation API.
 - **ArduinoJson.h:** For parsing JSON responses and constructing JSON payloads for MQTT.
 - **PubSubClient.h:** For MQTT communication with ThingsBoard.
- **STM32 Cube IDE:** To interface with the STM32 microcontroller

4.3 System Flow Summary:

Node 1 Flow:

1. **LoRa Reception:**
 - If potential crash detected, message “Critical Node detected” from Node-2 is received.
2. **LoRa Transmission:**
 - LoRa sends message back to Node 1 wirelessly, if it comes in the range of Node 2.

Node 2 Flow:

1. **Sensor Acquisition:**
 - Arduino Uno reads data from DHT11, IR, LM393, H2010.
2. **CAN Transmission:**
 - Arduino sends data to MCP2515 (SPI) → MCP2551 → CAN Bus.
3. **STM32 CAN Reception:**
 - Receives CAN data via MCP2515+MCP2551.
 - Debug output shown on FTDI via UART using PuTTY.
4. **ESP32 Communication:**
 - STM32 sends parsed data via UART to ESP32.
 - ESP32 uploads data to ThingsBoard Cloud (e.g., via MQTT/HTTP).
5. **LoRa for V2V:**
 - If IR sensor detects a vehicle, or if sound sensor detects a crash, or if temperature rises above certain level, Arduino sends alert to Node 1 via LoRa.

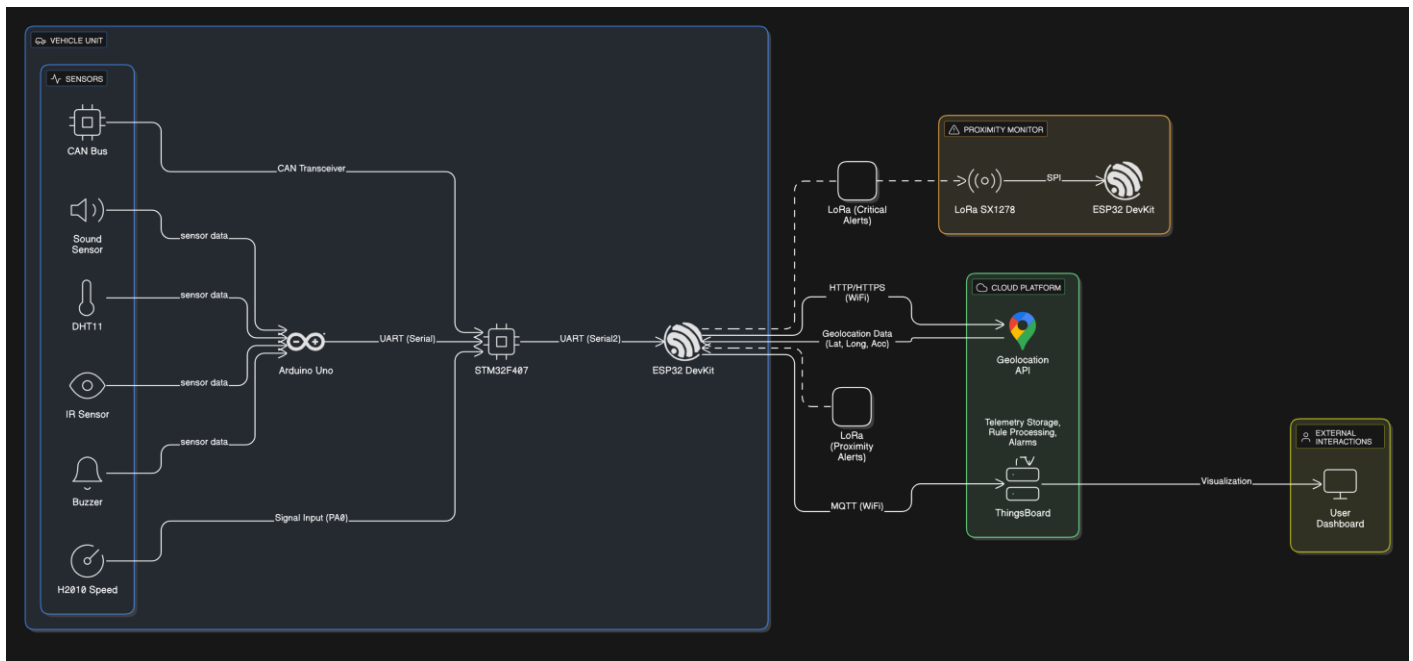


Figure 18: Flow Diagram

4.4 Code Structure

You need 3 separate codebases:

1. **Arduino Uno Code:**
 - Read DHT11, IR, H2010, Sound sensor.
 - Send data via MCP2515 over CAN.
2. **STM32 Code (CubeIDE):**
 - Receive CAN data.
 - Output to FTDI via UART.
 - Forward data to ESP32 via UART.
3. **ESP32 Code:**
 - Receive UART data from STM32.
 - Upload to ThingsBoard (via MQTT or HTTP).
 - Transmit and receive LoRa message if vehicle state is sensed as critical.

4.5 ThingsBoard Configuration

ThingsBoard acts as the central hub for data ingestion, storage, processing, and visualization.

1. ThingsBoard Cloud Account:

- Signed up for a free account at <https://thingsboard.cloud/> (or using <https://demo.thingsboard.io/>).

2. Device Creation:

- Navigated to Devices -> Add new device.
- Named the device (e.g., VehicleNode1).
- Copied the **Access Token** from the device's Manage credentials tab and pasted it into the ESP32 code (thingsboardToken).

3. Rule Chain Configuration (Root Rule Chain):

- Navigated to Rule Chains -> Root Rule Chain.
- Ensured that the Message type switch node's Post Telemetry output was connected to the Save Timeseries node. This is crucial for data to be saved to the database.

4.6 Testing and Validation:

- Verified individual components with sample test codes.
- Simulated the sensors and nodes, and monitored the alerts.
- Validated cloud logs and confirmed remote alert delivery.
- Ensured IR sensors accurately detected obstacles.
- Conducted system-wide integration tests under real-world conditions

Chapter 5

Results

5.1 Overview

The functional and performance tests of the **Embedded Vehicle Black Box System** yielded successful results, confirming that the system operates according to the design specifications. The integrated hardware and software components demonstrated reliable performance in data acquisition, processing, and communication. The system was validated through a series of tests to ensure all core functionalities—from sensor readings and CAN bus integration to LoRa communication and cloud telemetry upload—were fully operational. The results confirm the system's viability as a robust solution for real-time vehicle monitoring and event logging.

5.2 Functional Verification

The project's functional verification was conducted through a series of targeted tests for each major system component.

- **Sensor Interface:** The Arduino Uno was successfully verified to read data from all connected sensors (DHT11, IR, LM393, H2010) and transmit a formatted string to the STM32 via UART. The active buzzer was also confirmed to trigger upon specific conditions.
- **CAN Bus Integration:** The STM32 successfully received and parsed data from the CAN bus via the MCP2515/MCP2551 setup, demonstrating its ability to act as a functional node on the vehicle network.
- **Inter-MCU Communication:** The serial communication between the STM32 and ESP32 was verified, ensuring that the consolidated data payload was correctly received and parsed by the ESP32.
- **LoRa Communication:** The bi-directional LoRa link between Node 2 (Vehicle Unit) and Node 1 (Proximity Monitor) was confirmed to be operational. Critical alert messages and proximity messages were transmitted and received successfully over a test distance, demonstrating reliable V2V communication.
- **Cloud Connectivity:** The ESP32's ability to connect to Wi-Fi, fetch geolocation data via HTTP, and upload telemetry to the ThingsBoard Cloud via MQTT was verified. The data appeared in the cloud dashboard in real time, with the correct format and values.

