TECHNISCHE UNIVERSITÄT CHEMNITZ

Department of Electrical Engineering and Information Technology

Chair of Measurement and Sensor Technology

Project Documentation

"Project Lab Embedded Systems"

Group:	15
Members:	Singarayacheruvu Sreekar - 846217
	Karthick Sri Sreniketh - 856968
	Akanksha Undale - 837125
	Kanchana Nagakumar - 862624
Project:	Smart Battery Management System for Li-ion Batteries.
Date:	02-07-2025

Table of Contents

List of Figures

List of Tables

1 Abstract

This project presents the development of a Smart Battery Management System (BMS) for Li-ion batteries using Arduino, integrated into a robot car platform. The system combines Coulomb counting with voltage and current sensing to monitor the battery's State of Charge (SoC) accurately. A key feature is the use of impedance measurement to assess battery's State of Health (SoH) and detect degradation early. Arduino handles real-time data acquisition and control, making the system efficient and modular. The project enhances battery safety and reliability while offering hands-on experience in embedded systems, electronics, and smart energy management.

2 Member Responsibilities

Singarayacheruvu Sreekar: -

Robot Car Assembly and Hardware: Collaborated on assembling the robot car, leading mechanical and electrical integration, and tested its stability as a BMS testbed.

Circuit and Code Implementation: Designed and built circuits for current/voltage and impedance measurements; implemented Arduino C++ code for Coulomb Counting, measurement data processing, and impedance spectroscopy on Arduino.

LTspice Simulation: Conducted LTspice simulations for current/voltage measurement and impedance circuits, optimizing component values and validating performance against practical results.

Karthick Sri Sriniketh: -

Robot Car Assembly Support: Collaborated on assembling the robot car, focusing on electrical connections and testing its compatibility with BMS circuitry.

Conducted theoretical research to design mathematical models and frameworks for Coulomb Counting and current/voltage measurement, providing key insights that informed both software development and circuit design.

LTspice Simulation: Performed LTspice simulations for measurement circuits, incorporating theoretical insights and validating designs with practical outcomes.

Akanksha Undale: -

Robot Car Assembly Support**:** Contributed to assembling the robot car, assisting with battery integration and testing its functionality as a BMS platform.

Impedance Spectroscopy Research: Researched theoretical models for impedance spectroscopy, providing mathematical frameworks and data visualization methods for battery health analysis.

Software and Documentation Support: Supported impedance spectroscopy code development with algorithms and provided theoretical insights for documentation.

Kanchana Nagakumar: -

Robot Car Assembly and Research Support: Collaborated on assembling the robot car, ensuring battery and circuitry integration, and assisted with research on impedance spectroscopy, including data analysis and visualization techniques.

Led the development of comprehensive project documentation, including technical reports, Presentations, design overviews, and supporting visual materials to guide the project's progress and communication. Created detailed Fritzing schematics for BMS circuits, enabling clear visualization.

Team Coordination: Organized team meetings, managed a shared repository for code and documentation, and facilitated communication among team members for task alignment.

3 Introduction

Lithium-ion batteries are widely used in modern electronic devices and electric vehicles due to their high energy density and long cycle life. However, their performance and safety heavily depend on effective battery management. This project focuses on designing and implementing a Smart Battery Management System (BMS) using an Arduino-based platform, integrated into a robot car. The primary goal is to monitor and protect the battery by accurately estimating its State of Charge (SoC), measuring real-time voltage and current, and assessing battery's State of Health (SoH) through impedance measurement.[7] By combining hardware and software components, the system ensures efficient battery usage, estimates battery life, and provides a foundation for further research in smart energy systems.[10]

4 Project Overview

The Smart Battery Management System (BMS) project aims to develop an intelligent monitoring and control system for Li-ion batteries, implemented on a robot car platform using Arduino. The main goal is to ensure battery safety, optimize performance, and extend lifespan through real-time monitoring and diagnostics. It also calculates things like the battery's charge level and remaining operating time, providing critical information to enhance energy efficiency and prevent unexpected power loss.

4.1 Tasks Description

Task 1 - Assembly of the Robot Car

This task involves assembling a mobile robot platform that will serve as the testbed for the Smart BMS. It involves assembling key components like the motors, wheels, and chassis, along with integrating the power supply to bring the whole system together.

Task 2 - Addition of Current and Voltage Measurement Circuit

This task involves designing and implementing analog or digital circuits to measure the battery's real-time voltage and current. The data will be used for monitoring, protection, and SoC calculations.

Task 3 - Implementation of Coulomb Counting

Coulomb counting is a common technique for estimating a battery’s state of charge by tracking how much current flows in and out over time. This task includes setting up a current measurement and writing Arduino code to track charge usage accurately.

Task 4 – Theoretical Research of Impedance Spectroscopy Circuit & Determination Program

Impedance measurement helps to analyze the battery health. This task includes the research performed to find a method to measure the battery's response, and filtering, then analyzing the data.

Task 5 - Documentation

This task involves compiling all project aspects— methods, designs, schematics, code, results, and analysis—into a structured document or report and presentations. It ensures the project is ready for presentation and can be taken into consideration for future development.

5 Functional Description

5.1 SoC Estimation

State of Charge (SoC): SoC refers to how much electric charge has flowed into or out of a battery relative to its full capacity.

It is generally expressed as a percentage:

$$\text{SoC}\ (\,\%\,) = \frac{\text{Remaining Capacity}}{\text{Nominal Capacity}} \times 100$$

Determining SoC:

- ➢ Chemical method
- ➢ Voltage method
- ➢ Current Integration Method (also known as Coulomb counting)
- ➢ Combined approaches
- ➢ Kalman filtering
- ➢ Pressure Method

- Chemical methods determine the state of charge (SoC) in lead-acid batteries by measuring the electrolyte's specific gravity or refractive index, which changes with sulfuric acid concentration during charge/discharge.
- The voltage method estimates battery SoC using the manufacturer-provided voltage-SOC curve, but its accuracy is limited by current and temperature effects and is less reliable for batteries with flat voltage profiles.[11]

- The current integration (coulomb counting) method estimates SoC by time-integrating current flow, but it requires periodic recalibration due to measurement drift and lack of an absolute reference.[13]
- The combined approach merges voltage and coulomb counting methods for improved SoC accuracy, as seen in Maxim's Model Gauge m3 chips like the MAX17050 used in Nexus devices.
- The Kalman filtering method combines voltage and current data with a battery model to dynamically estimate SoC, adjusting trust in each input in real time for improved accuracy.
- The pressure method estimates SoC in some NiMH batteries by detecting internal pressure rise during charging, often using a pressure switch and considering Peukert's law for improved accuracy.

Implementation method used in this project: **Combined approach (Voltage + Coulomb counting)**

As Panasonic NCR18650PF battery is used in this project, we only have Voltage vs Discharge Capacity characteristics curve, Initial SoC is determined through Voltage-based method and it is used further in Coulomb counting. Hence, we chose the combined approach.[1][15]

Coulomb counting: Coulomb Counting is a method used to estimate the SoC of a battery by integrating the current flowing into or out of the battery over time.

The general principle:

$$\mathrm{SoC}(t) = \mathrm{SoC}(0) \pm \frac{1}{C_{\text{nominal}}} \int_0^t I(t)\ dt$$

Where:

- $\mathrm{SoC}(0)$ is the initial State of Charge
- Current $I(t)$ is the battery current at time t
- C_{nominal} is the nominal capacity of the battery
- "+" is used during charging, "-" is used during discharging

Coulomb Counting Equations:

During Charging:

$$\mathrm{SoC}(t) = \mathrm{SoC}(0) + \frac{1}{C_{\text{nominal}}} \int_0^t I(t)\ dt$$

During Discharging:

$$\mathrm{SoC}(t) = \mathrm{SoC}(0) - \frac{1}{C_{\text{nominal}}} \int_0^t I(t)\ dt$$

Determination of Initial SoC:

Coulomb counting requires an accurate initial SoC. This is typically determined using the battery's voltage vs discharge capacity curve provided in the manufacturer's datasheet.

Using Manufacturer's Discharge Curve:

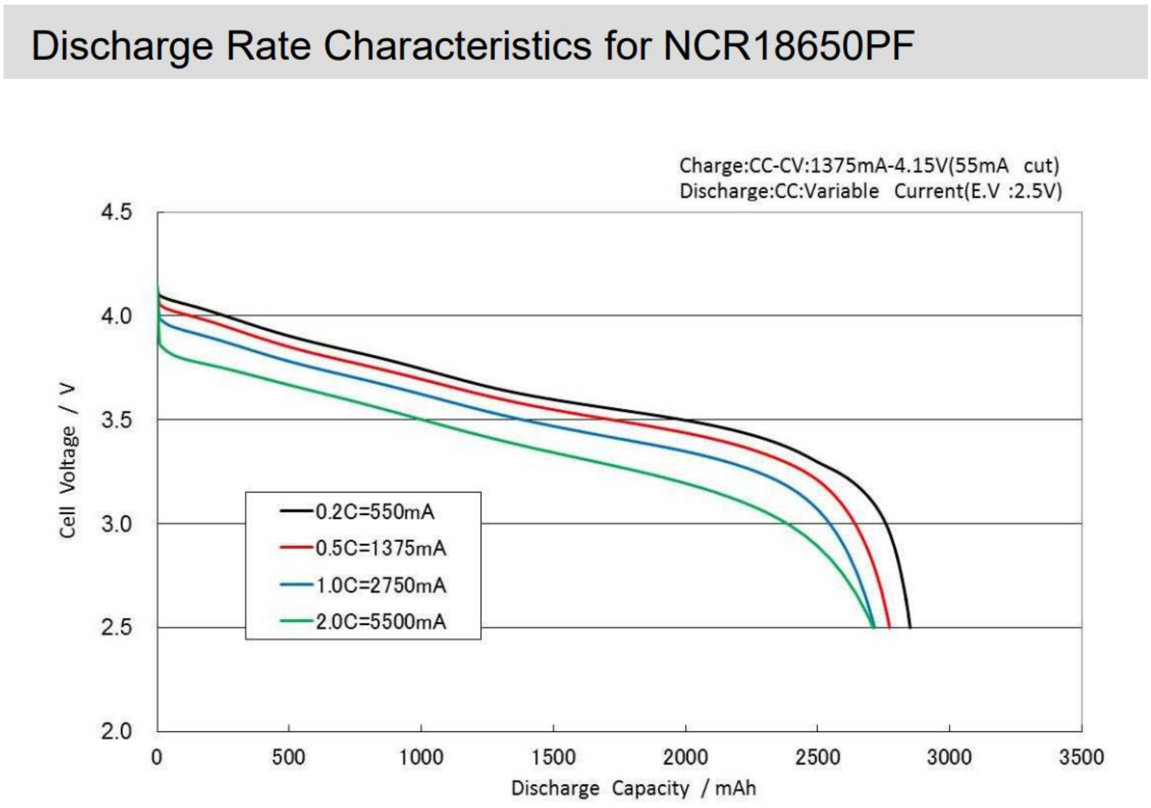


Fig. No 1 :- Voltage vs Discharge capacity: Discharge rate curve fir NCR18650PF

Measure battery voltage and discharge current.
Calculate the discharge rate:

$$\text{Discharge Rate (C)} = \frac{I}{C_{\text{nominal}}}$$

Refer to the manufacturer's discharge curve (e.g., NCR18650PF) corresponding to the measured discharge rate.

From the curve, identify the Discharge Capacity (Ah) corresponding to the measured voltage.

Compute the Initial Discharged SoC:

$$\text{Initial Discharged SoC} = \left(\frac{\text{Discharge Capacity}}{\text{C}_{\text{nominal}}} \right) \times 100$$

Derive the Initial Remaining SoC:

$$\text{Initial Remaining SoC} = 100 - \text{Initial Discharged SoC}$$

Use in Coulomb Counting:

If the battery is charging, use: -

$$\mathrm{SoC}(0) = \text{Initial Discharged SoC}$$

If the battery is discharging, use: -

$$\mathrm{SoC}(0) = \text{Initial Remaining SoC}$$

Scenario	SoC Equation	Initial SoC Used
Charging	$\mathrm{SoC}(t) = \mathrm{SoC}(0) + \frac{1}{C_{\text{nominal}}} \int_0^t I(t)\ dt$	Initial Discharged SoC
Discharging	$\mathrm{SoC}(t) = \mathrm{SoC}(0) - \frac{1}{C_{\text{nominal}}} \int_0^t I(t)\ dt$	Initial Remaining SoC

Table 1:- Soc Coulomb Counting

5.2 SoH Estimation

State of Health (SoH): State of Health (SoH) is a measure of a battery's condition relative to its ideal, as-new state, reflecting factors like capacity and performance; it starts near 100% and declines with usage, aging, and environmental stress.

SoH is often defined as:

$$\mathrm{SoH} = \frac{\text{Current Capacity or Performance Metric}}{\text{Nominal or Initial Value}} \times 100\%$$

Determining SoH:

As SoH isn't tied to a specific physical property, its calculation varies across the industry, with designers using different parameters or combinations to estimate it.

Parameters

- Internal resistance / impedance / conductance
- Capacity
- Voltage
- Self-discharge
- Ability to accept a charge
- Number of charge–discharge cycles
- Age of the battery
- Temperature of battery during its previous uses
- Total energy charged and discharged

Battery management system designers assign custom weights to parameters influencing SoH, and the exact evaluation method is often proprietary or a trade secret.

Proposed method in this project: Impedance spectroscopy approach

Impedance spectroscopy Techniques:

- Electrochemical Impedance Spectroscopy (EIS)

The most widely used method, applying a small-amplitude sinusoidal signal over a range of frequencies to analyze battery behavior in the frequency domain.[3]

- Multisine Excitation

Uses a sum of sine waves at different frequencies to probe the battery, enabling faster data acquisition than traditional EIS.

