GERMAN - MONGOLIAN INSTITUTE
FOR RESOURCES AND TECHNOLOGY

The present work was submitted to
the German-Mongolian Institute of Resources and Technology

DESIGN OF LITHIUM BATTERY MANAGEMENT SYSTEM (BMS)

Bachelor's Thesis

By

Jargal Gurbadam

Study program: Mechatronics Engineering

Student ID: B2100283

1st Supervisor: Ph.D. Odbileg Norovrenchin

2nd Supervisor: M.A. Oyunbileg Shirendev

Ulaanbaatar/Nalaikh

2025

GERMAN - MONGOLIAN INSTITUTE FOR RESOURCES AND TECHNOLOGY

The present work was submitted to
the German-Mongolian Institute of Resources and Technology

DESIGN OF LITHIUM BATTERY MANAGEMENT SYSTEM (BMS)

Bachelor's Thesis

By

Jargal Gurbadam

Study program: Mechatronics Engineering

Student ID: B2100283

1st Supervisor: Ph.D. Odbileg Norovrenchin

2nd Supervisor: M.A. Oyunbileg Shirendev

Ulaanbaatar/Nalaikh

2025

Statutory Declaration

Gurbadam Jargal	B2100283
Last Name, First Name	Student ID Number

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

DESIGN OF LITHIUM BATTERY MANAGEMENT SYSTEM (BMS)

I did not use any sources other than those stated. In case that the work is additionally submitted on a data medium, I declare that the written and the electronic form are completely identical. The work was not submitted in the same or similar form to any examination authority.

Nalaikh, Ulaanbaatar, 28 May 2025	
Place, Date	Signature

Table of Contents

Abstract

This thesis presents the design and prototyping of a low-cost Battery Management System (BMS) for lithium-ion cells using an Arduino Uno platform. The project focuses on implementing essential BMS functions, including voltage, current, and temperature monitoring, along with fault detection for overvoltage, undervoltage, overcurrent, and overtemperature conditions. Readings are acquired through affordable and widely available sensors and processed in real-time to flag unsafe operating states.

A physical prototype was assembled on a breadboard using modular components such as voltage dividers, ACS712 current sensors, and DS18B20 temperature sensors. Charging was handled by a TP4056 module, and monitored parameters were displayed via the Serial Monitor. Additionally, a MATLAB-based thermal simulation was conducted to validate safe temperature behavior during operation. While full charge-discharge testing was not performed, the system logic and component integration were verified through controlled simulations and code-based validation.

The results confirm that a simple, expandable BMS can be constructed using low-cost hardware and embedded software. This work lays the foundation for future improvements such as automatic cutoff, multi-cell support, and wireless monitoring.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor and advisor Professor of Mechatronics Engineering Odbileg N., and my second supervisor and advisor Master of Educational Research Ms. Oyunbileg Sh. for their support and guidance. Their immense knowledge and experience in their respective fields were irreplaceable pillars to help me put this paper together. I appreciate them not only for their technical assistance but also for their countless counselings as mentor figures in my life. Their personal guidance will be a stepping stone for me to become a better engineer and more importantly a better person.

Furthermore, I would like to extend my thanks to all of the professors, teachers, staff, the rectors and fellow students at the German-Mongolian Institute of Technology. Only this university is worthy to have been graced by your presence. It has been a wonderful experience to be taught and looked after by all of you. I am honored to have been a part of this academic palace for even a short amount of time as my bachelor years have allowed me.

I would also like to thank the German-Mongolian Institute of Technology, and the German Academic Exchange Service also known as DAAD for their belief in me and my studies, and for allowing me to overlook any financial burdens during my studies with their scholarships throughout the years.

Last but not least, I have to thank my friends and family for supporting and believing in me the entire way. It brings me much joy to have had you all by my side, and I predict with full certainty you will continue to bring me much more happiness and success in the future. Your camaraderie and love give me the strength and courage to always look forward to what's next, and to keep going no matter what.

I dedicate this milestone to all of you, this success would not have been possible without you. Once more, my deepest gratitude to everyone.

List of figures

List of tables

List of codes

1. Introduction

1.1. Background

Ever since the first electrochemical battery, the voltaic pile made in the 1800 by Italian physicist Alessandro Volta, electric battery technologies have been developed at a steady rate. The objective of this development is to improve the battery life and efficiency while maintaining safe operations at all times. Following the path of this development, we have since 2010 reached an all time high in battery demand in no small part due to the emergence of the smart devices that are a mainstay in our lives today. Major reasons for this meteoric rise are as mentioned above are smart devices but also the emergence of the electrification of transportation options (e.g. EVs, trains) and many large-scale electric grids, backed largely by the initiative of decarbonization globally. In the midst of all this, there has been an industry favorite standard in the form of the lithium-ion battery.

The lithium-ion battery also known as Li-ion battery is a rechargeable battery that uses reversible exchange of the Li^+ ions between anode and cathode. What makes the Li-ion battery so special is that it has a high cell voltage, high energy and power densities and low self-discharge, while also having a longer expected cycle and calendar lifetime than other battery technologies. Though these batteries come in comparatively high initial costs but due to their reusability, which is also better environmentally compared to disposable batteries, they have become a go-to for a majority of our electronic devices (mobile phones, laptops, cameras etc.), cordless electric tools, and electric vehicles.

However, one large takeaway from the Li-ion batteries is that safety-wise, it can vary from being completely harmless to extremely dangerous in varying conditions. The problem is that it is very unstable at high temperatures, and it is very specific about its charge level limits. When conditions such as overcharging, overheating, puncture, or a short-circuit are met, thermal runaway may most likely happen. Also, because of its very specific limits regarding charge levels, cell degradation and structural damage is imminent when uncontrolled. It is of utmost importance that Li-ion cells are always monitored and its operating conditions regulated during use attributed to the dangers aforementioned. Battery Management Systems (BMS) for Li-ion batteries are created with these aims in mind, to ensure safe, efficient use of the cells by constant monitoring and control of specific parameters, optimizing performance, lifespan and preventing

conditions that may lead to degradation, thermal runaway, and any other potential failures.

This thesis focuses on the design aspects and prototyping of a lithium-ion battery management system. The work involves the selection and circuitry of key hardware components, development of monitoring and fault detection logic, and testing through both simulation and physical implementation. The aim of the thesis is to contribute toward safer, longer-lasting, and more sustainable batteries through battery management systems.

1.2. Motivation

The modern transition to electric transportation, renewable energy systems, and a heavy reliance on portable electronics has dramatically increased the demand for efficient and reliable energy storage. Lithium-ion batteries have become central to this shift due to their high energy density and performance, but their safety and longevity still remain as ongoing challenges to combat against.

Battery-related incidents — from appliances not working to catastrophic thermal events — often stem from lackluster monitoring and/or control. While Battery Management Systems (BMS) are a proven and the go-to solution, many remain unreliable, poorly-made or conversely, expensive or too complex for small-scale use. There is a need for simplified, reliable, cost-effective BMS prototypes that can be implemented, tested, and adapted by students, researchers, or makers.

This project is motivated by the opportunity to design a working prototype of a small lithium-ion battery pack BMS that addresses fundamental needs: voltage and temperature monitoring, fault detection — all while remaining accessible and reproducible using widely available tools like Arduino and other electronic elements. By building a prototype grounded in real-world use cases and validating it through simulation and testing, this thesis contributes a practical stepping stone toward safer, smarter battery integration.

Moreover, as lithium battery usage continues to scale, effective BMS design plays an increasingly important role in minimizing e-waste, preventing battery degradation, and supporting the environmental sustainability of energy systems. These motivations align with both technical innovation and global responsibility.

1.3. Objectives

The objective of this thesis is to design and prototype a simplified Battery Management System (BMS) for lithium-ion battery packs while also delving into current trends and technologies in the BMS field. The project balances practical implementation with research and investigation. The specific objectives are:

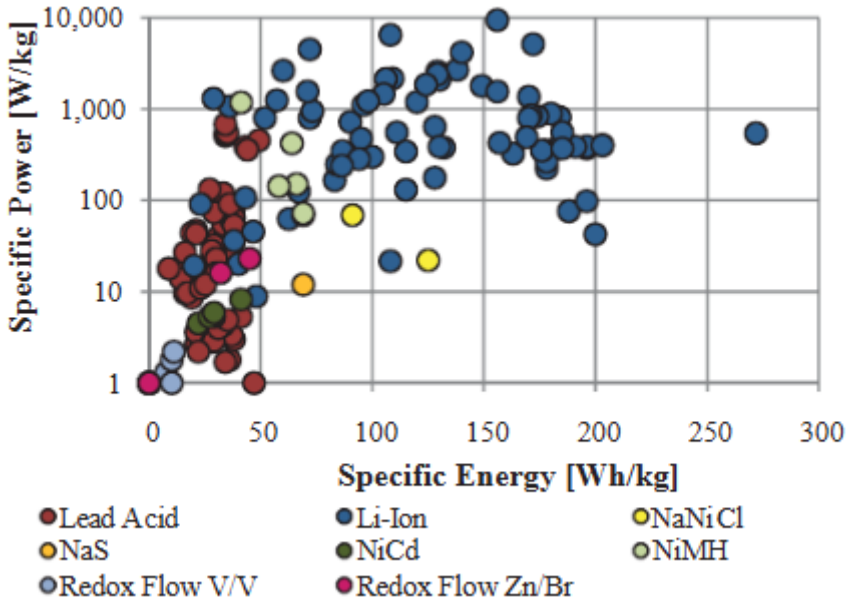
- Design and construct a functional BMS prototype using an Arduino microcontroller for voltage and temperature monitoring, fault detection, and basic cell balancing.
- Implement and test fault detection logic capable of identifying overvoltage, undervoltage, and temperature conditions based on sensor data.
- Simulate battery behavior and system response using MATLAB and LTSpice to validate the design under various conditions before physical testing.
- Evaluate limitations and propose recommendations for improving prototype performance, extending BMS functionality, or adapting it for more advanced applications (e.g., EVs or second-life battery use).