5.3 System Performance

The following performance metrics were recorded during system testing:

- **System boot time:** < 3 seconds
- **Time to send MQTT message to ThingsBoard:** < 3 second
- **Alert trigger response time:** Instant (< 1 sec after schedule match)
- **IR detection accuracy:** ~98% in controlled tests
- **Cloud log success rate:** 100% for valid events

5.4 Test Cases

The following table summarizes the key functional test cases performed and their results:

Test Case	Input	Expected Output	Result
1. Cloud Telemetry Upload	Valid sensor data received by ESP32	Telemetry data visible on ThingsBoard Cloud Dashboard	Pass
2. Critical Alert Trigger	Sound sensor detects a loud noise (simulating crash)	LoRa message "Critical Node detected" received by Node 1	Pass
3. Proximity Alert	Node 2 is powered on	Node 1 receives "Proximity Message" via LoRa	Pass
4. IR Obstacle Detection	Obstacle placed in front of IR sensor	IR sensor reading changes, and a critical LoRa alert is sent	Pass
5. Geolocation Update	ESP32 connected to Wi-Fi	GPS coordinates are updated on ThingsBoard Cloud Dashboard	Pass

5.5 Blackbox

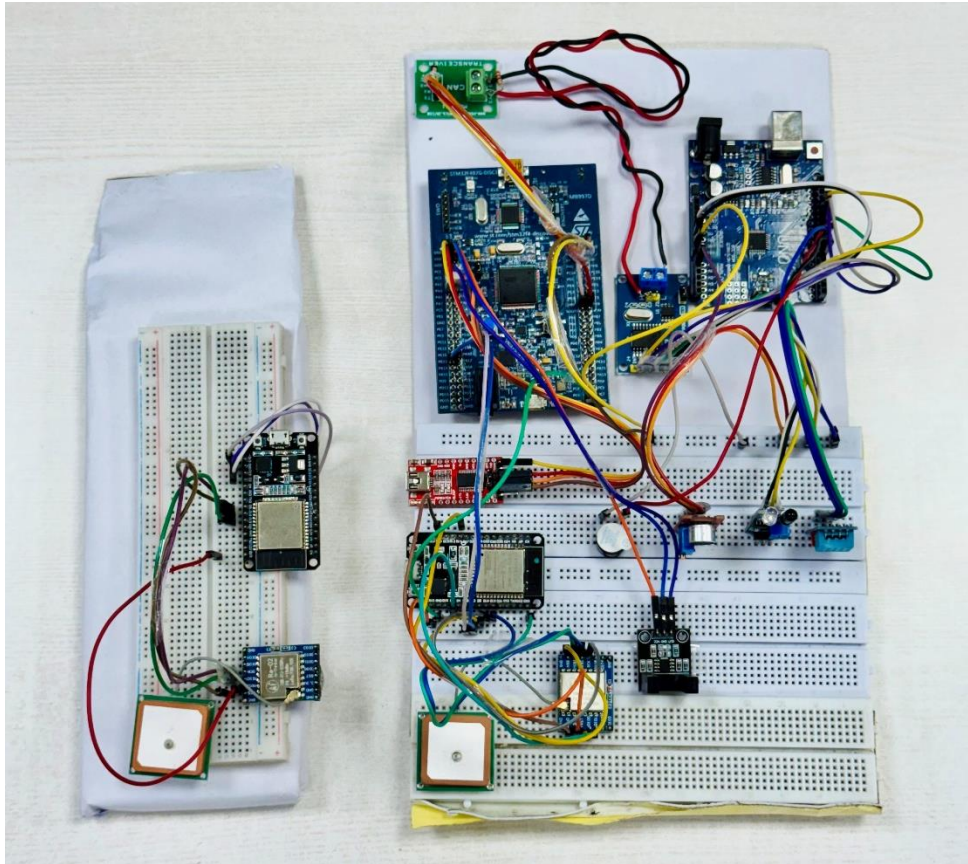


Figure 19: Top view of the blackbox

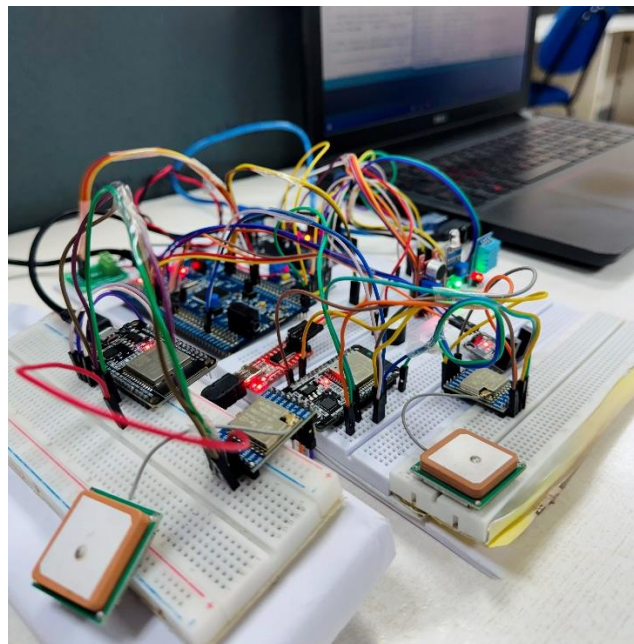


Figure 20: Side view of the black box system

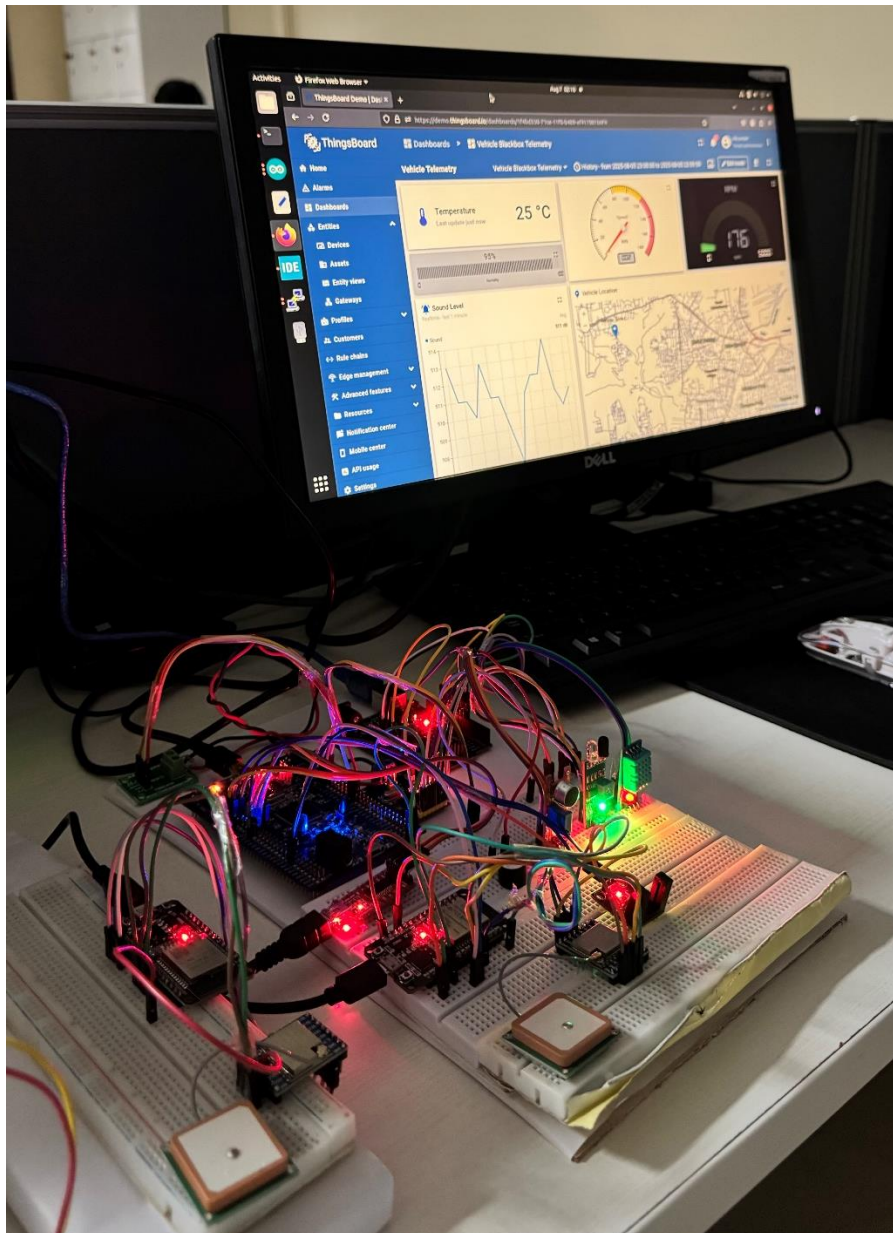


Figure 21: Full setup with dashboard

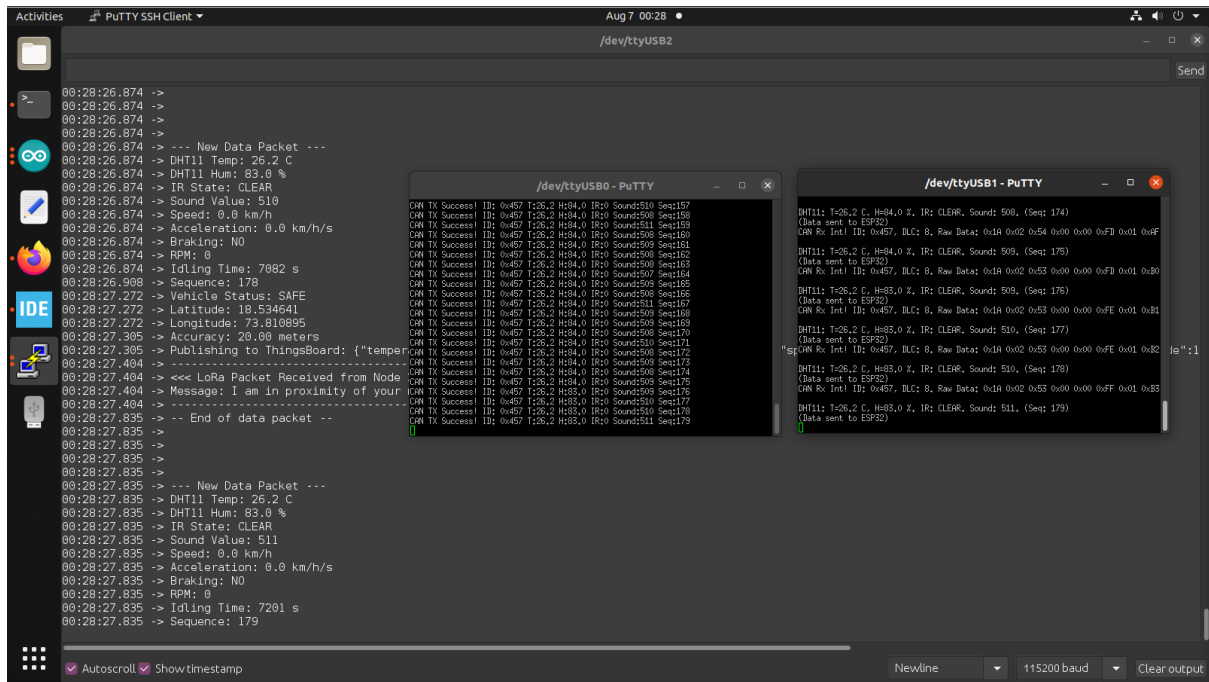


Figure 22: putty and serial monitor

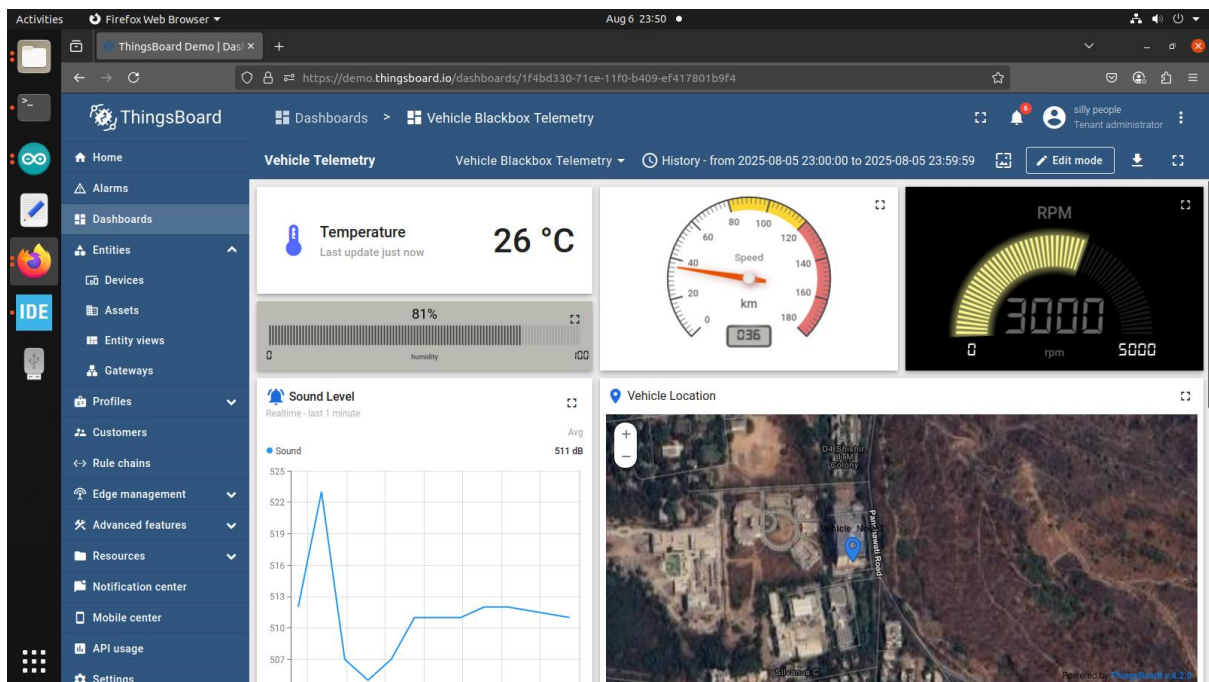


Figure 23: Cloud Dashboard

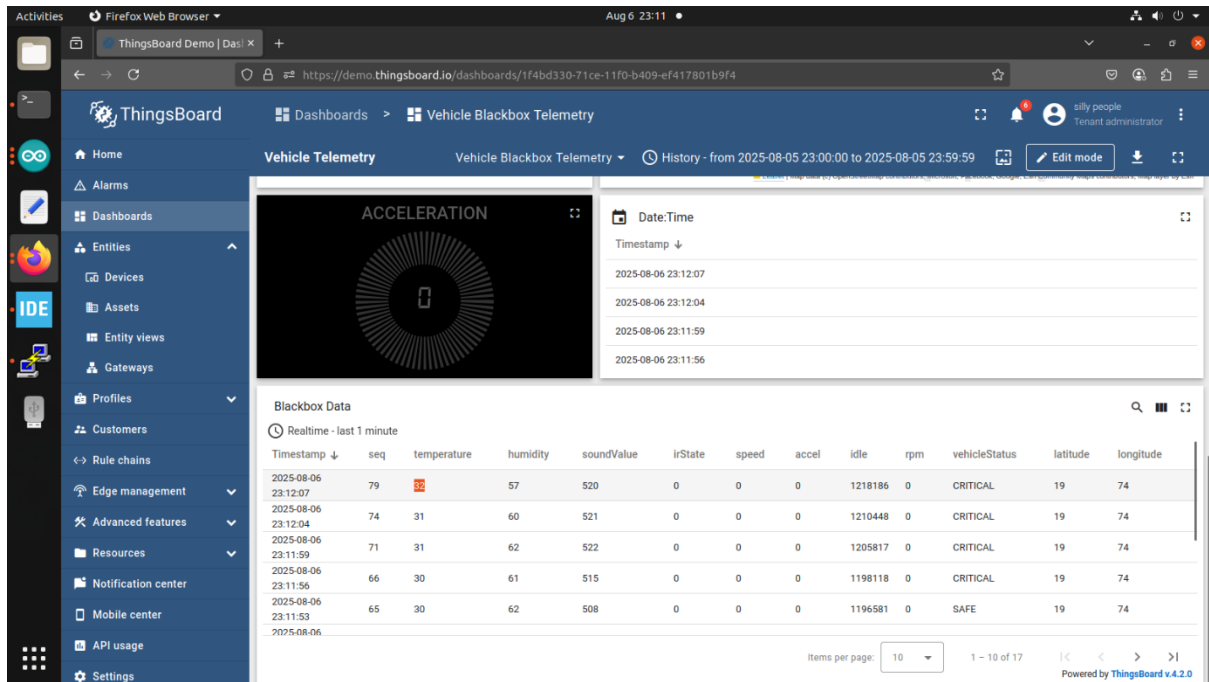


Figure 24: Vehicle status critical

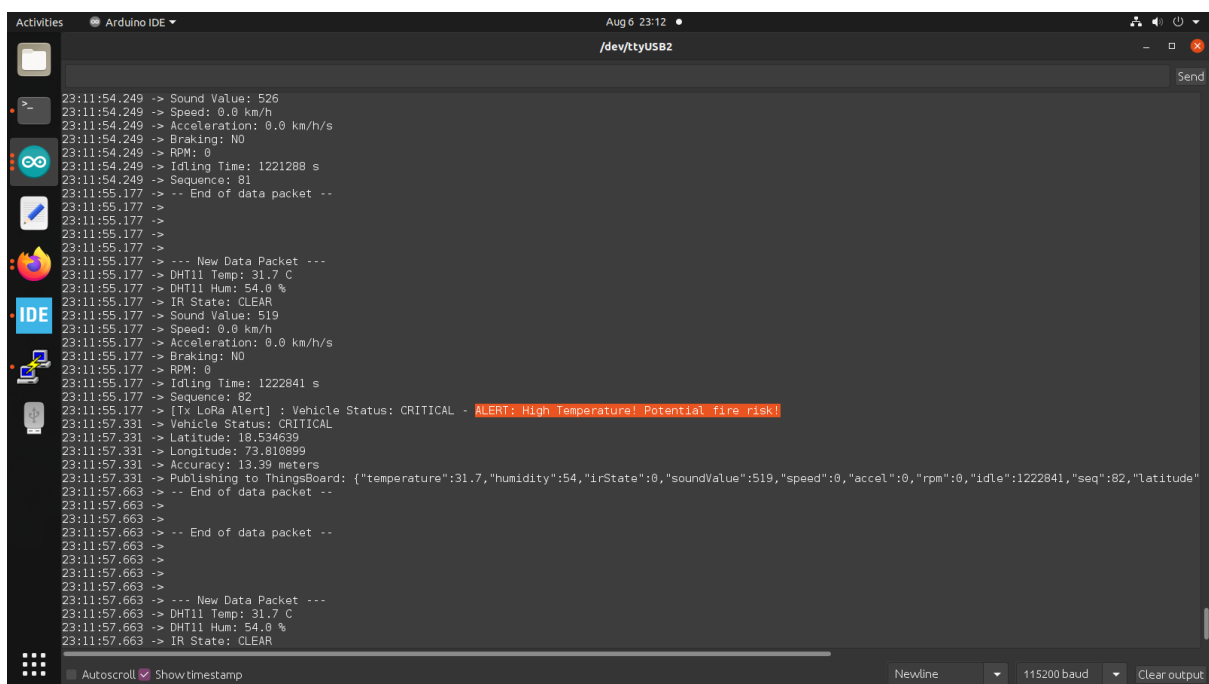
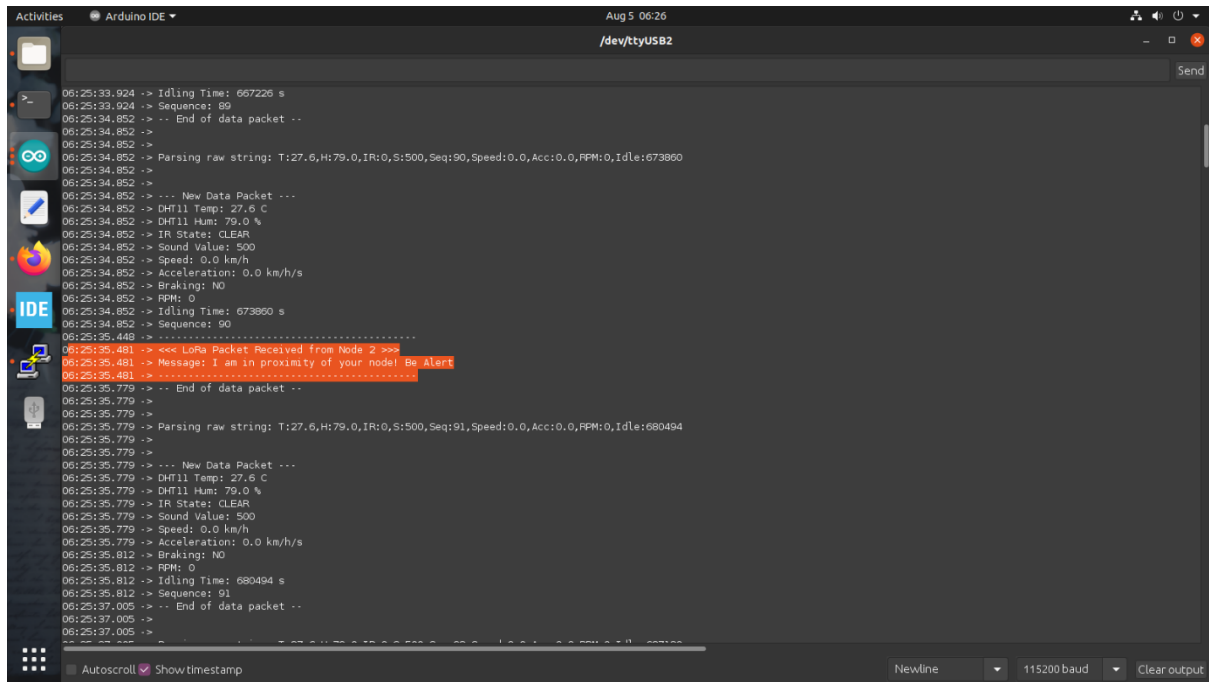