- Electrochemical Noise Analysis (ENA)

Measures spontaneous fluctuations in voltage and current to infer impedance-related properties without applying external perturbations.

Proposed technique in this project: Electrochemical Impedance Spectroscopy

Electrochemical Impedance Spectroscopy

1. Perform EIS
   - Apply a small-amplitude AC signal over a range of frequencies.
2. Perform Fourier Transform
   - Apply a Fast Fourier Transform (FFT) to the time-domain input and output signals to convert them into the frequency domain.
   - Calculate the impedance $Z(f)$ at each frequency using:

$$Z(f) = \frac{V(f)}{I(f)}$$

3. Analyze the Impedance Spectrum
4. Extract SoH-Relevant Parameters:
   - Rs (Series Resistance)
   - Rct (Charge Transfer Resistance)
   - Cdl (Double-Layer Capacitance)
5. Compare to Reference Data: Measured vs. Baseline Values
   - Calculate the ratios of baseline to measured values:

$$\frac{R_{s,0}}{R_s}, \quad \frac{R_{ct,0}}{R_{ct}}, \quad \frac{C_{dl}}{C_{dl,0}}$$

- A higher Rs or Rct indicates degradation.
- A lower Cdl suggests loss of active surface area.

6. Calculate SoH:

- For a new battery, SoH can be estimated like this:

$$\mathrm{SoH}_{\mathrm{impedance}} = \left(\frac{R_{ct,\mathrm{new}}}{R_{ct,\mathrm{measured}}}\right) \times 100\%$$

- For running battery, using a weighted combination:

$$\mathrm{SoH} = w_1\left(\frac{R_{s,\mathrm{new}}}{R_{s,\mathrm{measured}}}\right) + w_2\left(\frac{R_{ct,\mathrm{new}}}{R_{ct,\mathrm{measured}}}\right) + w_3\left(\frac{C_{dl,\mathrm{measured}}}{C_{dl,\mathrm{new}}}\right)$$

Where $w_1 + w_2 + w_3 = 1$, and weights are chosen based on sensitivity or application.

6 System Design and Simulation

In this project we utilized LTspice for simulation purposes to thoroughly analyze the circuit's behavior.[9] This high-performance SPICE simulation software allowed me to model the DC gear motor using resistors (R_{Driver}, R_{Shunt}) and an inductor (L_{Motor}), as well as simulate the operational amplifier-based sensing circuits, including U2 for voltage sensing and U4 for current sensing. The transient analysis (. tran 100ms) conducted in LTspice enabled me to observe dynamic responses, such as current and voltage transients, ensuring the accuracy of outputs like V_{Shunt} and $B_{voltage}$ /2. This simulation approach was crucial for validating the BMS's ability to monitor battery parameters and control the motor effectively under varying conditions.[8]

In this project we also used Fritzing, an open-source tool, to create and document both the schematic diagram and breadboard layout for the system. Using Fritzing, I created a clear visual representation of the physical assembly, incorporating components such as the 2S Panasonic NCR18650PF batteries, TL051CP op-amps, 15K and 30K resistors, BPR56 0.1Ω shunt resistor, L298N motor driver, Arduino Uno R3, and DC gear motor. This tool facilitated the prototyping process by providing an editable and intuitive diagram, allowing for easy modifications and a comprehensive understanding of the circuit's layout on breadboards. Together, LTspice and Fritzing provided a robust framework for designing, simulating, and refining the BMS system to meet the project’s requirements.

Circuit Diagram:

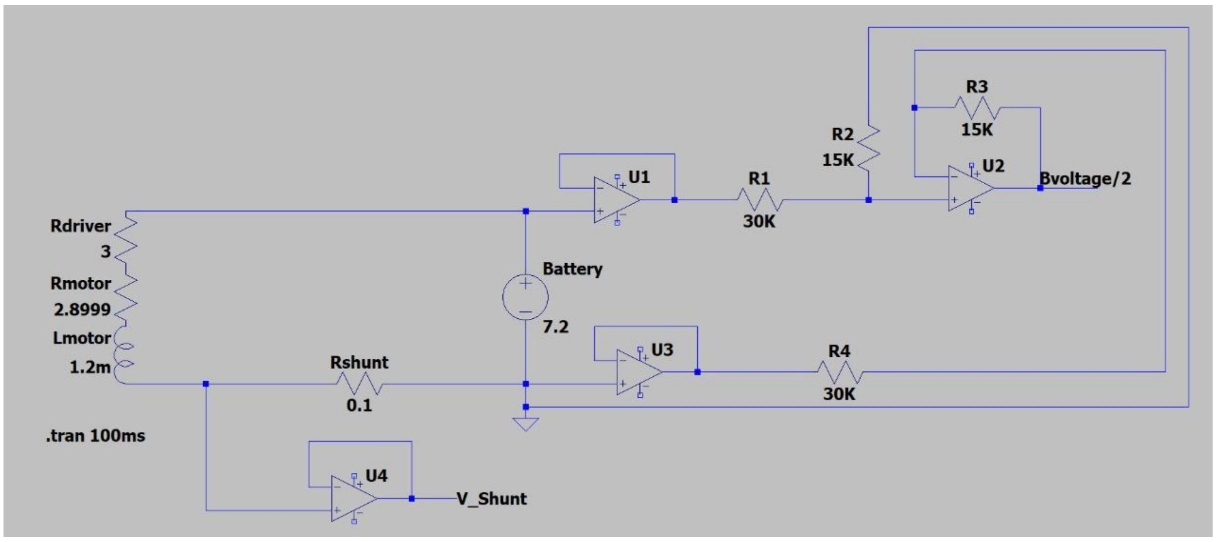


Fig. No 2: - Circuit Diagram in LTspice

Voltage Measurement:

Two voltage followers or buffers are used to extract voltage from battery terminals. The differential voltage from these terminals is fed to the subtractor with a gain factor of 0.5 (As Arduino Uno R3 can't take 7.2 V to an analogRead Pin). The Subtractor output is multiplied by the gain factor in the software to realize the original battery voltage.

Actual Battery Voltage:

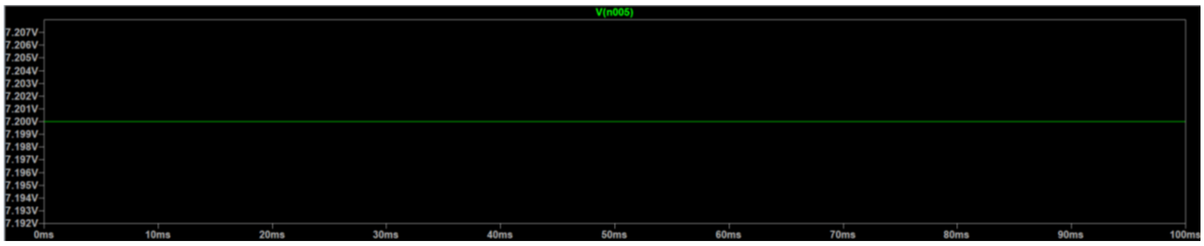


Fig. No 3: - Output of Voltage at Battery +ve Terminal

Voltage Measurement to Arduino with a gain factor = 0.5:

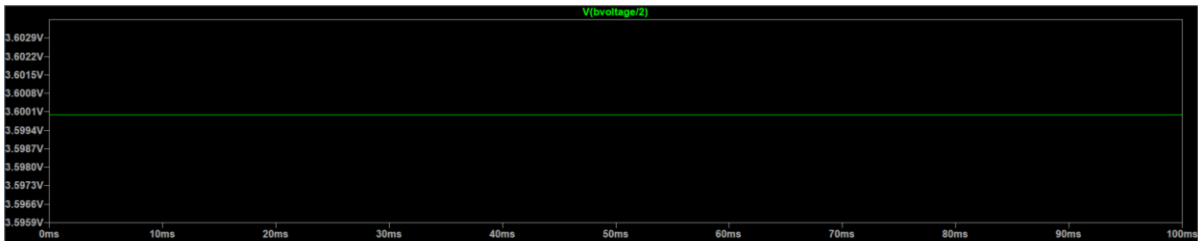


Fig. No 4: - Output of Subtractor (Voltage Measurement circuit)

Current Measurement:

The Shunt Resistor is added in series to the motor replicated circuit, to measure current flowing in the motor circuit. The voltage follower circuit / buffer is connected to the $\boldsymbol{R_{Shunt}(+ve)}$ terminal which measures $V_{Shunt}(+ve)$, as $R_{Shunt}(-ve)$ terminal is connected to the Ground $V_{Shunt}(-ve) = 0$.

Therefore,

$$V_{Shunt} = V_{Shunt}(+ve) - V_{Shunt}(-ve)$$
$$V_{Shunt} = V_{Shunt}(+ve) - 0$$
$$V_{Shunt} = V_{Shunt}(+ve)$$

Now, $\boldsymbol{V_{Shunt}}$ is fed to the Arduino Analog Pin to read Voltage drop across the Shunt Resistor. As Current is same in the series circuit, current determined with V_{Shunt} will be current flowing in the series circuit.

$$Current(I) = \frac{V_{Shunt}}{R_{shunt}}$$

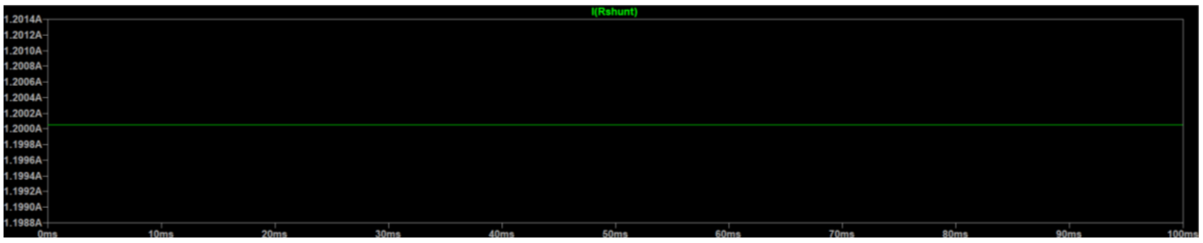


Fig. No 5: - Current across Shunt Resistor

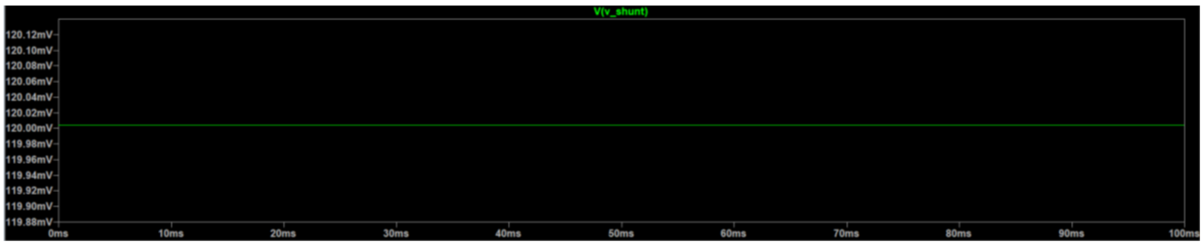


Fig. No 6:- Output of V_{Shunt} for Current Measurement

Schematic Diagram:

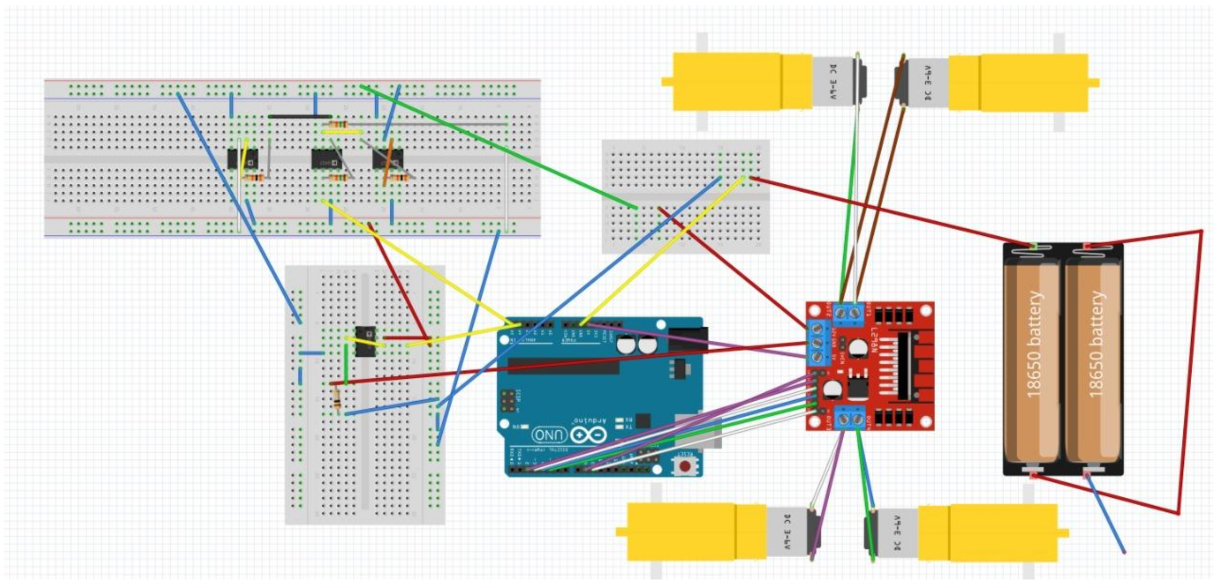


Fig. No 7: - Schematic Diagram in Fritzing

7 Hardware Description

The hardware setup for this project is designed to monitor and analyze the performance of a DC motor system powered by a 7.2V battery pack. The core of the circuit involves a motor load modeled using a combination of resistive and inductive elements: a 3 Ω resistor (Rdriver) representing driver resistance, a 2.8999 Ω resistor (Rmotor) simulating the motor winding resistance, and a 1.2 mH inductor (Lmotor) representing motor inductance. A 0.1 Ω shunt resistor (Rshunt) is installed in series with the motor to allow current measurement by detecting the voltage drop across it. This voltage is buffered using an operational amplifier (U4) configured as a voltage follower to produce the signal labeled V_Shunt, allowing accurate current sensing without loading the signal.