2. Literature review

2.1. Overview on Li-ion Batteries

Lithium-ion (Li-ion) batteries have become the dominant energy storage technology across a wide range of applications, including portable electronics, electric vehicles (EVs), and renewable energy storage systems. In a report, it is noted that the li-ion demand was set to increase by 26% from the end of 2023 to the end of 2024, and that the demand was at an all time high at 1TWh. It is also emphasised on the same report, that the BESS (Battery Energy Storage System) market growth dwarfs the growth of EV's even though the latter's market makes the majority of the market share. The demand for li-ion in stationary energy storage applications has grown from 7% to 15% in four years from 2020-2024, marking itself as the most rapidly growing demand in the battery market [1]. The Li-ion widespread adoption is largely attributed to their high energy density, long lifecycle, and low self-discharge rate compared to older chemistries such as nickel-metal hydride (NiMH) or lead-acid batteries [2]. On the plot shown on Figure 1, it is visible that the li-ion battery technologies are mostly on the high end on both axes.

Figure 1: Ragone plot of different batteries. *Reproduced from Stenzel et al. [2], based on data from KIT/FZJ*



In 2023, the global market was dominated by lead-acid batteries, taking up a whopping 68% of the market shares of all rechargeable battery types, and 16% taken up by the li-ion battery. However, it is noted that the demand for the Li-ion battery is rising rapidly, with the market in favor of its 5000 charge cycles compared to the underwhelming 1200 charge cycles of its lead-acid counterpart, ultimately resulting in a lower total cost at the end of its lifecycle [3]. Various sources also report that lithium-ion batteries can charge from 20% to 80% in as little as 2 to 3 hours, while equivalent lead-acid batteries may take 6 hours or more, depending on the charger and battery design [4][5].

Regarding the general working principle of the Li-ion battery, they operate based on the reversible movement of lithium ions between two electrodes in an electrolyte: a graphite anode and a lithium metal oxide or phosphate cathode, and the electrolyte is a lithium salt in an organic solvent. During charge, lithium ions migrate from the cathode to the anode through the electrolyte, where they are stored in the layered graphite structure. Simultaneously, electrons flow through the external circuit, completing the charge transfer. During discharging, the process is reversed: lithium ions move back to the cathode, and the stored energy is released as electrical power.

Each cell typically provides a nominal voltage of 3.6–3.7 V, with a full-charge voltage of up to 4.2 V and a discharge cutoff around 3.0 V. The electrolyte, often a lithium salt dissolved in an organic solvent, facilitates ionic conduction while the separator prevents physical contact between the electrodes, minimizing the risk of internal short circuits. The overall performance of a lithium-ion cell is influenced by factors such as temperature, current load, and the number of charge/discharge cycles, all of which contribute to its gradual capacity loss and degradation over time.

Due to their sensitive electrochemical nature, lithium-ion batteries require careful management to operate safely and efficiently. Overcharging can lead to hazardous elevated temperatures and potential thermal runaway — a dangerous cyclic process in which escalated temperatures increase the flowing current, which in turn increases temperature, while over-discharging may cause permanent capacity loss or internal structural damage. Lithium plating is one degrading mechanism, where metallic lithium deposits onto the anode surface instead of being properly absorbed into the graphite's structure. This occurs mostly in high-rate charging and low temperatures and leads to reduced capacity and potentially in severe cases, an internal short circuit. Eventually, this leads to lithium dendrites, forming thin needle-like structures that may pierce through the separators and cause cell failure. Additionally, in multi-cell configurations, slight variations in capacity or resistance can cause individual cells to become unbalanced. These risks highlight the critical role of a Battery Management System (BMS), which monitors voltage, current, and temperature parameters to ensure safe operation and optimal battery health throughout its lifecycle [2].

2.2. Lithium-ion Charging

Lithium-ion batteries are charged using a two-stage process known as the stages Constant Current/Constant Voltage (CC/CV). In the constant current phase, the battery is charged at a fixed current until the voltage across the terminals reaches the upper charge limit—typically 4.2 V per cell. During this stage, the battery accepts current at a steady rate and charges rapidly [7].

Once the terminal voltage reaches 4.2 V, the charger transitions into the constant voltage phase, where the voltage is held steady while the charging current gradually tapers off. Charging is considered complete when the current falls below a specified cutoff threshold (often around 0.05C to 0.1C of the battery's rated capacity).

This method ensures that the battery reaches full capacity without overcharging, which could otherwise lead to lithium plating, heat buildup, or reduced cycle life.

2.3. Functions of a Battery Management System (BMS)

A Battery Management System (BMS) is an essential control unit responsible for maintaining the safety, reliability, and efficiency of lithium-ion battery packs. As the complexity of battery-powered systems increases—particularly in electric vehicles (EVs), grid storage, and renewable integration—so does the importance of the BMS. It ensures optimal operation by performing several critical functions, outlined below:

- Voltage Monitoring
  The BMS continuously measures the voltage of each individual cell or group of cells. By doing so, it ensures that no cell exceeds its upper voltage limit (typically around 4.2 V for Li-ion) or drops below its minimum safe threshold (typically around 3.0 V). This helps prevent overcharging and over-discharging, both of which can lead to capacity loss or safety hazards.

- Current Monitoring
  Measuring current flow allows the BMS to track charging and discharging activity. Excessive current—due to short circuits, faulty loads, or aggressive charging—can damage cells and increase the risk of thermal events. The BMS can shut down or throttle the system in response to abnormal current levels.

- Temperature Monitoring
  Lithium-ion battery performance and safety are strongly affected by temperature. The BMS uses sensors to track cell and ambient temperature and may cut off charging or discharging if the temperature falls outside safe operating ranges (typically 0–45°C for charging and –20–60°C for discharging). This helps prevent overheating and thermal runaway.

- State of Charge (SOC) Estimation
  The BMS estimates the remaining charge in the battery using methods such as coulomb counting, open-circuit voltage measurement, or more advanced algorithms like Kalman filters. Accurate SOC estimation is crucial for determining runtime, managing charge cycles, and extending battery life.

- Cell Balancing

  Over time, individual cells in a battery pack can drift apart in voltage due to manufacturing tolerances and aging. The BMS equalizes these voltages through passive balancing (dissipating energy through resistors) or active balancing (redistributing energy between cells), ensuring consistent performance and maximizing usable capacity.

- Protection and Fault Detection

  The BMS serves as a protective interface that responds to fault conditions such as overvoltage, undervoltage, overcurrent, and overtemperature. Upon detection, it can trigger warnings, disable charging/discharging, or shut the system down entirely to prevent damage or fire.

- Data Logging and Communication

  Advanced BMS designs include data logging features that record operational history, fault events, and usage patterns. Many also support communication via CAN bus, UART, or I2C protocols, allowing integration with vehicle control units or cloud-based monitoring platforms. In summary, the BMS acts as both the guardian and manager of the battery system. It balances safety, performance, and longevity through a combination of real-time monitoring, protective action, and intelligent control.

2.4. State of Charge (SOC)

State of Charge (SOC) refers to the estimated amount of usable energy remaining in a battery, expressed as a percentage of its total capacity. An SOC of 100% indicates a fully charged battery, while 0% represents a fully discharged state. Accurate SOC estimation is critical for effective battery management, as it influences system decisions such as charging, discharging, load control, and safety responses.

SOC cannot be measured directly; instead, it is inferred using mathematical models and real-time data. Several methods are commonly used for SOC estimation:

- Coulomb Counting

  This method calculates SOC by integrating the current flowing in and out of the battery over time. While simple, it is prone to cumulative errors due to sensor drift and requires an accurate starting SOC value.

- Open Circuit Voltage (OCV) Method
  The OCV method estimates SOC based on the voltage of the battery when it is at rest (i.e., no load or charging). Each battery chemistry has a known OCV–SOC relationship. However, this method requires the battery to be idle for a significant period, which is often impractical in real-time systems.
- Model-Based Estimation (e.g., Kalman Filters)
  Advanced estimation methods such as the Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), or Adaptive Filters combine voltage, current, and temperature data with battery models to dynamically predict SOC. These approaches offer high accuracy and robustness, especially under varying load and temperature conditions.
- Machine Learning Approaches
  Emerging SOC estimation methods use neural networks, support vector machines, or regression models trained on historical battery data. These are especially useful in complex or nonlinear systems where physical modeling is difficult.

An accurate SOC estimation algorithm ensures optimal performance and prevents harmful conditions such as deep discharge or overcharging, both of which degrade battery life. For applications like electric vehicles or renewable energy systems, reliable SOC feedback is also essential for range prediction and system-level energy management.

2.5. State of Health (SOH)

State of Health (SOH) is a key performance metric used to assess the overall condition and aging status of a battery compared to its original, nominal condition. It is usually expressed as a percentage, where 100% indicates that the battery is operating at full capacity and efficiency, and values below that reflect degradation due to age, usage, or environmental stress.

Mathematically, SOH is often defined as:

$$SOH = \left(\frac{C\ actual}{C\ rated}\right) \times 100\%$$

Where:

- *C actual* is the battery's current maximum capacity

- *C rated* is the rated capacity when new

As batteries undergo charge-discharge cycles, their capacity and internal resistance change due to electrochemical aging, mechanical stress, and side reactions. Monitoring SOH is essential for ensuring long-term reliability, planning maintenance or replacement, and avoiding unexpected failures.

Common SOH Estimation Methods:

- Capacity Fade Measurement
  In controlled conditions, SOH can be evaluated by performing a full charge/discharge cycle and measuring the delivered capacity. This method is accurate but not suitable for real-time systems.
- Internal Resistance Tracking
  As batteries age, their internal resistance tends to increase. Many BMS units estimate SOH by tracking changes in internal impedance, especially during charging or high-current events.
- Model-Based Estimation
  Algorithms such as Kalman filters, particle filters, or equivalent circuit models can infer SOH in real time using voltage, current, and temperature data combined with aging models.
- Data-Driven and Machine Learning Approaches
  Advanced SOH estimation techniques involve machine learning models trained on large datasets, enabling accurate predictions even under variable operating conditions.

A reliable SOH estimation algorithm enables predictive maintenance, enhances user safety, and supports optimal utilization of battery energy throughout its lifecycle. In systems such as electric vehicles, SOH also plays a role in warranty enforcement and resale value assessment.

2.6. Cell Balancing

In multi-cell battery packs, especially those using lithium-ion chemistry, individual cells often exhibit small differences in capacity, internal resistance, and aging rate. Over time and through repeated charge-discharge cycles, these differences lead to imbalance—where some cells become fully charged or discharged earlier than others. If left unaddressed, cell imbalance can reduce the overall usable capacity of the

battery pack, increase the risk of overcharging or over-discharging certain cells, and accelerate degradation.

