OPTIMAL CHARGING AND DISCHARGING SCHEDULES OF LI-ION BATTERY FOR ENHANCED SOH

A SUMMER INTERNSHIP PROJECT REPORT

submitted by

DEVANATHAN J	**311422106010**	**Meenakshi College of Engineering**

Under the guidance of

Dr.N.PAPPA
Dr.S.SUTHA
Dr.SABITHA RAMAKRISHNAN

in partial fulfillment for the completion

of

Summer internship SISEA - 2025

DEPARTMENT OF INSTRUMENTATION ENGINEERING
MADRAS INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY::CHENNAI - 600 044

JULY 2025

BONAFIDE CERTIFICATE

Certified that this summer internship project report "**Optimal Charging and Discharging Schedules of Li-ion Battery for Enhanced SoH**" is the Bonafide work of

DEVANATHAN J 3rd Year B.E, ECE. Meenakshi College of Engineering who carried out the project work under my supervision.

Certified further, that to the best of my knowledge the work reported here does not form part of any project on the basis of which degree or award was conferred on an earlier occasion on this or any other candidate.

Dr.S.SRINIVASAN	**Dr.N.PAPPA** **Dr.S.SUTHA** **Dr.SABITHA RAMAKRISHNAN**
HEAD & PROFESSOR	**PROJECT GUIDE**
Dept of Instrumentation Engg.	Dept of Instrumentation Engg.
MIT- Anna University	MIT- Anna University
Chrompet,Chennai-600 044.	Chrompet,Chennai-600 044.

ACKNOWLEDGEMENT

We would like to record our sincere thanks to **Prof. P. Jayshree**, Dean, MIT, for having given consent to carry out the summer internship work at Madras Institute of Technology Campus, Anna University.

We wish to express our sincere appreciation and gratitude to **Dr. S. Srinivasan**, Professor Head of the Department, Department of Instrumentation Engineering, MIT, for permitting us to use the facility at the Department.

We would like to thank **RUSA 2.0 (PO6)** for providing with internship to work in the funded project titled " **Improved SoH by Soft Sensors and Controls for Battery Management Systems of Electric Vehicles** ".

We wish to express our sincere gratitude to **Dr N. Pappa**, Professor, Team Co-Ordinator, RUSA 2.0 (PO6) for giving us an opportunity to work under her. Her extensive vision, practical inputs and creative thinking has been a source of inspiration throughout this project.

We would like to thank **Dr. S. Sutha**, Professor, Member, RUSA 2.0 (PO6) Department of Instrumentation Engineering, MIT for her constructive criticism throughout our internship.

We would like to thank **Dr. Sabitha Ramakrishnan**, Professor, for her insightful guidance and continuous support throughout our internship.

We are extremely grateful to **Mrs. P. Anbumalar**, Project associate, RUSA 2.0 (PO6) Department of Instrumentation Engineering, MIT who helped us in successful completion of this internship.

We would like to express our sincere gratitude to **Ms. Sabarika**, **Ms. O. Arunpriya**, **Dr. S. Meyyappan**, and **Mr. A. Elavarasan** for their unwavering assistance, constant support, and encouragement, all of which played a significant role in the successful completion of our project and contributed greatly to our overall learning experience.

We sincerely thank **Dr. G. Ra. Gokul**, Chairman; **Dr. S. Anandakumar**, Principal; and **Mrs. R. Ananda Praba**, Head of the Department, **Meenakshi College of Engineering**, for their support and for providing the facilities necessary to complete this internship.

DEVANATHAN J (311422106010)

ABSTRACT

Electric Vehicles (EVs) are rapidly becoming an essential part of sustainable transportation. With the increasing adoption of EVs, the need for efficient battery monitoring and smart charging infrastructure is more critical than ever. This project proposes a optimal charging and discharging schedule of Li-ion battery for enhanced State of Health (SoH).

The system leverages the ESP32 microcontroller to collect live data from a Battery Management System (BMS) via Bluetooth, including State of Charge (SoC), battery voltage, current, and temperature. The data collected is then transmitted over Wi-Fi and uploaded to Google Sheets, Firebase Fire store, and visualized through Blynk Console, enabling easy monitoring and analysis.

To maintain optimal battery performance, the system enforces intelligent charging behavior—when the SOC falls below 20 percent, the system triggers a blinking alert indicating that charging is needed, and when the SOC exceeds 80 percent, it automatically notifies to stop charging to prevent overcharging and prolong battery life.

A Google Apps Script handles real-time data logging, alert generation, and filtering logic for dynamic visualization. The entire system is built and tested in a simulated environment, offering a low-cost and scalable solution for intelligent EV energy management.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Electric vehicles (EVs) are emerging as a sustainable alternative to fossil fuel-based transport. However, to ensure efficiency, longer battery life, and user convenience, smart battery monitoring and intelligent charging systems are essential. With the growing EV population, it is crucial to deploy solutions that monitor battery health, optimize charging, and avoid congestion at stations.

This project introduces a real-time cloud-based system using ESP32 as an IoT gateway between a Battery Management System (BMS) and platforms like Google Sheets, Blynk Console, and Firebase. Battery data (SoC, voltage, current) is collected via Bluetooth and transmitted over Wi-Fi. The system applies the 20–80 SoC rule to extend battery life by avoiding deep discharge and overcharging.

Smart algorithms analyse the data to recommend optimal charging stations based on SoC, queue length, and estimated wait time. By integrating real-time data with cloud automation and decision logic, the system reduces range anxiety, prevents station overload, and improves EV usability.

1.2 OBJECTIVE

The overall objective of the work is to develop optimal charging and discharging schedule for Li-ion battery to enhance its SoH. With this background the following sub-objectives are formulated:-

- To develop a cloud-integrated monitoring system that reads BMS data (voltage, current, SoC) in real-time from an EV and uploads it to online platforms using ESP32.
- To detect low battery conditions (SoC < 20%) and high temperature warnings and automatically recommend the most optimal charging station.
- To visualize and log the EV battery data in Google Sheets, Firebase Firestore, and Blynk Console for analytics and tracking.
- The 20–80 SoC rule encourages maintaining the battery charge between 20% and 80% to extend battery life and optimize performance, with LED alerts indicating when this range is approached.

1.3 NEED FOR THIS SYSTEM

As EV adoption increases, smart energy management becomes vital for charging infrastructure. Static and manual charging decisions often lead to overload, long waiting times, or underutilized stations. To ensure a seamless and automated solution, a lightweight and accessible microcontroller like the ESP32 can be used to collect and transmit BMS data without the need for bulky hardware.
Additionally, by simulating intelligent scheduling based on real-world parameters (SoC, location, queue), this system lays the foundation for a scalable EV network suitable for campuses, smart cities, and fleet systems.

1.4 SCOPE OF THE WORK

The project uses ESP32 with Bluetooth and Wi-Fi capabilities to create a fully simulated but practical smart EV monitoring and scheduling prototype. The system is designed to:

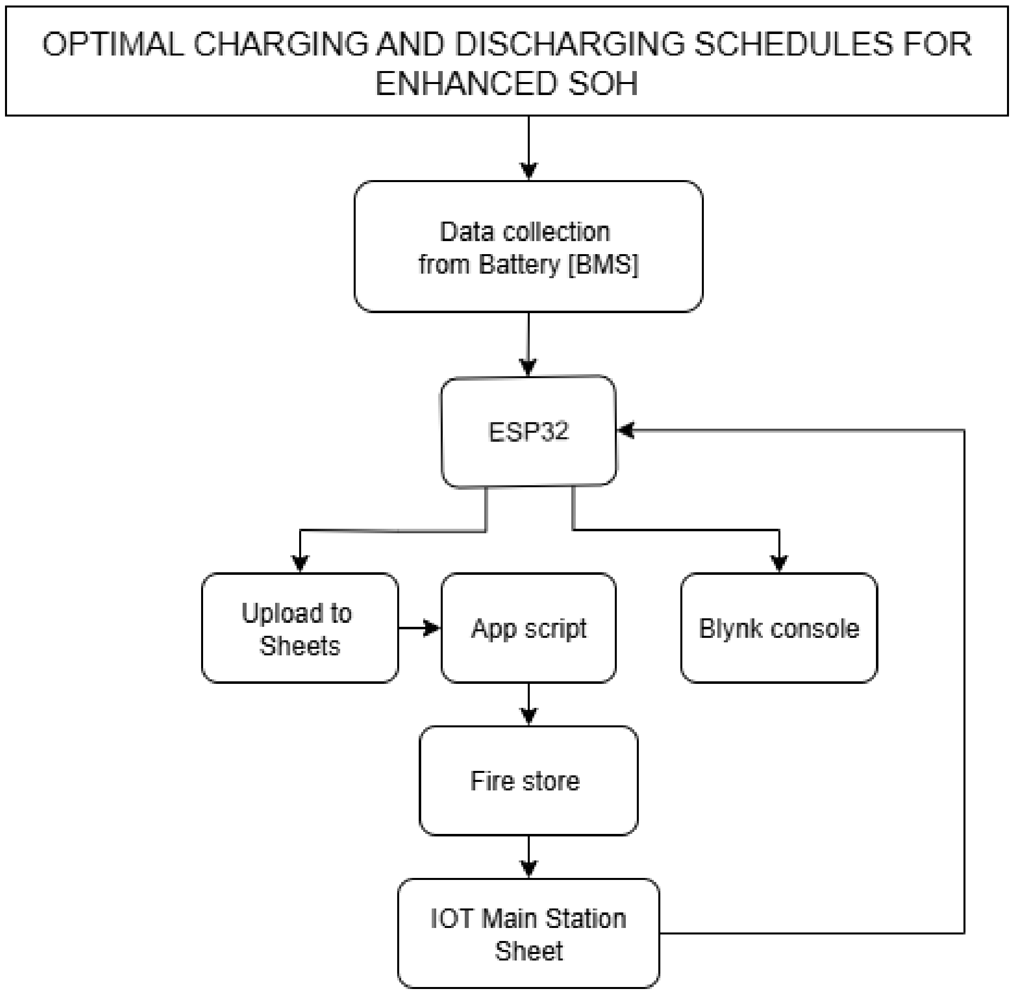

OPTIMAL CHARGING AND DISCHARGING SCHEDULES FOR ENHANCED SOH
Data collection from Battery [BMS]
ESP32
Upload to Sheets
App script
Blynk console
Fire store
IOT Main Station Sheet

Figure 1.4.1: Tree diagram representing the full scope of the project, from battery data acquisition to cloud storage and visualization.

- Communicate with a Daly BMS to extract battery data.
- Use Google Apps Script, Firebase, and Blynk Console to upload and visualize the data.
- Trigger actions (like alerts and recommendations) when SoC drops below a threshold or if temperature exceeds safe limits.
- Store the data in Main station for future recommendations.

This system is entirely cloud-based and can be scaled across multiple EVs or integrated with mobile apps and dashboards for real-time decision-making.

1.5 ORGANIZATION OF THESIS

The structure of the thesis/project report is organized as follows:

- **CHAPTER 2**: Reviews existing systems and literature relevant to EV monitoring and smart charging.
- **CHAPTER 3**: Explains key technologies including ESP32, Firebase, BMS protocols, and cloud platforms used.
- **CHAPTER 4**: Describes the system architecture, data flow, and methodology including BLE communication and Google Sheets integration.
- **CHAPTER 5**: Details the simulation of smart charging recommendations and backend logic for queue and time-based scheduling.
- **CHAPTER 6**: Presents the outcomes, dashboards, and visualizations of real-time battery and station data.
- **CHAPTER 7**: Concludes with insights, limitations, and suggested future improvements such as mobile app integration and live GPS.

CHAPTER 2

LITERATURE SURVEY

2.1 Real-time Monitoring Using Python and Raspberry Pi

Source: Espinosa, F., & Tan, S. (2020)

Espinosa and Tan (2020) introduced a real-time monitoring and logging system for lithium-ion battery health using Python and Raspberry Pi. The study outlines how embedded platforms like Raspberry Pi can be combined with analog-to-digital converters and Python-based scripts to monitor key battery parameters such as voltage, current, temperature, and State of Charge (SoC). Their architecture includes a sensor module connected via I2C and SPI buses, interfacing with a Python-based application for real-time logging and dashboard display.

The primary advantage presented is the system's ability to function autonomously with minimal computational overhead, making it ideal for small-scale deployments in research or educational EV prototypes. Espinosa and Tan emphasized the importance of continuous data logging in diagnosing battery anomalies and extending battery life. Their implementation ensures scalability for larger applications and offers remote access features, aligning with the trend toward smart diagnostics in embedded systems.

2.2 IoT-Based Battery Health Monitoring for EVs

Source: Chatterjee A., Sinha R. (2019)

Chatterjee and Sinha (2019) presented a detailed IoT-based architecture for monitoring electric vehicle battery health. The framework utilizes microcontroller platforms interfaced with sensors to capture temperature, voltage, SoC, and cycle count data. Their system is designed to transmit real-time telemetry using MQTT

over Wi-Fi to a centralized cloud server, ensuring remote access and mobile viewing capabilities.
The authors emphasized predictive maintenance capabilities made possible by trend analysis on logged data. Their approach addresses issues related to thermal runaway and aging degradation by leveraging historical datasets. The system also integrates cloud analytics for proactive alert generation. The paper's contribution lies in merging low-cost hardware and scalable IoT platforms, proposing a viable solution for commercial EV fleets seeking real-time monitoring and control.

2.3 Applications of the Blynk IoT Platform

Source: Kaur, G., & Singh, J. (2021)

Kaur and Singh (2021) conducted a comprehensive review of the Blynk IoT platform, particularly its suitability for remote system monitoring. Their paper examined various use cases across domains like smart agriculture, home automation, and battery monitoring systems. Blynk's architecture consists of mobile app widgets, a cloud server, and microcontroller libraries, allowing real-time interaction between users and IoT devices.

The review emphasizes Blynk’s low-code approach and mobile-first design, which facilitates easy deployment without needing extensive backend configuration. When applied to battery monitoring systems, the platform enables the visualization of voltage, SoC, and temperature parameters via graphs and gauges. The researchers suggest that Blynk’s extensibility and open-source foundation make it a practical choice for students, hobbyists, and early-stage commercial products.

2.4 Google Sheets-Based IoT Logging Using Raspberry Pi

Source: Mahajan, S., & Kumar, A. (2022)

Mahajan and Kumar (2022) explored a novel way of leveraging Google Sheets as a cloud-based logging system for IoT applications. Their work uses Raspberry Pi to collect data from temperature and battery sensors, then logs that data into Google

Sheets using HTTP POST requests and Google Apps Script. The system is advantageous for being cost-effective, serverless, and easy to share among users. Their implementation illustrates how API quotas, authentication methods, and update intervals can be optimized for stable performance. The paper highlights use cases in classroom teaching, low-budget prototypes, and home monitoring systems. Mahajan and Kumar argue that this approach, while simple, can form the backend for larger-scale real-time analytics dashboards or be extended with Google Data Studio for visualization.