To monitor battery voltage, the setup includes another op-amp (U3) which buffers the raw battery voltage, followed by a resistor divider formed by R1 (30 kΩ) and R2 (15 kΩ) to scale the voltage down. This divided voltage is then buffered again using another op-amp (U2), providing a stable low-impedance output labeled as Bvoltage/2, suitable for ADC inputs in microcontroller-based monitoring systems. Additional resistors R3 and R4 (15 kΩ and 30 kΩ) are part of this voltage conditioning stage, ensuring appropriate signal scaling. The use of op-amps in buffer configuration across the circuit ensures signal integrity, isolating the sensing circuitry from loading effects. All components, including the battery, motor model, resistors, and op-amps, are implemented on a breadboard, enabling real-time signal observation and data acquisition for tasks such as power estimation and state-of-charge (SoC) analysis in battery-powered systems.

7.1 Component List

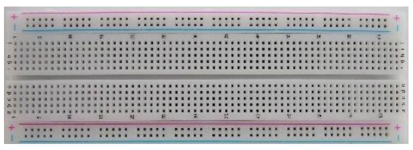
Name	Component	Description
Arduino Uno R3		The Arduino Uno R3, powered by the ATmega328P, is a versatile microcontroller board capable of supporting advanced BMS tasks like Electrochemical Impedance Spectroscopy.
Panasonic NCR18650PF		High-capacity 3.6V Li-ion battery with typical capacity of 2900mAh, arranged in series for 7.2V Nominal voltage output.[15] It provides power for all the components involved in this operation.
L298N Motor Driver		A Dual-H Bridge Motor Driver which can control the DC Motor's Speed and Direction rated upto 46V and 4A of current.[16]
Gear Motor		The DC Gear Motor drives the mechanical parts in this project which is controlled by the Motor Driver via PWM.
TL051CP OP- AMP		JFET-input operational amplifier, which helps providing gain and amplifying voltage and current signals for accurate measurements.
BPR56 0.1 Ohm 5W Shunt Resistor		Shunt resistor mainly used for Current Measurement.
Resistors		Resistor is electrical component that limits or regulates flow of Current.
Breadboard		Helps in easy prototyping, connecting and testing of all the components used in this project.

Table 2: - List of Components

7.2 Hardware setup

To provide a comprehensive overview of the images utilized in this Smart Battery Management System (BMS) project, I will describe the key setups and components captured, reflecting the hands-on development process. These visuals showcase the practical implementation of the system, highlighting the integration of hardware elements and their configurations as part of the experimental and educational framework outlined in the project documentation. Each description will focus on the physical arrangement, key components, and their roles in the BMS functionality, offering insight into the iterative prototyping approach employed.

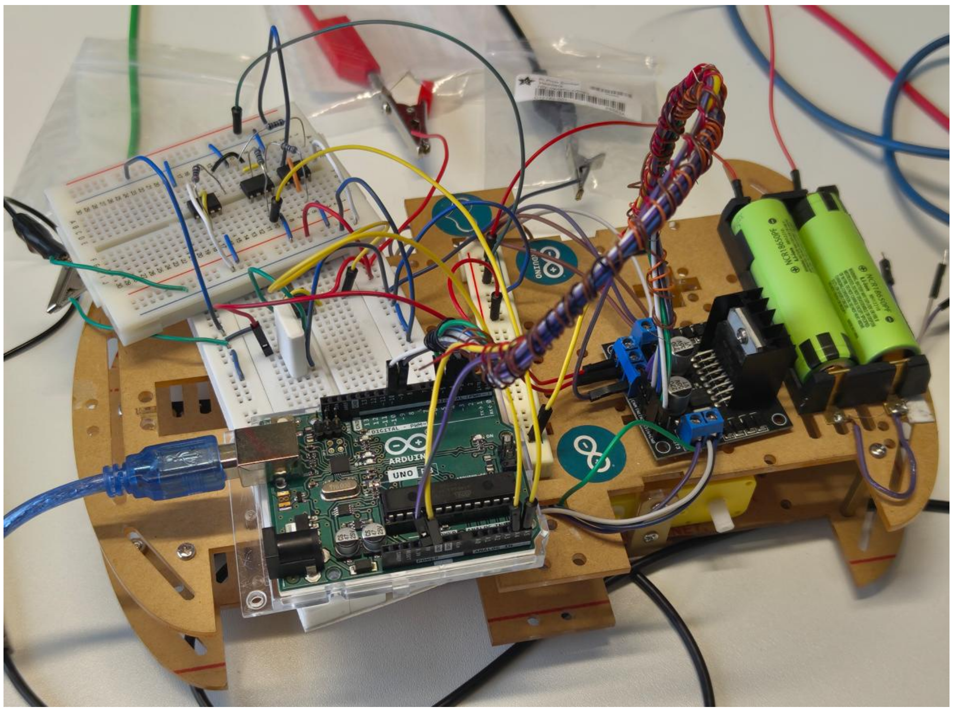


Fig. No 8: - Overall Hardware setup

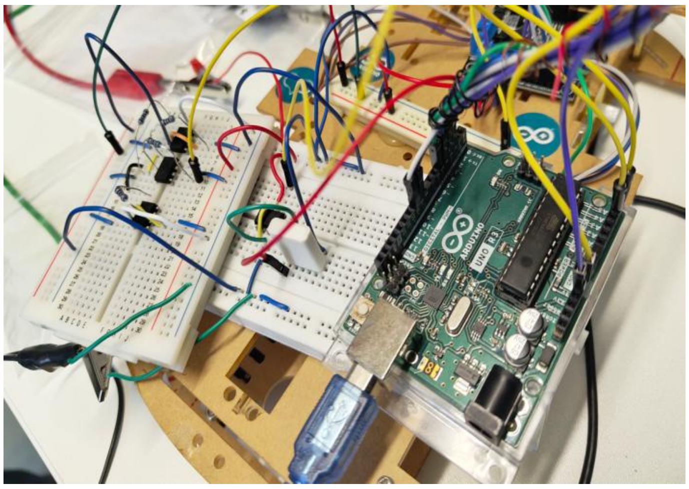


Fig. No 9: - Breadboard and Arduino Uno connections

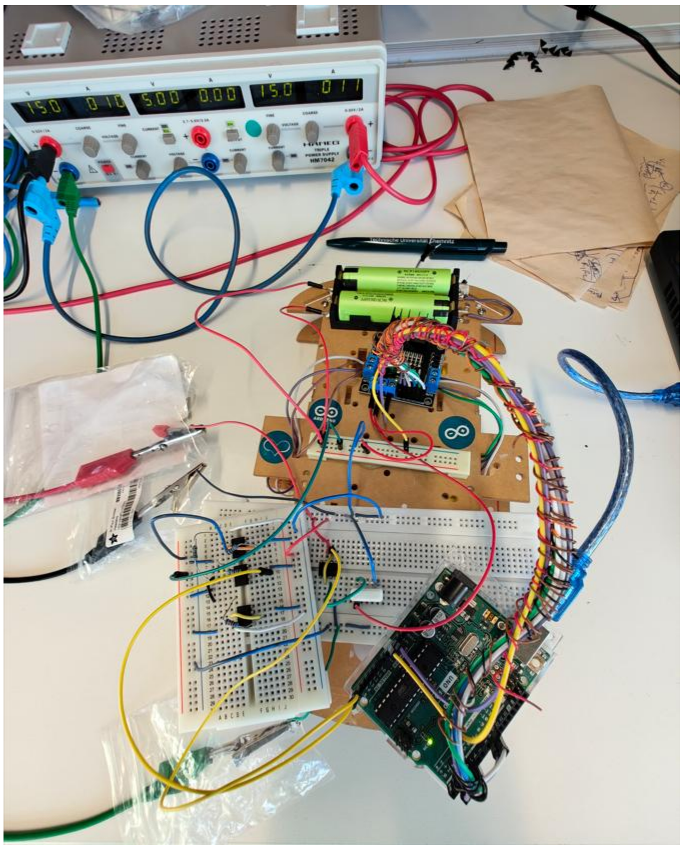


Fig. No 10: - Hardware setup connection with Power supply

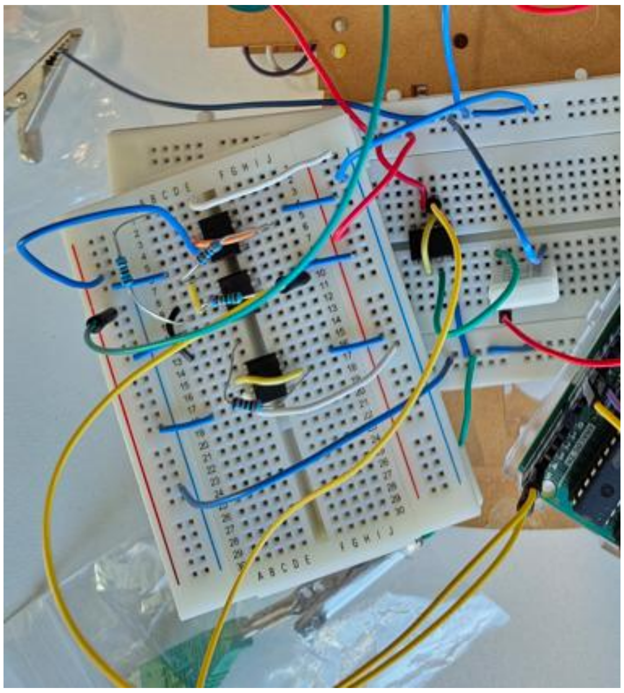


Fig. No 11: - Subtractor and Shunt resistor circuits

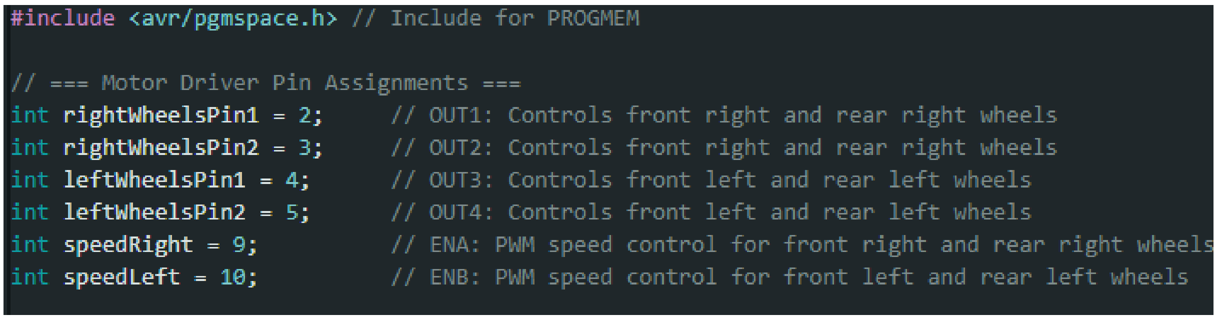
8 Software Description

The software for the Smart Battery Management System (BMS) project is designed to enhance the functionality of the hardware setup, leveraging the Arduino Uno R3 as the central processing unit. With the Arduino code written primarily in C/C++, the code processes analog signals from the TL051CP op-amps for current and voltage sensing, implementing algorithms such as Coulomb counting to estimate the State of Charge (SoC) and potentially impedance spectroscopy for advanced battery health monitoring. It controls the L298N motor driver via PWM and digital outputs to adjust the DC gear motor's speed and direction based on real-time battery data, ensuring safe operation within specified voltage limits (2.5V–4.2V per cell). The software also includes provisions for iterative testing and refinement, aligning with the project's developmental goals.