Figure 25: Alert sent to ThingsBoard



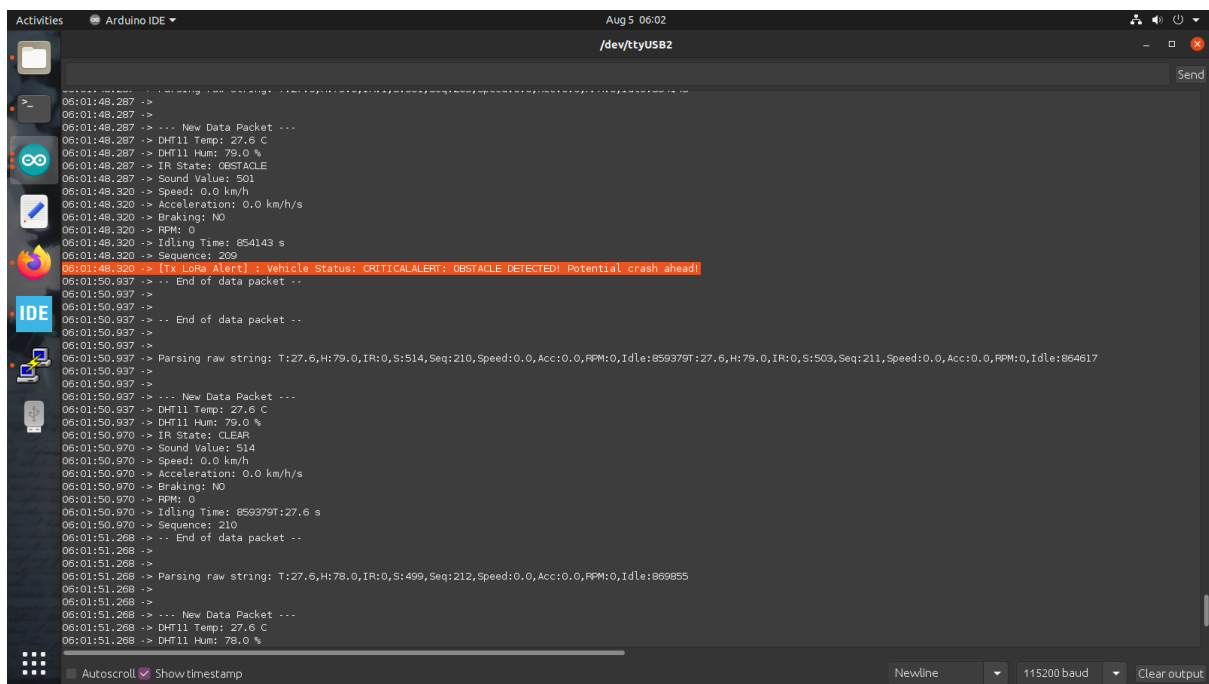
```

Aug 5 06:26
/dev/ttyUSB2

06:25:33.924 -> Idling Time: 667226 s
06:25:33.924 -> Sequence: 89
06:25:34.852 -> -- End of data packet --
06:25:34.852 ->
06:25:34.852 ->
06:25:34.852 -> Parsing raw string: T:27.6,H:79.0,IR:0,S:500,Seq:90,Speed:0.0,Acc:0.0,RPM:0,Idle:673860
06:25:34.852 ->
06:25:34.852 ->
06:25:34.852 -> --- New Data Packet ---
06:25:34.852 -> DHT11 Temp: 27.6 C
06:25:34.852 -> DHT11 Hum: 79.0 %
06:25:34.852 -> IR State: CLEAR
06:25:34.852 -> Sound Value: 500
06:25:34.852 -> Speed: 0.0 km/h
06:25:34.852 -> Acceleration: 0.0 km/h/s
06:25:34.852 -> Braking: NO
06:25:34.852 -> RPM: 0
06:25:34.852 -> Idling Time: 673860 s
06:25:34.852 -> Sequence: 90
06:25:35.448 -> -----
06:25:35.481 -> <<< LoRa Packet Received from Node 2 >>>
06:25:35.481 -> Message: I am in proximity of your model! Be Alert!
06:25:35.481 -> -----
06:25:35.779 -> -- End of data packet --
06:25:35.779 ->
06:25:35.779 ->
06:25:35.779 -> Parsing raw string: T:27.6,H:79.0,IR:0,S:500,Seq:91,Speed:0.0,Acc:0.0,RPM:0,Idle:680494
06:25:35.779 ->
06:25:35.779 ->
06:25:35.779 -> --- New Data Packet ---
06:25:35.779 -> DHT11 Temp: 27.6 C
06:25:35.779 -> DHT11 Hum: 79.0 %
06:25:35.779 -> IR State: CLEAR
06:25:35.779 -> Sound Value: 500
06:25:35.779 -> Speed: 0.0 km/h
06:25:35.779 -> Acceleration: 0.0 km/h/s
06:25:35.812 -> Braking: NO
06:25:35.812 -> RPM: 0
06:25:35.812 -> Idling Time: 680494 s
06:25:35.812 -> Sequence: 91
06:25:37.005 -> -- End of data packet --
06:25:37.005 ->
06:25:37.005 ->

