Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

# ELG4913 – ELECTRICAL ENGINEERING PROJECT

# FINAL REPORT

**Prepared for:**

Professor Emil M. Petriu

2023-04-01

**Prepared By:**

Nima Mehrjoonezhad - 300027431

Kaiyi Yuan - 8617972

Josiah Bigras - 300125987

Lidan Huang - 300092104

Kaicheng Zhang - 8748689

# Table of Contents

Project Charter

## Document Change Control

As this project unfolds, the project team gains more information and understanding about the project and its implementation. Consequently, the project charter may have to be adjusted as the scope of the project becomes more precise and the deliverables are better understood.

This section is used to document any changes and serves to control the development and distribution of revisions to the project charter. It should be used together with a change management process and a document management system. Document management procedures of the sponsoring organization should be applied to determine when versions and subversions must be created. This practice keeps an accurate history of the original document that was first approved.

## Executive Summary

This project is Initiated in support of uOttawa Electrical Engineering and Computer Science Department, in association with Professor Emil M. Petriu and teaching assistant Haseeb Ur Rehman. Our objective is to develop a solution to the electrical scooter battery inconvenience and inefficiency due to the climate characteristics of Canada; a group of 5 members will be assigned to this project.

Based on the analysis of project goals, project objectives, major milestones, key deliverables, primary risks, and estimated total costs, the project proposal has been approved by the Electrical Engineering and Computer Science Department.

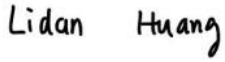
## Authorization

By signing below, I agree to the assigned roles, the description of the project, and the project deliverables and outcomes presented in the project charter.

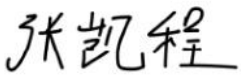
Client name: Eslin Ustun Karatop

Project Manager signature:

Key Stakeholders signature:







System Requirements Specification

## Project summary

As everything is going electric, e-scooter is getting popular around the campus, and many students choose to commute across the campus on an e-scooter. Still, it's not a choice during the winter due to the harsh climate characteristic here in Ottawa. Our group believes this problem can be solved with an integrated add-on system that manipulates the battery status, so the primary purpose of this project is to create a more efficient and convenient way for users to ride their e-scooter during the winter. The integrated system will allow users to have direct monitoring of the status of the battery and automatic control to maintain the battery status at optimal conditions. The project will be built and tested with a Nickel Metal Hydride battery pack which can be installed on the e-scooter as an uninterrupted power supply.

## Project Requirements

Our project requirements are divided into business, user, and design requirements. Based on these requirements, we designed our system for the battery pack. The system requirements show the battery pack and add-on system via the block diagram. It contains the project's hardware components and other software resources.

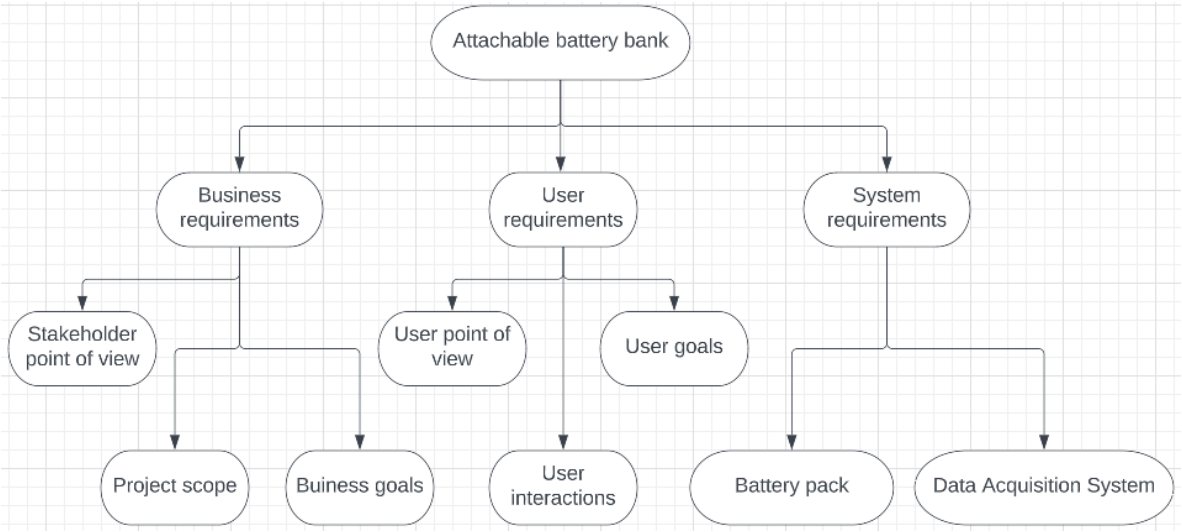



*Figure 1: Block Diagram for Major Project Requirements.*

## Functional Requirements

To achieve the above functions, we designed the hardware and software part of the battery pack. Our project includes but not limited to the following functions:

- Provide enough energy needed by the e-scooter (Provide 24V to e-scooter, 5V to Raspberry Pi).
- Rechargeable.
- Power output is stable, it will not be affected by the battery state of charge
- Measure voltage, temperature, speed of the scooter, energy consumption in the current route; control contactor, pre-charge.
- Protection against Over-charge, over-discharge, over-current, short circuits, and extreme temperatures.
- Minimize current to keep the wire diameter small and reduce resistivity.
- State-of-charge (SOC) estimation, power-limit computation, balance/equalize cells.
- Interface: Range estimation, communications, data recording, reporting.
- Thermal control in different environments.
- Locate and track e-scooters.

## Hardware Requirements

The hardware contains the components needed for the battery pack, such as batteries, DC-DC converter, battery charger discharger control module, charging controller protection module, and DC backup battery switching module. The block diagram also shows the model of the batteries and how they are arranged and assembled. Our project changed the original plan and decided to use Nickel Metal Hydride batteries instead of 18650 batteries. Because compared to lead acid batteries, NiMH batteries have higher power and energy density and a much longer life cycle. Also, it is rechargeable and completely safe. The power output is not affected by the battery's state of charge. Also, the university provides NiMH batteries for free, so we finally decided to replace our original plan.