2.5 Smart Charging Using GPS and SoC

Source: Lien, C. H., & Lin, Y. Y. (2020)

Lien and Lin (2020) proposed an intelligent charging architecture for EVs that incorporates GPS location and SoC data to dynamically allocate nearby charging stations. The system uses on-board GPS modules and battery monitoring units to send data to a centralized control system, which then calculates optimal routing to the most suitable charging station.

The paper emphasizes the significance of real-time data exchange, particularly in urban environments with high traffic density. Their routing algorithm accounts for distance, battery constraints, and estimated wait time. The research suggests that combining location awareness with energy data enhances EV efficiency and reduces grid load during peak hours. They also recommend future improvements using machine learning for demand prediction and resource optimization.

2.6 EV Charging Scheduling Using Distance, Queue, and SoC

Source: Sharma, V., & Patel, D. (2023)

Sharma and Patel (2023) developed a smart EV charging scheduler that uses a combination of distance to station, queue size, and current SoC to assign vehicles to the best available charging point. The proposed system dynamically evaluates

multiple variables and employs a decision-making algorithm to distribute vehicles efficiently.
Their simulation shows reduced overall wait time and increased system throughput compared to traditional first-come, first-serve methods. The authors note the importance of decentralized scheduling and edge computing for future deployments in smart cities. This model is scalable for highway networks and urban charging grids, making it particularly valuable for areas with high EV density and limited charging infrastructure.

2.7 RS485 Sensor Communication Using Python

Source: You, W., Lee, J., & Park, H. (2018)

You et al. (2018) focused on implementing low-cost industrial communication using RS485 protocol and Python programming. The system involves connecting multiple sensors over RS485 to a Raspberry Pi and decoding the data stream in real-time using Python scripts. RS485 is chosen due to its robustness, noise immunity, and long-range capabilities.

Their study finds RS485 particularly suitable for industrial automation and smart monitoring where wireless protocols may be unreliable. Python's role is central in providing a flexible and extensible interface for configuring, calibrating, and visualizing data. The authors highlight the protocol's relevance in battery monitoring systems in environments with high electromagnetic interference.

2.8 Fusion of Monitoring and Scheduling Architectures

By synthesizing insights from Mahajan et al., Kaur et al., Sharma et al., and Lien et al., a hybrid system is proposed wherein monitoring, logging, and charging scheduling are fused. For instance, real-time SoC data from a Raspberry Pi can be pushed to Google Sheets or Blynk, while simultaneously being used in smart scheduling algorithms based on GPS proximity and queue status.

Such a fusion allows a complete system with visibility (monitoring), accessibility (cloud logging), and intelligence (scheduling), suitable for smart cities. This integrated architecture paves the way for AI-based systems where EVs autonomously decide the most optimal charging path.

2.9 Practical Considerations in Embedded EV Monitoring

Derived from Espinosa, Chatterjee, and You et al.

Implementing EV monitoring systems on embedded platforms introduces several practical considerations: power efficiency, sensor calibration, communication reliability, and user interface design. Espinosa and Tan emphasize simplicity and power efficiency using Python, while Chatterjee and Sinha stress cloud connectivity and MQTT-based transport. You et al. highlight the value of industrial protocols in harsh environments.

Combining these insights reveals that the hardware-software co-design must focus on minimizing latency, maximizing uptime, and reducing deployment cost. Engineers should also factor in failover mechanisms and data redundancy.

2.10 Summary and Research Directions

Collectively, the reviewed works contribute to a growing ecosystem of intelligent, low-cost, and scalable EV battery management systems. Major research directions include:

- AI-driven prediction of battery failure and charging demand
- Integration of blockchain for secure EV-data exchange
- Use of LPWANs (e.g., LoRa, NB-IoT) in place of Wi-Fi or RS485
- User-focused UI/UX in platforms like Blynk and custom dashboards
- Federated learning models for predictive analytics without centralized datasets

The reviewed literature forms a strong foundation for future innovations in electric vehicle infrastructure and energy systems.

CHAPTER 3

PROPOSED METHODOLOGY

3.1 INTRODUCTION

The rapid growth of Electric Vehicles (EVs) has led to an increasing demand for intelligent battery monitoring and optimized charging infrastructure. Efficient battery health management and timely charging are critical to prolonging battery life and ensuring seamless vehicle operation. In this project, we propose a Smart EV Charging and Scheduling System that leverages real-time battery data collection using BLE, low-cost microcontrollers (ESP32 or Raspberry Pi), and Google Sheets for cloud-based logging and visualization.

The system captures vital battery parameters such as State of Charge (SoC), voltage, current, and temperature from a Daly Battery Management System (BMS). The data is transmitted wirelessly to an ESP32, which processes and sends it to a centralized cloud storage system via Google Sheets. The logged data is then used to generate low-battery alerts and provide optimal charging recommendations based on the vehicle's condition. This approach offers a scalable and affordable solution for battery monitoring and smart charging scheduling, suitable for real-world EV fleet management or individual users.

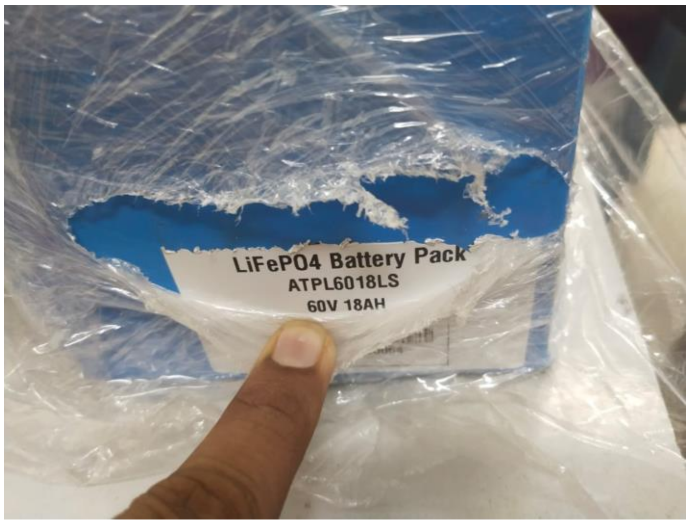


Figure 3.1.1: BMS (LiFeP04)

Figure 3.1.2: BMS

3.2 METHODOLOGY

The proposed system aims to monitor real-time battery parameters from an Electric Vehicle (EV) and utilize this data for intelligent charging recommendations. A Raspberry Pi is used as the central processing unit, interfaced with the Battery Management System (BMS) using an RS485-to-USB converter module to improve SoH of the Battery. Through the Raspberry Pi's USB port, a Python script sends predefined 13-byte command frames over the RS485 protocol to request data from the Daly BMS. The BMS responds with hexadecimal data packets containing key battery metrics such as State of Charge (SoC), voltage, current, and temperature. These raw responses are parsed and decoded in Python to extract human-readable values.

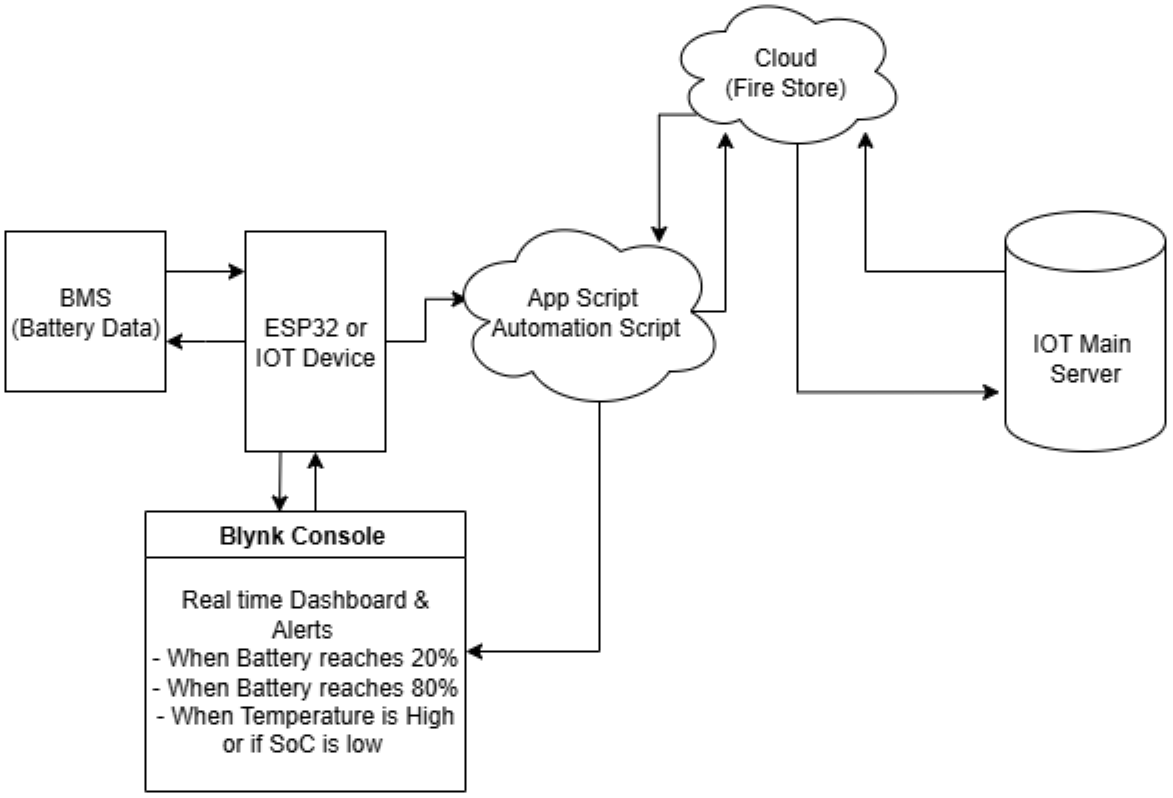


Figure 3.2.1: Methodology Block diagram

Once the data is validated and timestamped, it is formatted into a JSON structure and sent via HTTP POST to a Google Apps Script Web App endpoint. The Apps Script receives this data and logs it into a Google Sheet, creating a live and structured database.

This cloud-stored dataset can be used for generating alerts (e.g., SoC < 20%) and optimizing EV charging schedules based on real-time conditions. This methodology ensures accurate data acquisition using wired serial communication and a low-cost, programmable platform.

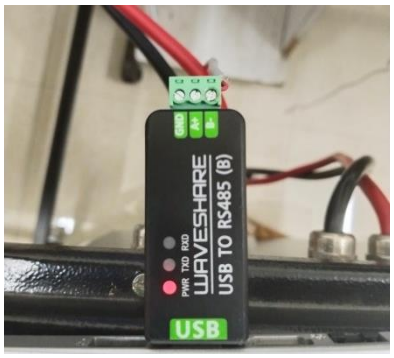


Figure 3.2.2: USB to RS485 module

3.3 DATASET DESCRIPTION

The dataset generated in this project is derived from real-time sensor values transmitted by the Battery Management System (BMS) of an Electric Vehicle. The Daly BMS provides crucial information about the state and health of the battery pack. The specific parameters collected are:

- **State of Charge (SoC)**: Indicates the remaining charge of the battery in percentage terms.
- **Battery Voltage (V)**: The overall pack voltage at the moment.
- **Charging/Discharging Current (A)**: Real-time current flowing into or out of the battery.
- **Temperature (°C)**: Readings from multiple NTC temperature sensors embedded in the battery pack.

Each data record includes a **timestamp** marking the exact time of data capture. The sampling rate is typically set at **30 seconds**, meaning new values are collected and stored every half minute. Over time, this creates a large time-series dataset that can be used for performance monitoring, early fault detection, or predictive scheduling of charging events.

The dataset serves two primary purposes:

1. **Battery Monitoring**: By analyzing live parameters, anomalies (e.g., low SoC or overheating) can be detected.
2. **Smart Charging Recommendation**: SoC and temperature data help determine when and where the EV should be routed for charging based on real-time conditions.

The final dataset is stored in **Google Sheets**, which acts as a lightweight, cloud-based database that can be visualized, shared, and integrated with other services such as Firebase or Google Data Studio.

3.4 DATA PREPROCESSING STEPS

Once the raw battery data is captured from the BMS, it must undergo preprocessing to ensure its reliability and usability for further operations like alert generation, trend analysis, or charging recommendations. The following preprocessing steps are implemented:

1. **Data Decoding**: The BMS sends raw hexadecimal data packets over BLE. These packets are decoded on the ESP32 using custom Python or Arduino functions that interpret the byte sequences according to Daly's communication protocol. For example, SoC is typically found in specific byte positions that are converted into floating-point numbers.
2. **Noise Filtering and Validation**: Occasionally, due to BLE instability or signal interference, corrupt or missing data might be received. To handle this:
   - Values like 0%, -1 A, or unrealistically high temperatures are rejected.
   - Only validated readings are stored or transmitted.

3. **Timestamp Generation**: The ESP32 attaches the current time (using NTP or internal timekeeping) to each record. This helps build a time series dataset essential for understanding charging/discharging trends over time.
4. **SoC Thresholding**: A logic layer checks if SoC falls below 20%. If it does:
   - The event is flagged.
   - A recommendation is prepared to guide the EV to the nearest charging station.
5. **Unit Normalization and Conversion**: For display and analysis, some values are converted to consistent units. For example:
   - Millivolts → Volts
   - Tenths of degrees → Celsius

These preprocessing steps are essential for ensuring that only clean, meaningful data is transmitted to the cloud and visualized by users.

Register 0x90 — Basic Battery Info (13 bytes)

Byte	Calculation	Formula	Example (Hex)	Explanation
0	Start Byte	0xA5	A5	Start of Frame
1	Device Address	0x40	40	BMS Address
2	Command	0x90	90	Register ID
3	Frame Length	-	08	Data length
4-5	Total Voltage	`((byte4<<8)		3124 to 312.4 V
6-7	Reserved/unused	-	00 00	Not used
8-9	Current	`(((byte8<<8)		(30000-30000) *0.1=0.0A
10-11	SoC	`((byte1-<<8)		8000 to 800.0 = 80.0%
12	Checksum	Sum(bytes[0:12]) & 0xFF	XX	Used for packet integrity check

Table 3.4.1: Hex Data Decoding Table for 0x90 Register

Register 0x95 — Cell Voltages (13 bytes per frame)

Note: There are **7 frames total**, so cell 1–19 are fetched across 7 requests:

- Frame 0 → Cell 1, 2, 3
- Frame 1 → Cell 4, 5, 6
- Frame 6 → Cell 19 (last)

Byte	Description	Formula	Example (Hex)	Explanation
0-3	Header info (A5, 40, 95, 08)	-	A5 40 95 08	Header and command
4	Frame number (0-6)	Used to identify cells in groups of 3	00	Fram 0= Cell 1,2,3
5-6	Cell Voltage 1	`((byte5 << 8)		3254 ➔ 3.254 V
7-8	Cell Voltage 2	`((byte5 << 8)	0C B8	3256 ➔ 3.256 V
9-10	Cell Voltage 3	`((byte5 << 8)	0C BA	3258 ➔ 3.258 V
11	Unused/Reserved	-	00	Not used
12	Checksum	sum(bytes[0:12]) & 0xFF	XX	Integrity check

Table 3.4.2: Hex Data Decoding Table for 0x95 Register

Register 0x96 — NTC Temperature Sensors (9 bytes per frame)

Byte	Description	Formula	Example (Hex)	Explanation
0-3	Header info (A5, 40, 96, 06)	-	A5 40 96 06	Header and command
4-9	NTC Raw Values (6 bytes)	raw_byte - 40	3C	Each byte is a temp sensor reading

Table 3.4.3: Hex Data Decoding Table for 0x96 Register

- The Daly BMS supports up to 10 NTC sensors, but your system uses **only 4**.
- You loop through up to **3 frames**, storing values until you reach 4 valid temps.