Cell balancing is a critical Battery Management System (BMS) function that ensures all cells in a series-connected pack maintain similar voltage levels. This allows the pack to operate safely and efficiently, maximizing both capacity and lifespan.

There are two types of cell balancing:

1. Passive Balancing

Passive balancing dissipates excess energy from higher-voltage cells as heat, usually through resistor-based bleed circuits. This method is simple and low-cost, making it common in small- to medium-scale applications.

- Advantages: Easy to implement, requires minimal hardware
- Disadvantages: Energy is wasted as heat, not suitable for large packs or high-efficiency systems

2. Active Balancing

Active balancing redistributes charge from higher-voltage cells to lower-voltage ones using capacitors, inductors, or DC–DC converters. This process conserves energy rather than wasting it as heat.

- Advantages: Energy-efficient, scalable for high-capacity systems
- Disadvantages: More complex, requires additional circuitry and cost

Cell balancing is typically performed:

- During charging (most common)
- Periodically at full charge (when cell voltages diverge most clearly)
- In some systems, even during idle or discharge phases

The BMS monitors the voltage of each cell and activates balancing only when the voltage deviation exceeds a certain threshold (e.g., ±50 mV). In passive systems, shunt resistors are activated to bleed current from cells that reach the upper threshold sooner than others.

Without cell balancing, a single weak or overcharged cell can compromise the safety and performance of the entire pack. Over time, this results in:

- Reduced overall pack capacity
- Faster aging of specific cells
- Increased risk of thermal or electrical faults

Balancing helps maintain the health of all cells within the pack, supports accurate State of Charge (SOC) estimation, and contributes to the longevity and reliability of the battery system.

2.7. Fault Detection and Protection

Fault detection is one of the most critical safety functions of a Battery Management System (BMS). It involves identifying abnormal conditions in voltage, current, temperature, and cell behavior that could pose a risk to battery performance, system reliability, or user safety. Early fault detection allows the BMS to trigger protective actions such as shutting down charging/discharging circuits, alerting users, or isolating faulty components.

Lithium-ion batteries are sensitive to several fault types, including overvoltage, undervoltage, overcurrent, short circuits, overtemperature, and internal cell failures. Without timely intervention, these faults can lead to reduced lifespan, cell damage, or hazardous events such as thermal runaway.

Common Types of Detectable Faults:

- Overvoltage
  Occurs when the voltage of one or more cells exceeds the safe maximum (e.g., 4.2 V for typical Li-ion cells). Prolonged overvoltage can lead to electrolyte breakdown or lithium plating.
- Undervoltage
  Occurs when a cell's voltage drops below its minimum safe level (e.g., ~3.0 V). Undervoltage stresses the cell and may cause permanent capacity loss or safety issues during recharge.
- Overcurrent
  Triggered by a current that exceeds design specifications, typically caused by short circuits, damaged loads, or malfunctioning chargers. Overcurrent can overheat the battery and damage internal connections.
- Short Circuit
  A severe form of overcurrent fault where current bypasses the load path,

potentially leading to immediate overheating or fire. High-speed detection is necessary to disconnect the circuit before damage occurs.

- Overtemperature
  Excessive heat during charging or discharging—especially in combination with high current—can accelerate degradation and trigger thermal runaway. Temperature sensors allow the BMS to suspend operation when limits are exceeded.
- Sensor Failure or Drift
  A well-designed BMS can also detect anomalies in its own sensing systems. For example, unrealistic voltage or temperature readings may indicate sensor disconnection, failure, or drift.

Fault Detection Methods:

- Threshold-Based Monitoring
  Compares real-time readings to preset safe limits. Simple and fast, but may miss gradual or intermittent faults.

- Model-Based Detection
  Uses battery models to predict expected behavior and compares it to actual performance. Deviations indicate potential faults.
- Machine Learning-Based Detection
  Advanced BMSs may use classification models or neural networks trained on historical fault data to detect complex or nonlinear failure modes.

Response Strategies:

Once a fault is detected, the BMS may:

- Disable charging or discharging
- Trigger fault or warning LEDs
- Log the fault for diagnostics
- Isolate the affected cell or module (in modular systems)
- Communicate fault data to an external controller via CAN, UART, or other protocols

By combining real-time sensing with responsive control logic, fault detection mechanisms significantly enhance the operational safety and resilience of lithium-ion

battery systems. This is particularly vital in high-stakes applications such as electric vehicles, aerospace systems, and stationary energy storage.

2.8. Existing BMS & Research Gaps

Numerous Battery Management System (BMS) architectures have been developed to address the needs of various applications, from consumer electronics to electric vehicles (EVs) and large-scale energy storage systems. These systems vary in complexity depending on factors such as battery size, cell configuration, and performance requirements.

In commercial applications, BMS designs typically include high-accuracy voltage and current sensors, temperature sensing, thermal management systems, and communication protocols such as CAN bus. Many automotive BMSs also include complex algorithms for state-of-charge (SOC), state-of-health (SOH), and remaining useful life (RUL) estimation. Additionally, advanced systems often support active balancing, wireless connectivity, and integration with vehicle control units or grid interfaces.

Despite these developments, several limitations remain in both research and commercial systems. One key challenge is balancing cost, complexity, and performance. Many existing solutions are proprietary, expensive, and difficult to adapt to custom or low-cost applications. For educational, research, or prototyping purposes, developers often lack access to flexible and open BMS platforms.

Another area with active research interest is improving the accuracy and robustness of SOC and SOH estimation. Traditional estimation methods such as Coulomb counting and voltage-based models can be inaccurate under dynamic load conditions. Research is ongoing into advanced algorithms, including Kalman filters, observer-based models, and machine learning approaches, to address these limitations.

Thermal management and fault detection remain critical areas as well. While most commercial BMSs include basic protection functions, accurate prediction and detection of thermal runaway, internal short circuits, and sensor failures are still under development. Additionally, the integration of real-time diagnostics and predictive maintenance features remains limited in lower-tier systems.

Recent trends also include the development of cloud-based BMSs that enable remote monitoring, firmware updates, and data analytics. While promising, such systems raise concerns about data security, latency, and infrastructure requirements, especially in large-scale deployments.

In summary, while significant progress has been made in BMS design, there is an ongoing need for cost-effective, modular, and adaptable systems that support accurate diagnostics, fault tolerance, and integration with intelligent energy platforms. This research aims to contribute to that space by designing and testing a simplified yet functional BMS prototype suitable for educational and low-cost experimental use.

3. Methodology

The Methodology chapter outlines the approach taken to design, simulate and prototype a simple Arduino-based Battery Management System for a simple 2-pack Li-ion battery cell. It describes the workflow from literature review to the system design, simulation, hardware implementation, and testing. Each phase is presented in detail in the following sections.

3.1. Overview of Methodology

The development of the Battery Management System (BMS) prototype in this project followed a structured, multi-phase process combining theoretical research, circuit design, simulation, hardware implementation, and testing. The methodology was selected to ensure the system could be validated at each stage before progressing to physical prototyping.

The process began with a literature review to establish a clear understanding of lithium-ion battery behavior, common BMS functions, and the challenges associated with monitoring and protecting battery packs. Based on this foundation, the system was designed to include essential features such as voltage and temperature monitoring, fault detection, and basic cell balancing.

Circuit components and microcontroller logic were first validated in software using MATLAB and LTSpice simulations. These tools allowed for rapid testing of theoretical behavior, such as charge/discharge curves, threshold-based protection, and balancing activation. Once the design was verified, a physical prototype was

assembled using an Arduino-based platform, voltage dividers, temperature sensors, and current sensors.

The final phase involved testing the prototype under controlled conditions to evaluate its performance, accuracy, and fault response. Results were used to assess the system's reliability and identify areas for potential improvement in future iterations.

3.2. System specifications

The main goal of this project was to design and prototype a basic Battery Management System (BMS) for lithium-ion batteries using low-cost and accessible components. To achieve this, the methodology was broken down into five main phases: literature review, system design, simulation, hardware prototyping, and testing. Each step built upon the last, with the aim of creating a working model that could monitor battery conditions, detect faults, and provide basic balancing functions.

The process began with reviewing existing research papers, technical documents, and commercial BMS products to understand the key functions and requirements of a BMS. This helped define what features the prototype would include and what the limitations would be, especially when working with simple hardware like Arduino and off-the-shelf sensors.

Once the background was clear, the design phase focused on selecting the right components and building the system's structure. This included choosing how to measure voltage and temperature, how to detect faults, and how to activate balancing circuits if needed. A block diagram and wiring plan were created to organize the connections between sensors, the microcontroller, and the batteries.

Before building anything, parts of the system were tested in simulation software. LTSpice was used to test analog circuits like voltage dividers and balancing resistors, while MATLAB was used to simulate battery behavior, charging and discharging curves, and SOC estimation models. This step helped confirm that the chosen circuit layout and logic would work in real life.

After simulations showed acceptable results, the hardware was assembled on a breadboard. An Arduino Uno was used as the main controller, and voltage dividers, thermistors, and current sensors were added to measure the state of the batteries.

Code was written to process the sensor data and carry out basic protection actions like cutting off the load or alerting the user when a fault occurred.

Finally, the system was tested by applying different conditions to see if the BMS responded correctly. Voltage and temperature readings were compared with multimeter data, fault detection was triggered manually to confirm it worked, and the balancing circuit was checked using mismatched cells. The results were recorded and used to evaluate the overall performance of the prototype.

3.3. Hardware Design

The hardware design for the BMS prototype should be kept simple and low-cost, especially since in Mongolia there aren't many readily available components and elements. Nonetheless, it should be enough to pass as a BMS — the primary goals being: charging the cells properly, monitoring at all times during operation, and being able to power a load.

3.3.1. System Architecture

The architecture of the prototype is a centralized BMS, built around an Arduino Uno microcontroller. The Arduino Uno is perfect for the prototype as it has pins to plug jumper wires into and changing wiring would be simple, and also wiring onto a breadboard would be more spaced out and organized. As the core, the Arduino is tasked with handling sensor data acquisition, forwarding to an HMI — be it an I2C 16x2 LCDisplay, or on the Arduino IDE on PC. Furthermore, the Arduino may control condition checking, and output control.