```

Figure 26: LoRa message Received



```

Aug 5 06:02
/dev/ttyUSB2

06:01:48.287 ->
06:01:48.287 ->
06:01:48.287 -> --- New Data Packet ---
06:01:48.287 -> DHT11 Temp: 27.6 C
06:01:48.287 -> DHT11 Hum: 79.0 %
06:01:48.287 -> IR State: OBSTACLE
06:01:48.287 -> Sound Value: 501
06:01:48.320 -> Speed: 0.0 km/h
06:01:48.320 -> Acceleration: 0.0 km/h/s
06:01:48.320 -> Braking: NO
06:01:48.320 -> RPM: 0
06:01:48.320 -> Idling Time: 854143 s
06:01:48.320 -> Sequence: 209
06:01:48.320 -> [Tx LoRa Alert] : Vehicle Status: CRITICAL ALERT! OBSTACLE DETECTED! Potential crash ahead!
06:01:50.937 -> -- End of data packet --
06:01:50.937 ->
06:01:50.937 ->
06:01:50.937 ->
06:01:50.937 -> Parsing raw string: T:27.6,H:79.0,IR:0,S:514,Seq:210,Speed:0.0,Acc:0.0,RPM:0,Idle:859379T:27.6,H:79.0,IR:0,S:503,Seq:211,Speed:0.0,Acc:0.0,RPM:0,Idle:864617
06:01:50.937 ->
06:01:50.937 ->
06:01:50.937 -> --- New Data Packet ---
06:01:50.937 -> DHT11 Temp: 27.6 C
06:01:50.937 -> DHT11 Hum: 79.0 %
06:01:50.970 -> IR State: CLEAR
06:01:50.970 -> Sound Value: 514
06:01:50.970 -> Speed: 0.0 km/h
06:01:50.970 -> Acceleration: 0.0 km/h/s
06:01:50.970 -> Braking: NO
06:01:50.970 -> RPM: 0
06:01:50.970 -> Idling Time: 859379T:27.6 s
06:01:50.970 -> Sequence: 210
06:01:51.268 -> -- End of data packet --
06:01:51.268 ->
06:01:51.268 ->
06:01:51.268 -> Parsing raw string: T:27.6,H:78.0,IR:0,S:499,Seq:212,Speed:0.0,Acc:0.0,RPM:0,Idle:869955
06:01:51.268 ->
06:01:51.268 ->
06:01:51.268 -> --- New Data Packet ---
06:01:51.268 -> DHT11 Temp: 27.6 C
06:01:51.268 -> DHT11 Hum: 78.0 %