3.5 BLE-BASED BATTERY DATA COLLECTION

In this phase, **Bluetooth Low Energy (BLE)** is used for data acquisition from the BMS. BLE offers a power-efficient and low-latency wireless communication method, which is ideal for embedded EV applications.
The ESP32 microcontroller is programmed to act as a **BLE central client**, and the Daly BMS operates as a BLE peripheral. The communication flow is as follows:

1. **Device Discovery**: The ESP32 begins scanning for available BLE peripherals and identifies the BMS based on its MAC address or advertised name.
2. **Connection Establishment**: Once the BMS is found, the ESP32 initiates a secure connection and subscribes to the relevant characteristics.
3. **Characteristic Read**: Specific service UUIDs and characteristics are used to read battery metrics like:
    - Total voltage (0x90)
    - Current (0x95)
    - SoC (0x96)
    - Cell-wise voltages and temperature sensors

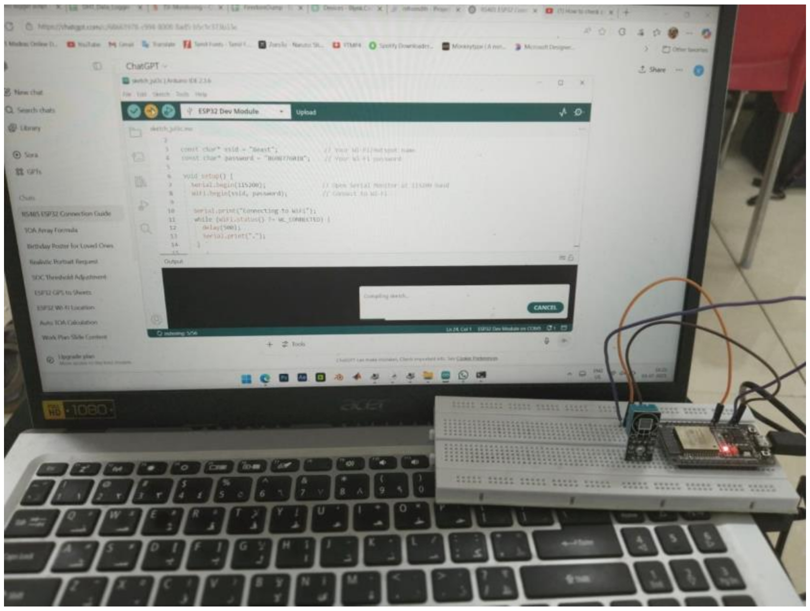


Figure 3.5.1: BMS data with /Esp32 module

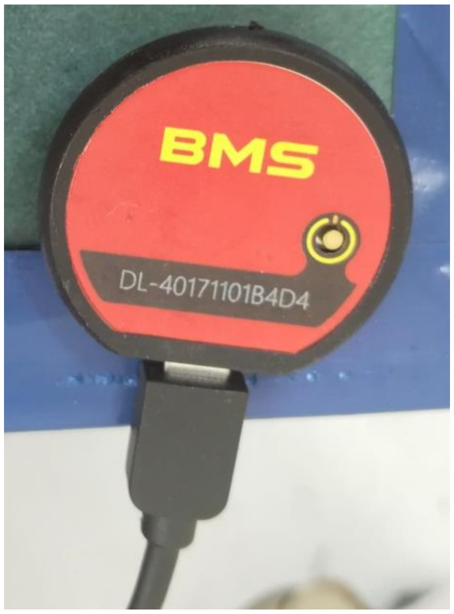
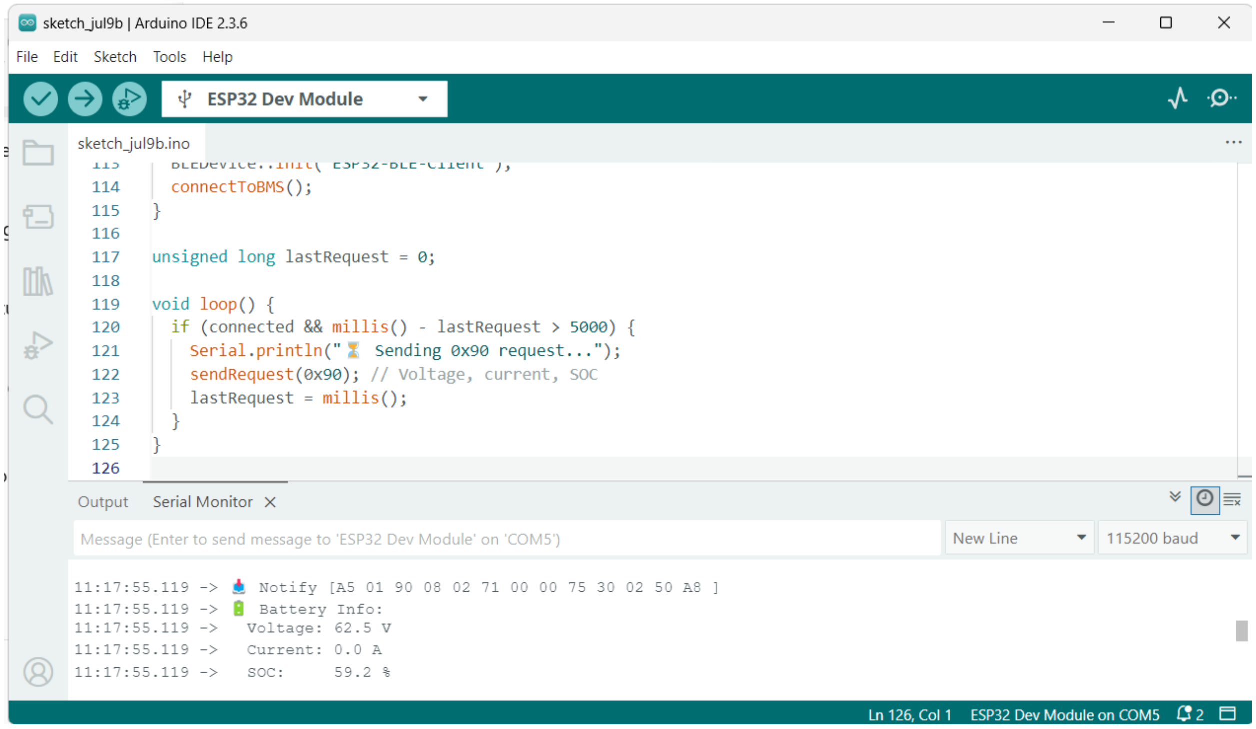


Figure 3.5.2: BLE module

4. **Data Buffering**: Once the values are read, they are stored temporarily in ESP32's memory to ensure consistency.
5. **BLE-Wi-Fi Switching**: To prevent interference between BLE and Wi-Fi operations (both use the 2.4 GHz band), BLE communication is stopped before activating the Wi-Fi module.

This BLE-based method avoids the need for USB or wired connections, allowing wireless communication between the EV and monitoring systems.


sketch_jul9b | Arduino IDE 2.3.6
File Edit Sketch Tools Help
ESP32 Dev Module
sketch_jul9b.ino
Output
Serial Monitor
Message (Enter to send message to 'ESP32 Dev Module' on 'COM5')
Ln 126, Col 1 ESP32 Dev Module on COM5 2

```
  BLEDevice::init("ESP32-BLE-Client");
  connectToBMS();
}

unsigned long lastRequest = 0;

void loop() {
  if (connected && millis() - lastRequest > 5000) {
    Serial.println("⏳ Sending 0x90 request...");
    sendRequest(0x90); // Voltage, current, SOC
    lastRequest = millis();
  }
}
```

New Line | 115200 baud

```
11:17:55.119 -> 📥 Notify [A5 01 90 08 02 71 00 00 75 30 02 50 A8 ]
11:17:55.119 -> 🔋 Battery Info:
11:17:55.119 ->   Voltage: 62.5 V
11:17:55.119 ->   Current: 0.0 A
11:17:55.119 ->   SOC:      59.2 %
```

Figure 3.5.3: Arduino IDE Code

3.6 DATA TRANSMISSION TO GOOGLE SHEETS

After BLE data acquisition and preprocessing, the ESP32 transitions to **Wi-Fi mode** to send the data to the cloud. The goal is to log this real-time information into a **Google Sheet** using a **Google Apps Script Web App**.
The transmission workflow is as follows:

1. **Wi-Fi Connection**: The ESP32 connects to a predefined Wi-Fi network stored in its flash memory.
2. **HTTP POST Request Formation**:
    - A JSON object is created with key-value pairs for voltage, current, SoC, temperature, and timestamp.
    - This object is sent as the body of an HTTP POST request to the Apps Script Web App URL.
3. **Apps Script Web App**:
    - A backend Apps Script is deployed as a Web App with the doPost(e) function.

    - The script receives the request, parses the JSON payload, and appends the values into a row in Google Sheets.
4. **Data Logging and Visualization**:
    - The Google Sheet automatically logs the new data.
    - Conditional formatting or charts can be applied to visualize SoC trends, detect anomalies, or trigger alerts (e.g., highlighting SoC < 20%).
5. **Multi-EV Integration**:
    - With slight modifications, the Web App can support multiple EVs, each sending its data with a unique Vehicle ID.

This seamless integration between ESP32 and Google Sheets enables real-time monitoring without any third-party platform dependency (e.g., Blynk, ThingSpeak), offering complete control over the data.

Data Sent to Google Sheets

Once the following data is decoded:

- voltage, current, SoC (from 0x90)
- cell voltages (from 0x95)
- temps (from 0x96)

It's sent in this **JSON format**:

json

CopyEdit

```json
{
  "voltage": 51.2,
  "current": -3.1,
  "soc": 76.0,
  "cells": [
    [1, 3.254],
    [2, 3.256],
    ...
  ],
  "temps": [26, 27, 25, 28]
}
```

CHAPTER 4

RESULTS AND DISCUSSIONS

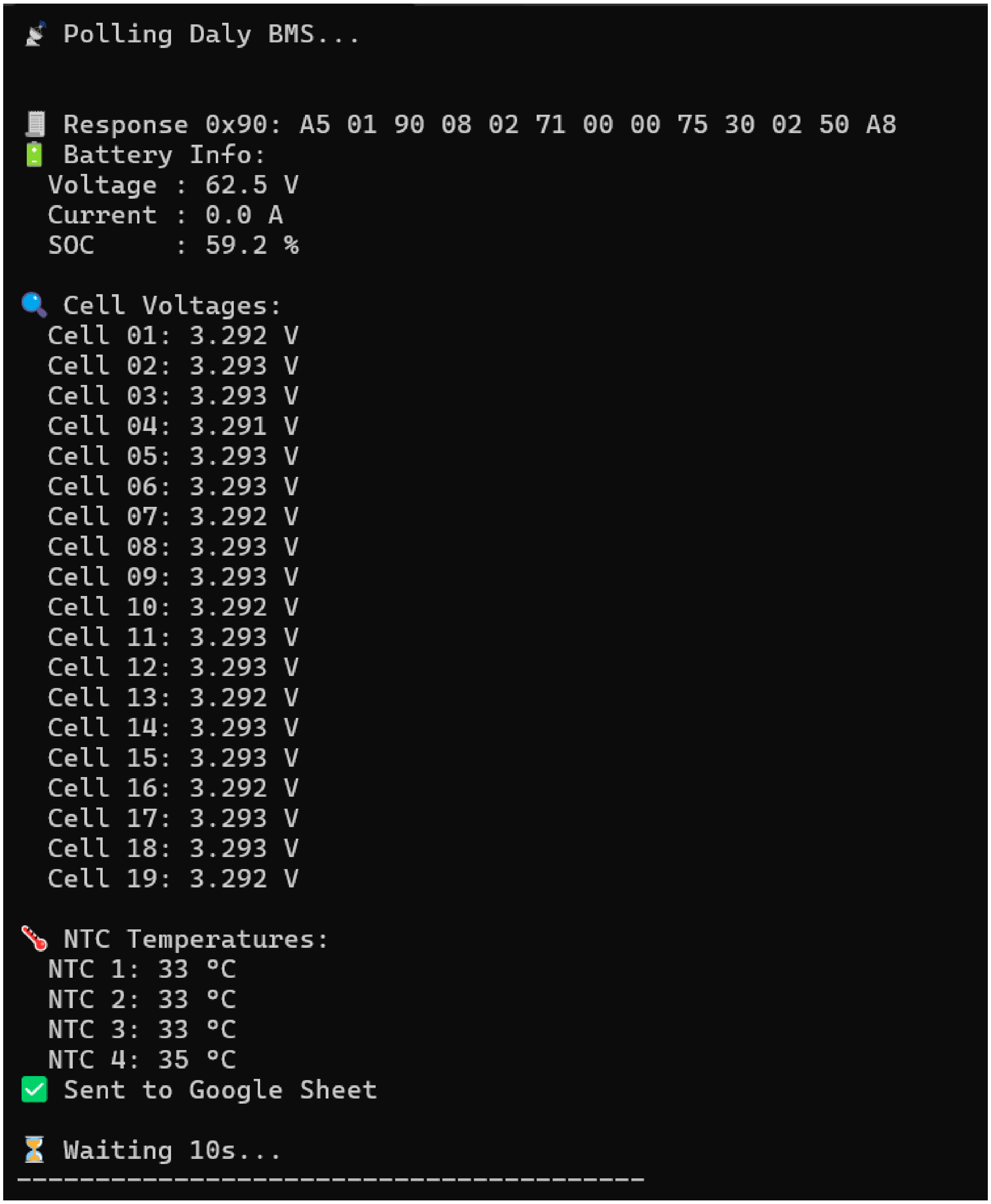
The successful implementation of the Smart EV Battery Monitoring and Charging Recommendation System demonstrates how real-time data from a Battery Management System (BMS) can be captured, processed, and visualized using low-cost hardware and cloud tools. The following subsections present the key outcomes of the system, focusing on data visualization, cloud integration, and system responsiveness.