8.1 SoC Estimation Code

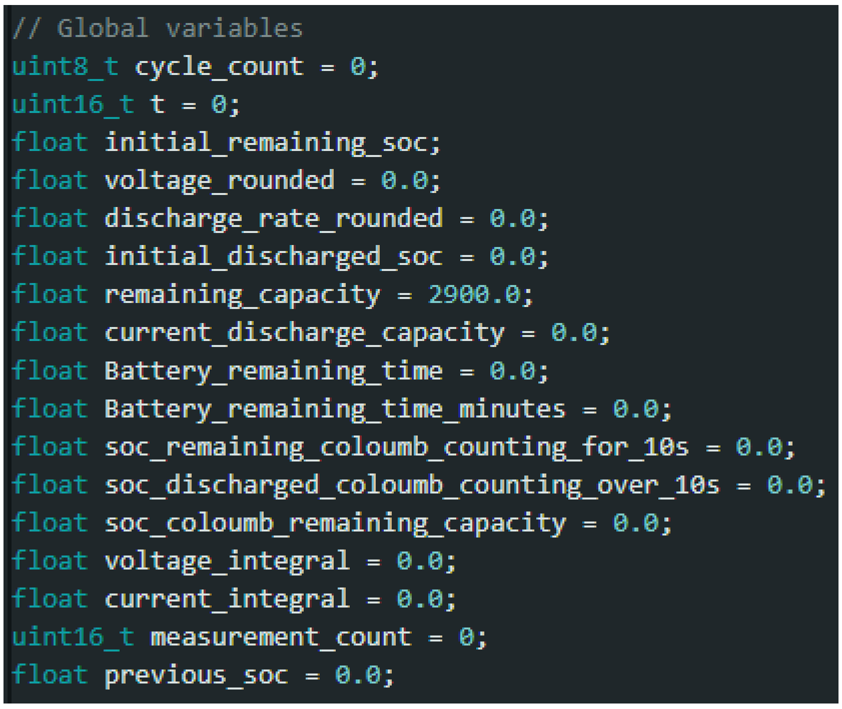
Arduino Program Flow:

- Header inclusion
- Pin assignments
- Global Variables
- User-defined functions
- void setup ()
- void loop ()



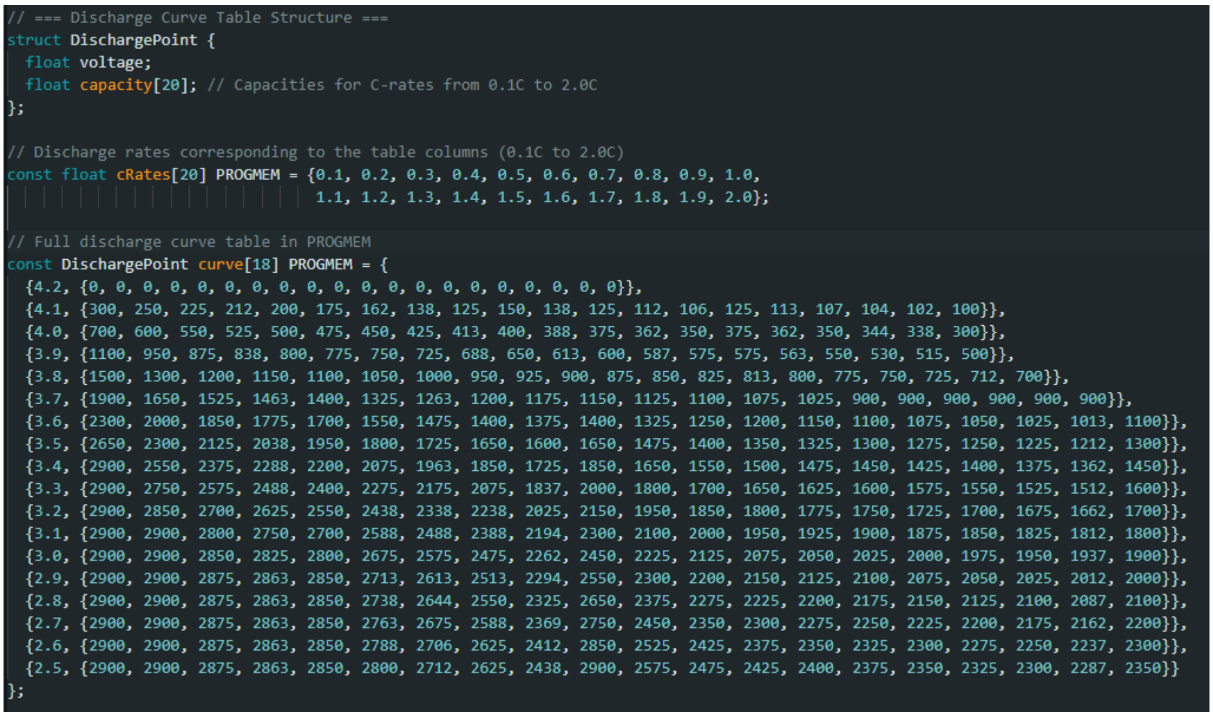
```
#include <avr/pgmspace.h> // Include for PROGMEM

// === Motor Driver Pin Assignments ===
int rightWheelsPin1 = 2;     // OUT1: Controls front right and rear right wheels
int rightWheelsPin2 = 3;     // OUT2: Controls front right and rear right wheels
int leftWheelsPin1 = 4;      // OUT3: Controls front left and rear left wheels
int leftWheelsPin2 = 5;      // OUT4: Controls front left and rear left wheels
int speedRight = 9;          // ENA: PWM speed control for front right and rear right wheels
int speedLeft = 10;          // ENB: PWM speed control for front left and rear left wheels
```

Efficient use of flash memory and motor pin configuration enables optimized control of robot wheel direction and speed on AVR microcontrollers.



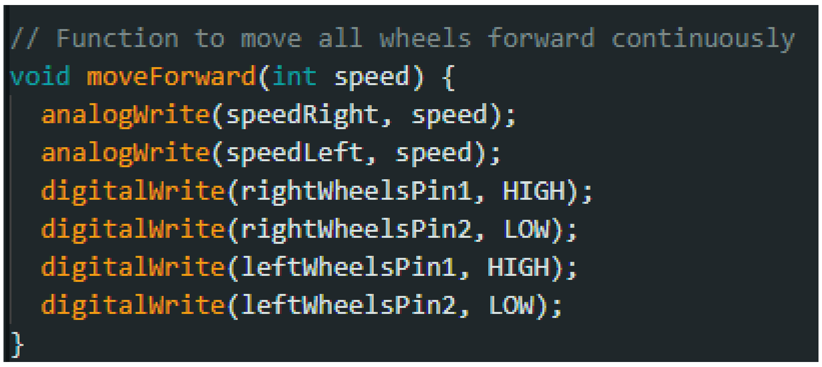
```
// Global variables
uint8_t cycle_count = 0;
uint16_t t = 0;
float initial_remaining_soc;
float voltage_rounded = 0.0;
float discharge_rate_rounded = 0.0;
float initial_discharged_soc = 0.0;
float remaining_capacity = 2900.0;
float current_discharge_capacity = 0.0;
float Battery_remaining_time = 0.0;
float Battery_remaining_time_minutes = 0.0;
float soc_remaining_coloumb_counting_for_10s = 0.0;
float soc_discharged_coloumb_counting_over_10s = 0.0;
float soc_coloumb_remaining_capacity = 0.0;
float voltage_integral = 0.0;
float current_integral = 0.0;
uint16_t measurement_count = 0;
float previous_soc = 0.0;
```

Battery tracking variables and memory-efficient datatypes enable precise monitoring and management of energy consumption and charge state in embedded systems.



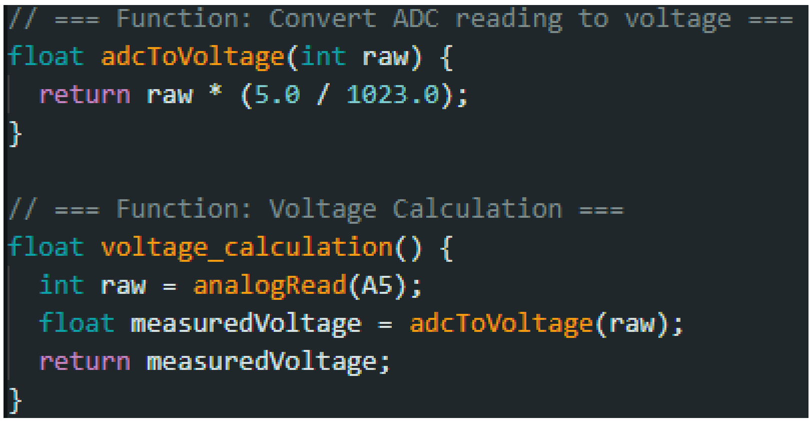
```
// === Discharge Curve Table Structure ===
struct DischargePoint {
  float voltage;
  float capacity[20]; // Capacities for C-rates from 0.1C to 2.0C
};

// Discharge rates corresponding to the table columns (0.1C to 2.0C)
const float cRates[20] PROGMEM = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
                                  1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0};

// Full discharge curve table in PROGMEM
const DischargePoint curve[18] PROGMEM = {
  {4.2, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}},
  {4.1, {300, 250, 225, 212, 200, 175, 162, 138, 125, 150, 138, 125, 112, 106, 125, 113, 107, 104, 102, 100}},
  {4.0, {700, 600, 550, 525, 500, 475, 450, 425, 413, 400, 388, 375, 362, 350, 375, 362, 350, 344, 338, 300}},
  {3.9, {1100, 950, 875, 838, 800, 775, 750, 725, 688, 650, 613, 600, 587, 575, 575, 563, 550, 530, 515, 500}},
  {3.8, {1500, 1300, 1200, 1150, 1100, 1050, 1000, 950, 925, 900, 875, 850, 825, 813, 800, 775, 750, 725, 712, 700}},
  {3.7, {1900, 1650, 1525, 1463, 1400, 1325, 1263, 1200, 1175, 1150, 1125, 1100, 1075, 1025, 900, 900, 900, 900, 900, 900}},
  {3.6, {2300, 2000, 1850, 1775, 1700, 1550, 1475, 1400, 1375, 1400, 1325, 1250, 1200, 1150, 1100, 1075, 1050, 1025, 1013, 1100}},
  {3.5, {2650, 2300, 2125, 2038, 1950, 1800, 1725, 1650, 1600, 1650, 1475, 1400, 1350, 1325, 1300, 1275, 1250, 1225, 1212, 1300}},
  {3.4, {2900, 2550, 2375, 2288, 2200, 2075, 1963, 1850, 1725, 1850, 1650, 1550, 1500, 1475, 1450, 1425, 1400, 1375, 1362, 1450}},
  {3.3, {2900, 2750, 2575, 2488, 2400, 2275, 2175, 2075, 1837, 2000, 1800, 1700, 1650, 1625, 1600, 1575, 1550, 1525, 1512, 1600}},
  {3.2, {2900, 2850, 2700, 2625, 2550, 2438, 2338, 2238, 2025, 2150, 1950, 1850, 1800, 1775, 1750, 1725, 1700, 1675, 1662, 1700}},
  {3.1, {2900, 2900, 2800, 2750, 2700, 2588, 2488, 2388, 2194, 2300, 2100, 2000, 1950, 1925, 1900, 1875, 1850, 1825, 1812, 1800}},
  {3.0, {2900, 2900, 2850, 2825, 2800, 2675, 2575, 2475, 2262, 2450, 2225, 2125, 2075, 2050, 2025, 2000, 1975, 1950, 1937, 1900}},
  {2.9, {2900, 2900, 2875, 2863, 2850, 2713, 2613, 2513, 2294, 2550, 2300, 2200, 2150, 2125, 2100, 2075, 2050, 2025, 2012, 2000}},
  {2.8, {2900, 2900, 2875, 2863, 2850, 2738, 2644, 2550, 2325, 2650, 2375, 2275, 2225, 2200, 2175, 2150, 2125, 2100, 2087, 2100}},
  {2.7, {2900, 2900, 2875, 2863, 2850, 2763, 2675, 2588, 2369, 2750, 2450, 2350, 2300, 2275, 2250, 2225, 2200, 2175, 2162, 2200}},
  {2.6, {2900, 2900, 2875, 2863, 2850, 2788, 2706, 2625, 2412, 2850, 2525, 2425, 2375, 2350, 2325, 2300, 2275, 2250, 2237, 2300}},
  {2.5, {2900, 2900, 2875, 2863, 2850, 2800, 2712, 2625, 2438, 2900, 2575, 2475, 2425, 2400, 2375, 2350, 2325, 2300, 2287, 2350}}
};
```

DischargePoint structure maps voltage-capacity pairs to model battery behavior under varying loads for accurate performance estimation.



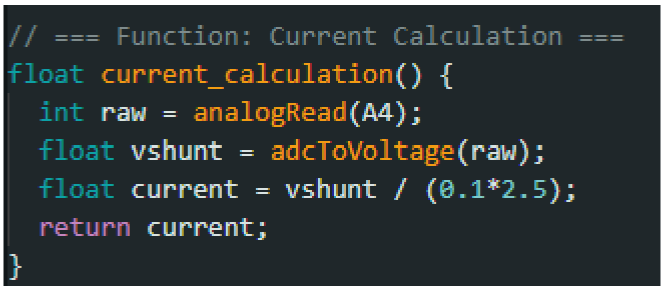
```
// Function to move all wheels forward continuously
void moveForward(int speed) {
  analogWrite(speedRight, speed);
  analogWrite(speedLeft, speed);
  digitalWrite(rightWheelsPin1, HIGH);
  digitalWrite(rightWheelsPin2, LOW);
  digitalWrite(leftWheelsPin1, HIGH);
  digitalWrite(leftWheelsPin2, LOW);
}
```

moveForward (int speed) configures motor direction and applies PWM signals to synchronize both wheels for forward motion.



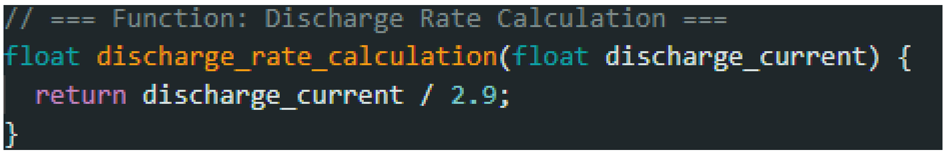
```
// === Function: Convert ADC reading to voltage ===
float adcToVoltage(int raw) {
  return raw * (5.0 / 1023.0);
}

// === Function: Voltage Calculation ===
float voltage_calculation() {
  int raw = analogRead(A5);
  float measuredVoltage = adcToVoltage(raw);
  return measuredVoltage;
}
```

adcToVoltage () and voltage_calculation () convert and process analog readings for accurate voltage monitoring via Arduino’s A5 input pin.