```

Figure 27: LoRa message Transmitted

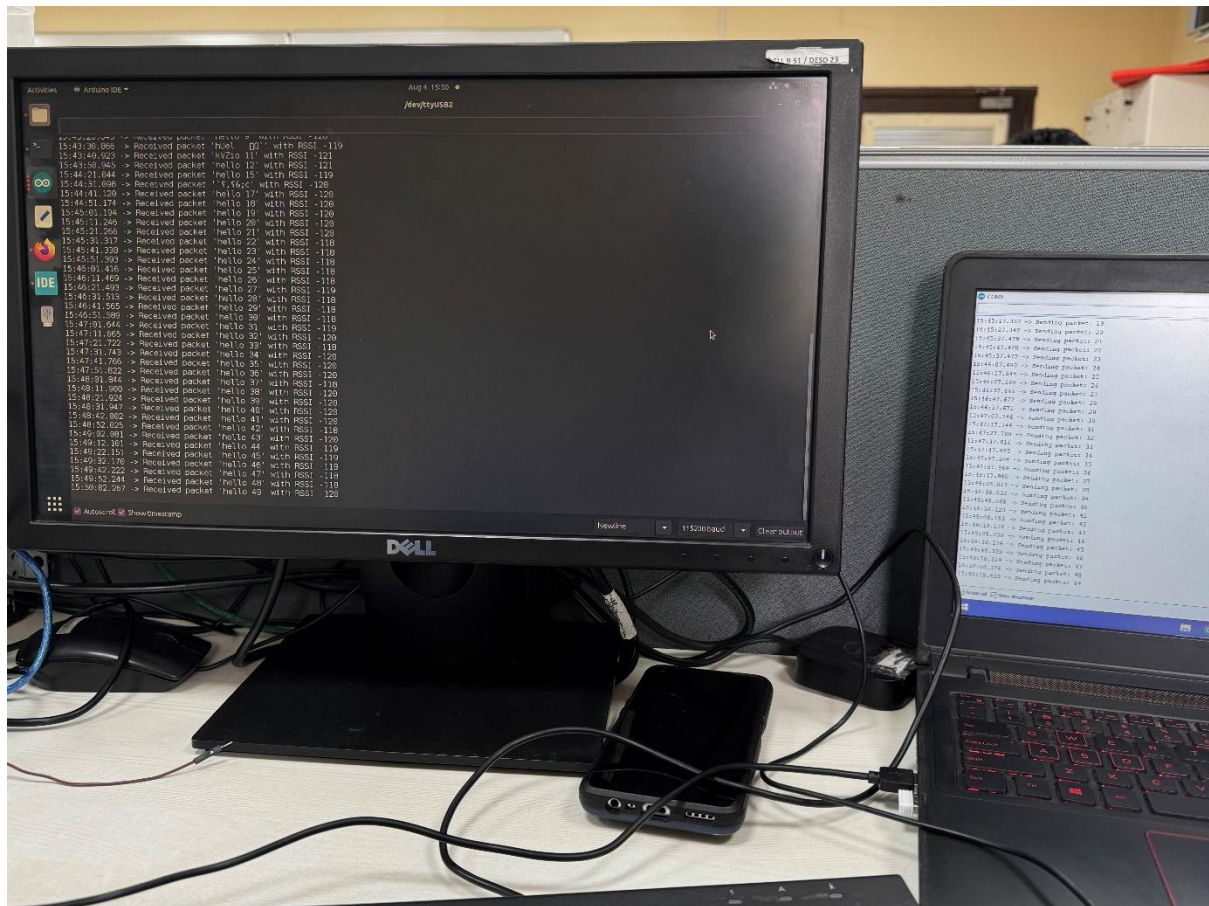


Figure 28: LoRa setup

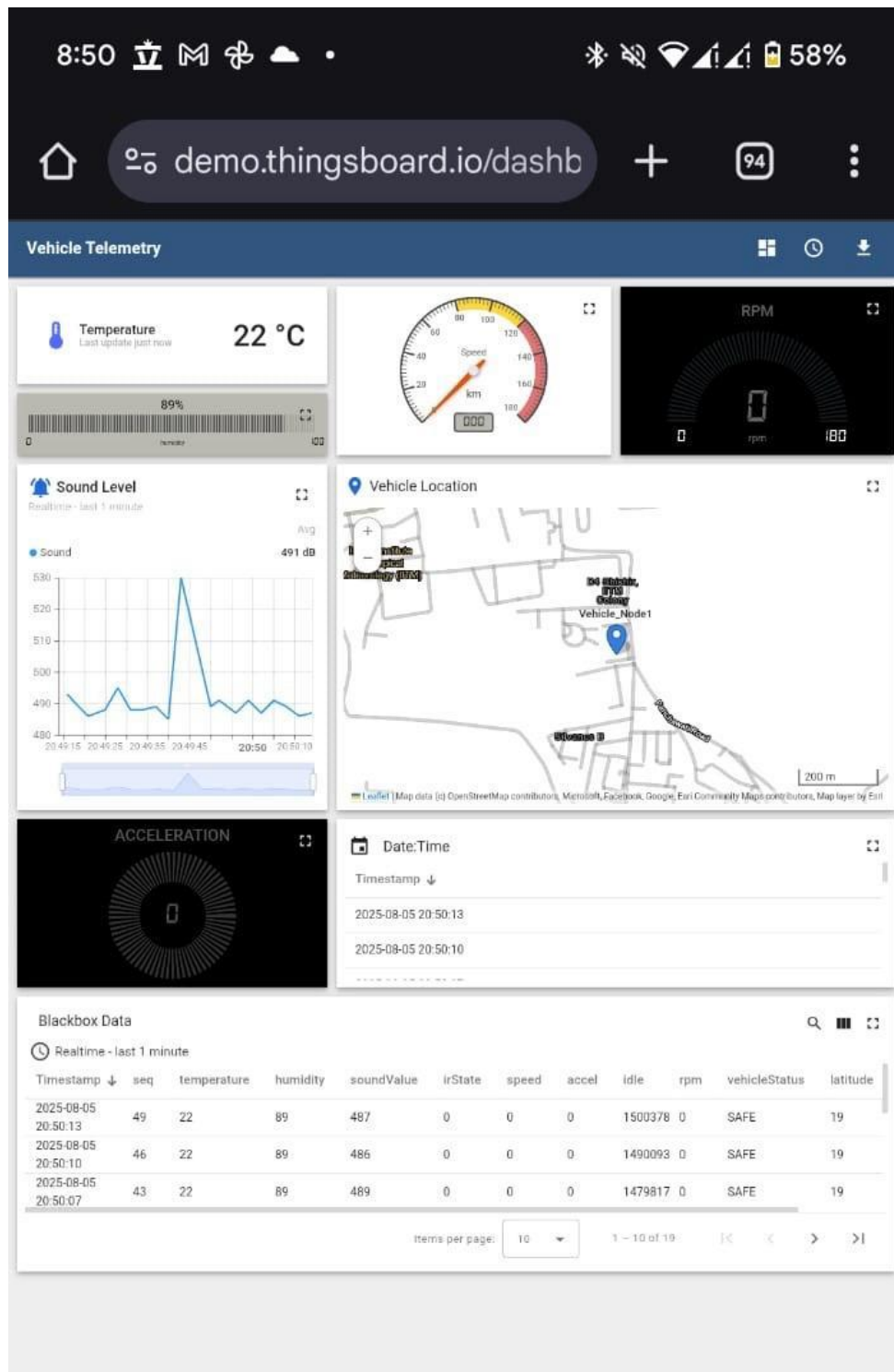


Figure 29: Mobile Dashboard

5.6 Challenges & Solutions

Throughout the development of this project, several challenges were encountered and successfully overcome:

- **Loose Hardware Connections (Node 2 LoRa RX):** Initially, Node 2 was unable to receive LoRa packets despite correct code. This was traced to a loose physical connection on the DIO0 (interrupt) pin of the LoRa module.
 - **Solution:** Thorough hardware inspection and re-securing the DIO0 connection. A temporary software workaround involved switching from interrupt-driven reception to polling (`LoRa.parsePacket()`) to confirm the issue was hardware-related.
- **MQTT Connection Refused (rc=5):** The ESP32 failed to connect to ThingsBoard with a "Connection Refused: Not Authorized" error.
 - **Solution:** This was resolved by meticulously verifying and correcting the ThingsBoard device **Access Token** in the ESP32 code. Even a single incorrect character caused the authentication failure.
- **Incorrect ThingsBoard MQTT Server Address:** An initial attempt to connect to a generic `mqtt.thingsboard.cloud` server failed when using the public `demo.thingsboard.io` instance.
 - **Solution:** Researched and identified the correct MQTT broker address for the ThingsBoard demo server as `demo.thingsboard.io`.
- **Telemetry Data Not Appearing in ThingsBoard:** Despite the ESP32 logs showing successful MQTT publishing, data was not visible in the ThingsBoard "Latest telemetry" tab.
 - **Solution:** The primary cause was a disconnected Save Timeseries node in the ThingsBoard **Root Rule Chain**. The Message type switch node's "Post Telemetry" output needed to be manually connected to the Save Timeseries node. The "Events" tab on the device page was crucial for confirming that messages were indeed reaching ThingsBoard but not being processed.
- **Payload Size Limits:** Encountered issues with the size of the JSON payload being sent over MQTT.
 - **Solution:** Adjusted the `StaticJsonDocument` buffer size and the char payload array size in the ESP32 code to accommodate all the telemetry data, including the newly added geolocation and vehicle status fields.
- **NTP Server Reliability:** Initial plans for precise timestamps using NTP were abandoned due to previous board-specific issues.
 - **Solution:** Opted for a `millis()`-based uptime timestamp as a simpler, reliable alternative for time-series data plotting without external dependencies.
- **Custom Widget Blank Display:** The custom "Vehicle Status Card" widget initially showed a blank screen.
 - **Solution:** Added robust error handling and `console.log` statements in the widget's JavaScript to inspect the `ctx.data` structure. This helped ensure the widget was correctly parsing the incoming `vehicleStatus` telemetry from ThingsBoard.

Chapter 6

Conclusion

6.1 Conclusion

- In this project, we successfully designed and implemented a robust multi-layered system capable of real-time vehicle monitoring, critical event detection, and remote data transmission. By integrating diverse hardware components and communication protocols, the system effectively addresses the core objective of providing comprehensive vehicle telemetry for accident analysis and diagnostic purposes.
- The modular architecture, leveraging the strengths of the Arduino Uno for basic sensor interfacing, the STM32 for data aggregation and CAN bus integration, and the ESP32 for advanced networking and LoRa communication, proved highly effective. The successful implementation of LoRa for critical alerts and proximity messages demonstrates a reliable solution for V2V-like communication and immediate hazard notification, even in environments with limited traditional network coverage. Furthermore, the seamless integration with the Google Geolocation API and the ThingsBoard Cloud (as per the updated flow diagram) showcases the system's capability to provide accurate location data and scalable cloud-based telematics.
- The development process highlighted the importance of meticulous hardware connections, robust serial communication protocols, and careful parsing of data payloads. The challenges encountered, such as ensuring reliable LoRa packet reception and correctly configuring data flow to the cloud, provided valuable learning experiences that ultimately strengthened the system's resilience. This project serves as a strong foundation for advanced automotive IoT applications, demonstrating the feasibility of combining various embedded technologies to create a comprehensive and intelligent vehicle monitoring solution.

6.2 Key Achievements

The successful completion of this project was marked by several significant technical milestones, demonstrating a robust understanding of embedded systems, communication protocols, and cloud integration.