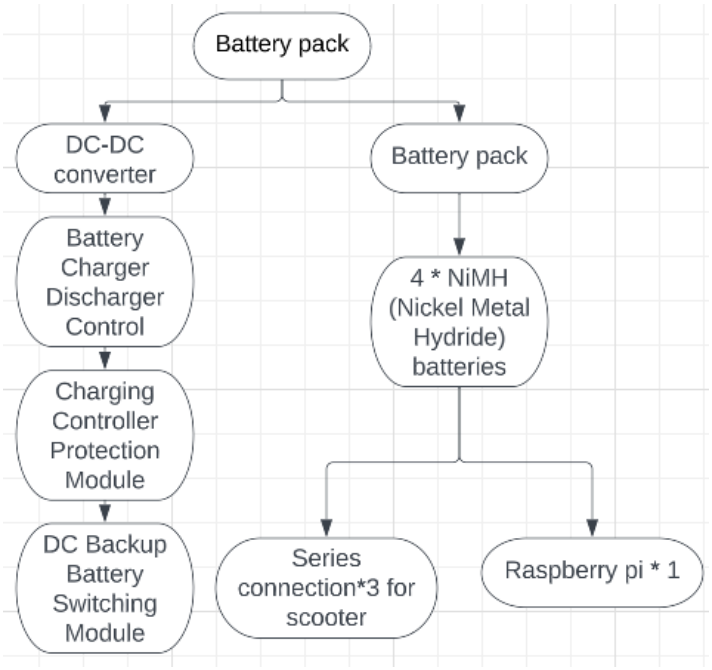



*Figure 2: Block Diagram for Hardware Requirements.*

Software Requirements

Its function is to collect data and control hardware. Also, it gives users insightful data about their battery consumption and acts as a feedback loop to ensure all the parameters are within the desired range (such as voltage, etc.). The temperature control system and GPS include three different sensors and cooperate with raspberry pi for data collection and automatic control.

Temperature Control System (Temperature Sensor):

The best working temperature for batteries is around 20℃. In a non-extreme environment, after working for dozens of minutes, the battery can rely on its heat to maintain a mild and comfortable "body temperature." Our battery pack has a temperature control system that can prevent the battery from overheating. The battery automatically disconnects when the battery temperature exceeds the set value or when the voltage is abnormal.

Voltage Control System (Voltage Sensor):

A voltage sensor is used to monitor the state of the batteries, as well as their discharge and charge rates. Using the voltage sensor, we can monitor when the batteries are active and how much voltage the circuit has. With these sensors, integrated with software design, we can demonstrate all the states of the battery to the user.

GPS System (GPS Sensor):

Positioning systems are integral to military applications and emergency responders locating people in need. GPS technology often comes into play in many areas we don't usually consider. For the GPS system, we are using the GPS sensor and the GSM module.

User Interface/Data Storage System (GSM module):

All data will be saved and uploaded to the cloud to help us follow up and upgrade the battery pack in the future, and some data, such as the remaining battery power, speed of the e-scooter, and battery consumption, will be displayed on the user interface to help users use it more conveniently.

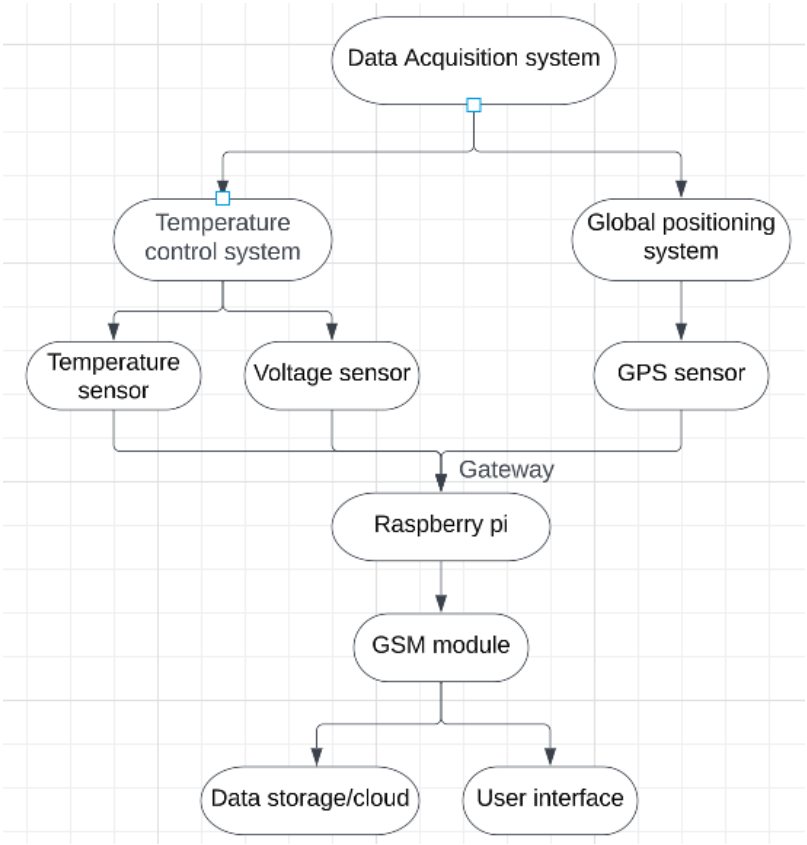



*Figure 3: Block Diagram for Software and Control System Requirements.*

## Non - Functional Requirements

**Performance and scalability:** The system returns results immediately. The performance is not significantly reduced with higher workloads.

**Portability and compatibility:** External battery pack, which can be easily fixed on the bottom plate of an e-scooter, is suitable for all types of e-scooters.

**Reliability/Maintainability:** The battery will have an average lifespan of 300-500 full charge cycles and usually takes 2-3 years for the average user.

**Localization:** All features of the battery are designed based on the Ottawa environment, It works fine in the extreme cold of Ottawa's winter.

**Usability:** It's simple to use; take it from home and connect it to the e-scooter charger with the plug in the wire of the battery pack. There are no additional operations required.

# Detailed Design

Designing the optimal battery system involves many design criterium to ensure a safe and user-friendly design. A smart battery pack is a type of battery that incorporates an intelligent management system to monitor its performance and provide accurate state-of-charge and state-of-health information. Designing this system involves several hardware and software components that will work together to optimize the battery's usability and performance. Some major considerations we took upon designing such a system are the battery selection, the battery management system, the charging circuit, the power supply, the user interface, and the mechanical design.

## Hardware

The hardware design of our system revolves around four major factors, the batteries selected, the battery management system, the charging circuit, and the power supply. For the first factor, due to the availability of Nickel-metal hydride batteries at the University of Ottawa, we've decided it would be an optimal replacement for our originally selected lithium-ion batteries. NiMH batteries provide an overall lower cost, are more environmentally friendly, are more robust, and they can be charged and discharged many more times compared to lithium-ion batteries due to their resilience against the battery memory effect (decreasing the battery capacity and they experience battery cycling).


*Figure 4: ONYX Nickel-Metal Hydride Battery Used in Project.*

We've decided the optimal way to configure the battery is to have three 7.2 V NiMH batteries (in series) to drive the electric scooter, and one 7.2 V NiMH battery to control the management system of our device.


*Figure 5: Battery No. 4 Charged State.*


*Figure 6: Battery No. 3 Charged State.*


*Figure 7: Battery No. 2 Charged State.*


*Figure 8: Battery No. 1 Charged State.*

The management system for the device will be both the charging circuits (which will have their own implemented over/under charge protection) and a raspberry pi 3B+ which will monitor the temperature, current, voltage, and health of the batteries. We've considered many configurations to properly maintain the batteries, and we've decided that this would be the easiest way to monitor and control the state of the batteries.