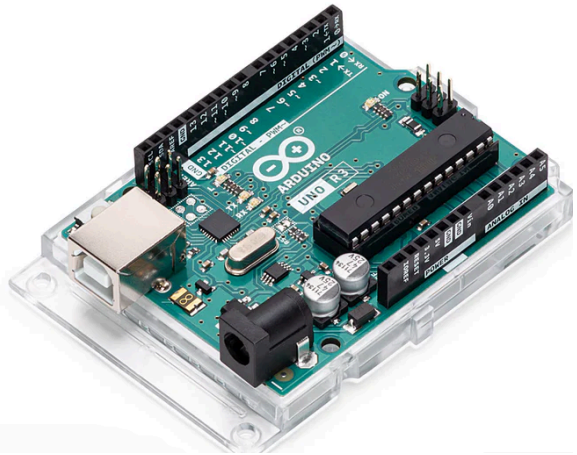
The system is designed to monitor two separate lithium-ion cells, their voltages and temperatures as they are charged and then discharged. The two lithium-ion cells can be both connected in series or parallel, both of which have their payoffs and problems, but for this work, it will only be designed as separate cells.

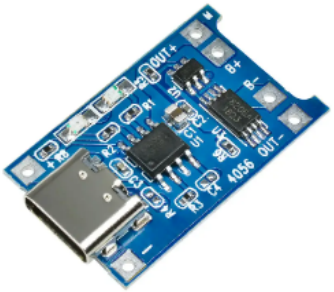
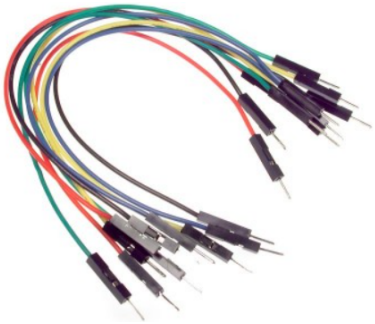
The sensing functions will be operated separately, with a 0-25V voltage sensor parallel to the cells and connected to an analog pin on the Arduino, the temperatures will be sensed through a DS18B20 sensor and connected to digital pins along with 4.7kOhm pull-up resistors, and the currents will be sensed by ACS712 modules at positions in the circuit where it can read the charging and discharging currents and

send the data to analog pins on the Arduino board. These will all be powered by the Arduino's 5V output through a breadboard setup.

3.3.2. Component Selection

The components selected for the prototype are simple, common marketed components for Arduino projects and the like. In this project however, the collected components were as below:

Components	Amount
Arduino Uno 	1
ACS712-20A Current Sensor 	2
0-25V Voltage Sensor 	2

TP4056 Li-ion Battery Charging Module 	2
DS18B20 Temperature Sensor 	2
4.7kOhm Resistor 	2
Jumper Wires 	~20
18650 Li-ion Battery	2

	
18650 Battery Holder 	2

Table 1: Components selected for the prototype

The Arduino Uno is a reliable, versatile microcontroller board based around the ATMega328P with 6 analog input pins and 14 digital input/output pins. It is a go-to choice for student projects and beginner-level electronics enthusiasts. Therefore, it was an easy choice to make it the main controller of the prototype.

The ACS712 is an AC/DC current sensor, hall-effect based linear current sensor with bidirectional sensing functionalities. It has 5A, 20A, and 30A versions, but the one used in this thesis is the 20A version. It outputs an analog voltage linearly proportional to the current flowing through it and is based at 2.5V when there is no current flowing. The output voltage increases or decreases with a sensitivity of 100mV/A when current flows through it. The sensor operates on 5V supply, perfectly compatible with the Arduino. It is a non-intrusive sensor with bidirectional current sensing which is suitable for reading charging and discharging modes. Moreover, it is widely used, making it also widely available.

The 0-25V Voltage sensor module is a simple analog voltage measurement device, used commonly to scale down the input voltage to 0-5V, safe for microcontroller configurations. It uses basic resistors to create a voltage divider circuit, stepping the voltage down to a fifth of its original value. The module includes a standard 3-pin interface, easy for breadboard and Arduino connections, while also operating on a 5V supply. It is a very simple and widespread component that gets the job done.

TP4056 is a linear charging module for single-cell Li-ion batteries. It provides constant-current/constant-voltage (CC/CV), and it is widely used as a low-cost, compact charging solution. The module has in-built charge status indicating LEDs, input overvoltage protection and battery over-discharge protection. The TP4056 operates on a separate 5V supply (usually USB) and charges the battery at a preset current and ends when the battery voltage reaches 4.2V. On the module, there are also output terminals which may be connected to the load the battery should supply.

DS18B20 is a one-wire digital temperature sensor. It has a wide operating range from -55°C to +125°C with an accuracy of ±0.5°C in the –10°C to +85°C range. Unlike analog thermal sensors, this one communicates via the 1-Wire digital protocol, allowing the data and power to be transmitted over a single data line. This also allows multiple DS18B20 sensors to output to one microcontroller pin. The sensor is used to monitor the battery temperature at all times as accurate and reliable data is of paramount importance during the charging process, making it one of the most vital parts in the whole prototype model. The sensor also very peculiarly needs a 4.7kOhm pull-up resistor to function properly, connected to its data line.

3.3.3. Circuit Design

The following circuit was designed to monitor both charging and discharging current of a 3.7 V Li-ion battery, while simultaneously measuring voltage levels and the temperature, and a placeholder load. This forms the core sensing mechanism for the prototype Battery Management System.

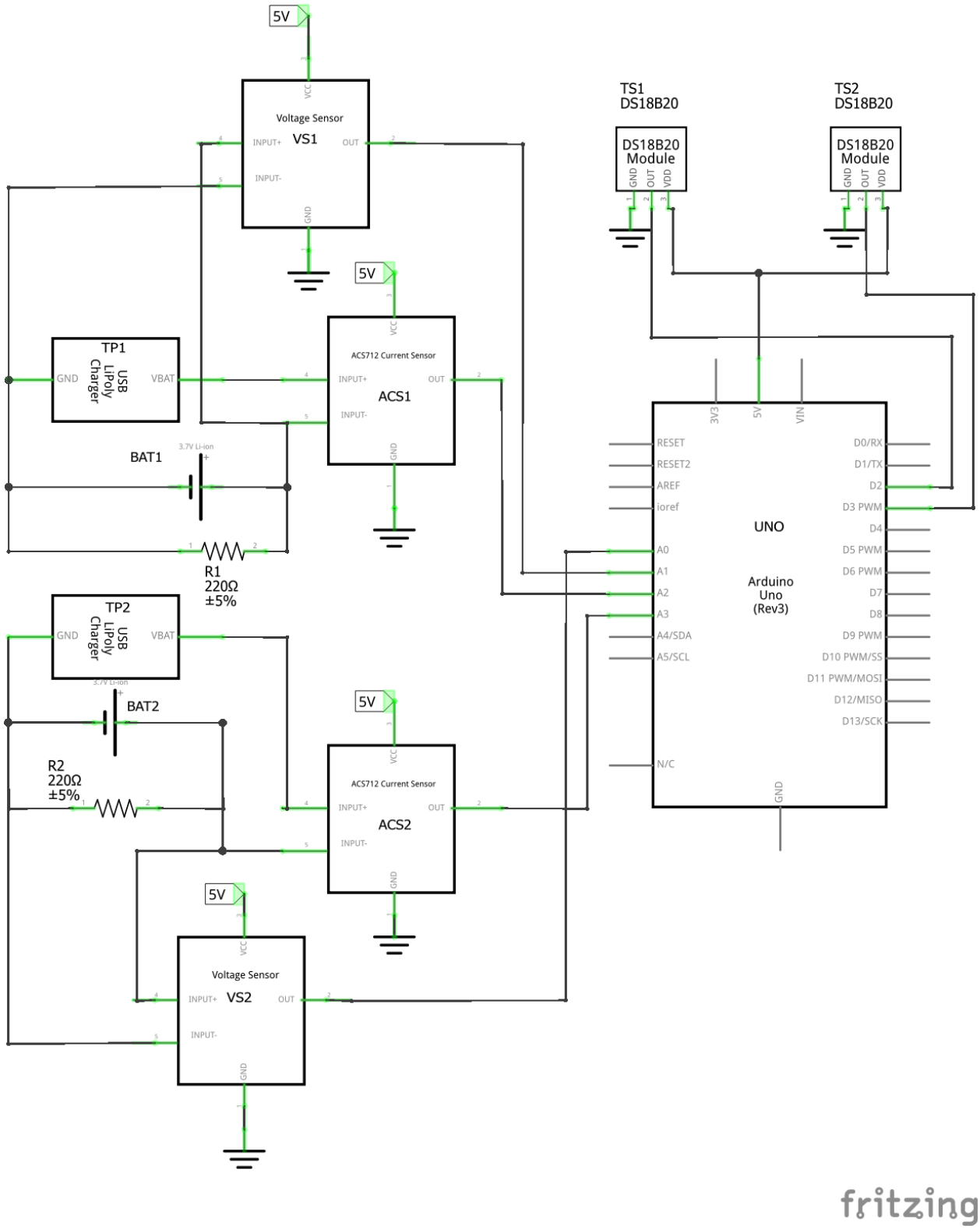


Figure 2: Circuit diagram of BMS prototype

The circuit includes two TP4056 charger modules, named TP1&2 which regulate charging current to the batteries BAT1&2. The ACS712 current sensors ACS1&2 are placed in series with the charger and the batteries' positive terminal,

allowing it to detect both charging and discharging currents from the battery. There are two DS18B20 temperature sensors TS1&2 which do not need any connection to the battery other than physically contacting the battery. There are two voltage sensor modules VS1&2 connected to the batteries in parallel for real-time voltage reading. All sensor outputs are connected to the Arduino for real-time processing.

3.4. Software and Algorithms

The entirety of the software component is based around the Arduino platform. It handles real-time data acquisition from the sensors, and deploys basic anomaly detection logic based on the sensor readings. The data is used to determine the battery's operational state and can be used to activate corresponding safety measures if necessary.