4.1 Real-Time Battery Parameter Visualization

Using Python scripts on a Raspberry Pi, battery parameters such as voltage, current, SoC, and temperatures from NTC sensors were successfully decoded from the Daly BMS via RS485 communication. These values were printed in a clean and readable format on the console every 10 seconds. For instance:

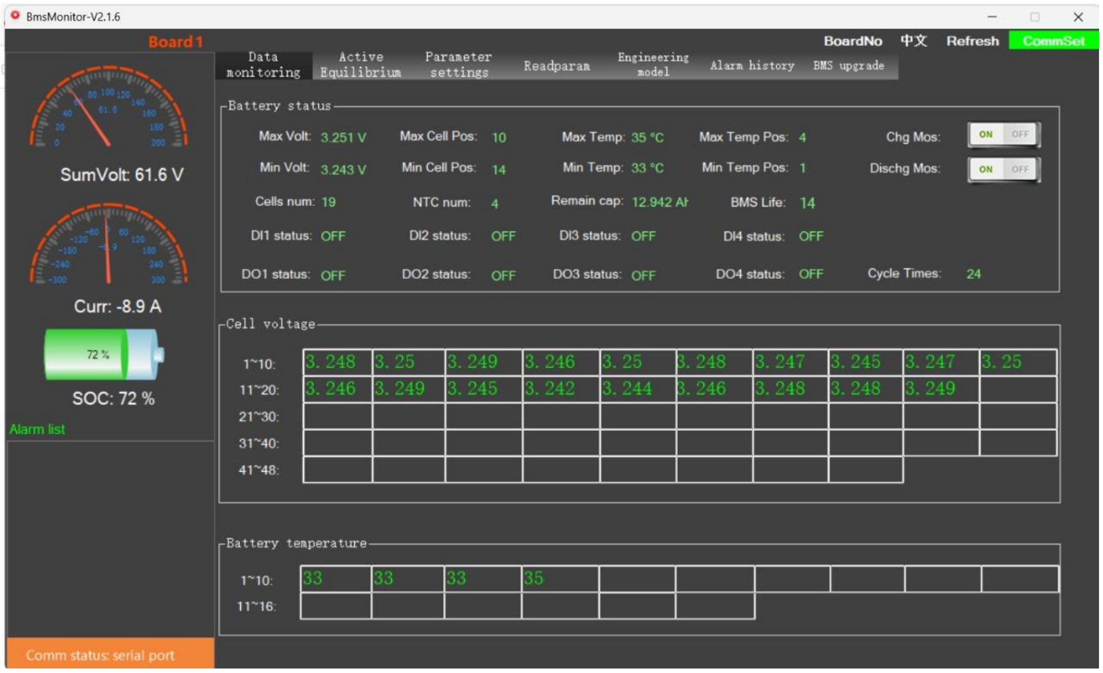
- Voltage: 51.2 V
- Current: -3.5 A (discharging)
- SoC: 74.0%
- Temperature: 26–29 °C

This real-time feedback loop allows immediate observation of the battery's condition, which is essential for identifying patterns such as fast discharges or overheating. This local visualization is crucial for development, debugging, and verifying sensor accuracy.



```
📡 Polling Daly BMS...


🧾 Response 0x90: A5 01 90 08 02 71 00 00 75 30 02 50 A8
🔋 Battery Info:
  Voltage : 62.5 V
  Current : 0.0 A
  SOC     : 59.2 %

🔍 Cell Voltages:
  Cell 01: 3.292 V
  Cell 02: 3.293 V
  Cell 03: 3.293 V
  Cell 04: 3.291 V
  Cell 05: 3.293 V
  Cell 06: 3.293 V
  Cell 07: 3.292 V
  Cell 08: 3.293 V
  Cell 09: 3.293 V
  Cell 10: 3.292 V
  Cell 11: 3.293 V
  Cell 12: 3.293 V
  Cell 13: 3.292 V
  Cell 14: 3.293 V
  Cell 15: 3.293 V
  Cell 16: 3.292 V
  Cell 17: 3.293 V
  Cell 18: 3.293 V
  Cell 19: 3.292 V

🌡 NTC Temperatures:
  NTC 1: 33 °C
  NTC 2: 33 °C
  NTC 3: 33 °C
  NTC 4: 35 °C
✅ Sent to Google Sheet

⏳ Waiting 10s...
----------------------------------------
```

Figure 4.1.1: BMS output using python script



SumVolt: 61.6 V

Curr: -8.9 A

SOC: 72 %

Battery status

Max Volt: 3.251 V   Max Cell Pos: 10   Max Temp: 35 °C   Max Temp Pos: 4   Chg Mos: ON
Min Volt: 3.243 V   Min Cell Pos: 14   Min Temp: 33 °C   Min Temp Pos: 1   Dischg Mos: ON
Cells num: 19   NTC num: 4   Remain cap: 12.942 Ah   BMS Life: 14
DI1 status: OFF   DI2 status: OFF   DI3 status: OFF   DI4 status: OFF
DO1 status: OFF   DO2 status: OFF   DO3 status: OFF   DO4 status: OFF   Cycle Times: 24

Cell voltage

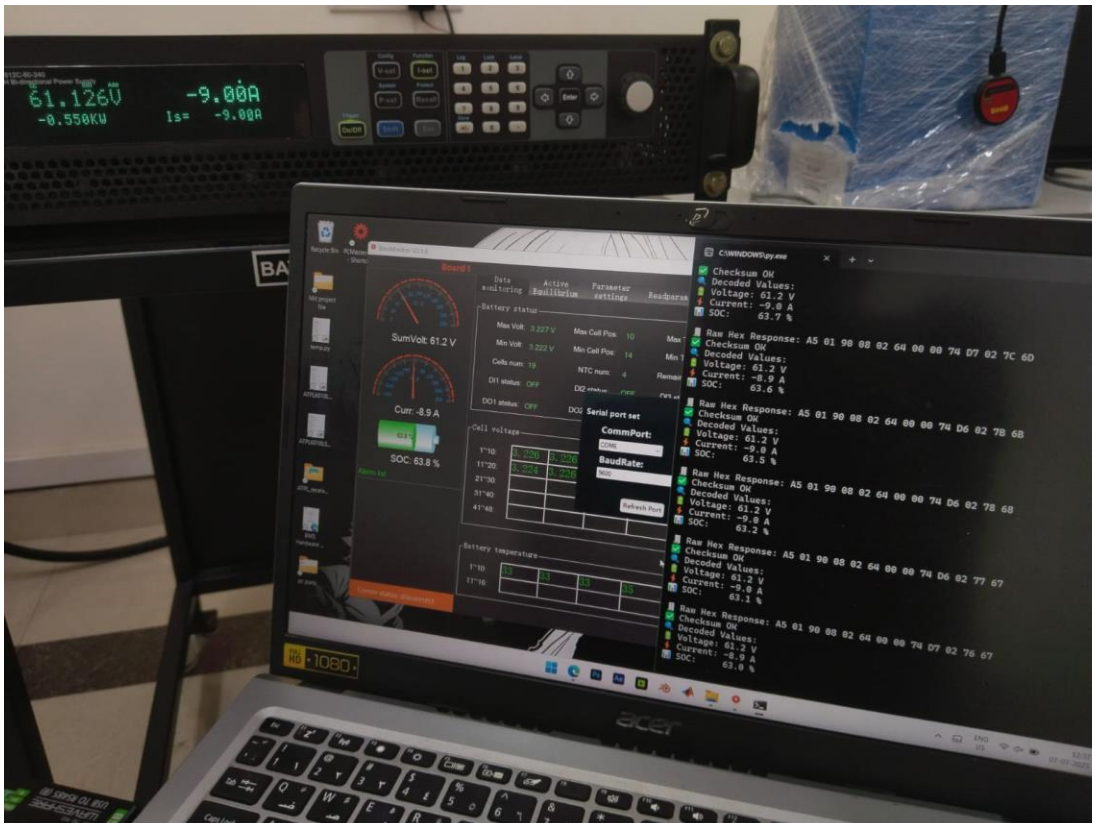
1~10:	3. 248	3. 25	3. 249	3. 246	3. 25	3. 248	3. 247	3. 245	3. 247	3. 25
11~20:	3. 246	3. 249	3. 245	3. 242	3. 244	3. 246	3. 248	3. 248	3. 249	
21~30:										
31~40:										
41~48:										

Battery temperature

1~10:	33	33	33	35						
11~16:										

Comm status: serial port

Figure 4.1.2: PC Master BMS Data



```
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:      63.7 %

Raw Hex Response: A5 01 90 08 02 64 00 00 74 D7 02 7C 6D
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -8.9 A
SOC:      63.6 %

Raw Hex Response: A5 01 90 08 02 64 00 00 74 D6 02 7B 6B
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:      63.5 %

Raw Hex Response: A5 01 90 08 02 64 00 00 74 D6 02 78 68
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:      63.2 %

Raw Hex Response: A5 01 90 08 02 64 00 00 74 D6 02 77 67
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:      63.1 %

Raw Hex Response: A5 01 90 08 02 64 00 00 74 D7 02 76 67
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -8.9 A
SOC:      63.0 %
```

Figure 4.1.3: Comparing PC master and python script

4.2 BMS Data Logging using Python

4.2.1 Overview

To monitor real-time battery parameters from the Daly Battery Management System (BMS), a Python script was developed that communicates over RS485. The script collects voltage, current, State of Charge (SoC), individual cell voltages, and temperature values. This data is parsed and uploaded to a Google Sheet using an HTTP POST request via a Google Apps Script Web App.

4.2.2 System Flowchart

The flowchart below outlines the data acquisition and upload process implemented in the Python script:

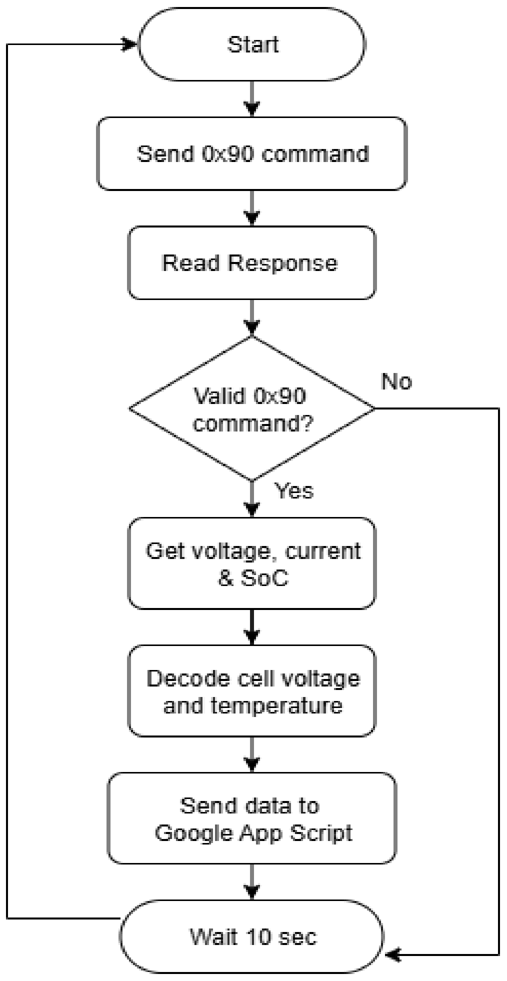

Start
Send 0x90 command
Read Response
Valid 0x90 command?
No
Yes
Get voltage, current & SoC
Decode cell voltage and temperature
Send data to Google App Script
Wait 10 sec

Figure 4.2.1: Python Script Flowchart for Battery Data Logging

4.2.3 Script Explanation

The script performs the following key functions:

- Serial Communication: Connects to the Daly BMS via RS485 on port COM6 at 9600 baud rate.
- Data Request & Response:
  Sends hex commands to fetch Voltage, current, SoC (0x90), Cell voltages (0x95), NTC temperature readings (0x96).
- Data Decoding: Raw hex values from BMS are decoded into human-readable values (e.g., volts, amps, percentage, °C).
- Cloud Logging: The decoded data is formatted as a JSON object and sent to Google Sheets through an Apps Script Web App.
- Loop: The entire process repeats every 10 seconds for continuous monitoring.

4.3 Structured Data Logging in Google Sheets

4.3.1 Overview

The pre-processed data was transmitted to Google Sheets using HTTP POST requests to a Google Apps Script Web App. Each row in the sheet represents a snapshot of battery health, logged every 10 seconds, including:

- Timestamp
- Voltage (V)
- Current (A)
- SoC (%)
- Cell voltages (1–19)
- Temperature

Figure 4.3.1: PC Master Data

Using built-in Google Sheets charts and conditional formatting, the data was visualized to track trends and highlight anomalies. For example:

- Conditional formatting turned SoC < 20% red for alerting.
- Line graphs showed gradual SoC decline during discharge.

This provided a lightweight and accessible cloud logging system without the need for complex databases or third-party platforms.