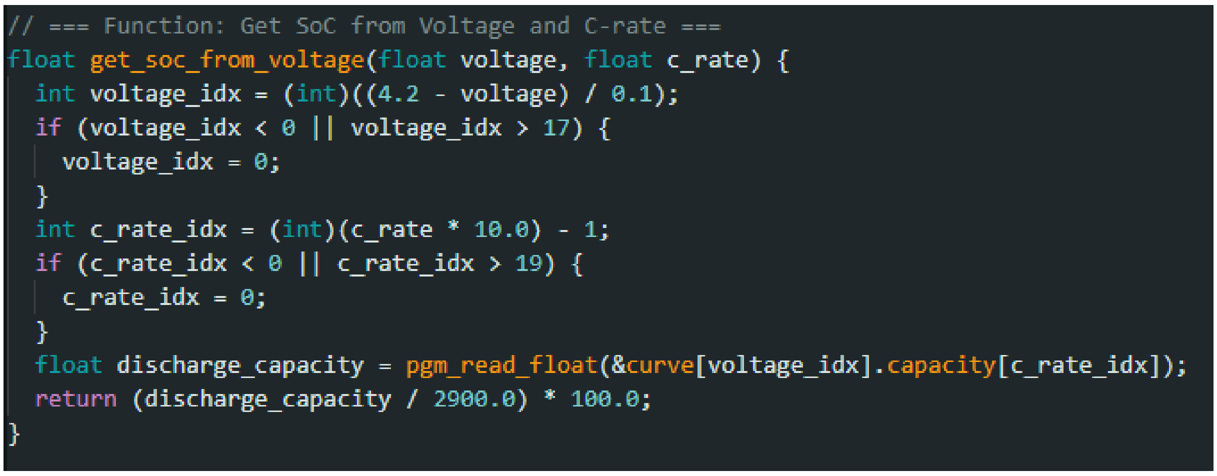
```
// === Function: Current Calculation ===
float current_calculation() {
  int raw = analogRead(A4);
  float vshunt = adcToVoltage(raw);
  float current = vshunt / (0.1*2.5);
  return current;
}
```

current_calculation () measures analog input on A4, converts it to voltage, and calculates current by applying Ohm’s law with a shunt resistor and gain factor.



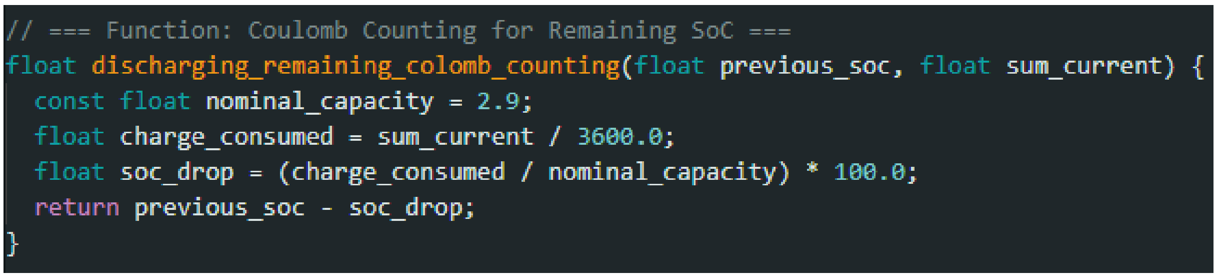
```
// === Function: Discharge Rate Calculation ===
float discharge_rate_calculation(float discharge_current) {
  return discharge_current / 2.9;
}
```

dischargeRate () calculates the battery's C-rate by dividing input current by its rated capacity, indicating how quickly energy is being depleted.



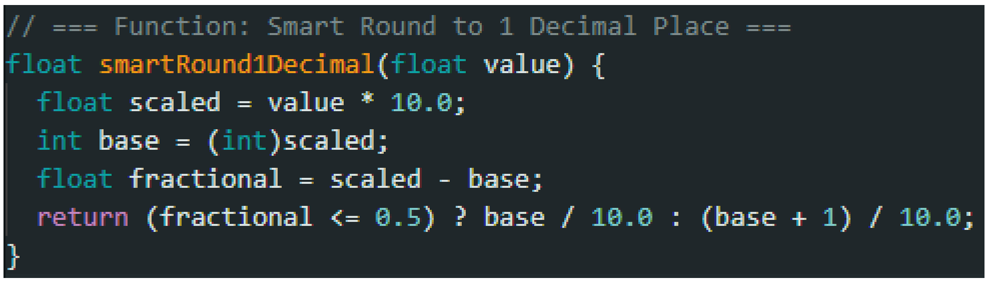
```
// === Function: Get SoC from Voltage and C-rate ===
float get_soc_from_voltage(float voltage, float c_rate) {
  int voltage_idx = (int)((4.2 - voltage) / 0.1);
  if (voltage_idx < 0 || voltage_idx > 17) {
    voltage_idx = 0;
  }
  int c_rate_idx = (int)(c_rate * 10.0) - 1;
  if (c_rate_idx < 0 || c_rate_idx > 19) {
    c_rate_idx = 0;
  }
  float discharge_capacity = pgm_read_float(&curve[voltage_idx].capacity[c_rate_idx]);
  return (discharge_capacity / 2900.0) * 100.0;
}
```

estimateSoC () computes battery State of Charge by indexing a lookup table with voltage and C-rate, retrieving capacity, normalizing it to 2900mAh, and applying boundary checks for safe evaluation.



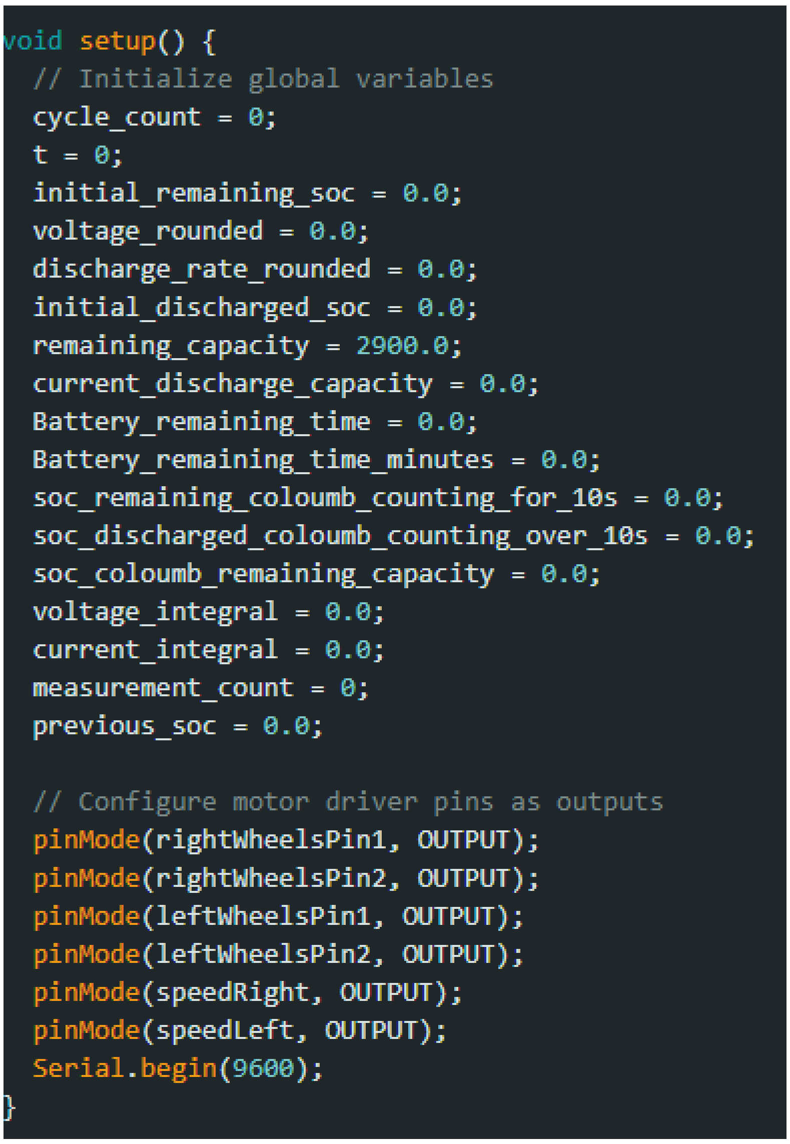
```
// === Function: Coulomb Counting for Remaining SoC ===
float discharging_remaining_colomb_counting(float previous_soc, float sum_current) {
  const float nominal_capacity = 2.9;
  float charge_consumed = sum_current / 3600.0;
  float soc_drop = (charge_consumed / nominal_capacity) * 100.0;
  return previous_soc - soc_drop;
}
```

estimateSoC_Coulomb () tracks battery usage by calculating charge consumption over time and updating the remaining SoC based on cumulative current.



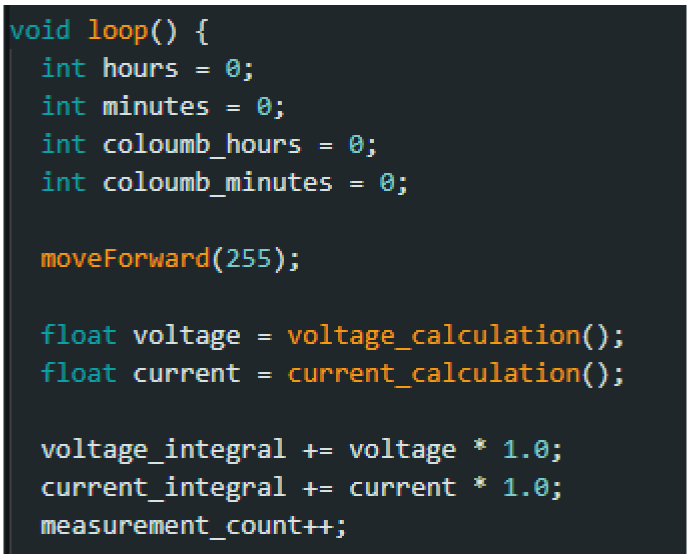
```
// === Function: Smart Round to 1 Decimal Place ===
float smartRound1Decimal(float value) {
  float scaled = value * 10.0;
  int base = (int)scaled;
  float fractional = scaled - base;
  return (fractional <= 0.5) ? base / 10.0 : (base + 1) / 10.0;
}
```

smartRound1Decimal () rounds a float to one decimal place using custom logic: down for ≤ 0.5 fractions, up for > 0.5 fractions.



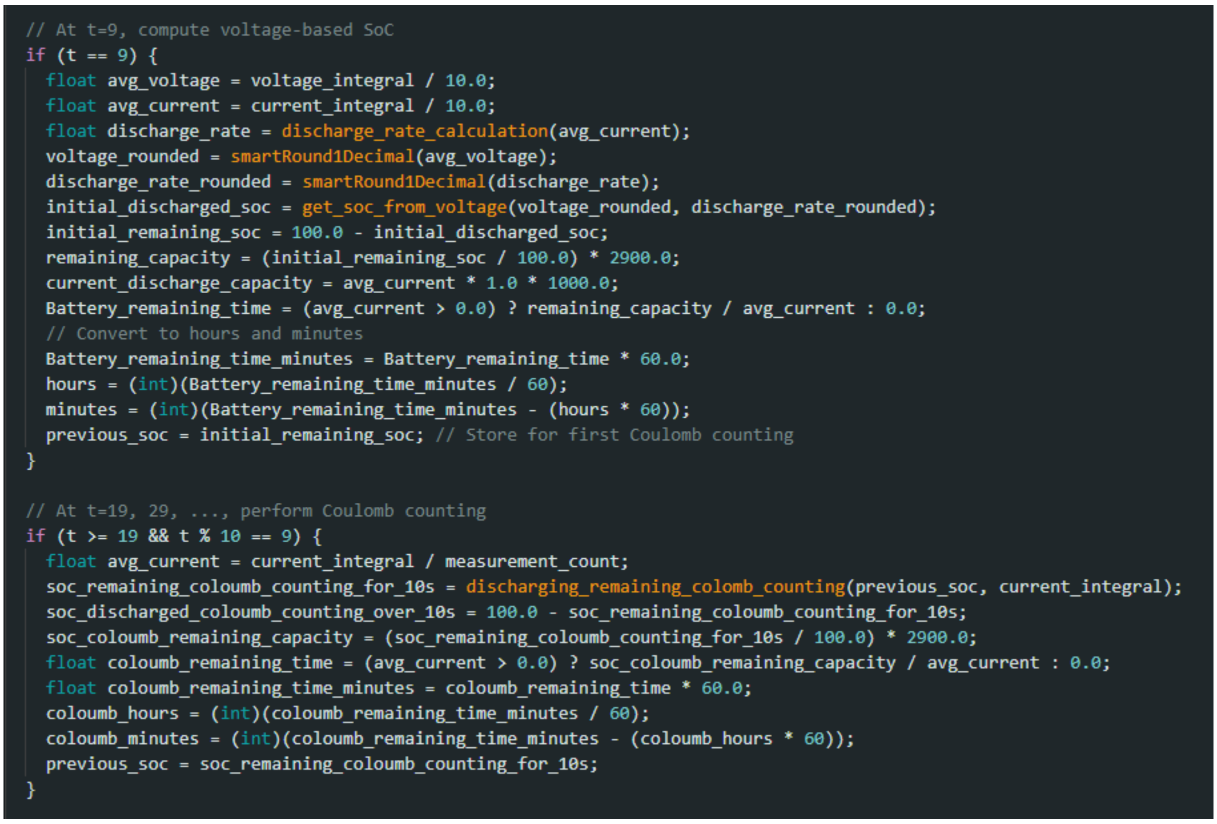
```
void setup() {
  // Initialize global variables
  cycle_count = 0;
  t = 0;
  initial_remaining_soc = 0.0;
  voltage_rounded = 0.0;
  discharge_rate_rounded = 0.0;
  initial_discharged_soc = 0.0;
  remaining_capacity = 2900.0;
  current_discharge_capacity = 0.0;
  Battery_remaining_time = 0.0;
  Battery_remaining_time_minutes = 0.0;
  soc_remaining_coloumb_counting_for_10s = 0.0;
  soc_discharged_coloumb_counting_over_10s = 0.0;
  soc_coloumb_remaining_capacity = 0.0;
  voltage_integral = 0.0;
  current_integral = 0.0;
  measurement_count = 0;
  previous_soc = 0.0;

  // Configure motor driver pins as outputs
  pinMode(rightWheelsPin1, OUTPUT);
  pinMode(rightWheelsPin2, OUTPUT);
  pinMode(leftWheelsPin1, OUTPUT);
  pinMode(leftWheelsPin2, OUTPUT);
  pinMode(speedRight, OUTPUT);
  pinMode(speedLeft, OUTPUT);
  Serial.begin(9600);
}
```

setup () initializes energy tracking variables and timing counters to ensure accurate and consistent battery monitoring from system start up.



```
void loop() {
  int hours = 0;
  int minutes = 0;
  int coloumb_hours = 0;
  int coloumb_minutes = 0;

  moveForward(255);

  float voltage = voltage_calculation();
  float current = current_calculation();

  voltage_integral += voltage * 1.0;
  current_integral += current * 1.0;
  measurement_count++;
```