3.4.1. Voltage, Current and Temperature Monitoring

- Voltage monitoring:

Battery voltages are monitored via the voltage sensor module connected in parallel to the batteries. The output of the sensor modules are connected to the A0 and A1 analog pins of the Arduino. The cells must operate within 3.0V and 4.2V separately. The Arduino reads the analog voltage using its in-built 10-bit ADC, and the actual voltage from the reading must be calculated with the 5:1 voltage reduction ratio in mind.

```cpp
float rawV1 = analogRead(A0);
float rawV2 = analogRead(A1);
float voltage1 = (rawV1 * 5.0 / 1023.0) * 5.0;
float voltage2 = (rawV2 * 5.0 / 1023.0) * 5.0;
```

Code 1: Arduino IDE code snippet for voltage sensor input

- Current Monitoring:

Currents are read by the ACS712 modules and are sent to the Arduino on the analog pins A2 and A3. The output from the current sensor is also converted with the Arduino's 10-bit ADC corresponding to a voltage from 0-5V, which would give us around 512 during 0A current cases at 2.5V. This raw reading must be first converted to its corresponding voltage value by the reference voltage of 5V and divided by the 10-bit ADC resolution of 1023.0. Afterwards, it is then needed to subtract the baseline

output voltage of 2.5V at no current flow condition to determine the deviation from the zero-current point. The result is then divided by the sensitivity of the current sensor which is 100mV/A to calculate the actual current in amperes.

```cpp
float rawC1 = analogRead(A2);
float rawC2 = analogRead(A3);
float voltageC1 = rawC1 * 5.0 / 1023.0;
float voltageC2 = rawC2 * 5.0 / 1023.0;
float current1 = (voltage - 2.5) / 0.100;
float current2 = (voltage - 2.5) / 0.100;
```

Code 2: Arduino IDE code for snippet for current sensor input

- Temperature monitoring

The temperature is sensed through the DS18B20 sensors, connected to digital pins D2 and D3 and the readings are taken at regular intervals. The OneWire and DallasTemperature libraries are used to simplify the communication and conversion.

```cpp
OneWire oneWire1(2);
OneWire oneWire2(3);
DallasTemperature sensor1(&oneWire1);
DallasTemperature sensor2(&oneWire2);

sensor1.requestTemperatures();
float temp1 = sensor1.getTempCByIndex(0);

sensor2.requestTemperatures();
float temp2 = sensor2.getTempCByIndex(0);
```

Code 3: Arduino IDE code for temperature sensor input

3.4.2. SOC Estimation Algorithm

Accurate SOC estimation is crucial for battery longevity, safe operation, and effective energy management. However, direct measurement of SOC is not possible; it must be estimated using indirect parameters such as voltage, current, and historical charge/discharge data.

In this prototype, a basic voltage-based SOC estimation algorithm is implemented. Since the system uses individual 1S Li-ion cells, each with its own

voltage sensor, SOC is estimated by mapping the measured voltage to a predefined lookup table based on typical discharge curves of Li-ion batteries. This method is simple and computationally efficient, though less accurate under dynamic load conditions.

The following voltage-to-SOC mapping may be used:

Voltage (V)	**Estimated SOC (%)**
4.20	100%
4.10	90%
4.00	80%
3.90	70%
3.80	60%
3.70	50%
3.60	40%
3.50	30%
3.40	20%
3.30	10%
≤ 3.20	0%

Table 2: Voltage to SOC lookup table

The lookup table approach used is based on the typical discharge curve of Li-ion cells, which exhibits a rather linear voltage profile for most of its usable capacity, but dropping significantly near the end of its discharge. Although this method does not account for load current, temperature or aging effects, it does provide a usable estimate for basic monitoring. The code part of the SOC estimation for the first battery would be:

```
cpp
```

```
float soc = 0;

if (voltage1 >= 4.20) soc1 = 100;
else if (voltage1 >= 4.10) soc1 = 90;
else if (voltage1 >= 4.00) soc1 = 80;
else if (voltage1 >= 3.90) soc1 = 70;
else if (voltage1 >= 3.80) soc1 = 60;
else if (voltage1 >= 3.70) soc1 = 50;
else if (voltage1 >= 3.60) soc1 = 40;
else if (voltage1 >= 3.50) soc1 = 30;
else if (voltage1 >= 3.40) soc1 = 20;
else if (voltage1 >= 3.30) soc1 = 10;
else if (voltage1 >= 3.20) soc1 = 0;
```

Code 4: Arduino IDE code snippet for SOC1 estimation

3.4.3. Fault Detection

Fault detection is a key function of any Battery Management System, enabling the system to respond to abnormal conditions before they lead to battery degradation, safety hazards, or failure. In this prototype, a threshold-based logic system is implemented in software to detect basic fault conditions in real-time using sensor data. Below is a table with the fault conditions, its triggering thresholds and the rationale behind it being regarded as a fault condition:

Fault Condition	Trigger Threshold	Rationale
Overvoltage	Voltage > 4.25V	Prevents overcharging; voltages above 4.20 V risk swelling, fire, or thermal runaway
Undervoltage	Voltage < 3.20V	Protects battery from deep discharge, which reduces capacity and shortens lifespan
Overcurrent	Current > ±4.0 A	High current causes excessive heating and may lead to internal damage or lithium plating
Overtemperature	Temperature > 60 °C	Indicates possible thermal runaway, poor heat dissipation, or high internal resistance

Table 3: Fault conditions and their thresholds

And its Arduino code snippet to implement on the first battery:

```
cpp
```

```
if (voltage1 > 4.25) {
  Serial.println("Fault: Overvoltage detected at battery
1!");
}

if (voltage1 < 3.20) {
  Serial.println("Fault: Undervoltage detected at battery
1!");
}

if (abs(current1) > 4.0) {
  Serial.println("Fault: Overcurrent detected at battery
1!");
}

if (temperature1 > 60) {
  Serial.println("Fault: Overtemperature detected at battery
1!");
}
```

Code 5: Arduino IDE code snippet for fault detection at battery 1

3.5. Simulations and Modeling

To support the design of the Battery Management System, a thermal simulation was conducted using MATLAB to evaluate whether the prototype could operate safely under expected current loads. Rather than simulating the full electrical system, this approach focused on verifying the thermal viability of the design based on realistic discharge and charging scenarios..

3.5.1. MATLAB Simulation

MATLAB is a powerful software which may be used as a computing or plotting tool capable of outputting simulation scenarios through graphical means. In this case, it is used as a thermal viability graphing test.

- Thermal Modeling

A thermal model was simulated using actual battery physical properties, including mass, internal resistance, and specific heat capacity. The model calculates the cumulative heat generated by Joule heating $P = I^2 \times R$ and converts this into temperature rise over time. Simulated current profiles of charging, discharging, and idle were applied to the model to estimate temperature changes under realistic conditions.

This helped verify that the selected high current of 3A would not produce unsafe temperature levels within the typical operation window.

The parameters for this simulation were taken from actual data available, such as specific heat of a li-ion battery (ranges from 800-1100 J/kg*K), the internal resistance (0.05 Ohms), mass (0.045kg), and assuming that the environmental condition is in a room (Surrounding temperature being 25°C). With these parameters, it is possible to find out the temperature rise by plugging them in an energy equation in the worst possible outcome (which is 100% of the energy turning to heat Q = Power × Time, then calculated to find the temperature difference by dT = Q / (Mass × Specific heat)) and then plotting the values on a Current-Time-Temperature graph, as shown below:

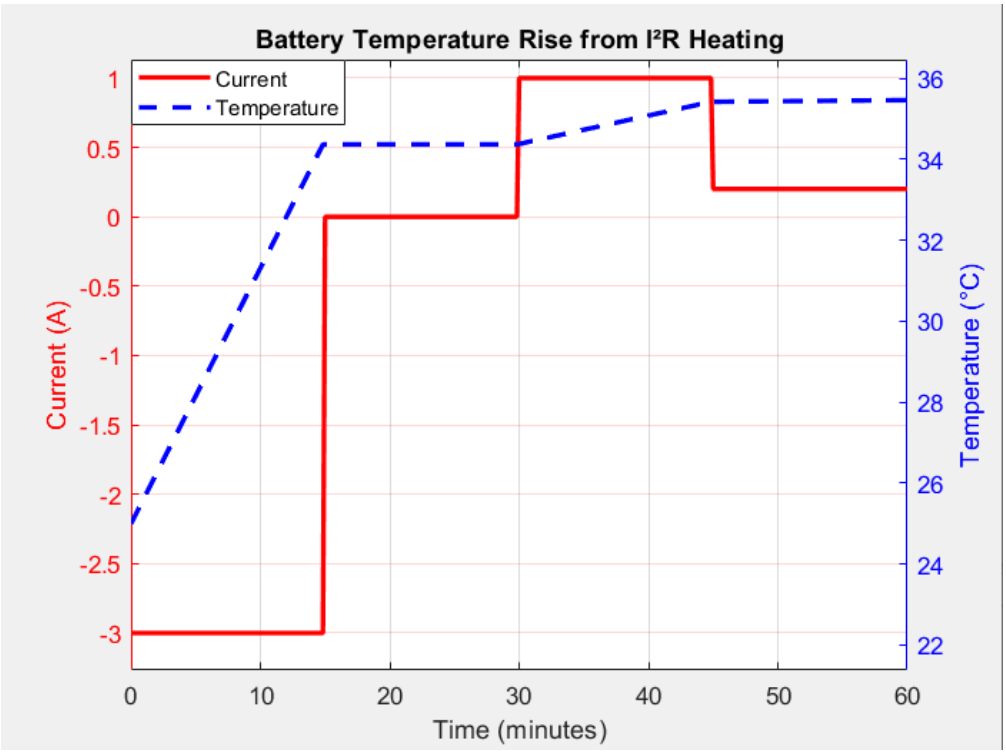


Figure 3: MATLAB simulation plot showing Current-Time-Temperature correlation

This graph shows that the battery would not go aboard safe operating temperatures under conditions of 3A discharging for 15 minutes, keeping idle for 15 minutes, 15 minutes of active 1A charging and 15 minutes of 0.2A floating charging. This is assuming that the battery won't dissipate any heat to the environment as well, and after the whole hour it ends up at a respectable, but safe ~36°C. The simulation can help the

inference that the prototype can safely handle sustained charge and discharge at these currents safely without triggering thermal faults or requiring any active cooling during use of the battery.

3.6. Summary

This chapter described the design and implementation outline of the BMS prototype, covering hardware integration, sensor-based monitoring, and fault detection logic. A MATLAB thermal simulation was also conducted to confirm safe temperature levels during typical operation. These methods establish a solid foundation for evaluating system performance in the next chapter.