4.3.2 System Flowchart

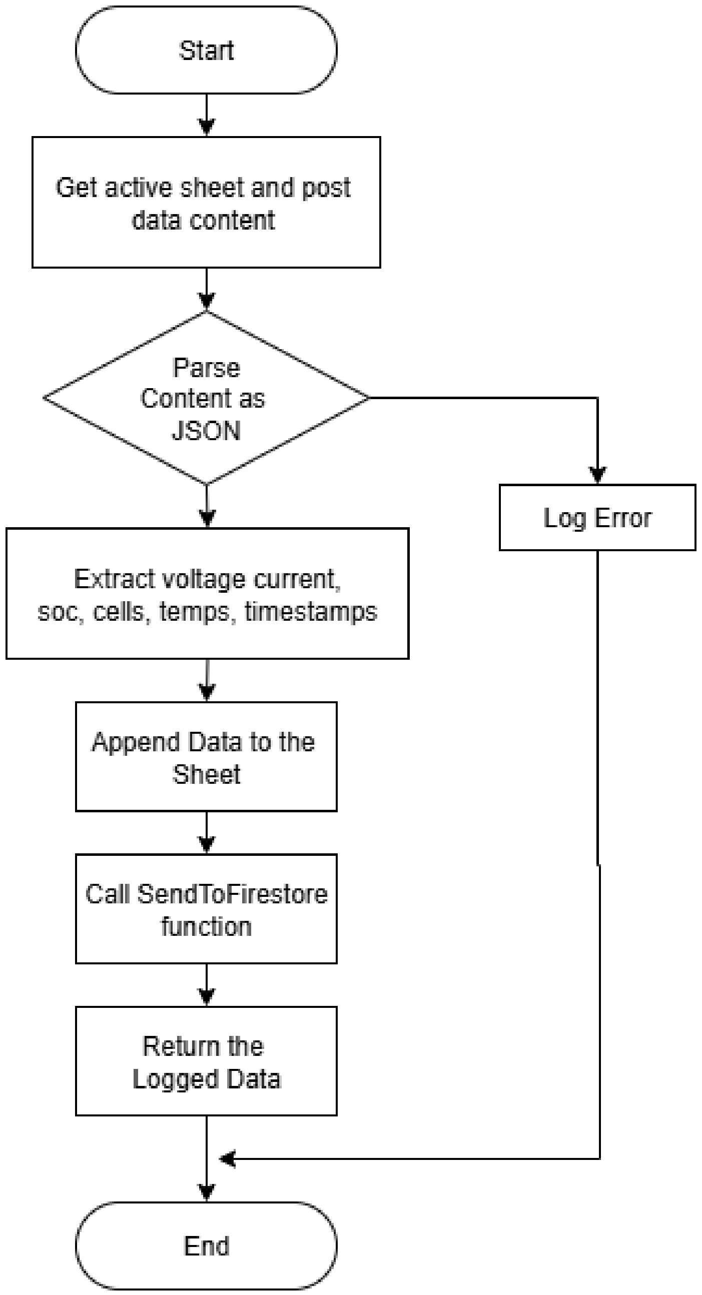
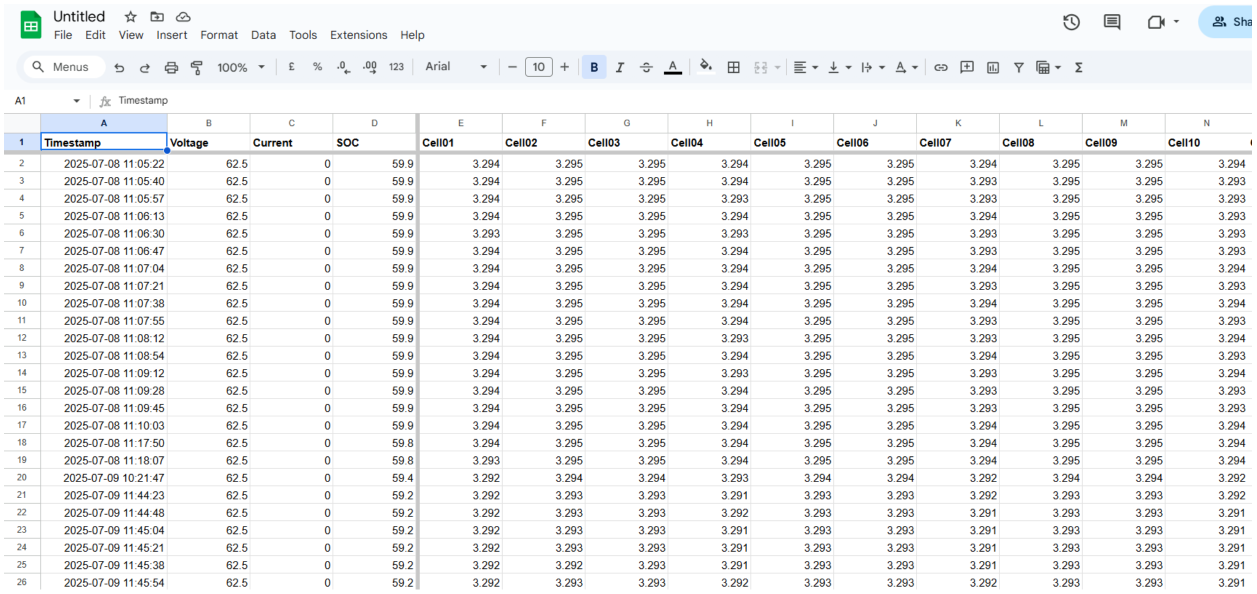
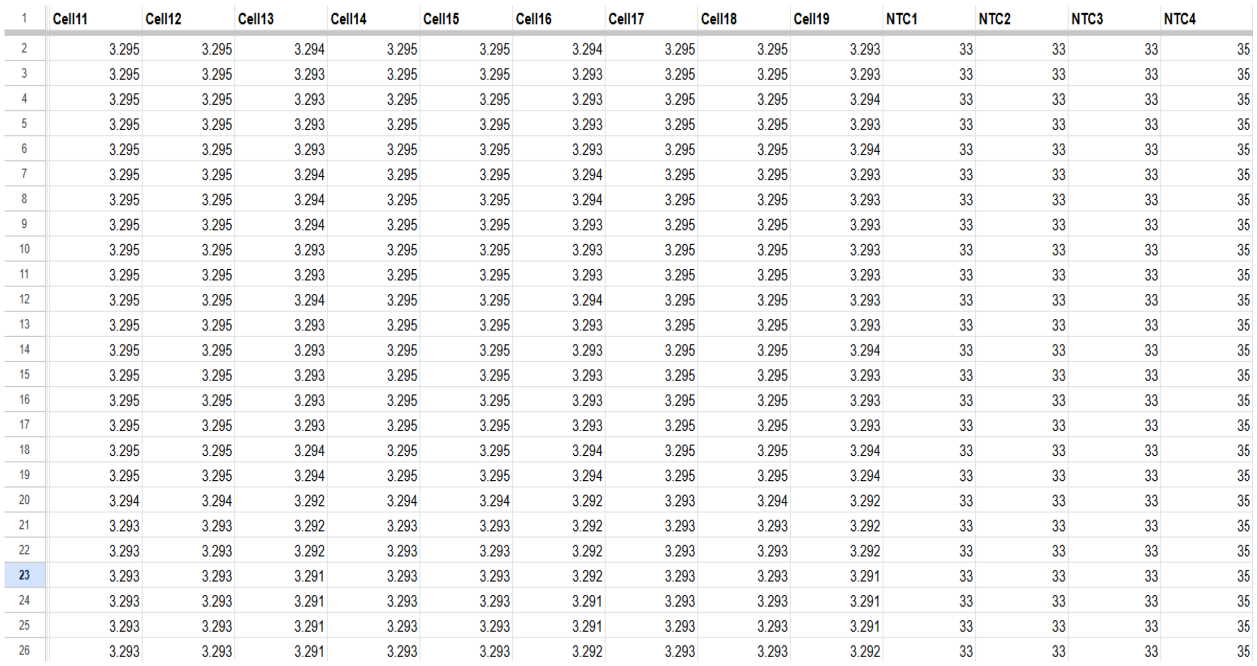

Start
Get active sheet and post data content
Parse Content as JSON
Log Error
Extract voltage current, soc, cells, temps, timestamps
Append Data to the Sheet
Call SendToFirestore function
Return the Logged Data
End

Figure 4.3.2: Flowchart of Data Logging and Fire store Upload via Google Apps Script


Untitled
File Edit View Insert Format Data Tools Extensions Help
Menus 100% £ % .0 .00 123 Arial 10
A1 Timestamp

	Timestamp	Voltage	Current	SOC	Cell01	Cell02	Cell03	Cell04	Cell05	Cell06	Cell07	Cell08	Cell09	Cell10
2	2025-07-08 11:05:22	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
3	2025-07-08 11:05:40	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
4	2025-07-08 11:05:57	62.5	0	59.9	3.294	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293
5	2025-07-08 11:06:13	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293
6	2025-07-08 11:06:30	62.5	0	59.9	3.293	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293
7	2025-07-08 11:06:47	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
8	2025-07-08 11:07:04	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
9	2025-07-08 11:07:21	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
10	2025-07-08 11:07:38	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
11	2025-07-08 11:07:55	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
12	2025-07-08 11:08:12	62.5	0	59.9	3.294	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.294
13	2025-07-08 11:08:54	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293
14	2025-07-08 11:09:12	62.5	0	59.9	3.294	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.294
15	2025-07-08 11:09:28	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
16	2025-07-08 11:09:45	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293
17	2025-07-08 11:10:03	62.5	0	59.9	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
18	2025-07-08 11:17:50	62.5	0	59.8	3.294	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
19	2025-07-08 11:18:07	62.5	0	59.8	3.293	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294
20	2025-07-09 10:21:47	62.5	0	59.4	3.292	3.294	3.294	3.293	3.294	3.294	3.292	3.294	3.294	3.292
21	2025-07-09 11:44:23	62.5	0	59.2	3.292	3.293	3.293	3.291	3.293	3.293	3.292	3.293	3.293	3.292
22	2025-07-09 11:44:48	62.5	0	59.2	3.292	3.293	3.293	3.292	3.293	3.293	3.291	3.293	3.293	3.291
23	2025-07-09 11:45:04	62.5	0	59.2	3.292	3.293	3.293	3.291	3.293	3.293	3.291	3.293	3.293	3.291
24	2025-07-09 11:45:21	62.5	0	59.2	3.292	3.293	3.293	3.291	3.293	3.293	3.291	3.293	3.293	3.291
25	2025-07-09 11:45:38	62.5	0	59.2	3.292	3.292	3.293	3.291	3.293	3.293	3.291	3.293	3.293	3.291
26	2025-07-09 11:45:54	62.5	0	59.2	3.292	3.293	3.293	3.292	3.293	3.293	3.292	3.293	3.293	3.291

Figure 4.3.3: Google sheets Receiving data from Esp32



	Cell11	Cell12	Cell13	Cell14	Cell15	Cell16	Cell17	Cell18	Cell19	NTC1	NTC2	NTC3	NTC4
2	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293	33	33	33	35
3	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
4	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.294	33	33	33	35
5	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
6	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.294	33	33	33	35
7	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293	33	33	33	35
8	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293	33	33	33	35
9	3.295	3.295	3.294	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
10	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
11	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
12	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.293	33	33	33	35
13	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
14	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.294	33	33	33	35
15	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
16	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
17	3.295	3.295	3.293	3.295	3.295	3.293	3.295	3.295	3.293	33	33	33	35
18	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294	33	33	33	35
19	3.295	3.295	3.294	3.295	3.295	3.294	3.295	3.295	3.294	33	33	33	35
20	3.294	3.294	3.292	3.294	3.294	3.292	3.293	3.294	3.292	33	33	33	35
21	3.293	3.293	3.292	3.293	3.293	3.292	3.293	3.293	3.292	33	33	33	35
22	3.293	3.293	3.292	3.293	3.293	3.292	3.293	3.293	3.292	33	33	33	35
23	3.293	3.293	3.291	3.293	3.293	3.292	3.293	3.293	3.291	33	33	33	35
24	3.293	3.293	3.291	3.293	3.293	3.291	3.293	3.293	3.291	33	33	33	35
25	3.293	3.293	3.291	3.293	3.293	3.291	3.293	3.293	3.291	33	33	33	35
26	3.293	3.293	3.291	3.293	3.293	3.292	3.293	3.293	3.292	33	33	33	35