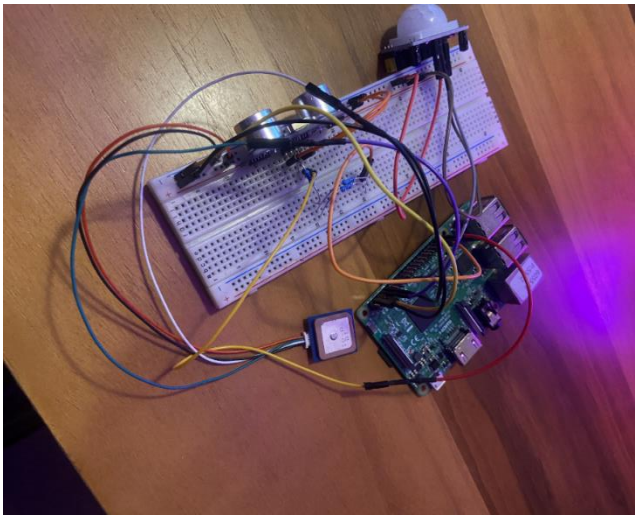


*Figure 9: Management and Monitoring System (Raspberry PI 3B+).*

The three batteries in series each have a nominal voltage of approximately 7.2V. However, these batteries when fully charged have a voltage of around 8.4V. This means that when these batteries are in series, we'll have a maximum and nominal voltage of 25.2V and 21.2V. As shown below.

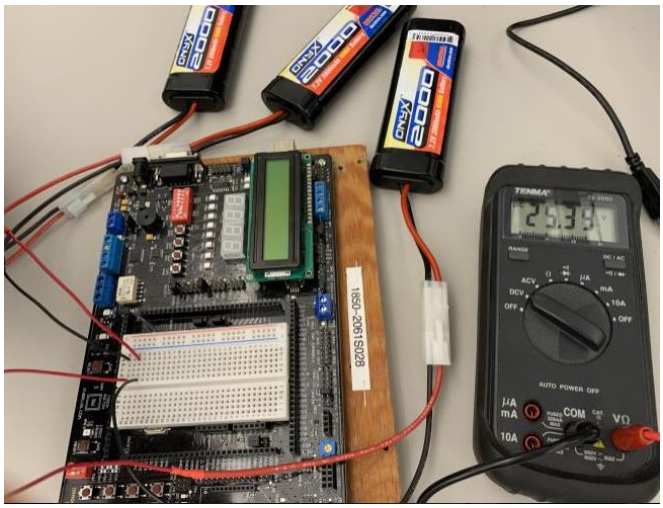



*Figure 10: Batteries No. 4, 3, and 1 in Series Configuration (24V).*

This voltage range is perfect for driving the electric scooter, thus if we want to charge this battery bank, we will use a 24-25V power supply and a 24V charging circuit. As for the management system, we have a single NiMH battery (7.2V nominal, 8.4V fully charged) which also needs to be charged from time to time, this can be done when the drive battery is charging. The 24V supply can be dropped using a BUCK voltage switching regulator. The Raspberry Pi (management system) runs on a voltage range of 3-5V DC, to account for this, we can add another voltage regulator to drop the battery voltage to a usable 5V. The following diagram demonstrates a simulation of this circuit in SIMULINK.

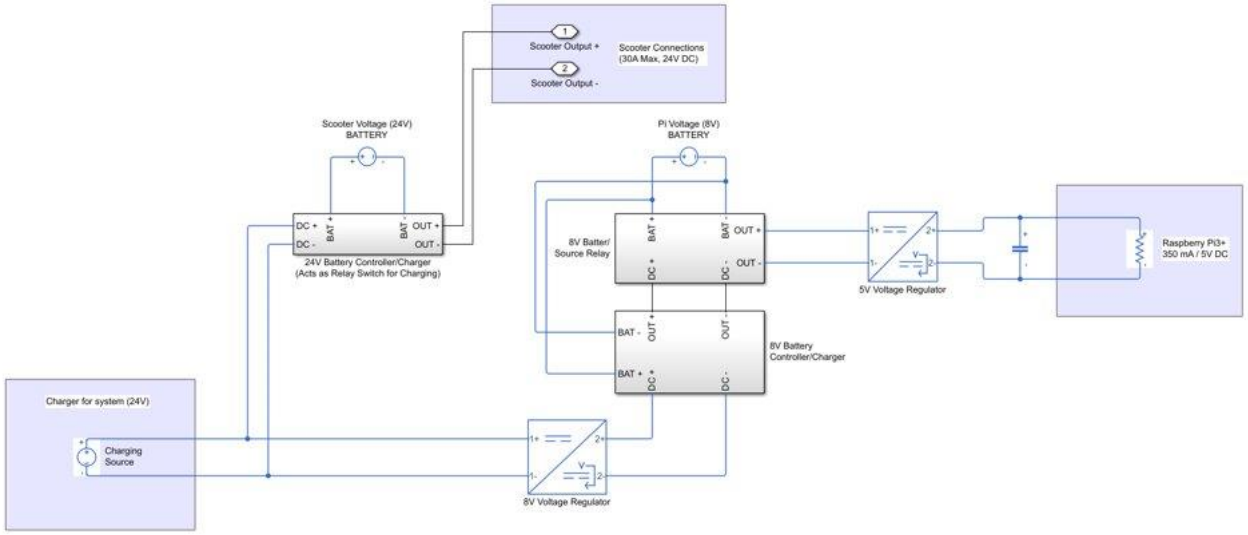



*Figure 11: Hardware Charging Circuit Simulation*

Upon charging the batteries, we need to switch the batteries from providing load to accepting a charge. This will effectively cut off the load of the batteries when charging. However, this poses a huge problem for our Raspberry Pi which needs to be monitoring the system constantly, especially when charging the batteries. To make up for this, we need to design a circuit that switches the Raspberry Pi's input to the source voltage when its battery is charging. This switching is measured to be around 0.7 seconds of downtime. To make up for this, we must add a capacitor in parallel with the pi, so that it may remain active during this switching period. Since the current draw of the circuit isn't very high, we can calculate the necessary capacitance as follows:

$$Capacitance = time_{discharge}(0.7) * \frac{current\ (0.35)}{voltage_i(5) - votlage_f(3)} \cong 100\ mF$$

Now, this capacitance is high, though we can simply use a supercapacitor to keep the input voltage to the Raspberry Pi between 5V and 3V. To achieve the switching between the battery and source and charging the battery, we've used a combination of a charging circuit (with voltage and current protection) and a load-switching circuit. Both these circuits are shown below.