4. Implementation and prototyping

4.1. Overview

This chapter details the physical implementation of the Battery Management System prototype, including hardware assembly, sensor integration, and embedded software deployment. The goal was to verify that the design could function reliably under real operating conditions using readily available components.

4.2. Hardware setup

4.2.1. Component list

The prototype was built using an Arduino Uno as the central controller, with the following components connected:

- Voltage sensors: for measuring cell voltage scaled to the Arduino’s ADC range
- ACS712 (20A): for current sensing
- DS18B20 sensors: for temperature monitoring
- TP4056 module: for charging control
- Resistive load: to simulate battery discharge
- LED: to visualize operation

All components were mounted on a breadboard and connected and soldered using jumper wires. Care was taken to ensure reliable power and signal connections.

4.2.2. System Schematic

The prototype is wired in a way where two single cell batteries are connected to the circuit as two separate parts. They are connected this way due to the complexity of the problem that is switching modes of the battery. If the two batteries are in series, though the rated voltage would double and open doors to operate higher voltage demanding loads, it is not possible to charge them as the TP4056 charger is for single battery cells only. On the other hand, if the two batteries are connected in parallel, it would not be possible to read their voltages separately, furthermore it is needed to balance both cells before connecting them in parallel. Due to these major oversights, it is rather difficult to connect the two batteries together- leading to a separate cell architecture.

Therefore, here is the simplified schematic of one battery connections:

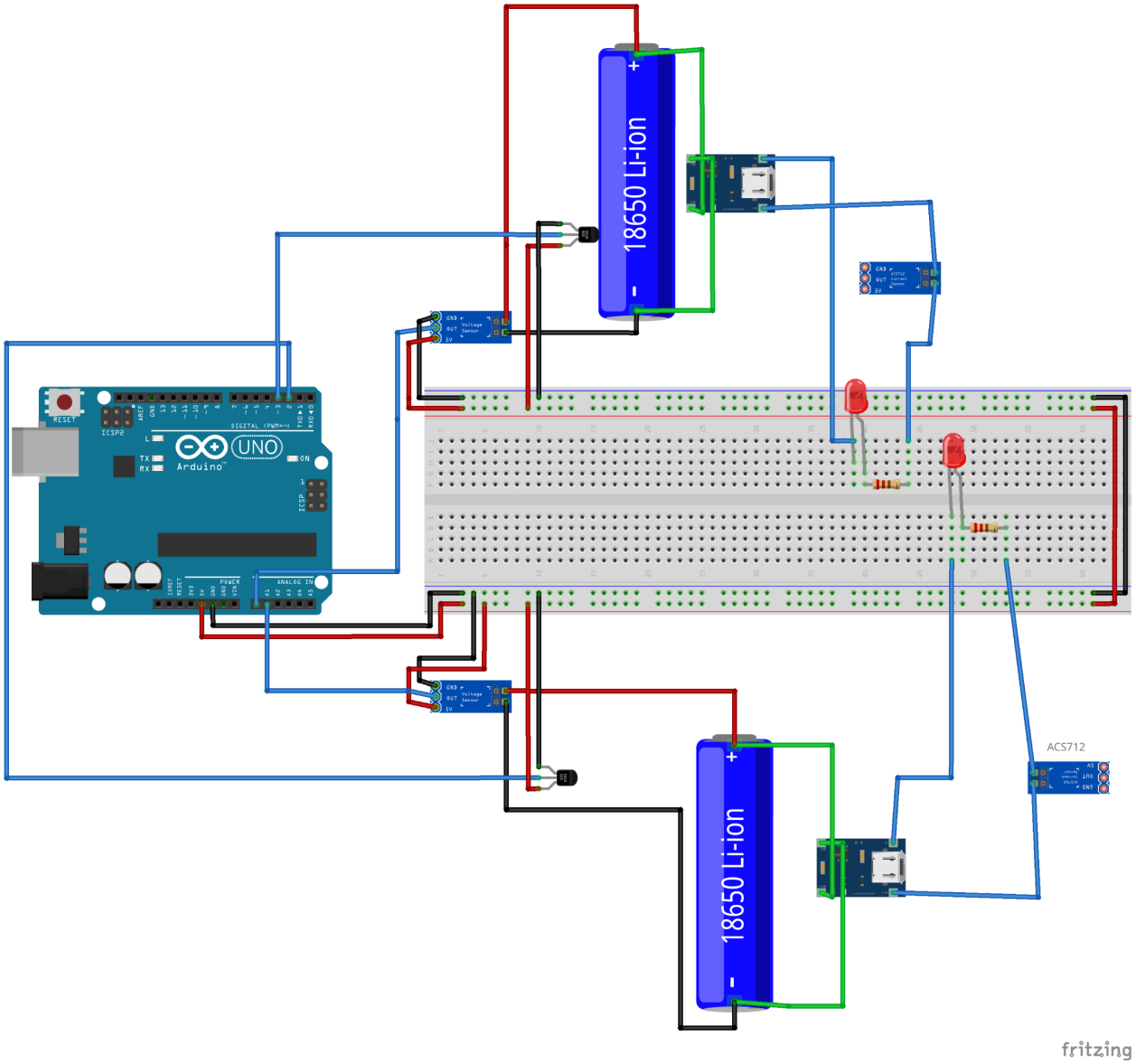


Figure 4: Overview of the prototype schematics

4.3. Software Implementation

The software for the Battery Management System prototype was developed using the Arduino programming environment. Its core purpose is to monitor real-time sensor data, process readings, and execute fault detection logic for safe battery operation. The code is deployed on an Arduino Uno, which interacts with voltage, current, and temperature sensors, and outputs fault status via the Serial Monitor or connected indicators.

4.3.1. Code architecture

The software for the Battery Management System was developed using the Arduino platform and is organized into four main sections: library inclusion, setup, main loop, and modular sensor processing and fault logic. The architecture emphasizes readability, modularity, and real-time response.

Part 1. Library Inclusion and Global Definitions

The program begins by including external libraries required for temperature sensing (`OneWire` and `DallasTemperature`) and defining global constants and pin assignments. This includes calibration values for voltage dividers, ACS712 sensors, and threshold values for fault detection.

cpp

```
// --- 1. Library Inclusion and Global Definitions ---
#include <OneWire.h>
#include <DallasTemperature.h>

// --- Pin Definitions ---
const int VOLTAGE_PIN_1 = A0;    // Voltage sensor 1
const int VOLTAGE_PIN_2 = A1;    // Voltage sensor 2
const int CURRENT_PIN_1 = A2;    // Current sensor 1
const int CURRENT_PIN_2 = A3;    // Current sensor 2
const int ONE_WIRE_BUS_1 = 2;    // Temperature sensor 1
const int ONE_WIRE_BUS_2 = 3;    // Temperature sensor 2

// --- OneWire and Temperature Sensors ---
OneWire oneWire1(ONE_WIRE_BUS_1);
OneWire oneWire2(ONE_WIRE_BUS_2);
DallasTemperature sensors1(&oneWire1);
DallasTemperature sensors2(&oneWire2);

// --- Calibration Constants ---
const float VREF = 5.0;              // Reference voltage for
Arduino
const int ADC_RESOLUTION = 1023;     // 10-bit ADC
const float VOLTAGE_DIVIDER_RATIO = 2.0;  // Adjust based on
resistor values

const float ACS712_SENSITIVITY = 0.1; // 100mV/A for 20A
version
const float ACS_ZERO_OFFSET = 2.5;    // 2.5V at 0A output
from sensor

// --- Fault Thresholds ---
```

```
const float OVERVOLTAGE_THRESHOLD = 4.25;
const float UNDERVOLTAGE_THRESHOLD = 3.20;
const float OVERCURRENT_THRESHOLD = 4.0;     // Amps
const float OVERTEMP_THRESHOLD = 60.0;       // °C
```

Code 6: Arduino IDE code snippet for main code Part 1

Part 2. Setup Function

The `setup()` function initializes communication with the Serial Monitor and configures the DS18B20 sensors. All pin modes are defined, and any connected indicator outputs (e.g., built-in LED) are prepared.

```
cpp
```

```
void setup() {
  Serial.begin(9600);
  sensors1.begin();
  sensors2.begin();
}
```

Code 7: Arduino IDE code snippet for main code Part 2

3. Loop Function (Main Logic Cycle)

The `loop()` function runs continuously and performs four core operations:

- Sensor data acquisition: Reads voltages, currents, and temperatures from two independent battery lines.
- Unit conversion: Converts analog readings to volts, amps, and Celsius using calibration constants.
- Fault detection: Compares each sensor value against safety thresholds to flag overvoltage, undervoltage, overcurrent, or overtemperature conditions.
- Data output: Prints all sensor values and any triggered fault messages to the Serial Monitor.

Each logical block is grouped for easy editing and future expansion. For example, voltage and current processing are handled separately for each battery channel, allowing easy scaling to more cells or more complex balancing logic.

```
cpp
```

```
void loop() {
  // --- Voltage Reading ---
  float voltage1 = analogRead(VOLTAGE_PIN_1) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;
  float voltage2 = analogRead(VOLTAGE_PIN_2) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;

  // --- Current Reading ---
  float sensorVoltage1 = analogRead(CURRENT_PIN_1) * VREF /
ADC_RESOLUTION;
  float sensorVoltage2 = analogRead(CURRENT_PIN_2) * VREF /
ADC_RESOLUTION;
  float current1 = (sensorVoltage1 - ACS_ZERO_OFFSET) /
ACS712_SENSITIVITY;
  float current2 = (sensorVoltage2 - ACS_ZERO_OFFSET) /
ACS712_SENSITIVITY;

  // --- Temperature Reading ---
  sensors1.requestTemperatures();
  sensors2.requestTemperatures();
  float temp1 = sensors1.getTempCByIndex(0);
  float temp2 = sensors2.getTempCByIndex(0);

  // --- Fault Detection ---
  bool fault = false;

  if (voltage1 > OVERVOLTAGE_THRESHOLD || voltage2 >
OVERVOLTAGE_THRESHOLD) {
    Serial.println("Overvoltage detected"); fault = true;
  }
  if (voltage1 < UNDERVOLTAGE_THRESHOLD || voltage2 <
UNDERVOLTAGE_THRESHOLD) {
    Serial.println("Undervoltage detected"); fault = true;
  }
  if (abs(current1) > OVERCURRENT_THRESHOLD || abs(current2)
> OVERCURRENT_THRESHOLD) {
    Serial.println("Overcurrent detected"); fault = true;
  }
  if (temp1 > OVERTEMP_THRESHOLD || temp2 >
OVERTEMP_THRESHOLD) {
    Serial.println("Overtemperature detected"); fault = true;
  }

  // --- Output Values ---
  Serial.print("V1: "); Serial.print(voltage1);
Serial.print(" V, ");
  Serial.print("V2: "); Serial.print(voltage2);
Serial.print(" V | ");
  Serial.print("I1: "); Serial.print(current1);
Serial.print(" A, ");
  Serial.print("I2: "); Serial.print(current2);
Serial.print(" A | ");
  Serial.print("T1: "); Serial.print(temp1); Serial.print("
```

```
°C, ");
  Serial.print("T2: "); Serial.println(temp2); Serial.print("
°C\n");

  delay(1000);
}
```