Figure 4.3.4: Google sheets Receiving data from Esp32

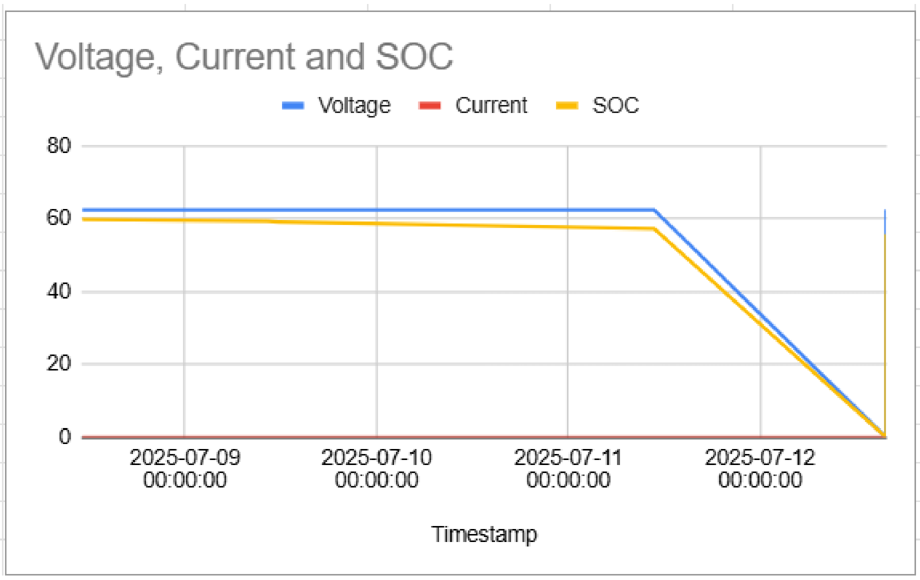


Figure 4.3.5: Voltage, Current and Soc graph

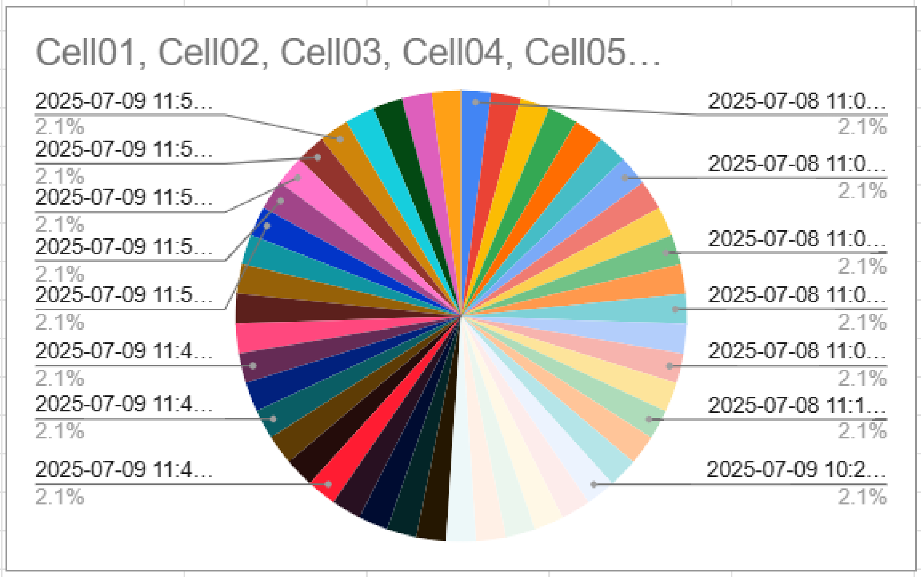


Figure 4.3.6: Data of 19 cells and their voltage

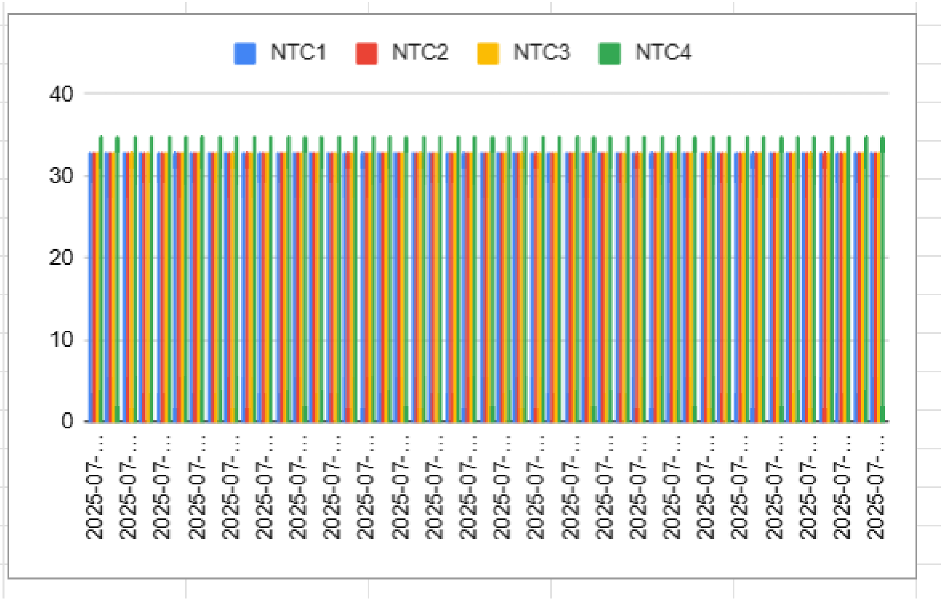


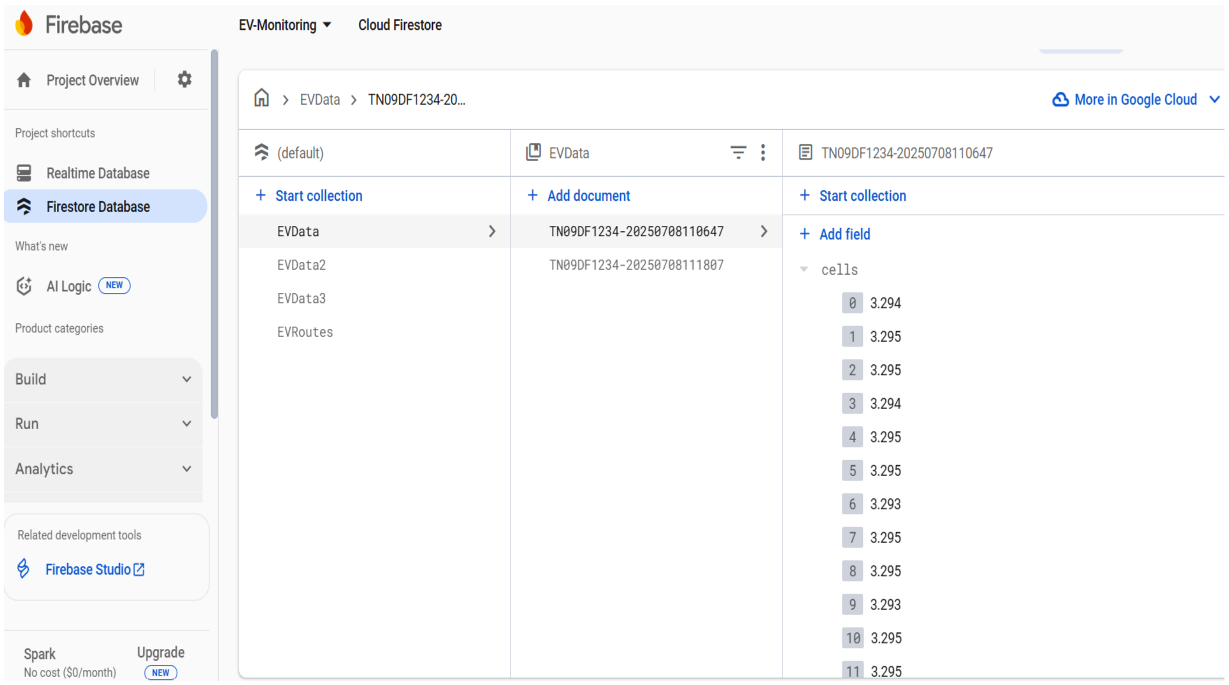
Figure 4.3.7: Four temperature sensor data

4.4 Syncing Data to Firebase Fire store (Optional Integration)

To support future scalability and multi-device integration, an optional step was tested where the same data was pushed to Firebase Fire store. This allows for:

- Centralized storage for multiple EVs
- Querying based on vehicle ID, location, or time
- Triggering serverless functions for alerts

The Firebase SDK or Apps Script can be extended to sync from Google Sheets to Fire store or directly from Raspberry Pi using the Firebase REST API. While not the core focus, this step establishes the groundwork for future multi-user dashboards or AI-based scheduling logic.



EVData > TN09DF1234-20...

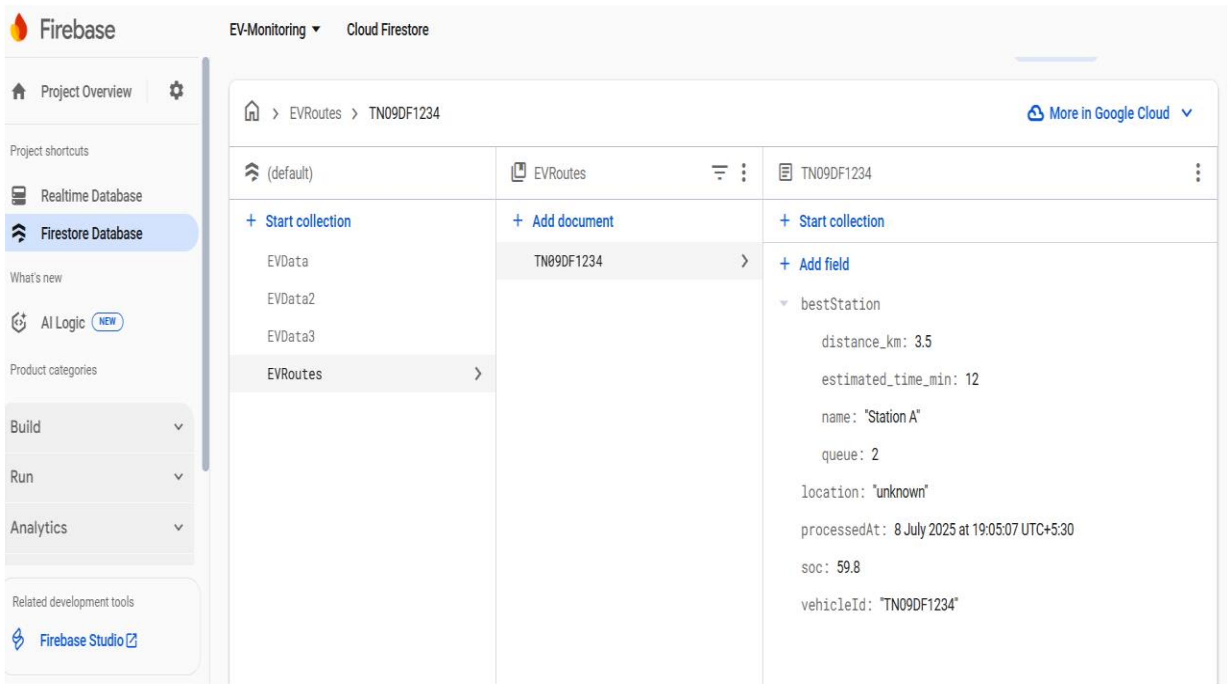
(default): EVData, EVData2, EVData3, EVRoutes

EVData: TN09DF1234-20250708110647, TN09DF1234-20250708111807

TN09DF1234-20250708110647

cells
0 3.294
1 3.295
2 3.295
3 3.294
4 3.295
5 3.295
6 3.293
7 3.295
8 3.295
9 3.293
10 3.295
11 3.295

Figure 4.4.1: Firebase data collection dashboards



EVRoutes > TN09DF1234

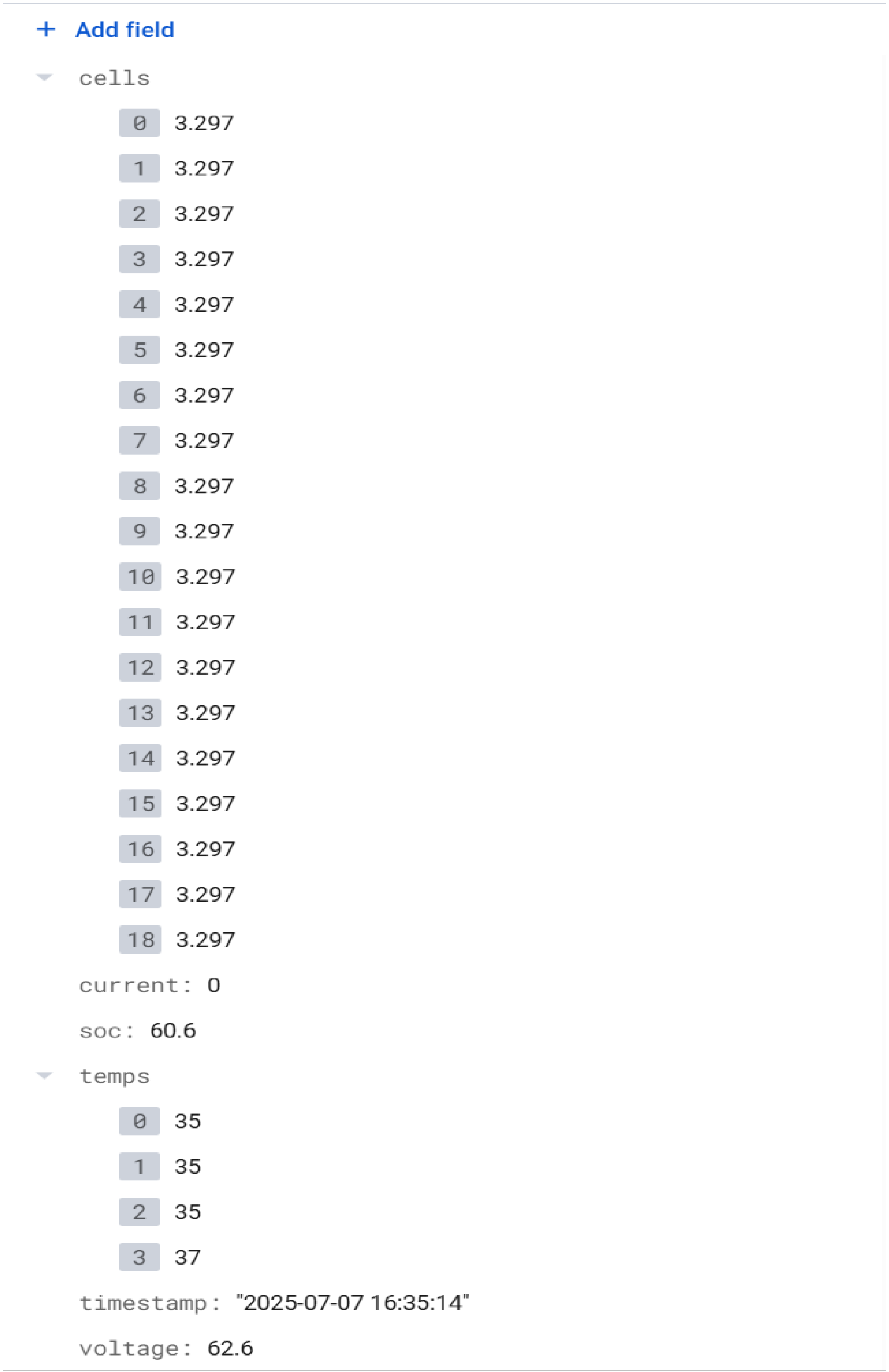
(default): EVData, EVData2, EVData3, EVRoutes

EVRoutes: TN09DF1234

TN09DF1234

bestStation
distance_km: 3.5
estimated_time_min: 12
name: "Station A"
queue: 2
location: "unknown"
processedAt: 8 July 2025 at 19:05:07 UTC+5:30
soc: 59.8
vehicleId: "TN09DF1234"

Figure 4.4.2: Firebase data receiving the best Charging station (future implementation)



```
cells
    0  3.297
    1  3.297
    2  3.297
    3  3.297
    4  3.297
    5  3.297
    6  3.297
    7  3.297
    8  3.297
    9  3.297
    10 3.297
    11 3.297
    12 3.297
    13 3.297
    14 3.297
    15 3.297
    16 3.297
    17 3.297
    18 3.297
current: 0
soc: 60.6
temps
    0  35
    1  35
    2  35
    3  37
timestamp: "2025-07-07 16:35:14"
voltage: 62.6
```

Figure 4.4.3: Collection Field in Fire Store:

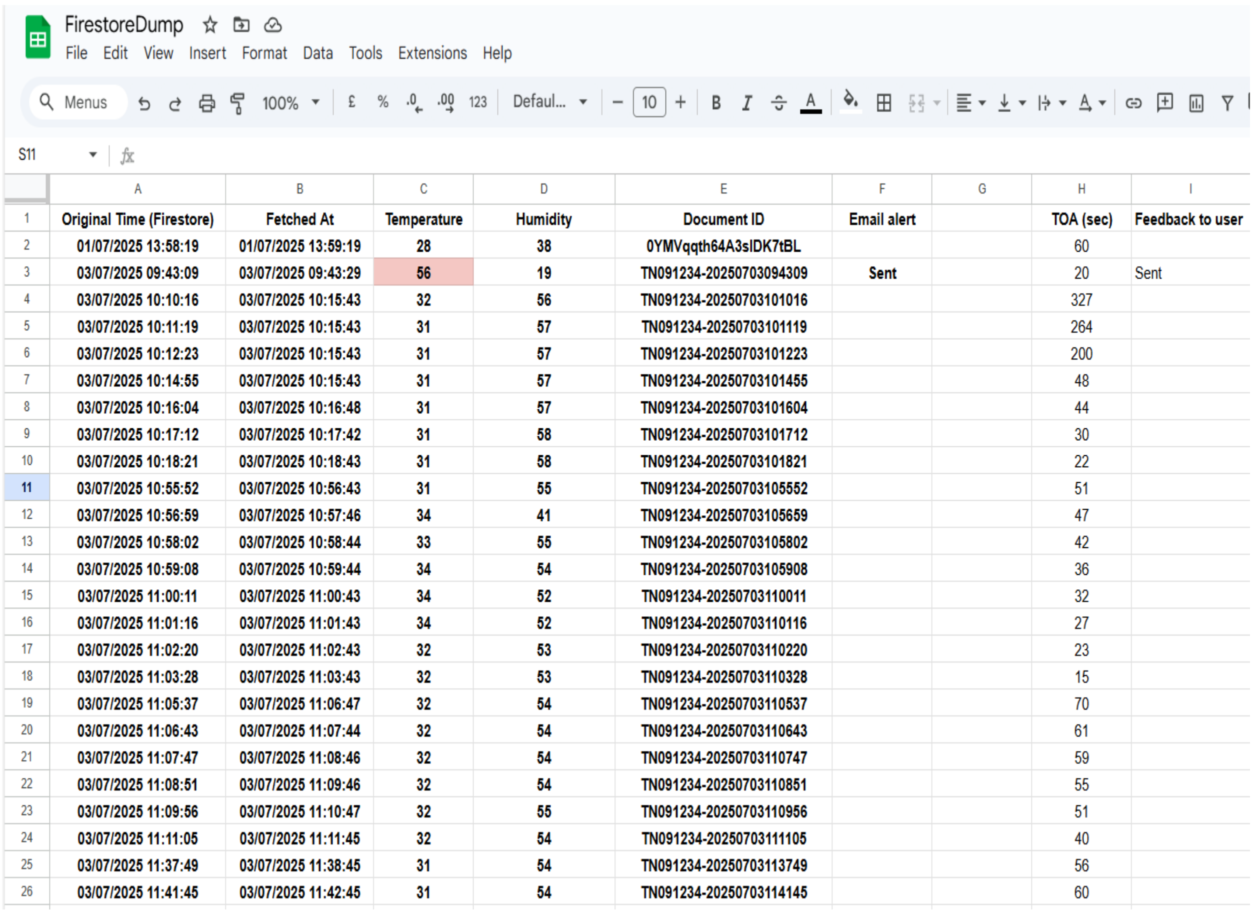
4.5 Central Data Retrieval at IoT Main Station

4.5.1 Overview

A simulated IoT central station (a separate dashboard sheet or app) was created to retrieve data from all EVs through Firebase or Google Sheets. This main station:

- Fetched live SoC, temperature, and location (if available)
- Compared available charging stations
- Estimated waiting queues and assigned best station

Although fully autonomous scheduling wasn't implemented, the design supports integration with GPS modules and server-side decision logic. The main station also acts as a hub for fleet management in future real-world deployment.