*Figure 12: Load Switching Circuit*

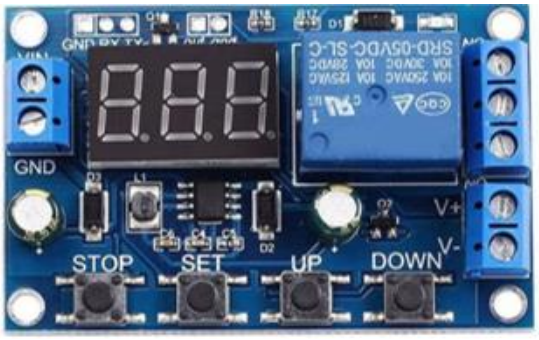



*Figure 13: Battery Charging Circuit (Raspberry Pi)*

The battery charging circuit uses a reference from the battery and determines if it is above or below the set Lower and Upper voltage bounds to open or close the relay (this circuit has over and under voltage protection, as well as monitoring). As for the load-switching circuit, we have the input voltage from the battery charging circuit relay and the input of the battery, if this circuit obtains a voltage higher than the battery voltage, it will switch the load from using the battery to using the source and begins charging the battery, this is where we have the 0.7-second switching delay. Both these circuits can handle up to 10A of current.

When charging the 24V battery cluster, we don't need to have an output while these batteries are charging, thus we can simply use a battery charging circuit with protection. The circuit we ended up using is demonstrated below. This circuit allows for up to 30A of continuous current (which can be limited) as well as the output voltage.


*Figure 14: Battery Charging Circuit (Raspberry Pi)*

For the power supply, as previously mentioned, we only need a 24V input DC power supply. The only voltage regulation needed is for the Raspberry Pi charger and Raspberry Pi inputs. For this, originally, we opted for using linear regulators (8V for the charging circuit and 5V for the Raspberry Pi), though this was very soon replaced with a switching modulated BUCK regulator due to the current limiting factors of the linear regulator and the heat losses due to its efficiency. Below are the comparisons and why we decided to change from using a linear regulator to a BUCK converter.

Figure 15: IC7808 and IC7805 Linear Reg.

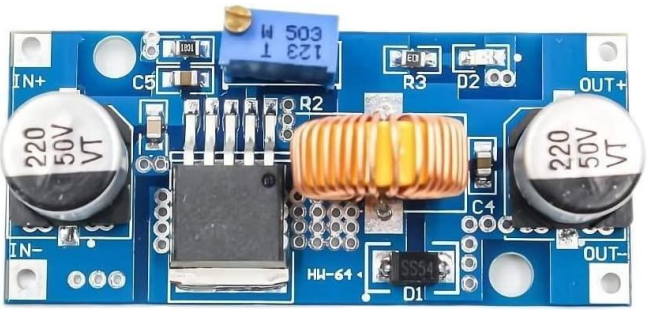



Figure 16: LM2596 BUCK Switching Reg.

This linear regulator, both 5V and 8V versions, is limited to a maximum current output of 1A continuous. We've attempted to use multiple in parallel for testing and have also equipped them with heatsinks to dissipate heat. However, they were still causing issues. Upon charging a single battery, the regulator would heat up so much it would decrease the output voltage to an unusable amount (4.5V for 8V reg).

The operating temperature of these regulators upon testing was approximately 80 degrees C. The efficiencies are calculated below.

Efficiency 8V: $\frac{V_{out}\,(8V)}{V_{in}(24V)} \cong 33\%\ Efficient$

Efficiency 5V: $\frac{V_{out}\,(5V)}{V_{in}(8V)} \cong 63\%\ Efficient$

This switching regulator uses a pulsed input to charge a capacitor. Using feedback, the PMW internally can adjust the output to match any set needs. This circuit we're using is limited to 5A which is more than enough for our uses. Upon charging a single battery, the circuit didn't heat up very much, the only component which heated up slightly was the inductor. The voltage was perfectly maintained, and the current output didn't affect the voltage.

The operating temperature of this regulator upon testing was assumed to be around 40-50 degrees C. The efficiency is provided by the manufacturer.

Efficiency 9V out, 24V in $\cong 89\%\ Efficient$

Efficiency 5V out, 9V in $\cong 90\%\ Efficient$

So far, all hardware-specific components have been tested and are functional, we are simply refining our circuits and trying to fit them all in a nice package so that they are presentable. There has been a more efficient way to design this charging and management circuit. For example, using the Raspberry Pi alone to manage relays which would charge and control the batteries. However, the design we have chosen functions well, and we will keep refining it to the best of our ability. More physical representations of the circuit will be shown at a later time. Below is a simple physical example of the charging circuit used for the Raspberry Pi.

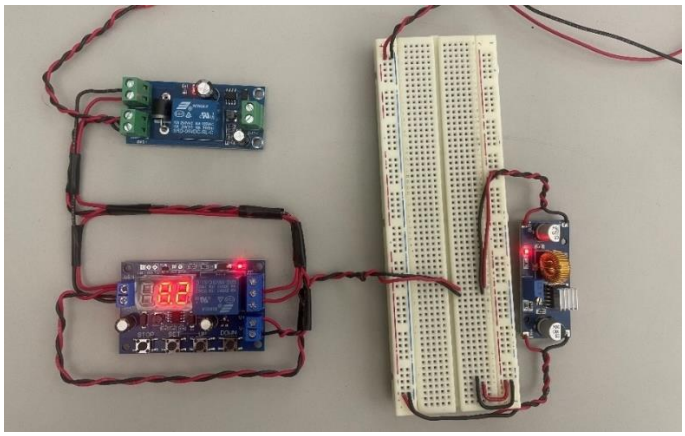


*Figure 17: Physical sample of charging circuit used for the Raspberry Pi.*

*Figure 18: Batteries Used for the Raspberry Pi & Scooter.*

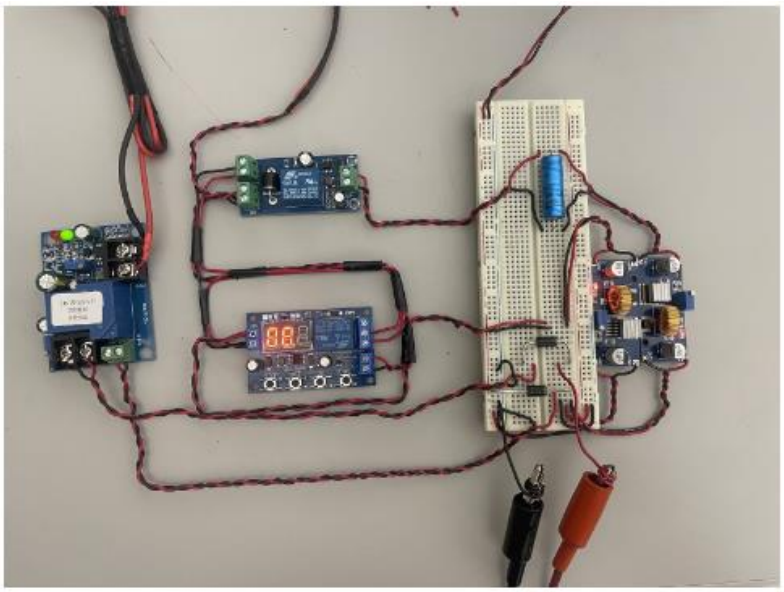


*Figure 19: Physical sample of charging circuit used for the Raspberry Pi & Scooter.*

In Figure 19 we can see the physical charging circuit for the scooter on the left. This charging circuit is being powered by a 24V source, and while the batteries are charging, the output terminals are cut off. The charging circuit for the Raspberry Pi is depicted in the center, this circuit is powered by a 9V source (down concerted from the original 24V rail). The Raspberry Pi circuit has a 'zero time' transfer switch which will transfer the load to the mains while the batteries are charging. This is in parallel with a capacitor to ensure no downtime on the raspberry pi. To ensure completely no downtime, a capacitor of 1mF is recommended.