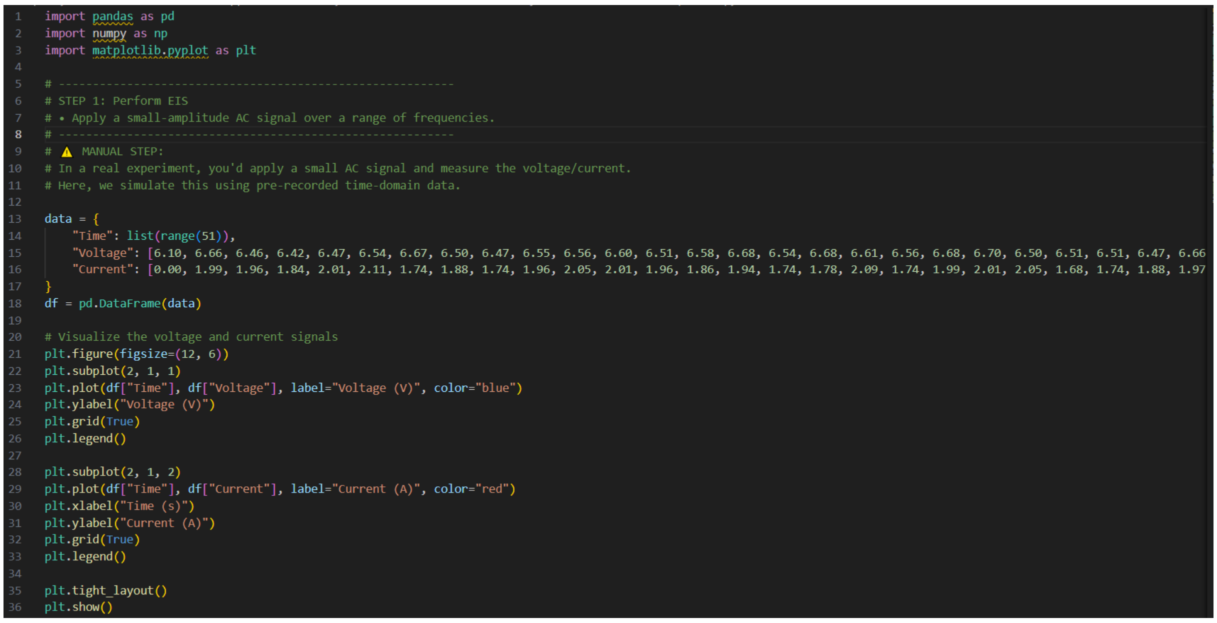
measure Loop () continuously collects voltage and current data while in motion, tracking samples for accurate energy and SoC evaluation.



```
// At t=9, compute voltage-based SoC
if (t == 9) {
  float avg_voltage = voltage_integral / 10.0;
  float avg_current = current_integral / 10.0;
  float discharge_rate = discharge_rate_calculation(avg_current);
  voltage_rounded = smartRound1Decimal(avg_voltage);
  discharge_rate_rounded = smartRound1Decimal(discharge_rate);
  initial_discharged_soc = get_soc_from_voltage(voltage_rounded, discharge_rate_rounded);
  initial_remaining_soc = 100.0 - initial_discharged_soc;
  remaining_capacity = (initial_remaining_soc / 100.0) * 2900.0;
  current_discharge_capacity = avg_current * 1.0 * 1000.0;
  Battery_remaining_time = (avg_current > 0.0) ? remaining_capacity / avg_current : 0.0;
  // Convert to hours and minutes
  Battery_remaining_time_minutes = Battery_remaining_time * 60.0;
  hours = (int)(Battery_remaining_time_minutes / 60);
  minutes = (int)(Battery_remaining_time_minutes - (hours * 60));
  previous_soc = initial_remaining_soc; // Store for first Coulomb counting
}

// At t=19, 29, ..., perform Coulomb counting
if (t >= 19 && t % 10 == 9) {
  float avg_current = current_integral / measurement_count;
  soc_remaining_coloumb_counting_for_10s = discharging_remaining_colomb_counting(previous_soc, current_integral);
  soc_discharged_coloumb_counting_over_10s = 100.0 - soc_remaining_coloumb_counting_for_10s;
  soc_coloumb_remaining_capacity = (soc_remaining_coloumb_counting_for_10s / 100.0) * 2900.0;
  float coloumb_remaining_time = (avg_current > 0.0) ? soc_coloumb_remaining_capacity / avg_current : 0.0;
  float coloumb_remaining_time_minutes = coloumb_remaining_time * 60.0;
  coloumb_hours = (int)(coloumb_remaining_time_minutes / 60);
  coloumb_minutes = (int)(coloumb_remaining_time_minutes - (coloumb_hours * 60));
  previous_soc = soc_remaining_coloumb_counting_for_10s;
}
```

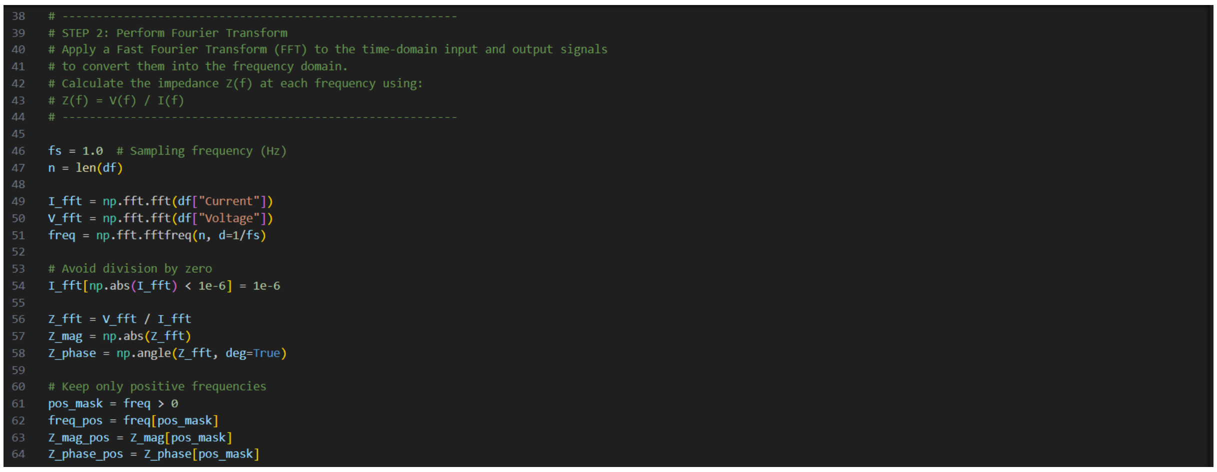
Combined SoC estimation averages sensor data, blends voltage-based and Coulomb counting methods, and calculates remaining runtime based on discharge rate.

8.2 SoH Estimation Code



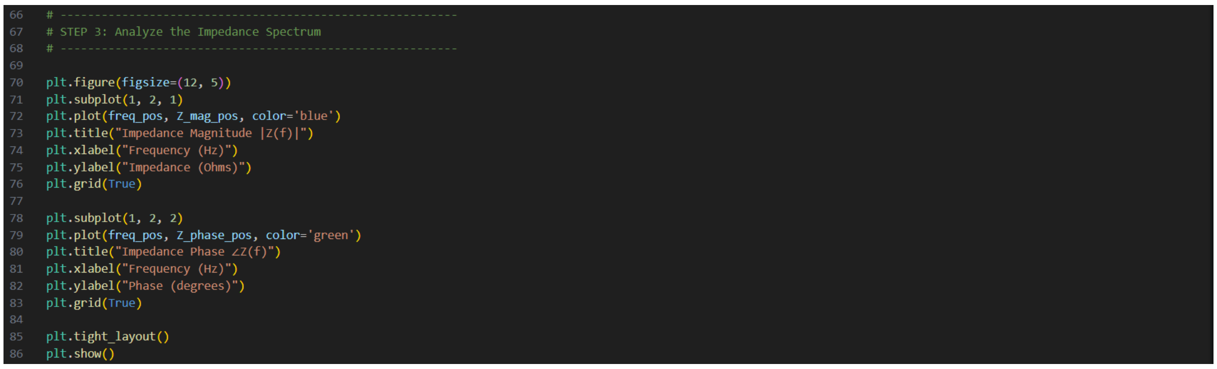
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# ----------------------------------------------------------
# STEP 1: Perform EIS
# • Apply a small-amplitude AC signal over a range of frequencies.
# ----------------------------------------------------------
# ⚠ MANUAL STEP:
# In a real experiment, you'd apply a small AC signal and measure the voltage/current.
# Here, we simulate this using pre-recorded time-domain data.

data = {
    "Time": list(range(51)),
    "Voltage": [6.10, 6.66, 6.46, 6.42, 6.47, 6.54, 6.67, 6.50, 6.47, 6.55, 6.56, 6.60, 6.51, 6.58, 6.68, 6.54, 6.68, 6.61, 6.56, 6.68, 6.70, 6.50, 6.51, 6.51, 6.47, 6.66
    "Current": [0.00, 1.99, 1.96, 1.84, 2.01, 2.11, 1.74, 1.88, 1.74, 1.96, 2.05, 2.01, 1.96, 1.86, 1.94, 1.74, 1.78, 2.09, 1.74, 1.99, 2.01, 2.05, 1.68, 1.74, 1.88, 1.97
}
df = pd.DataFrame(data)

# Visualize the voltage and current signals
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(df["Time"], df["Voltage"], label="Voltage (V)", color="blue")
plt.ylabel("Voltage (V)")
plt.grid(True)
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(df["Time"], df["Current"], label="Current (A)", color="red")
plt.xlabel("Time (s)")
plt.ylabel("Current (A)")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

It simulates and plots voltage and current data over time to mimic an electrical impedance spectroscopy experiment.



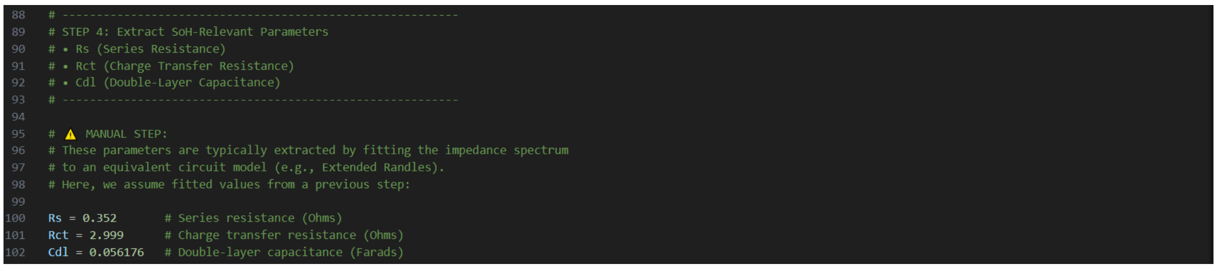
```python
# ----------------------------------------------------------
# STEP 2: Perform Fourier Transform
# Apply a Fast Fourier Transform (FFT) to the time-domain input and output signals
# to convert them into the frequency domain.
# Calculate the impedance Z(f) at each frequency using:
# Z(f) = V(f) / I(f)
# ----------------------------------------------------------

fs = 1.0  # Sampling frequency (Hz)
n = len(df)

I_fft = np.fft.fft(df["Current"])
V_fft = np.fft.fft(df["Voltage"])
freq = np.fft.fftfreq(n, d=1/fs)

# Avoid division by zero
I_fft[np.abs(I_fft) < 1e-6] = 1e-6

Z_fft = V_fft / I_fft
Z_mag = np.abs(Z_fft)
Z_phase = np.angle(Z_fft, deg=True)

# Keep only positive frequencies
pos_mask = freq > 0
freq_pos = freq[pos_mask]
Z_mag_pos = Z_mag[pos_mask]
Z_phase_pos = Z_phase[pos_mask]
```

The code performs a Fast Fourier Transform on voltage and current signals to compute frequency-dependent impedance magnitude and phase, while filtering out zero and negative frequencies.



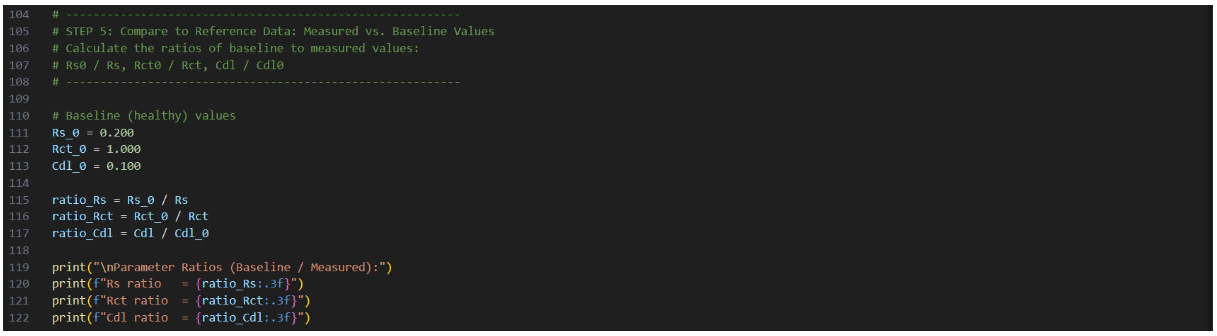
```python
# --------------------------------------------------------
# STEP 3: Analyze the Impedance Spectrum
# --------------------------------------------------------

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(freq_pos, Z_mag_pos, color='blue')
plt.title("Impedance Magnitude |Z(f)|")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Impedance (Ohms)")
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(freq_pos, Z_phase_pos, color='green')
plt.title("Impedance Phase ∠Z(f)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Phase (degrees)")
plt.grid(True)

plt.tight_layout()
plt.show()
```

The code creates two plots—one showing how impedance magnitude changes with frequency, and the other showing how impedance phase changes with frequency—helping visualize the behavior of an electrical system across a spectrum.