	A	B	C	D	E	F	G	H	I
1	Original Time (Firestore)	Fetched At	Temperature	Humidity	Document ID	Email alert		TOA (sec)	Feedback to user
2	01/07/2025 13:58:19	01/07/2025 13:59:19	28	38	0YMVqqth64A3sIDK7tBL			60	
3	03/07/2025 09:43:09	03/07/2025 09:43:29	56	19	TN091234-20250703094309	Sent		20	Sent
4	03/07/2025 10:10:16	03/07/2025 10:15:43	32	56	TN091234-20250703101016			327	
5	03/07/2025 10:11:19	03/07/2025 10:15:43	31	57	TN091234-20250703101119			264	
6	03/07/2025 10:12:23	03/07/2025 10:15:43	31	57	TN091234-20250703101223			200	
7	03/07/2025 10:14:55	03/07/2025 10:15:43	31	57	TN091234-20250703101455			48	
8	03/07/2025 10:16:04	03/07/2025 10:16:48	31	57	TN091234-20250703101604			44	
9	03/07/2025 10:17:12	03/07/2025 10:17:42	31	58	TN091234-20250703101712			30	
10	03/07/2025 10:18:21	03/07/2025 10:18:43	31	58	TN091234-20250703101821			22	
11	03/07/2025 10:55:52	03/07/2025 10:56:43	31	55	TN091234-20250703105552			51	
12	03/07/2025 10:56:59	03/07/2025 10:57:46	34	41	TN091234-20250703105659			47	
13	03/07/2025 10:58:02	03/07/2025 10:58:44	33	55	TN091234-20250703105802			42	
14	03/07/2025 10:59:08	03/07/2025 10:59:44	34	54	TN091234-20250703105908			36	
15	03/07/2025 11:00:11	03/07/2025 11:00:43	34	52	TN091234-20250703110011			32	
16	03/07/2025 11:01:16	03/07/2025 11:01:43	34	52	TN091234-20250703110116			27	
17	03/07/2025 11:02:20	03/07/2025 11:02:43	32	53	TN091234-20250703110220			23	
18	03/07/2025 11:03:28	03/07/2025 11:03:43	32	53	TN091234-20250703110328			15	
19	03/07/2025 11:05:37	03/07/2025 11:06:47	32	54	TN091234-20250703110537			70	
20	03/07/2025 11:06:43	03/07/2025 11:07:44	32	54	TN091234-20250703110643			61	
21	03/07/2025 11:07:47	03/07/2025 11:08:46	32	54	TN091234-20250703110747			59	
22	03/07/2025 11:08:51	03/07/2025 11:09:46	32	54	TN091234-20250703110851			55	
23	03/07/2025 11:09:56	03/07/2025 11:10:47	32	55	TN091234-20250703110956			51	
24	03/07/2025 11:11:05	03/07/2025 11:11:45	32	54	TN091234-20250703111105			40	
25	03/07/2025 11:37:49	03/07/2025 11:38:45	31	54	TN091234-20250703113749			56	
26	03/07/2025 11:41:45	03/07/2025 11:42:45	31	54	TN091234-20250703114145			60	

Figure 4.5.1: IOT main station Data collection sheet with email alert and indication for user feedback

4.5.2 System Flowchart

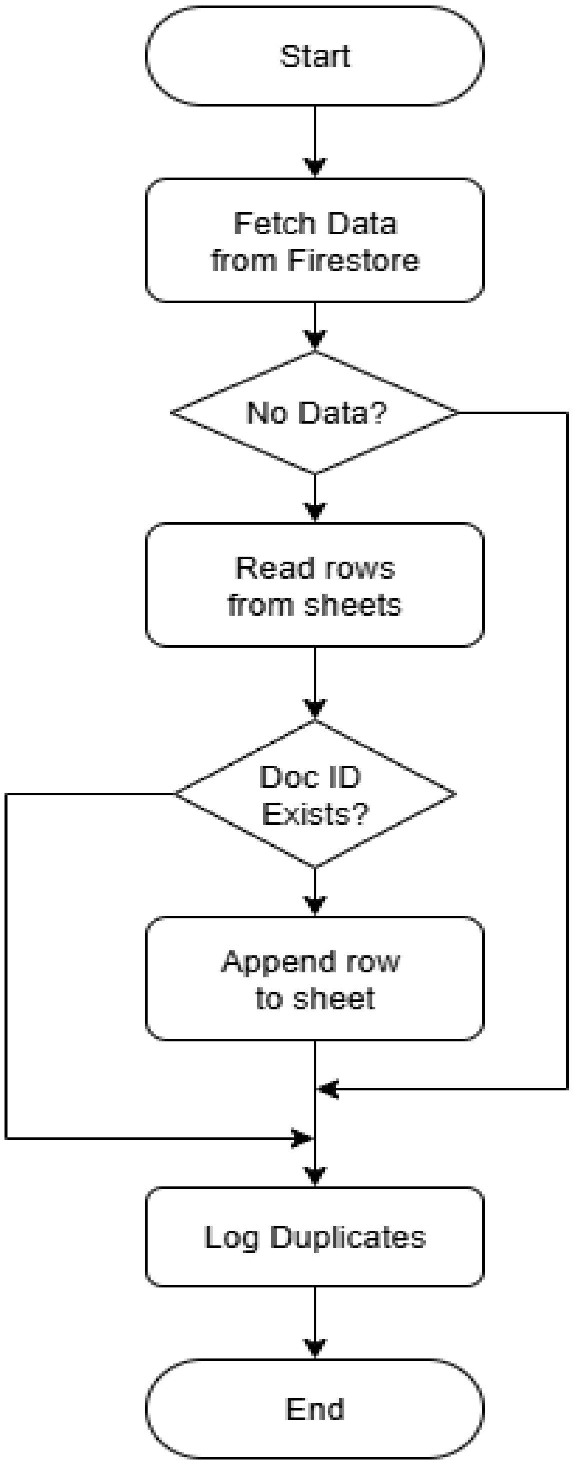
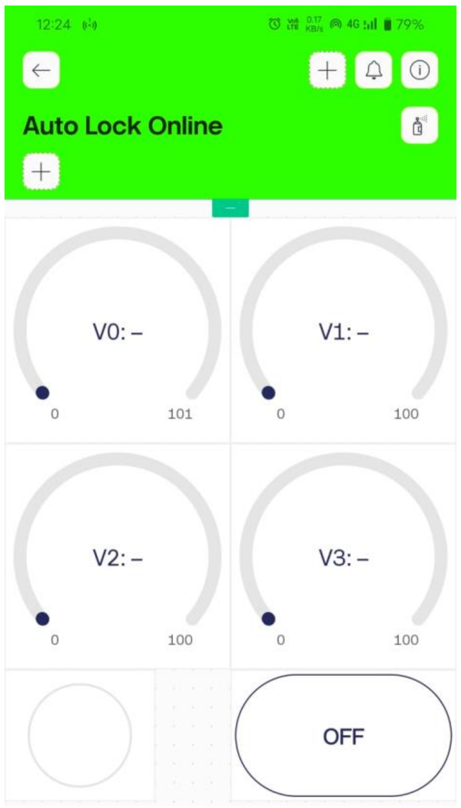


Figure 4.5.2: Flowchart of Google Apps Script for EV Data Fetching, Filtering, and Alert Management

4.6 Status and Alerts in Blynk Console

The Blynk IoT platform was used to simulate user notifications. The latest values were displayed in the Blynk mobile app using:

- Gauge widgets for SoC and voltage
- Indicator LEDs for alerts (e.g., SoC < 20%)
- Push notifications for urgent cases



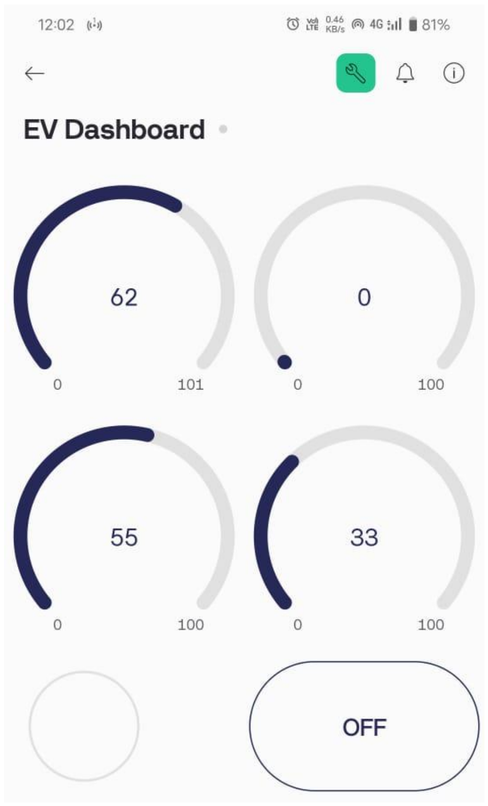


Figure 4.6.1: Blynk console android app Virtual pin configuration and Data receiving view

Every time data was uploaded to Google Sheets, it was also mirrored to the Blynk console using a simple API or webhook. This allowed remote users to get live battery insights directly on their phones without accessing technical dashboards.

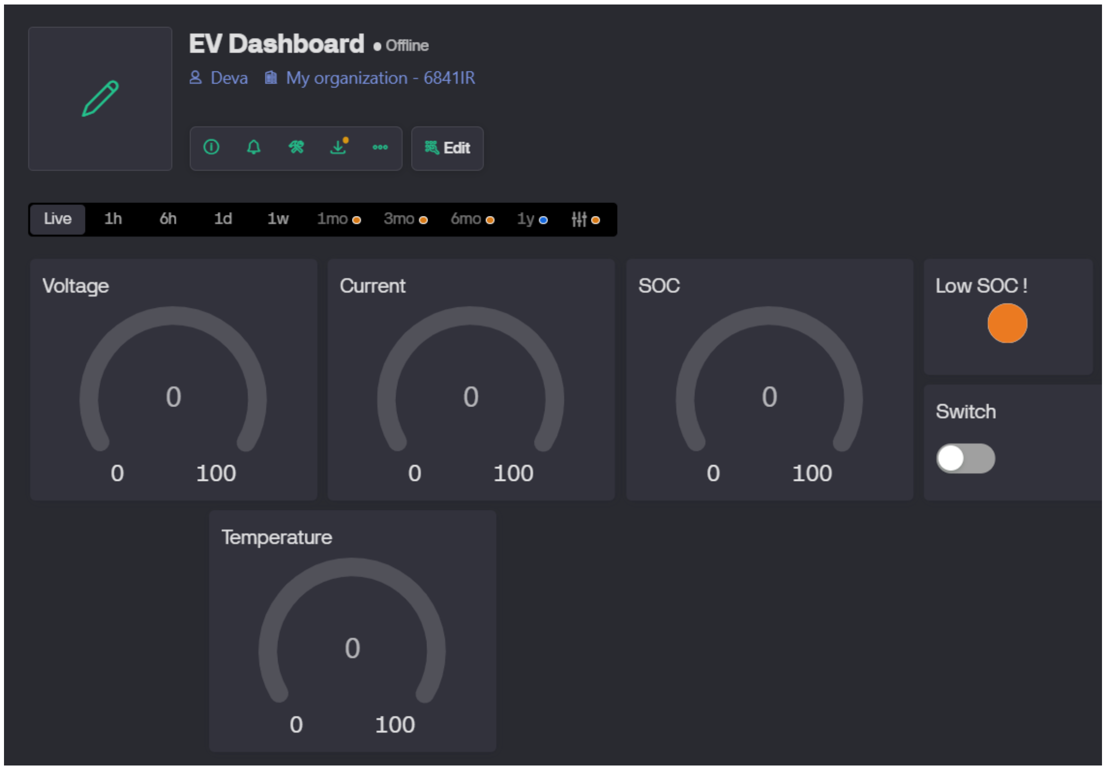


Figure 4.6.2: Blynk console web dashboard

4.7 SoC-Based LED Indication Logic (20–80 Rule)

To encourage healthy battery usage and maintain the optimal State of Charge (SoC) range, the system includes an LED-based alert mechanism within the Blynk Console. The logic is designed to give users early visual indications during both charging and discharging cycles:

- When the battery is **discharging** and the **SoC falls below 40%**, a **yellow LED** indicator is activated to notify the user that charging will soon be required.
- When the battery is **charging** and the **SoC exceeds 60%**, a **blue LED** indicator is triggered to remind the user that disconnection may be needed soon.

These thresholds serve as **pre-emptive warnings**, aligning with the widely recommended 20–80 SoC range, which helps preserve battery life and improve charging efficiency. The LED indicators ensure that users are made aware of upcoming SoC limits in real-time without needing to constantly monitor numerical values. This enhances user experience while supporting optimal energy management practices.