Data Acquisition System

As a component of our battery management system, we are constructing a data acquisition system. Our objective is to leverage this system to monitor the battery's condition and assess its overall health. Our intelligent battery provides users with data to enhance their riding experience. We will gather information on the battery's performance using multiple sensors integrated into the battery pack.

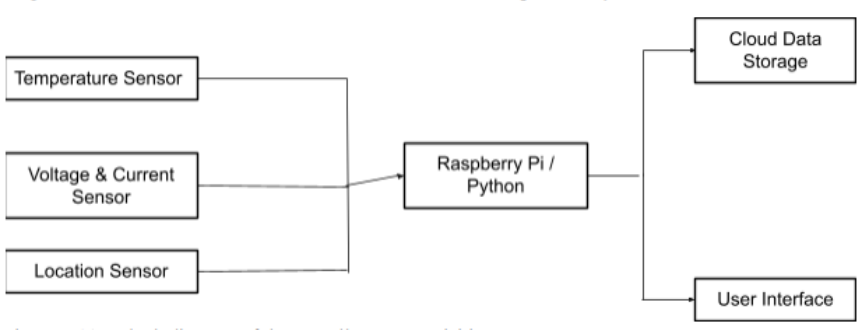



*Figure 20: Flow of the Data Acquisition System*

The diagram above illustrates the flow of the data acquisition system. The sensors are linked to the battery pack and collect data while it is in use. The data is subsequently transmitted to the Raspberry Pi, where it is processed and transformed into digital bits to be sent to the cloud. Once the data is transmitted to the cloud, it becomes accessible for use in a web application where we can exhibit valuable insights to the user. This system offers several advantages, such as enabling users to monitor the health of their battery pack, determine the remaining battery life for upcoming trips, assess power consumption, and locate the battery pack and scooter. Additionally, if feasible, we will incorporate an estimate of how much battery power is required for the user to complete a trip.

The data gathered from the sensors will be integrated into a feedback loop via the Raspberry Pi GPIO pins, which will enable us to respond appropriately to the data received. For instance, if the temperature of the battery surpasses the manufacturer's recommended temperature, we will turn off the battery and notify the user to allow the battery to cool down before reusing it. This feedback loop enables us to maintain our system in an optimal state continually.

Sensors:

Our strategy is to employ four sensors to gather data from the scooter. Although each sensor collects data independently, this data can be analyzed in the backend to extract valuable insights. The sensors we are using are:

1. Temperature sensor - To prevent overheating of the battery pack, we will employ a temperature sensor to monitor the pack's temperature. By reading the temperature, we can ensure that the temperature remains within acceptable limits.
2. Voltage sensor - utilized to ensure that the voltage remains within the desired range.
3. Current sensor - utilized to provide feedback on the current levels, which aids our system in preventing overheating occurrences.
4. GPS sensor - By utilizing this sensor, we can collect and analyze data in real-time, which corresponds with the scooter's speed. With the assistance of backend calculations, we can provide users with valuable insights on how their battery is being utilized.

Sensor Specifications:

Temperature Sensor:

Temperature sensors gauge temperature by using electrical signals. These sensors feature two metals that generate an electrical voltage or resistance when a temperature variation occurs. Temperature sensors operate by measuring the voltage across the diode terminals. As the voltage rises, the temperature also increases, which leads to a voltage drop between the transistor terminals and the emitter (in a diode). We will be using the ASAIR AM2302 temperature sensor for our project.

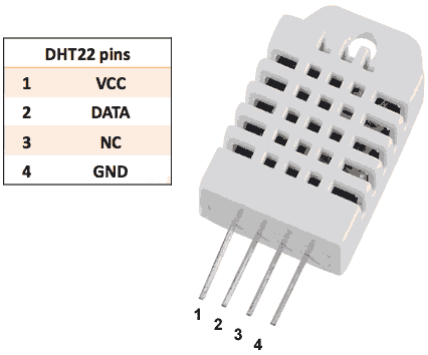



*Figure 21: ASAIR AM2302 temperature sensor*

| Parameter | Specifications |
|---|---|
| **Voltage Supply** | 3 to 5V power and I/O |
| **Temperature Range** | -40 to 80°C with ±0.5°C accuracy |
| **Max Current** | 2.5mA max current use during conversion (while requesting data) |

Voltage Sensor & Current Sensor:

There will be 2 sets of voltage and current sensors, one for each battery set. These sensors take a 0-25V input and 0-30A input and sends a 0-5V signal to the pi for sensor. In the case of the current sensor, an output 66mV/A, and for the case of the voltage sensor, and output of 50mV/V.

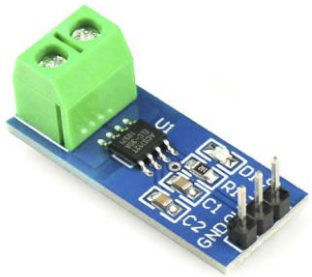



*Figure 22: 0-30A Current Sensor*

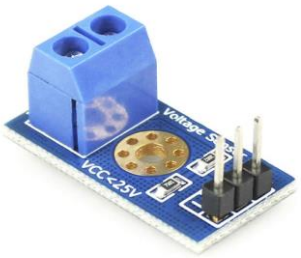



*Figure 23: 0-25V Voltage Sensor*

Location Sensor (GPS):

GPS sensors consist of receivers equipped with antennas that employ a satellite-based navigation system utilizing a network of satellites orbiting the earth. These satellites transmit signals to the GPS sensors (receivers) to provide a precise location. We will employ a Location sensor to monitor the user's location. The selected Location sensor is the Beitian BN-220, and its specifications are listed below.