Code 8: Arduino IDE code snippet for main code Part 3

4.3.2. Voltage Sensing

Voltage sensing is implemented using analog inputs on pins `A0` and `A1`, connected to two external voltage sensor modules with a 5:1 divider ratio. These modules reduce the input voltage to a safe level that can be read by the Arduino's 0–5 V ADC. The sensing is done using the `analogRead()` function, followed by a scaling calculation to convert the result back to actual voltage. The relevant code is:

```
cpp
```

```
float voltage1 = analogRead(VOLTAGE_PIN_1) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;
float voltage2 = analogRead(VOLTAGE_PIN_2) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;
```

Code 9: Voltage Sensor code for actual voltage reading

Where:

- `VOLTAGE_PIN_1` and `VOLTAGE_PIN_2` are connected to A0 and A1
- `VREF` is the reference voltage (5.0 V)
- `ADC_RESOLUTION` is 1023 (for 10-bit ADC)
- `VOLTAGE_DIVIDER_RATIO` is 5.0 (due to the sensor design)

This calculation converts the ADC reading into an actual voltage. The measured voltage is then compared to threshold values: 3.20 V for undervoltage and 4.25 V for overvoltage. If either threshold is breached, a fault flag is triggered in the software for safety monitoring.

4.3.3. Current Sensing

Current sensing is achieved using two ACS712 20A Hall-effect sensors connected to analog pins A2 and A3. These sensors output a voltage centered around 2.5 V when no current is flowing. As current increases in either direction (positive or negative), the output voltage shifts proportionally.

The following lines in the code handle the reading and conversion of current:

```cpp
float sensorVoltage1 = analogRead(CURRENT_PIN_1) * VREF / ADC_RESOLUTION;
float sensorVoltage2 = analogRead(CURRENT_PIN_2) * VREF / ADC_RESOLUTION;
float current1 = (sensorVoltage1 - ACS_ZERO_OFFSET) / ACS712_SENSITIVITY;
float current2 = (sensorVoltage2 - ACS_ZERO_OFFSET) / ACS712_SENSITIVITY;
```

Code 10: Current Sensor code for actual current reading

Where:

- `CURRENT_PIN_1` and `CURRENT_PIN_2` correspond to A2 and A3
- `VREF` is 5.0 V
- `ADC_RESOLUTION` is 1023
- `ACS_ZERO_OFFSET` is 2.5 V (no-current baseline)
- `ACS712_SENSITIVITY` is 0.1 V/A (for 20A version)

The analog voltage is first scaled to the actual sensor output voltage. Then, the zero-current offset is subtracted, and the result is divided by the sensitivity to obtain the current in amperes.

These current values are continuously monitored and compared to a ±4.0 A threshold. If exceeded, the software flags an overcurrent fault for protection purposes.

4.3.4. Temperature Sensing

Temperature sensing is implemented using two DS18B20 digital temperature sensors connected to digital pins 2 and 3. These sensors use the OneWire protocol

and provide temperature readings with a typical accuracy of ±0.5 °C. The Arduino communicates with each sensor using the DallasTemperature library.

The following lines in the code handle temperature requests and retrieval:

```
cpp
```

```
sensors1.requestTemperatures();
sensors2.requestTemperatures();
float temp1 = sensors1.getTempCByIndex(0);
float temp2 = sensors2.getTempCByIndex(0);
```

Code 11: Temperature Sensor implementation code

Here:

- `sensors1` and `sensors2` refer to two separate OneWire buses, allowing each sensor to be read independently
- `requestTemperatures()` initiates a measurement
- `getTempCByIndex(0)` returns the temperature in Celsius

These readings are taken once per loop cycle and compared to a predefined threshold of 60 °C. If either temperature exceeds this limit, an overtemperature fault is triggered in the system.

This setup provides reliable thermal monitoring of the battery cells to prevent overheating during charging or discharging.

4.3.5. Fault Detection Logic

The fault detection logic continuously checks for four key battery safety violations: overvoltage, undervoltage, overcurrent, and overtemperature. These checks are performed on both battery lines once per loop cycle using simple conditional statements.

The relevant code block is:

```
cpp
```

```
bool fault = false;

if (voltage1 > OVERVOLTAGE_THRESHOLD || voltage2 >
OVERVOLTAGE_THRESHOLD) {
  Serial.println("Overvoltage detected"); fault = true;
}
if (voltage1 < UNDERVOLTAGE_THRESHOLD || voltage2 <
UNDERVOLTAGE_THRESHOLD) {
  Serial.println("Undervoltage detected"); fault = true;
}
if (abs(current1) > OVERCURRENT_THRESHOLD || abs(current2) >
OVERCURRENT_THRESHOLD) {
  Serial.println("Overcurrent detected"); fault = true;
}
if (temp1 > OVERTEMP_THRESHOLD || temp2 > OVERTEMP_THRESHOLD)
{
  Serial.println("Overtemperature detected"); fault = true;
}
```

Code 12: Simple fault detection logic for voltage, current, and temperature

4.4. Prototype Assembly and Integration

The BMS was prototyped on a solderless breadboard, with an Arduino Uno as the centerpiece. All sensors were connected by jumper wires, with soldering on the battery holders' leading wires and the TP4056 B+ and B- terminals. Jumper wires allowed for easy debugging and flexibility in arrangement.

First, the TP4056 modules were soldered on the battery holders' leading wires. The B+ to the positive terminals and the B- to the negative terminals. Then, the sensors are connected to their respective pins on the Arduino board, the voltage sensors connected to A0 and A1, the current sensors connected to A2 and A3, the temperature sensors connected to the digital pins 2 and 3. Then the breadboard is connected to the GND pin and the 5V pin on the long lines, allowing them to supply the sensors. Then two discharge loads consisting of a 200 Ohm resistor and an LED to indicate the current flowing through.

The current sensors were positioned between the TP4056's output and the load to capture discharge current accurately. Voltage divider outputs were routed to the Arduino's analog pins, while the DS18B20 sensors were mounted contacting the batteries to capture surface temperature changes. All code was uploaded to the Arduino, and the Arduino IDE is outputting the values continuously.

4.5. Safety Considerations

Due to the inherent risks associated with Li-ion batteries, safety was always a crucial consideration during the implementation of the prototype. The charging safety was relegated to the TP4056 charging module, which provided built-in overvoltage protection, charge current limitation, and automatic termination at full charge. Breadboard wiring and component connections were double-checked to prevent shorts, reverse polarity, or unstable contacts. All tests were conducted under supervision with access to multimeters and manual shutoff if needed. At moments such as soldering it was also important to keep clear of any safety hazards that may occur.

4.6. Summary

This chapter went over the physical implementation and software development of the Battery Management System prototype. The system was assembled using readily available components including an Arduino Uno, TP4056 charger, voltage and current sensors, and digital temperature sensors. Each sensor was integrated into the circuit to enable real-time monitoring of battery conditions.

Software logic was developed to acquire sensor data, process it into readable units, and detect fault conditions based on predefined thresholds. Voltage, current, and temperature readings were continuously logged via the Serial Monitor, and appropriate flags were triggered for any abnormal values.

The prototype successfully demonstrated key BMS functions such as overvoltage, undervoltage, overcurrent, and overtemperature detection. The modular structure and clear sensor integration lay the foundation for further improvements such as automated cutoff, data logging, or wireless monitoring.

5. Testing and Results

5.1. Overview

This chapter outlines the testing procedures and observed results from the Battery Management System prototype. Due to safety and time constraints, full-scale discharge and charge cycle tests were not conducted. However, the individual sensing modules and fault detection logic were tested under controlled and simulated conditions to evaluate system functionality.

5.2. Sensor Verification and Observations

Voltage Sensing**:**

The other battery was used in series to apply known voltages to the voltage sensor inputs. The system successfully read and displayed the voltages via the Serial Monitor. The measured error was within ±0.1 V, confirming that the 5:1 voltage divider and ADC conversion logic were functioning as expected.

Current Sensing:

Fixed resistive loads were applied to simulate known current draw conditions. The ACS712 sensors responded accordingly, with minor deviations near zero current due to the sensor's analog offset. Readings were consistent with external multimeter measurements, within ±0.2 A.

Temperature Sensing:

Room temperature readings were confirmed by comparing the DS18B20 output to a digital thermometer. Heating the sensor manually (e.g., by hand) resulted in visible temperature changes on the Serial Monitor, confirming responsiveness and real-time tracking.

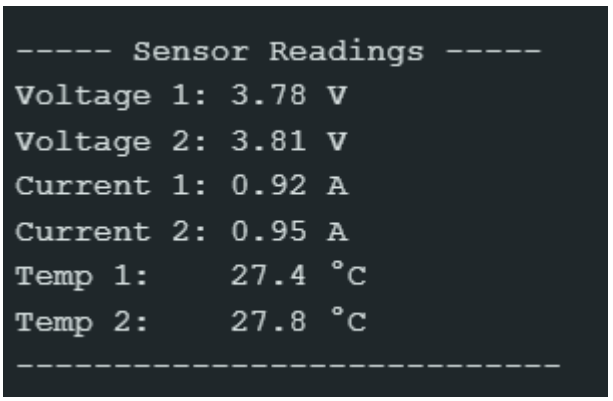


Figure 5: Sensor Readings at idle charging

5.3. Fault Detection Behaviour

To verify the system's fault detection capabilities, sensor values were hardcoded in the Arduino firmware to simulate abnormal conditions.