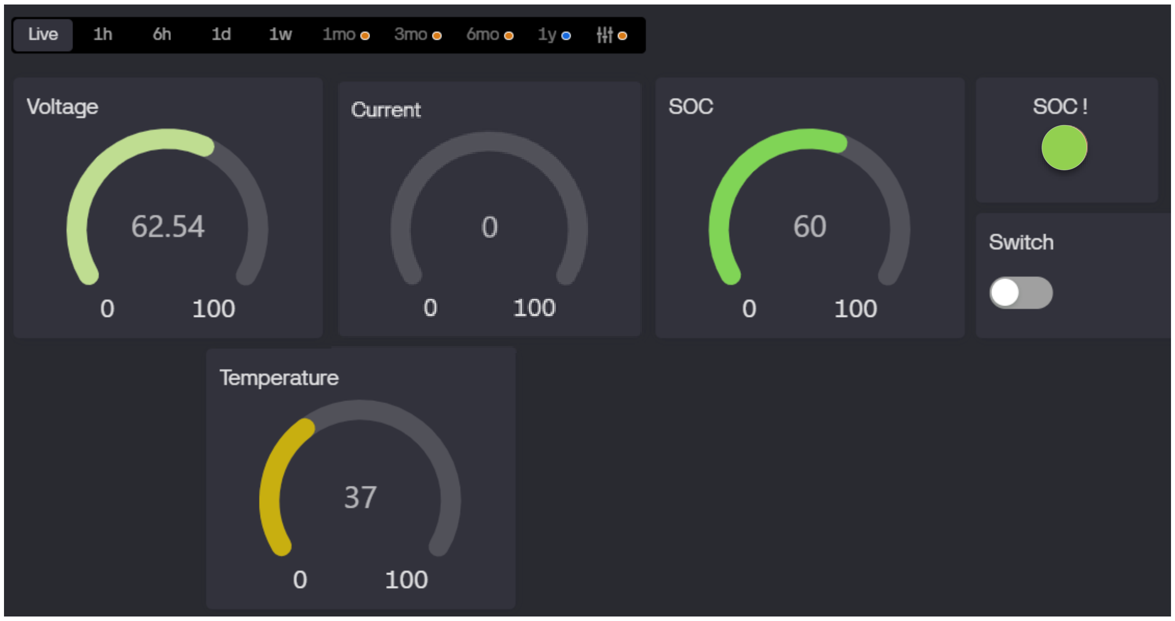


Figure 4.7.1: Alert when charging (SoC above 60)

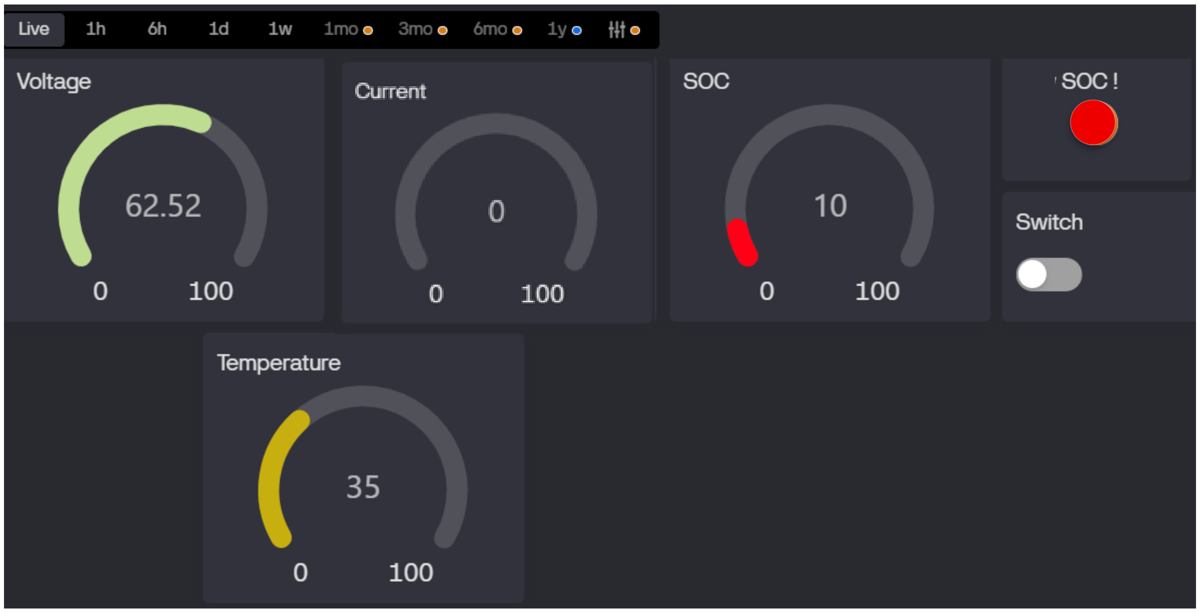


Figure 4.7.2: Alert when discharging (SoC below 40)

CHAPTER 5

SUMMARY, CONCLUSION AND FURURE WORK

5.1 SUMMARY

This project successfully demonstrates a real-time battery monitoring and cloud-integrated data logging system for Electric Vehicles (EVs) using a Daly Battery Management System (BMS). The main goal was to extract critical battery parameters such as State of Charge (SoC), total voltage, current, individual cell voltages, and temperature readings from the BMS via RS485 communication, using a Raspberry Pi and a USB-to-RS485 converter.

Using Python scripts, the Raspberry Pi periodically polled the BMS with specific hexadecimal request frames, decoded the received raw hex responses, and converted them into human-readable values. These values were then validated and formatted into structured JSON and sent via HTTP POST to a Google Apps Script Web App. This allowed seamless logging of battery data into Google Sheets, which served as a lightweight and effective cloud database. Visualization tools in Google Sheets (charts, conditional formatting) helped in observing trends such as SoC drop, or temperature rise in real time.

5.2 CONCLUSION

To enhance user accessibility, the same data was also pushed to the Blynk IoT console, enabling remote monitoring of battery health parameters through a mobile interface. Blynk widgets displayed values in real time, while push notifications or alerts could be triggered under specific conditions (e.g., SoC < 20%).

In conclusion, the project achieved its objective of building a reliable, low-cost, and scalable EV battery monitoring system. By leveraging open-source tools and simple hardware, the system enables proactive battery management, data logging, and real-time visualization—laying the groundwork for more intelligent electric vehicle infrastructure to increase the SoH of the Battery

5.3 FUTURE WORK

Although the current system meets the core functionality of monitoring and logging, several extensions can transform it into a full-fledged smart EV management platform. The following future improvements are envisioned:

- Charging Station Communication Future versions should enable direct communication with charging stations to retrieve dynamic information like charger availability, queue length, and power output. This two-way interaction will allow the EV to make data-driven decisions about when and where to charge.

- Route Optimization and Scheduling Algorithms can be developed to recommend the best charging station based on real-time SoC, GPS location, station distance, expected queue times, and power output. These may include shortest-path algorithms, traffic-aware scheduling, or optimization heuristics for fleet-level decisions.

- GPS Integration for Smart Decision-Making Adding a GPS module to the Raspberry Pi will allow real-time location tracking of the EV. Combined with the charging station data, this can enable dynamic route recommendations and geofenced alerts for charging needs.

- Custom Mobile App Development A dedicated mobile app can be created to replace Blynk, offering better UI/UX and expanded features like map-based station search, charging session history, and personalized alerts. This also opens the door to integrating payment or reservation systems.

- Predictive Maintenance using Machine Learning With continuous data collection, machine learning models such as LSTM networks can be trained to detect anomalies, predict battery degradation, and estimate Remaining Useful Life (RUL)—leading to safer and more efficient EV operation.

These future upgrades will elevate the system from just a monitoring tool to a fully intelligent, responsive, and autonomous EV charging assistant.

REFERENCES

1. Raspberry Pi Foundation (2022), *Raspberry Pi 4 Model B Technical Specifications*, Raspberry Pi Ltd.

2. Google Developers (2023), *Apps Script Web Apps – Creating and Deploying Web Applications*, https://developers.google.com/apps-script/guides/web.

3. Espinosa, F., & Tan, S. (2020), "Real-time Monitoring and Logging of Li-Ion Battery Health Parameters using Python and Raspberry Pi", *Journal of Embedded Systems*, Vol. 14, No. 2, pp. 112–119.

4. Chatterjee A., Sinha R., (2019), "IoT-based Battery Health Monitoring for Electric Vehicles", *International Journal of Energy Research*, Vol. 43, No. 6, pp. 2842–2855.

5. Kaur, G., and Singh, J. (2021), "A Review on Applications of Blynk IoT Platform for Remote Monitoring Systems", *International Conference on Smart Technologies*, Vol. 12, No. 4, pp. 65–70.

6. Mahajan, S., Kumar, A. (2022), "Design and Implementation of Google Sheets-based IoT Logging System using Raspberry Pi", *International Journal of Computer Applications*, Vol. 181, No. 31, pp. 25–30.

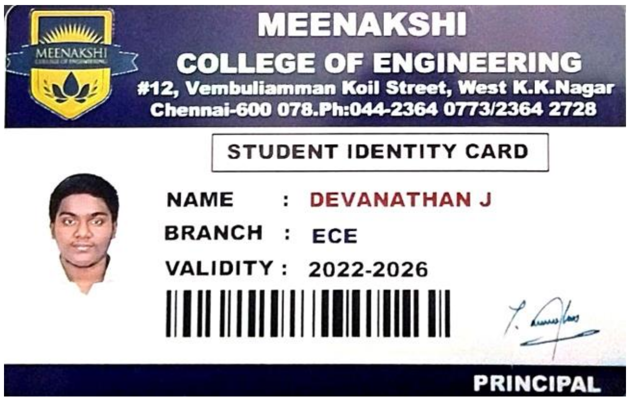
7. Lien, C. H., Lin, Y. Y. (2020), "Smart Charging Architecture for Electric Vehicles Based on GPS and SoC Data", *IEEE Access*, Vol. 8, pp. 119765–119776.

8. Sharma, V., Patel, D. (2023), "A Smart EV Charging Scheduling Model Using Distance, Queue, and SoC Constraints", *Journal of Electrical and Computer Engineering*, Vol. 2023, Article ID 9234567, pp. 1–9.

9. You, W., Lee, J., & Park, H. (2018), "Low-cost Implementation of RS485-based Industrial Sensor Communication Using Python", *Sensors and Actuators A: Physical*, Vol. 271, pp. 240–246.

APPENDIX

/



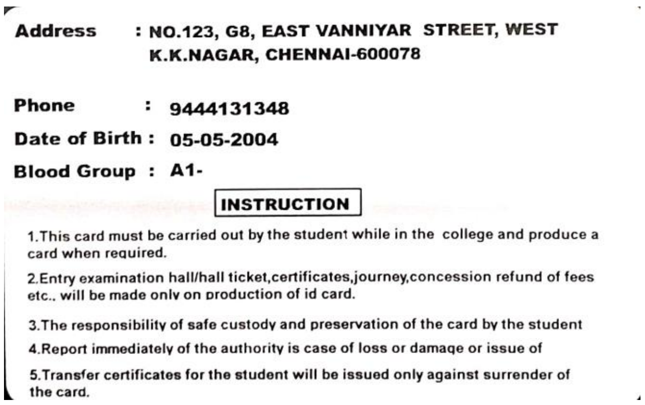
MEENAKSHI COLLEGE OF ENGINEERING
#12, Vembuliamman Koil Street, West K.K.Nagar
Chennai-600 078.Ph:044-2364 0773/2364 2728

STUDENT IDENTITY CARD

NAME	:	DEVANATHAN J
BRANCH	:	ECE
VALIDITY	:	2022-2026

[illegible]

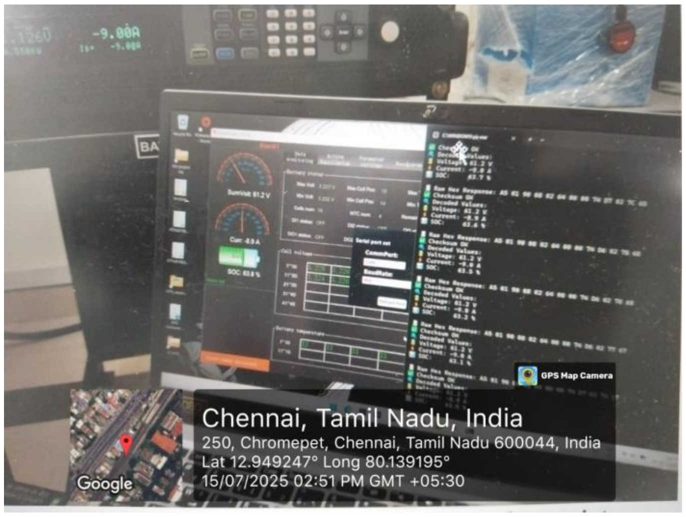
PRINCIPAL



Address	:	NO.123, G8, EAST VANNIYAR STREET, WEST K.K.NAGAR, CHENNAI-600078
Phone	:	9444131348
Date of Birth	:	05-05-2004
Blood Group	:	A1-

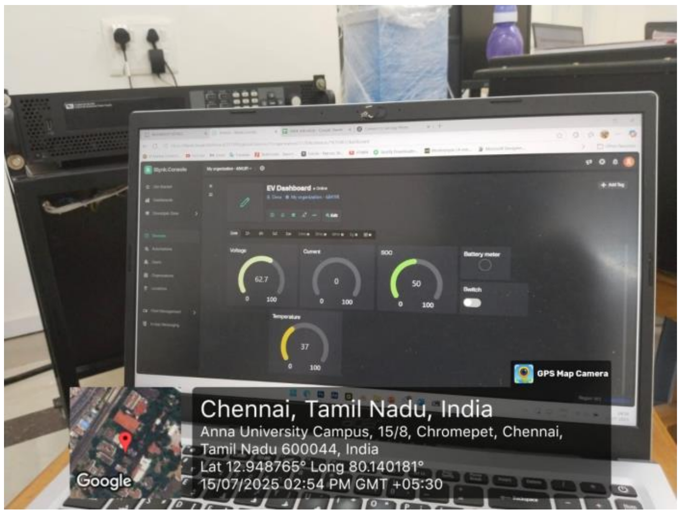
INSTRUCTION

1.This card must be carried out by the student while in the college and produce a card when required.
2.Entry examination hall/hall ticket,certificates,journey,concession refund of fees etc., will be made only on production of id card.
3.The responsibility of safe custody and preservation of the card by the student
4.Report immediately of the authority is case of loss or damage or issue of
5.Transfer certificates for the student will be issued only against surrender of the card.



```
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:     63.7 %

Raw Hex Response: [illegible]
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -8.9 A
SOC:     63.6 %

Raw Hex Response: [illegible]
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:     63.5 %

Raw Hex Response: [illegible]
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:     63.2 %

Raw Hex Response: [illegible]
Checksum OK
Decoded Values:
Voltage: 61.2 V
Current: -9.0 A
SOC:     63.1 %

Raw Hex Response: [illegible]
Checksum OK
Decoded Values:
```

GPS Map Camera
Chennai, Tamil Nadu, India
250, Chromepet, Chennai, Tamil Nadu 600044, India
Lat 12.949247° Long 80.139195°
15/07/2025 02:51 PM GMT +05:30
Google



EV Dashboard
Voltage
62.7
Current
0
SOC
50
Battery meter
Switch
Temperature
37
GPS Map Camera
Chennai, Tamil Nadu, India
Anna University Campus, 15/8, Chromepet, Chennai, Tamil Nadu 600044, India
Lat 12.948765° Long 80.140181°
15/07/2025 02:54 PM GMT +05:30
Google