*Figure 24: Beitian BN-220 GPS Sensor*

| Parameters | Specifications |
|---|---|
| **Voltage Supply** | 3V - 5.5 V (typically 5V) |
| **Current Output** | 50 mA |
| **Operating Temperature** | -40 to +85℃ |
| **Max Altitude** | 50,000 m |
| **Max Velocity** | 515 m/s |

After collecting the data, it will be transferred to the Raspberry Pi via its input/output pins. The Raspberry Pi will process this signal, which will be transformed into a digital signal and then transmitted to the cloud.

In the subsequent section, we will delve deeper into the process of the signals gathered by the sensor and how they are converted into digital signals for the computer to interpret. We will also provide additional information about the Cloud platform and the User Interface components of the system in this section.

Gateway:

Upon data collection, the Raspberry Pi acts as a gateway and transfers the information to Azure IoT Hub, which serves as the big data streaming service of Azure. It is specifically designed for high-throughput data streaming scenarios, where billions of requests may be received per day. When the data enters the IoT Hub, the Event Grid is triggered, which notifies us that new data is available for processing.

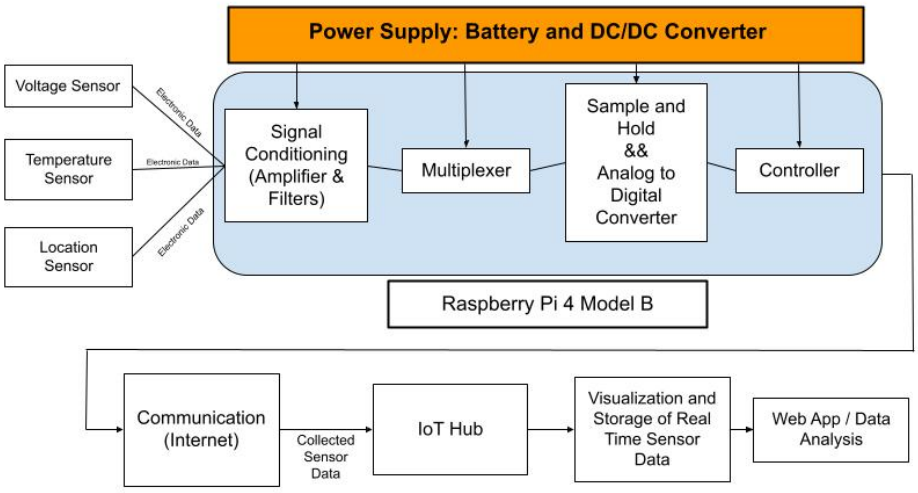



*Figure 25: Data Collection for The System*

After the data is collected, it is sent to the Microsoft Azure Cloud platform using the Raspberry Pi 3 B+. The collected data is stored in the cloud and can be accessed remotely at any time. Data processing is done in the Raspberry Pi before the signal is sent to the cloud. The signal first goes through a filtering and amplification process, followed by a multiplexer that allows only the relevant inputs to pass through. The signal then passes through a sample and hold circuit in conjunction with an Analog to Digital converter to convert the signal into binary values that can be interpreted by the computer. The digital signal is then transmitted to the cloud through GSM module communication between the Raspberry Pi and Microsoft Azure IoT Hub.

GSM Module:

Since we are working on Raspberry Pi based projects, when we talk about a true IoT project then it becomes a must thing to consider, so it is imperative to interface GSM SIM800L for providing internet to our Pi just like a modem using GSM GPRS. For this project, we are using Point to Point Protocol (PPP) and node to node communication for communicating with the GSM serially. After doing this project we'll not need to connect your pi with Wi-Fi or ethernet. Here's the connection list showing the interface of GSM module with raspberry Pi 3.

| **GSM Module** | **Raspberry Pi 3** |
|---|---|
| Rx | Tx of the Raspberry Pi |
| Tx | RX of the Raspberry Pi |
| GND | GND of the Raspberry Pi |
| | GND of the Buck |
| VCO | 4.2 V (2596 Ic output) |

Hence, we have connected all the pins accordingly. For a better understanding, the connection is demonstrated in the form of diagram.

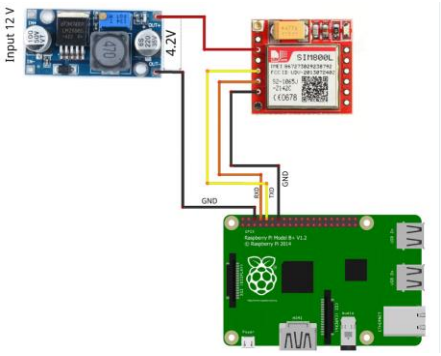



*Figure 26: GSM Module Configuration for Raspberry PI.*

The GSM module was installed successfully, we have chosen WaveShare Hat GSM module to integrate with our project; due to the limitation of the geographic reason, we could not get the appropriate SIM card to enable the wireless communication via GSM module.

Azure:

Microsoft Azure IoT Hub is a cloud-based service that acts as a central message hub for bi-directional communication between our data visualization software and the Raspberry Pi. As a Platform-as-a-Service (PaaS) managed service, IoT Hub offers various features and capabilities for managing, monitoring, and analyzing device-to-cloud and cloud-to-device communication.

Once the data is collected from the sensors, it is processed and sent to IoT Hub through a Wi-Fi connection. IoT Hub then stores and manages the data, allowing it to be accessed remotely at any time. This enables us to monitor the state of the battery in real-time and provide insights to the user about its usage.

One of the main advantages of using IoT Hub is its ability to handle high volumes of data from multiple devices. IoT Hub is designed for high throughput data streaming scenarios where customers may send billions of requests per day. This means that our system can scale easily and accommodate a large number of users and data points.

IoT Hub also offers various security features such as per-device authentication, firewall protection, and role-based access control. This ensures that our data is kept secure and only authorized users have access to it.

In summary, by using IoT Hub as our central message hub, we can collect, store, and manage data from our battery monitoring system in an efficient and secure manner. This allows us to provide real-time insights to our users about the state of their battery and ensure optimal performance and safety.

The initial steps in our project involved setting up the Raspberry Pi with the Raspbian operating system and writing a Python script to read data from the sensors. We utilized the Azure IoT Hub SDK library to establish a connection between the Raspberry Pi and the Azure IoT Hub, allowing us to send device-to-cloud messages and view them in the Azure portal terminal.

Moving forward, we plan to incorporate the Azure IoT Hub Service SDK and Azure Function to process the data collected by our sensors in the backend, using the ASP. Net Core framework. This will involve the development of APIs to be utilized in the front end, which will be built using React.

To store the collected data, we plan to implement a relational database, such as Azure SQL Server. This will allow for efficient and organized storage and retrieval of the data, as well as seamless integration with the rest of our Azure services.

Overall, our approach involves utilizing various Azure services and technologies to develop a comprehensive and efficient system for monitoring the state of a battery through sensor data collection and cloud-based processing.

IoT Hub Service:

Setting up an IoT Hub in Azure is a critical step in integrating IoT data into our web app. The IoT Hub is a central point of communication for our IoT devices, enabling them to send data to the cloud for processing and analysis.