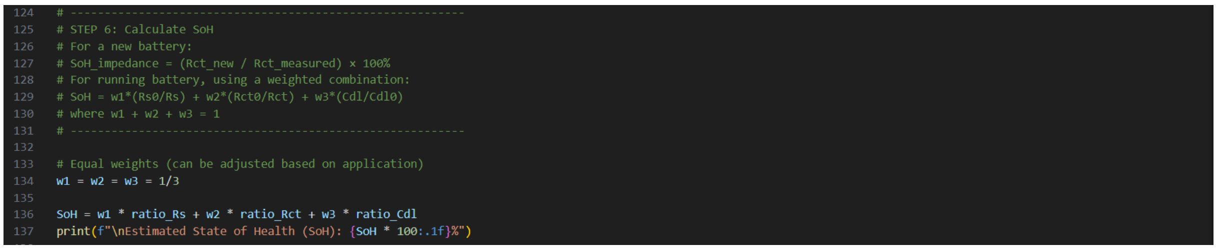
```python
# --------------------------------------------------------
# STEP 4: Extract SoH-Relevant Parameters
# • Rs (Series Resistance)
# • Rct (Charge Transfer Resistance)
# • Cdl (Double-Layer Capacitance)
# --------------------------------------------------------

# ⚠ MANUAL STEP:
# These parameters are typically extracted by fitting the impedance spectrum
# to an equivalent circuit model (e.g., Extended Randles).
# Here, we assume fitted values from a previous step:

Rs = 0.352       # Series resistance (Ohms)
Rct = 2.999      # Charge transfer resistance (Ohms)
Cdl = 0.056176   # Double-layer capacitance (Farads)
```

The code assigns pre-determined values for Rs, Rct, and Cdl—key impedance parameters—to support analysis of battery health, assuming these values were previously fitted using an equivalent circuit model.



```python
# --------------------------------------------------------
# STEP 5: Compare to Reference Data: Measured vs. Baseline Values
# Calculate the ratios of baseline to measured values:
# Rs0 / Rs, Rct0 / Rct, Cdl / Cdl0
# --------------------------------------------------------

# Baseline (healthy) values
Rs_0 = 0.200
Rct_0 = 1.000
Cdl_0 = 0.100

ratio_Rs = Rs_0 / Rs
ratio_Rct = Rct_0 / Rct
ratio_Cdl = Cdl / Cdl_0

print("\nParameter Ratios (Baseline / Measured):")
print(f"Rs ratio   = {ratio_Rs:.3f}")
print(f"Rct ratio  = {ratio_Rct:.3f}")
print(f"Cdl ratio  = {ratio_Cdl:.3f}")
```

The code calculates the ratios of baseline (healthy) values to the measured impedance parameters to assess how the current system compares to its expected optimal condition.



```python
# --------------------------------------------------------
# STEP 6: Calculate SoH
# For a new battery:
# SoH_impedance = (Rct_new / Rct_measured) × 100%
# For running battery, using a weighted combination:
# SoH = w1*(Rs0/Rs) + w2*(Rct0/Rct) + w3*(Cdl/Cdl0)
# where w1 + w2 + w3 = 1
# --------------------------------------------------------

# Equal weights (can be adjusted based on application)
w1 = w2 = w3 = 1/3

SoH = w1 * ratio_Rs + w2 * ratio_Rct + w3 * ratio_Cdl
print(f"\nEstimated State of Health (SoH): {SoH * 100:.1f}%")
```

This segment of the code calculates the battery's **State of Health (SoH)** using weighted ratios of impedance parameters: series resistance, charge transfer resistance, and double-layer capacitance then outputs the result as a percentage to evaluate overall battery condition.

9 Results

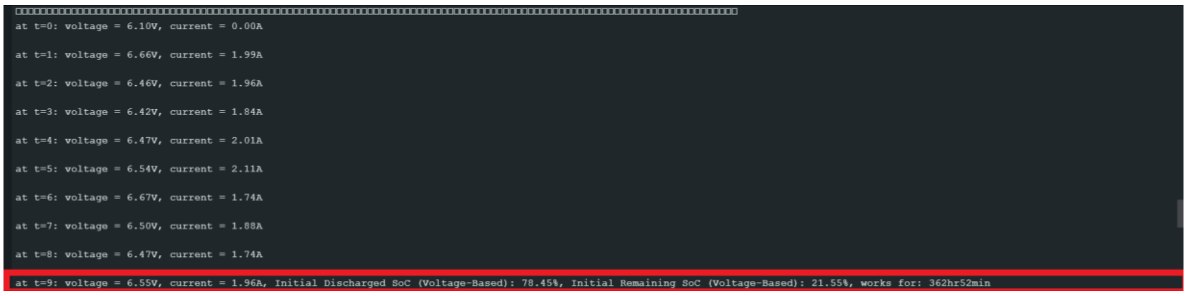
For SoC Estimation: Combined approach (Voltage + Coulomb counting)

Step 1: Voltage and Current are measured as proposed in System Design

Step 2: Initial SoC is determined by Voltage (average value for 10 sec for stability) vs Discharge characteristics curve table

Step 3: Coulomb Counting is performed with Initial SoC taken from previous step and Current Integration over 10 seconds.

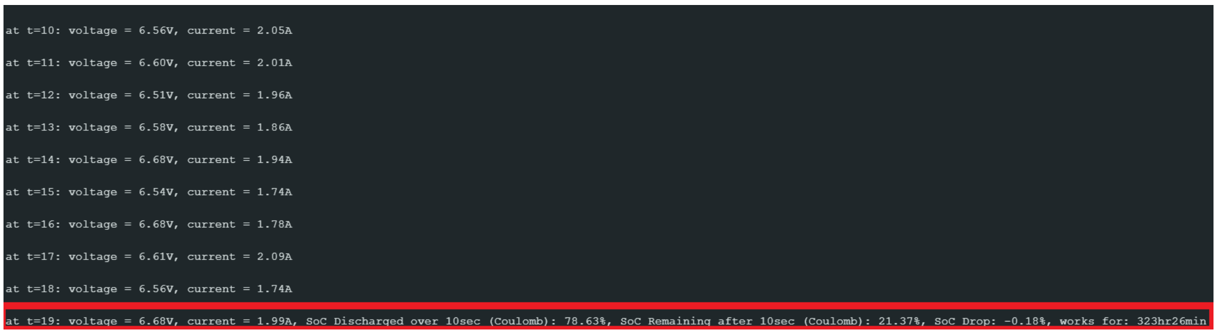
Step 4: Calculating discharged SoC, remaining SoC, SoC drop and operating time.



```
at t=0: voltage = 6.10V, current = 0.00A

at t=1: voltage = 6.66V, current = 1.99A

at t=2: voltage = 6.46V, current = 1.96A

at t=3: voltage = 6.42V, current = 1.84A

at t=4: voltage = 6.47V, current = 2.01A

at t=5: voltage = 6.54V, current = 2.11A

at t=6: voltage = 6.67V, current = 1.74A

at t=7: voltage = 6.50V, current = 1.88A

at t=8: voltage = 6.47V, current = 1.74A

at t=9: voltage = 6.55V, current = 1.96A, Initial Discharged SoC (Voltage-Based): 78.45%, Initial Remaining SoC (Voltage-Based): 21.55%, works for: 362hr52min
```

Fig. No 12: - Output of calculated Initial SoC

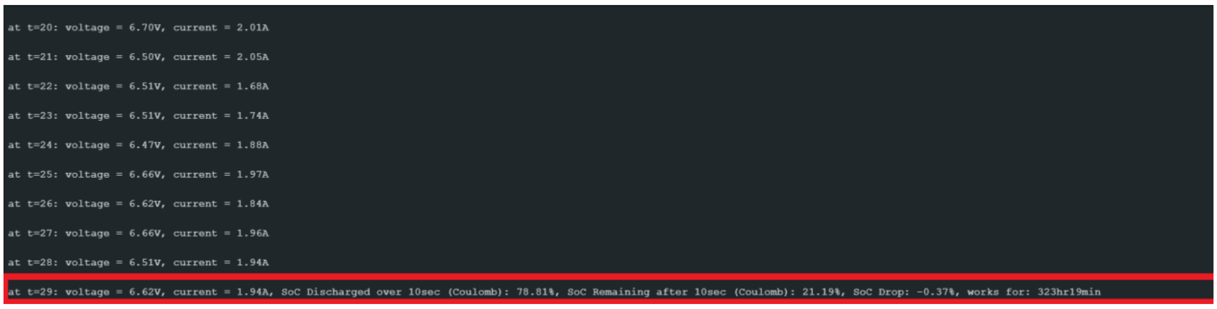
Here, the Initial Discharged SoC (Voltage-Based), Initial Remaining SoC (Voltage-Based), operating time is calculated over the average of 10 voltage and current values (from t=0 to t=9).



```
at t=10: voltage = 6.56V, current = 2.05A

at t=11: voltage = 6.60V, current = 2.01A

at t=12: voltage = 6.51V, current = 1.96A

at t=13: voltage = 6.58V, current = 1.86A

at t=14: voltage = 6.68V, current = 1.94A

at t=15: voltage = 6.54V, current = 1.74A

at t=16: voltage = 6.68V, current = 1.78A

at t=17: voltage = 6.61V, current = 2.09A

at t=18: voltage = 6.56V, current = 1.74A

at t=19: voltage = 6.68V, current = 1.99A, SoC Discharged over 10sec (Coulomb): 78.63%, SoC Remaining after 10sec (Coulomb): 21.37%, SoC Drop: -0.18%, works for: 323hr26min
```

Fig. No 13: - Output of calculated SoC (t=10-19)

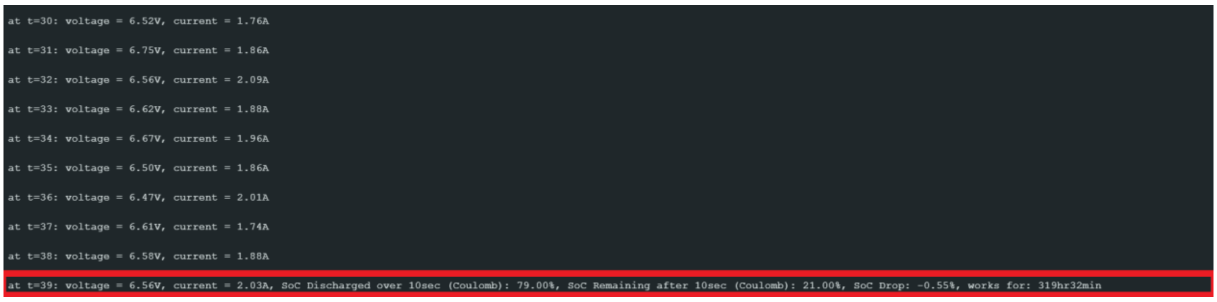
For the next iteration, SoC Discharged over 10 sec (Coulomb), SoC remaining after 10sec (Coulomb), SoC Drop and operating time is calculated by Coulomb Counting from t=10 to t=19, considering Initial SoC from previous step.



```
at t=20: voltage = 6.70V, current = 2.01A

at t=21: voltage = 6.50V, current = 2.05A

at t=22: voltage = 6.51V, current = 1.68A

at t=23: voltage = 6.51V, current = 1.74A

at t=24: voltage = 6.47V, current = 1.88A

at t=25: voltage = 6.66V, current = 1.97A

at t=26: voltage = 6.62V, current = 1.84A

at t=27: voltage = 6.66V, current = 1.96A

at t=28: voltage = 6.51V, current = 1.94A

at t=29: voltage = 6.62V, current = 1.94A, SoC Discharged over 10sec (Coulomb): 78.81%, SoC Remaining after 10sec (Coulomb): 21.19%, SoC Drop: -0.37%, works for: 323hr19min
```

Fig. No 14: - Output of calculated SoC continued (t=20-29)