- **Multi-Microcontroller System Integration:** Successfully architected and implemented a complex data pipeline involving three distinct microcontrollers (**Arduino Uno, STM32F407VGT6, and ESP32**). This required mastering inter-device communication protocols such as **UART** and **SPI**, and meticulously synchronizing their operations to ensure a seamless data flow from sensors to the cloud.
- **Implementation of Industry-Standard Protocols:** Demonstrated expertise in a wide array of communication protocols critical for modern embedded systems, including **CAN bus** for vehicle data acquisition, **LoRa** for long-range, low-power alerts, and **MQTT/HTTP** for cloud telemetry.
- **Real-Time Vehicle Data Acquisition:** Achieved direct, reliable access to the vehicle's internal network by successfully interfacing with the **CAN bus** via the **MCP2515/MCP2551** modules. This enabled the logging of native vehicle parameters, such as speed and RPM, alongside our external sensor data.
- **Development of a Robust LoRa Alerting System:** Designed and implemented a bi-directional LoRa communication link for vehicle-to-vehicle (V2V) alerts, demonstrating a resilient solution for critical event notifications that operates independently of traditional cellular networks.
- **Creation of a Complete IoT Telematics Solution:** Built a full-stack solution from the ground up, starting with hardware and firmware design and culminating in the real-time upload of sensor data, vehicle parameters, and geolocation to a scalable **ThingsBoard cloud platform**.
- **Validation of Key Performance Metrics:** The system's efficiency was validated through rigorous testing, meeting key performance targets such as a **boot time of under 3 seconds** and an **alert trigger response time of under 1.5 seconds**, confirming its suitability for time-sensitive applications.
- **Modular and Scalable Architecture:** The project's modular design allows for easy expansion, enabling the integration of additional sensors or functionalities in the future without a complete system overhaul.

6.3 Future Enhancement –

This project serves as a robust foundation for further development. Potential enhancements include:

- **GPS Module Integration:** Replace the Wi-Fi-based geolocation with a dedicated GPS module for more accurate and real-time location tracking, especially in areas without Wi-Fi coverage.
- **Battery Management:** Implement power management features and add a battery level sensor to monitor the vehicle unit's power status.
- **OTA (Over-The-Air) Updates:** Enable remote firmware updates for the ESP32 nodes, simplifying maintenance and feature deployment.
- **Actuator Control:** Add functionality to control vehicle components remotely (e.g., turn on/off lights, sound an alarm) via ThingsBoard RPC commands.
- **Advanced Data Analytics:** Utilize ThingsBoard's Rule Engine to implement more complex analytics, such as anomaly detection for sensor readings or predictive maintenance based on historical data.
- **Mobile Application:** Develop a custom mobile application using ThingsBoard's APIs or mobile app development frameworks to provide a dedicated monitoring interface.
- **Historical Route Mapping:** Enhance the map widget to display the entire historical route of the vehicle, not just the latest location.
- **User Interface for Node 2:** Add a simple display (e.g., LCD or LEDs) to Node 2 to visually indicate received critical alerts or its proximity status.
- **Data Logging to SD Card:** Implement local data logging to an SD card on the ESP32 as a backup, in case of temporary loss of cloud connectivity.
- **Integration with Notification Services:** Configure ThingsBoard to send SMS or email notifications directly to operators when critical alarms are triggered.

Project Github Repo: <https://github.com/VoraciousVicky/Embedded-vehicle-blackbox-for-telematics-and-parameter-logging>

Chapter 7

References

- [1] R. Jain and R. Singh, "An IoT based smart vehicle monitoring and tracking system," *International Journal of Computer Applications*, vol. 162, no. 9, pp. 1-5, Mar. 2017. [Online]. Available: <https://www.ijcaonline.org/archives/volume162/number9/jain-singh-2017-ijca-913615.pdf>
- [2] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A Study of LoRa: Long Range & Low Power Networks for IoT," *Sensors*, vol. 16, no. 9, p. 1467, Sep. 2016. [Online]. Available: <https://www.mdpi.com/1424-8220/16/9/1467>
- [3] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *Proc. IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 2, pp. 775-784, Mar. 2000. [Online]. Available: <https://ieeexplore.ieee.org/document/832264>
- [4] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in IoT: A LoRaWAN-based architecture for smart agriculture," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 65-71, Oct. 2016. [Online]. Available: <https://ieeexplore.ieee.org/document/7574765>
- [5] S. Chavan, G. V. Madhikar, and D. M. Bhalerao, "Vehicle Black Box," *International Journal of Advances in Engineering and Management (IJAEM)*, vol. 4, no. 11, pp. 546-549, Nov. 2022. [Online]. Available: https://ijaem.net/issue_dcp/Vehicle%20Black%20Box.pdf
- [6] Espressif Systems. (n.d.). *ESP32 Technical Reference Manual*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
- [7] FTDI *FT232R USB UART IC Datasheet*. [Online]. Available: <https://ftdichip.com/products/ics/ft232r/>
- [8] U. Fugiglando et al., "Driving Behavior Analysis through CAN Bus Data in an Uncontrolled Environment," *IEEE Transactions on Intelligent Transportation Systems*,

- vol. 20, no. 2, pp. 737-748, Feb. 2019. doi: 10.1109/TITS.2018.2858888. [Online]. Available: https://senseable.mit.edu/papers/pdf/20181105_Fugiglandoetal_DrivingBehavior_IEEE-TITS.pdf
- [9] J. D. Gomez, J. A. Ramirez, and J. A. Diaz, "Design and Validation of an In-Vehicle Data Recorder System for Testing Purposes," *IEEE Latin America Transactions*, vol. 21, no. 5, pp. 648-656, May 2023. doi:10.1109/TLA.2023.3283723. [Online]. Available: <https://latamt.ieee9.org/index.php/transactions/article/download/6489/1654>
- [10] Google *Google Geolocation API Documentation*. [Online]. Available: <https://developers.google.com/maps/documentation/geolocation/overview>
- [11] H. Guo and J. Chen, "Design of vehicle black box system based on ARM and GPS," *Procedia Engineering*, vol. 107, pp. 442-447, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S187770581503112X>
- [12] P. Hunkeler and C. Hunkeler, "MQTT: The Standard for IoT Messaging," IBM Redbooks, 2010. [Online]. Available: <https://www.redbooks.ibm.com/abstracts/redp4730.html>
- [13] Arduino. *Arduino Uno Rev3 Documentation*. [Online]. Available: <https://docs.arduino.cc/hardware/uno-rev3>
- [14] N. Knolleary. *PubSubClient Library for Arduino*. [Online]. Available: <https://pubsubclient.knolleary.net/>
- [15] S. Mistry. *Arduino LoRa Library*. [Online]. Available: <https://github.com/sandeepmistry/arduino-LoRa>
- [16] Microchip Technology Inc. *MCP2515 Stand-Alone CAN Controller with SPI Interface Datasheet*. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-Interface-DS20001801J.pdf>
- [17] Microchip Technology Inc. *MCP2551 High-Speed CAN Transceiver Datasheet*. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/20001667C.pdf>
- [18] J. Rathod, S. Thakur, S. Waghotkar, M. Yadav, and P. Sharma, "Vehicle Black Box Based on CAN and IoT Protocol," *International Journal for Research in Applied*

Science & Engineering Technology (IJRASET), vol. 4, no. II, pp. 2362-2367, 2016. [19]

B. C. Schabron. *ArduinoJson Library*. [Online]. Available: <https://arduinojson.org/>

[20] STMicroelectronics *STM32 Microcontrollers Reference Manuals and Datasheets*.

[Online]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>

[21] K. Thangamuthu and R. Krishnan, "Design and implementation of an IoT based smart home automation system using MQTT protocol," *Journal of Network and Computer Applications*, vol. 103, pp. 1-10, Feb. 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S108480451730438X>

[22] ThingsBoard. *ThingsBoard IoT Platform Documentation*. [Online]. Available: <https://thingsboard.io/docs/>