To create an IoT Hub resource in the Azure portal, we will first need to sign into our Azure account and navigate to the IoT Hub service. From there, we can create a new IoT Hub resource, specifying a unique name that will be used as part of the domain name for the IoT Hub endpoint.

We will also need to select the subscription, resource group, and region where the IoT Hub will be hosted. The subscription determines the billing for the IoT Hub resource, while the resource group is a logical container for managing related Azure resources. The region determines the physical location of the IoT Hub, which can affect performance and compliance requirements.

Once the IoT Hub is created, we will need to manage it in the Azure portal, configuring security and access policies to ensure that only authorized devices and users can connect to the IoT Hub and access its data. This may involve configuring firewall rules, setting up shared access policies, and configuring identity and access management (IAM) roles.

Finally, we will need to monitor device connections and data to ensure that our IoT devices are sending data correctly and that the data is being processed and analyzed as expected. This may involve using Azure IoT Hub monitoring tools or third-party monitoring tools to view telemetry data, diagnostic logs, and device-to-cloud messages. By monitoring our IoT Hub, we can identify issues and optimize our IoT solution for maximum performance and efficiency.

Once we have set up our IoT Hub in Azure, our next step is to configure our IoT devices to send data to the cloud. This involves working closely with the Raspberry Pi to ensure that the device is configured correctly and can connect securely to the IoT Hub.

To connect our devices to the IoT Hub, we will need to configure each device with a unique device ID, a device key, and a connection string that specifies the IoT Hub endpoint and the security credentials for the device. We can use Azure IoT Device SDKs or third-party SDKs to help us configure our devices and establish a secure connection to the IoT Hub.

For example, if we are using Azure IoT Device SDKs, we can leverage the Azure IoT Device Provisioning Service (DPS) to register our devices and generate the necessary credentials and connection strings. We can also use Azure IoT Hub Device Twins to manage the configuration and state of our devices remotely, enabling us to update device firmware or settings without requiring physical access to the device. This is the current progress we have made in building the system.

Power BI:

After the telemetry data has been sent to the IoT Hub using MQTT protocol, we will redirect the data to Power BI dashboard using "Message Routing".

Message routing is the process of directing messages from a source to one or more destinations based on certain criteria or rules. In our project, we used Azure Service Bus, a messaging service in Azure, to enable reliable communication between different components in a distributed application. The message routing feature of Azure Service Bus allowed us to dynamically route messages to Power BI, a cloud-based business analytics service provided by Microsoft, based on the content of the message or the current state of the system.

Power BI is a powerful data analysis and visualization tool that allows us to create interactive reports and dashboards from various data sources. By integrating Power BI with our Azure Service Bus messaging system, we were able to process and analyze the messages in real-time and display the insights and visualizations in Power BI.

To accomplish this, we first created a connection between Azure Service Bus and Power BI. We then defined the message routing rules in Azure Service Bus to send the relevant messages to Power BI. When a message was received by Power BI, it was processed and visualized according to the predefined rules and reports.

This approach to message routing and data visualization allowed us to create a scalable and flexible distributed application that could handle large volumes of messages and provide real-time analytics and visualization capabilities through Power BI. It also ensured that messages were sent only to the intended recipient, improving the overall efficiency and reliability of the system.

Simulations:

Simulation 1:

For our first simulation we focused on the signal processing of the raspberry pi. We decided to simulate all the circuits that play a role in converting the signal from analog (original form) to digital (final form).

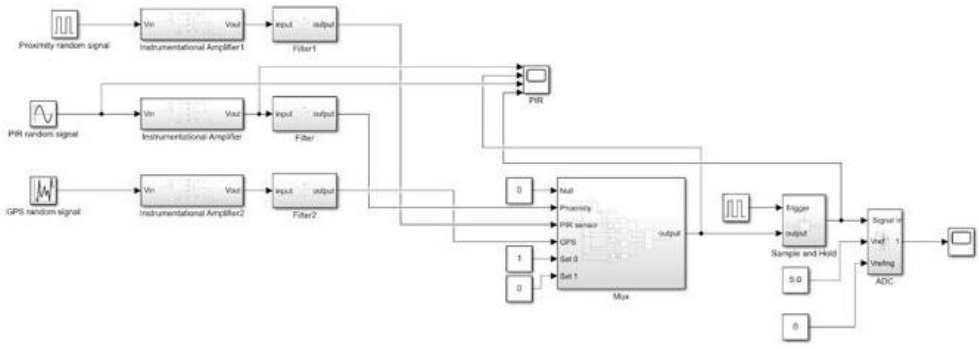



*Figure 27: Simulink circuit of the entire system.*

For our sensor signal, we used a random signal generator for the GPS sensor and other forms of signals such as square pulse and sinusoidal wave. These signals represent the analog signal output by the sensors. As mentioned before, the first step of signal processing is an amplifier. After the desired signal has been amplified, it goes to a filter that removes any undesired noise that might have been added to the signal along the way. Once the signal is clean and amplified, the multiplexer decides which signals are let through to the next step. We only need certain signals at specific times; this decision-making is determined by the code we wrote in python. Once the signal has permission to pass through the multiplexer, it enters the sample and holds. The sample and hold work in combination with analog to digital converter to convert the analog signal into binary form for the computer to understand. Let's look at what is inside of those blocks.

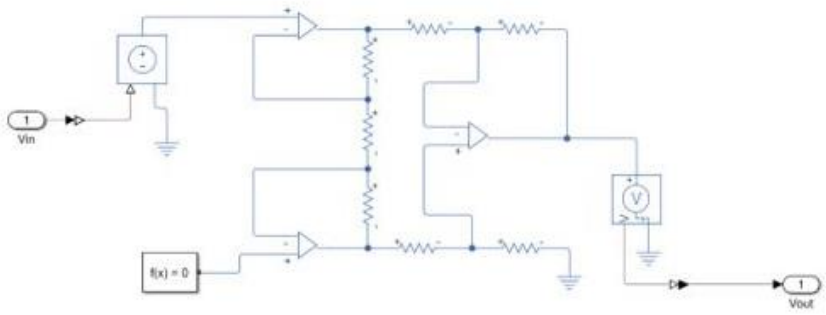



*Figure 28: Amplifier circuit.*

We picked an instrumental amplifier to simulate our amplifier block; this is due to the high input impedance characteristic of the circuit. An instrumental amplifier consists of two non-inverting amplifiers connected to a voltage follower, also called a buffer. Since we are feeding the bottom amplifier with f(x) = 0, this will amplify the difference between our input voltage and zero, which is just our input voltage. So, this circuit amplifies the desired sensor signal. After the signal has been amplified, it gets passed to a filter.