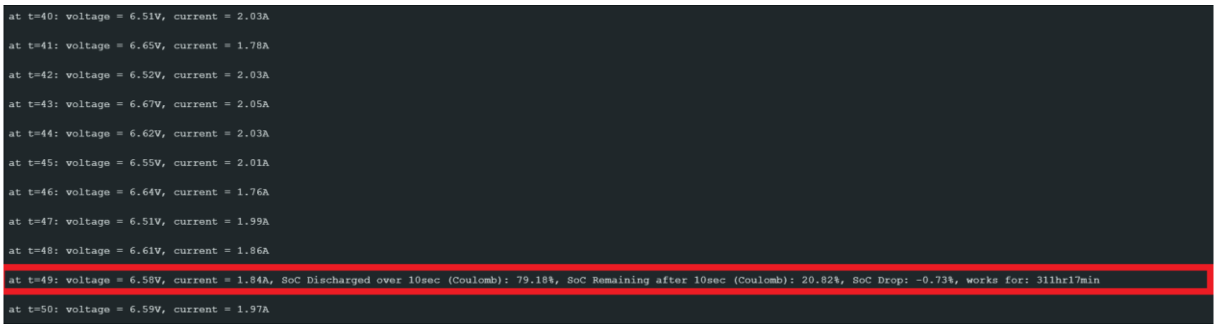
For the next iteration, SoC Discharged over 10 sec (Coulomb), SoC remaining after 10sec (Coulomb), SoC Drop and operating time is calculated by Coulomb Counting from t=20 to t=29, considering SoC remaining after 10 seconds from previous step.



```
at t=30: voltage = 6.52V, current = 1.76A

at t=31: voltage = 6.75V, current = 1.86A

at t=32: voltage = 6.56V, current = 2.09A

at t=33: voltage = 6.62V, current = 1.88A

at t=34: voltage = 6.67V, current = 1.96A

at t=35: voltage = 6.50V, current = 1.86A

at t=36: voltage = 6.47V, current = 2.01A

at t=37: voltage = 6.61V, current = 1.74A

at t=38: voltage = 6.58V, current = 1.88A

at t=39: voltage = 6.56V, current = 2.03A, SoC Discharged over 10sec (Coulomb): 79.00%, SoC Remaining after 10sec (Coulomb): 21.00%, SoC Drop: -0.55%, works for: 319hr32min
```

Fig. No 15: - Output of calculated SoC continued (t=30-39)

For the next iteration, SoC Discharged over 10 sec (Coulomb), SoC remaining after 10sec (Coulomb), SoC Drop and operating time is calculated by Coulomb Counting from t=30 to t=39, considering SoC remaining after 10 seconds from previous step.



```
at t=40: voltage = 6.51V, current = 2.03A

at t=41: voltage = 6.65V, current = 1.78A

at t=42: voltage = 6.52V, current = 2.03A

at t=43: voltage = 6.67V, current = 2.05A

at t=44: voltage = 6.62V, current = 2.03A

at t=45: voltage = 6.55V, current = 2.01A

at t=46: voltage = 6.64V, current = 1.76A

at t=47: voltage = 6.51V, current = 1.99A

at t=48: voltage = 6.61V, current = 1.86A

at t=49: voltage = 6.58V, current = 1.84A, SoC Discharged over 10sec (Coulomb): 79.18%, SoC Remaining after 10sec (Coulomb): 20.82%, SoC Drop: -0.73%, works for: 311hr17min

at t=50: voltage = 6.59V, current = 1.97A
```

Fig. No 16: - Output of calculated SoC continued (t=40-49)

For the next iteration, SoC Discharged over 10 sec (Coulomb), SoC remaining after 10sec (Coulomb), SoC Drop and operating time is calculated by Coulomb Counting from t=40 to t=49, considering SoC remaining after 10 seconds from previous step.

Total SoC Discharged over 50 sec **= 79.18%,** SoC remaining after 50 sec **= 20.82%,** SoC Drop **= 0.73%,** operating time **= 311hr 17min**

For SoH Estimation: Electrochemical Impedance Spectroscopy

Current and Voltage Plot collected over time:

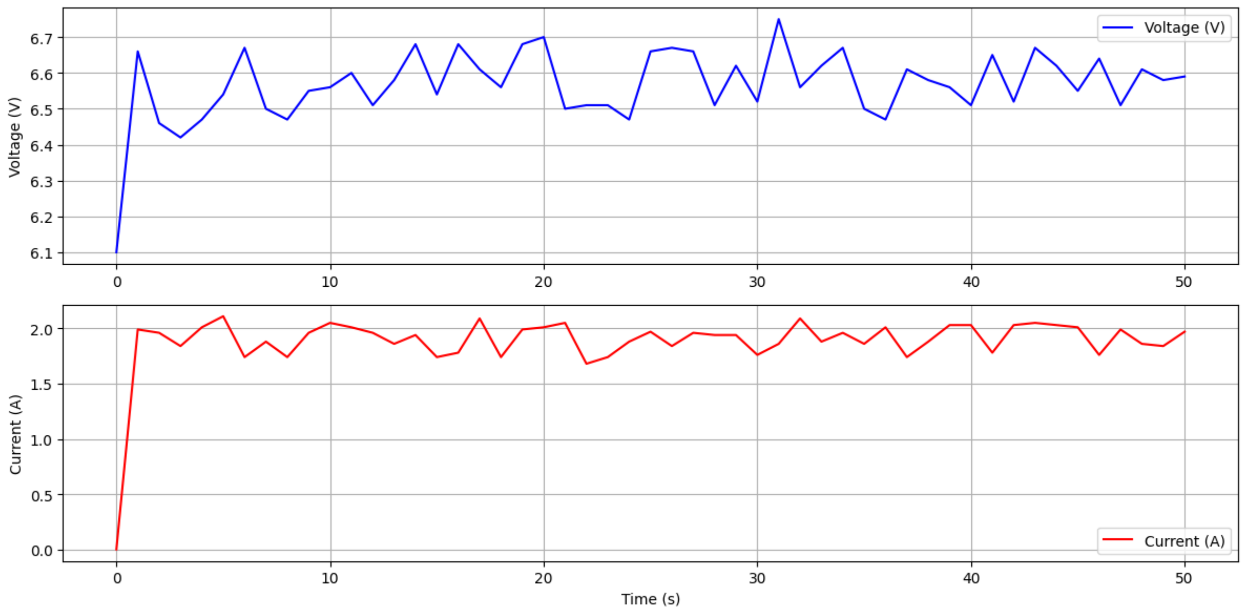


Fig. No 17: - Voltage and Current curves of Battery over time

In the future, we plan to implement a complete Electrochemical Impedance Spectroscopy (EIS) process to estimate the State of Health (SoH) of electrochemical cells. This involves applying a small-amplitude AC signal over a range of frequencies and capturing the resulting voltage and current responses. By performing a Fast Fourier Transform (FFT), we will convert these time-domain signals into the frequency domain and calculate the impedance spectrum using the relation Z(f)=V(f)/I(f). From this spectrum, key parameters such as series resistance R_s, charge transfer resistance R_ct, and double-layer capacitance C_dl will be extracted. These parameters will then be compared with reference values from a new or baseline cell to assess degradation.[14] Specifically, increased values of R_s or R_ct, and decreased C_dl, will indicate aging or performance loss. To quantify the SoH, we will use either a direct ratio for new cells or a weighted combination of all three parameters for in-service batteries. The weights will be chosen based on their sensitivity and relevance to the application, ensuring a more accurate and comprehensive estimation of battery health.

10 Conclusion and Future Scope

This project successfully created a Smart Battery Management System (BMS) tailored for Li-ion batteries, implemented on an Arduino-based platform and incorporated into a robotic car. It combines Voltage method with Coulomb counting—to precisely estimate the State of Charge (SoC), enabling accurate monitoring of battery use and runtime forecasting.

To estimate the State of Health (SoH), we plan to incorporate an Electrochemical Impedance Spectroscopy (EIS) approach. This will allow us to evaluate key impedance-based parameters and assess battery degradation over time, providing a reliable and non-invasive method for monitoring long-term battery health.[6]

Modelling and prototyping were supported through LTSpice simulations and Fritzing schematic designs. The Arduino software managed real-time data collection, system control, and SoC estimation, while Python scripts handled the impedance data processing for SoH evaluation.

In summary, the project achieves real-time battery diagnostics and monitoring, establishing a scalable platform for future developments in the fields of Impedance Spectroscopy and Battery Management Systems.

Future Scope:

The Smart Battery Management System (BMS) developed in this project provides a strong foundation for further advancements in intelligent energy storage diagnostics. Future work can focus on improving estimation accuracy, scalability, and system intelligence through the integration of modern computational techniques and advanced battery technologies.

To enhance State of Charge (SoC) estimation, Kalman filtering methods—such as the Extended Kalman Filter (EKF) or Unscented Kalman Filter (UKF)—can be employed.[1] These filters enable real-time sensor fusion of current, voltage, and temperature data, significantly reducing estimation errors caused by noise, drift, and dynamic load variations.

For State of Health (SoH) prediction, machine learning algorithms can be trained on large datasets comprising historical impedance, voltage, current, and temperature profiles. Models like Random Forests and Support Vector Machines (SVM) can uncover complex degradation patterns, offering predictive insights into battery aging and potential failure modes.[2]

Deep learning techniques, such as Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN), can be introduced to model non-linear battery dynamics. These models learn temporal dependencies in charge/discharge behavior and enable accurate long-term forecasting of SoC and SoH under diverse operational conditions.[5]

Support for regenerative battery systems can be incorporated to accommodate applications involving bidirectional power flow, such as electric vehicles or robotic platforms with braking energy recovery. This requires adapting the SoC algorithms and charging logic to account for intermittent energy input during regenerative events.

Furthermore, online impedance spectroscopy can be optimized using adaptive excitation signals and embedded learning algorithms. This will enable continuous SoH tracking without interrupting regular battery operation, reducing overhead and improving efficiency.[4]

The platform can also be scaled to support multi-cell battery packs, with modular architecture and wireless communication for distributed monitoring.[12] Integration with cloud platforms can facilitate remote diagnostics, data logging, and predictive maintenance across battery fleets.

Finally, thermal and safety management can be improved through the use of predictive analytics. By analyzing temperature trends and operational anomalies using machine learning, the system can pre-emptively detect signs of thermal runaway, internal shorts, or other critical failures.

These enhancements would position the BMS as a comprehensive, intelligent solution suitable for modern energy systems across automotive, robotics, and IoT applications.

11 References

[1] Fahmy, H. M., Swief, R. A., Hasanien, H. M., Alharbi, M., Maldonado, J. L., & Jurado, F. (2023). Hybrid state of charge estimation of lithium-ion battery using the Coulomb counting method and an adaptive unscented Kalman filter. Energies, 16(14), 5558. https://doi.org/10.3390/en16145558

[2] Wang, C., Zhang, X., Yun, X., & Fan, X. (2023). A novel hybrid machine learning Coulomb counting technique for state of charge estimation of lithium-ion batteries. Journal *of* Energy Storage, 63, 107081. https://doi.org/10.1016/j.est.2023.107081

[3] Liu, Y., Wang, L., Li, D., & Wang, K. (2023). State-of-health estimation of lithium-ion batteries based on electrochemical impedance spectroscopy: A review. Protection and Control of Modern Power Systems, 8, 41. https://doi.org/10.1186/s41601-023-00314-w

[4] Li, A. G., Fahmy, Y. A., Wu, M. M., & Preindl, M. (2022, June). Fast time-domain impedance spectroscopy of lithium-ion batteries using pulse perturbation. In 2022 IEEE Transportation Electrification Conference and Expo (ITEC) (pp. 155–160). IEEE. https://doi.org/10.1109/ITEC53557.2022.9813868

[5] Jones, P. K., Stimming, U., & Lee, A. A. (2022). Impedance-based forecasting of lithium-ion battery performance amid uneven usage. Nature Communications, 13, 4806. https://doi.org/10.1038/s41467-022-32422-w

[6] Sangiri, J. B., Kulshreshtha, T., Ghosh, S., Maiti, S., & Chakraborty, C. (2022). A novel methodology to estimate the state-of-health and remaining-useful-life of a Li-ion battery using discrete Fourier transformation. Journal of Energy Storage, 46, 103849. https://doi.org/10.1016/j.est.2021.103849

[7] Analog Devices. (n.d.). A closer look at state of charge and state health estimation techniques. Analog Devices. https://www.analog.com/en/resources/technical-articles/a-closer-look-at-state-of-charge-and-state-health-estimation-tech.html

[8] Telmasre, T. K., Goswami, N., Concepción, A., Kolluri, S., Pathak, M., Morrison, G., & Subramanian, V. R. (2022). Impedance response simulation strategies for lithium ion battery models. Current Opinion in Electrochemistry, 36, 101140. https://doi.org/10.1016/j.coelec.2022.101140

[9] Carthy, K. M., Gullapalli, H., & Kennedy, T. (2021). Review—Use of impedance spectroscopy for the estimation of Li ion battery state of charge, state of health and internal temperature. Journal of The Electrochemical Society, 168(8), 080517. https://doi.org/10.1149/1945-7111/ac1a85

[10] Monolithic Power Systems, Inc. (n.d.). The importance of state of charge (SOC) and state of health (SOH) in battery management systems. https://www.monolithicpower.com/en/learning/resources/the-importance-of-state-of-charge-and-state-of-health-in-battery-management-systems

[11] Texas Instruments. (2021, October). State of charge estimation using smart battery charger (SLUAAA1). https://www.ti.com/lit/ta/sszt475/sszt475.pdf

[12] Lisboa Cardoso, L. A., da Costa, E. F., Scherer, H. F., & Ando Junior, O. H. (2021). An open-source battery management system: Concept and development. IEEE Latin America Transactions, 19(7), 1164–1171. https://doi.org/10.1109/TLA.2021.9461844

[13] Wu, K. H., et al. (2025). Lithium-ion battery state of charge estimation using improved coulomb counting method with adaptive error correction. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 239(2), 123–135. https://doi.org/10.1177/09544070231210562

[14] Zhang, M., Liu, Y., Li, D., Cui, X., Wang, L., Li, L., & Wang, K. (2023). Electrochemical Impedance Spectroscopy: A New Chapter in the Fast and Accurate Estimation of the State of Health for Lithium-Ion Batteries. Energies, 16(4), 1599. https://doi.org/10.3390/en16041599

[15] Panasonic. (2016). Data sheet – NCR18650PF lithium-ion battery [PDF]. Panasonic Energy Co., Ltd. Retrieved from https://actec.dk/media/documents/70FC46554038.pdf

[16] ST Microelectronics. (2023). L298 – Dual full-bridge driver datasheet (DS0218 Rev 5) [PDF]. ST Microelectronics. Retrieved from https://www.st.com/resource/en/datasheet/l298.pdf