Fault Type	**Simulated Condition**	**Expected Outcome**	**Observed**
Overvoltage	Sensor value > 4.25 V	Serial print	Triggered

Undervoltage	Sensor value < 3.20 V	Serial print	Triggered
Overcurrent	Simulated load >±4.0 A	Serial print	Triggered
Overtemperature	DS18B20 value >60 °C	Serial print	Triggered

These test cases confirmed that the logic correctly identifies unsafe operating conditions and provides real-time alerts via the Serial Monitor

5.4. Summary

All sensors were functioning correctly and with a predictable margin of error, the Arduino IDE's serial monitor provided continuous, acceptable values of output for voltage, current, and temperature. Fault conditions were successfully triggered using hardcoded fake data. No physical overheating or unstable behaviour was observed during the entire sensor testing or mock fault detection testings.

6. Conclusion

From the beginning of this thesis to its completion, the objective remained clear: to develop a simple, low-cost prototype of a Battery Management System (BMS) that could reliably perform the core functions expected of such a system. The initial step was to define these core functions, which included monitoring voltage, current, and temperature, and detecting anomalies that could indicate unsafe operating conditions. Upon identifying these functional requirements, the next task was to select affordable and widely available components capable of fulfilling each role. This led to the use of voltage sensor modules, ACS712 current sensors, and DS18B20 temperature sensors — components commonly used in hobbyist and educational embedded systems.

Following component selection, the focus shifted to designing a practical circuit that could serve as a working prototype of a BMS. This involved integrating the chosen sensors into a configuration that would allow the Arduino Uno to acquire, interpret, and respond to real-time data. Physical prototyping followed, with all components assembled on a breadboard and interfaced with the Arduino. Each part of the system was tested for functionality, with particular attention given to ensuring correct pin assignments, voltage safety, and logical sensor behavior. Through this iterative and hands-on process, a functional prototype was built that represents a basic yet operational BMS framework.

Although full-scale physical testing of the system under active charge and discharge conditions was not performed, the prototype's behavior was evaluated through controlled tests and simulated scenarios. A MATLAB-based thermal model was used to estimate temperature rise under realistic current loads, confirming that the system would remain within safe thermal limits during operation. Additionally, the Arduino firmware was tested with simulated sensor inputs to verify the accuracy of fault detection logic and real-time data reporting via the Serial Monitor. These tests demonstrated that the core BMS logic functioned correctly under expected conditions.

However, several limitations remain. The system currently lacks automatic response mechanisms such as load cutoff or charge disconnect in the event of a fault. Moreover, due to an oversight, it was not possible to develop a design of a circuit which would have allowed for cell-balancing to take place as the two single cell batteries do not allow for cell-balancing, which was also a main function of the BMS. The two battery cells connected in parallel would have made separate voltage readings impossible, and initial cell balancing would have been required for the cells to be connected in parallel in the first place. Having the cells in series would allow for separate cell voltage readings, but the TP4056 disallows multiple cell charging. Given plenty of resources, and research it seems to be possible to use a DPDT switch to allow for series-to-parallel switching battery connections.

It may be noted that there are numerous potential additions and improvements that could enhance the current BMS design. These include integrating automatic cutoff mechanisms using MOSFETs, implementing data logging for long-term performance tracking, and adding wireless communication capabilities for remote monitoring. Further enhancements could also involve expanding the system to support multiple cells with active or passive balancing, refining calibration routines for increased measurement accuracy, and optimizing the code structure for lower power consumption and better real-time performance. Each of these improvements would contribute to making the BMS more robust, scalable, and suitable for practical deployment in real-world battery-powered systems.

In conclusion, this project successfully demonstrates that a functional and expandable Battery Management System can be built using low-cost components and embedded logic, laying a strong foundation for more advanced designs in future iterations.

References

[1] Energy Storage News. *Annual lithium-ion demand surpasses 1TWh for the first time* [Internet]. 2024 Dec 20 [cited 2025 Apr 14]. Available from: https://www.ess-news.com/2024/12/20/annual-lithium-ion-demand-surpasses-1-twh-for-the-first-time/

[2] Stenzel P, Sterner M, Vandrei N, et al. *Database development and evaluation for techno-economic assessments of electrochemical energy storage systems* [Internet]. 2014 [cited 2025 Apr 14]. Available from: https://www.researchgate.net/publication/269297116_Database_development_and_evaluation_for_techno-economic_assessments_of_electrochemical_energy_storage_systems

[3] Precedence Research. *Rechargeable Batteries Market Size, Share, Growth Report 2024–2033* [Internet]. Precedence Research; 2024 [cited 2025 Apr 14]. Available from: https://www.precedenceresearch.com/rechargeable-batteries-market

[4] Battery University. *BU-409: Charging Lithium-ion* [Internet]. 2022 [cited 2025 Apr 30]. Available from: https://batteryuniversity.com/article/bu-409-charging-lithium-ion

[5] Redway Battery. *How do charging times differ between 12V LiFePO4 and lead-acid batteries?* [Internet]. 2024 [cited 2025 Apr 30]. Available from: https://www.redway-tech.com/how-do-charging-times-differ-between-12v-lifepo4-and-lead-acid-batteries/

[6] R. Suganya. *Understanding lithium-ion battery management systems in electric vehicles: Environmental and health impacts, comparative study, and future trends: A review.* [Internet]. [ResearchGate].

[7] Yilmaz M, Krein PT. *Review of charging power levels and infrastructure for plug-in electric and hybrid vehicles*. *Electronics* [Internet]. 2023;12(19):4095. Available from: https://www.mdpi.com/2079-9292/12/19/4095

[8] Gao Y, Xu J, Zhang X, Yang Z. *Evaluation of charging methods for lithium-ion batteries*. *Energies* [Internet]. 2022;15(9):3212. Available from: https://www.mdpi.com/1996-1073/15/9/3212

[9] Battery Design. *Constant Current – Constant Voltage Charging* [Internet]. [cited 2025 Apr 30]. Available from: https://www.batterydesign.net/constant-current-constant-voltage-charging/

Appendix

Complete Arduino Code for Battery Management System Prototype

cpp

```
#include <OneWire.h>
#include <DallasTemperature.h>

// --- Pin Definitions ---
const int VOLTAGE_PIN_1 = A0;    // Voltage sensor 1
const int VOLTAGE_PIN_2 = A1;    // Voltage sensor 2
const int CURRENT_PIN_1 = A2;    // Current sensor 1
const int CURRENT_PIN_2 = A3;    // Current sensor 2
const int ONE_WIRE_BUS_1 = 2;    // Temperature sensor 1
const int ONE_WIRE_BUS_2 = 3;    // Temperature sensor 2

// --- OneWire and Temperature Sensors ---
OneWire oneWire1(ONE_WIRE_BUS_1);
OneWire oneWire2(ONE_WIRE_BUS_2);
DallasTemperature sensors1(&oneWire1);
DallasTemperature sensors2(&oneWire2);

// --- Calibration Constants ---
const float VREF = 5.0;              // Reference voltage for
Arduino
const int ADC_RESOLUTION = 1023;     // 10-bit ADC
const float VOLTAGE_DIVIDER_RATIO = 2.0;  // Adjust based on
resistor values

const float ACS712_SENSITIVITY = 0.1; // 100mV/A for 20A
version
const float ACS_ZERO_OFFSET = 2.5;    // 2.5V at 0A output
from sensor

// --- Fault Thresholds ---
const float OVERVOLTAGE_THRESHOLD = 4.25;
const float UNDERVOLTAGE_THRESHOLD = 3.20;
const float OVERCURRENT_THRESHOLD = 4.0;     // Amps
const float OVERTEMP_THRESHOLD = 60.0;       // °C

// --- 2. Setup Function ---
void setup() {
  Serial.begin(9600);
  sensors1.begin();
  sensors2.begin();
}

// --- 3. Loop Function (Main Logic Cycle) ---
void loop() {
  // --- Voltage Reading ---
  float voltage1 = analogRead(VOLTAGE_PIN_1) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;
```

```
  float voltage2 = analogRead(VOLTAGE_PIN_2) * VREF /
ADC_RESOLUTION * VOLTAGE_DIVIDER_RATIO;

  // --- Current Reading ---
  float sensorVoltage1 = analogRead(CURRENT_PIN_1) * VREF /
ADC_RESOLUTION;
  float sensorVoltage2 = analogRead(CURRENT_PIN_2) * VREF /
ADC_RESOLUTION;
  float current1 = (sensorVoltage1 - ACS_ZERO_OFFSET) /
ACS712_SENSITIVITY;
  float current2 = (sensorVoltage2 - ACS_ZERO_OFFSET) /
ACS712_SENSITIVITY;

  // --- Temperature Reading ---
  sensors1.requestTemperatures();
  sensors2.requestTemperatures();
  float temp1 = sensors1.getTempCByIndex(0);
  float temp2 = sensors2.getTempCByIndex(0);

  // --- Fault Detection ---
  bool fault = false;

  if (voltage1 > OVERVOLTAGE_THRESHOLD || voltage2 >
OVERVOLTAGE_THRESHOLD) {
    Serial.println("Overvoltage detected"); fault = true;
  }
  if (voltage1 < UNDERVOLTAGE_THRESHOLD || voltage2 <
UNDERVOLTAGE_THRESHOLD) {
    Serial.println("Undervoltage detected"); fault = true;
  }
  if (abs(current1) > OVERCURRENT_THRESHOLD || abs(current2)
> OVERCURRENT_THRESHOLD) {
    Serial.println("Overcurrent detected"); fault = true;
  }
  if (temp1 > OVERTEMP_THRESHOLD || temp2 >
OVERTEMP_THRESHOLD) {
    Serial.println("Overtemperature detected"); fault = true;
  }

  // --- Output Values ---
  Serial.print("V1: "); Serial.print(voltage1);
Serial.print(" V, ");
  Serial.print("V2: "); Serial.print(voltage2);
Serial.print(" V | ");
  Serial.print("I1: "); Serial.print(current1);
Serial.print(" A, ");
  Serial.print("I2: "); Serial.print(current2);
Serial.print(" A | ");
  Serial.print("T1: "); Serial.print(temp1); Serial.print("
°C, ");
  Serial.print("T2: "); Serial.println(temp2); Serial.print("
°C\n");
  delay(1000);
}
```