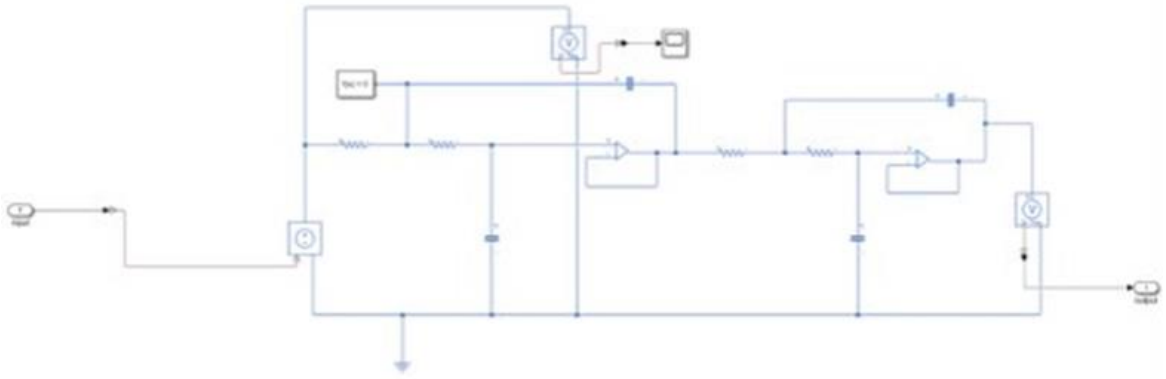


*Figure 27: Filter circuit.*

We used a bandpass filter in our filter block. A bandpass filter's characteristic function is designed to block undesired high-frequency noise and pass the desired sensor signal. This filter makes the signal a lot cleaner and more accurate. In addition, this process decreases the chance of error when encoding the signal into bits in the analog-to-digital converter. Next block is multiplexer.

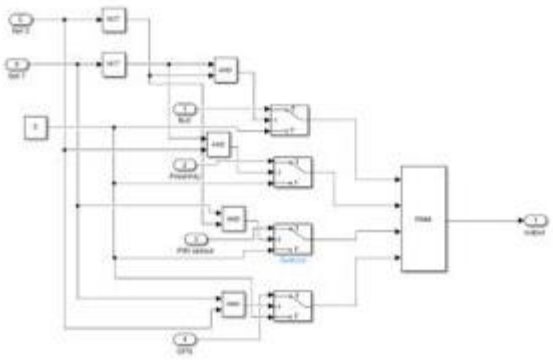


*Figure 29: Multiplexer circuit.*

Next, we look at the multiplexer circuit block. A multiplexer is a combinational logic circuit with multiple inputs and one output signal. It allows a signal to pass through based on the selected inputs. Like the other circuits, the multiplexer is in the processor and is controlled by the CPU. This means the code we write on, for example, python three, will eventually tell the multiplexer what to do and which signals to pass at what time.

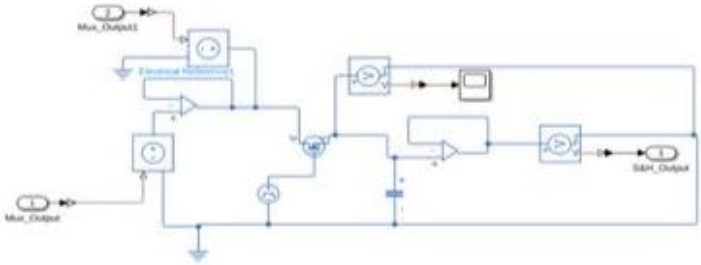


*Figure 30: Sample and Hold circuit.*

The next block is a sample and hold. The sample and hold circuit's job is to sample the signal at a given time and hold that sampled value. This circuit is combined with analog to digital converter to convert the signal to digital form.

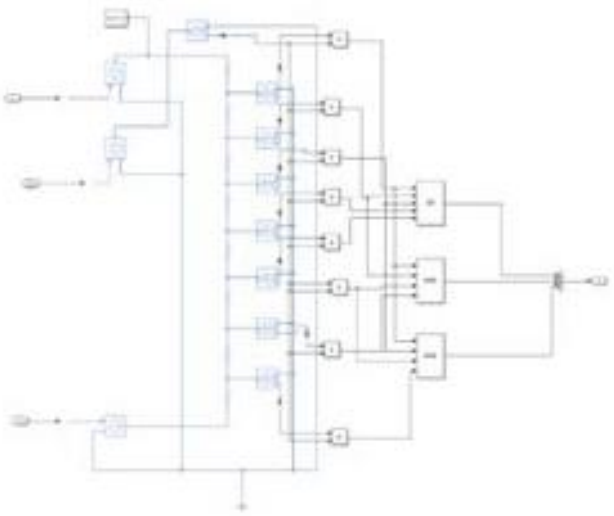


*Figure 31: Analog to Digital Converter circuit.*

The analog-to-digital converter is the last signal processing stage on the Raspberry Pi. It has three steps: the sample and hold, followed by a quantizer and a decoder. The quantizer and decoder use the sampled values to determine the combination of 0s and 1s to represent the analog signal. After that, the digital signal is to be interpreted and sent to the cloud.

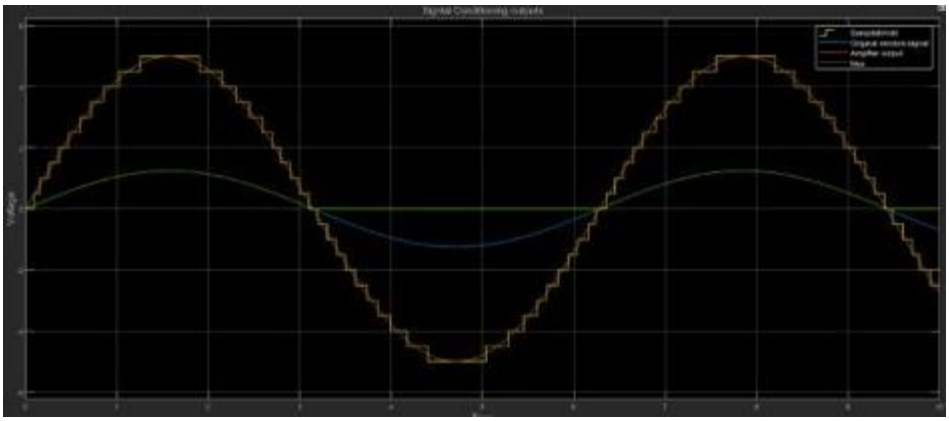


*Figure 32: Signal at every step of signal processing stages.*

Above signal is the output of the signal in each step of the simulation. The blue line is the original signal which we can see operates at $\pm 1.5V$ $\pm 1.5V$ , and yellow is the output of the quantizer. This simulation allowed us to better understand the role of each circuit in the signal processing phase and to see what happens to the signal after it is collected by the sensors.

Simulation 2:

For our second simulation, we used the online Raspberry Pi Azure IoT Simulator to set up our configuration and test the Raspberry Pi python code.

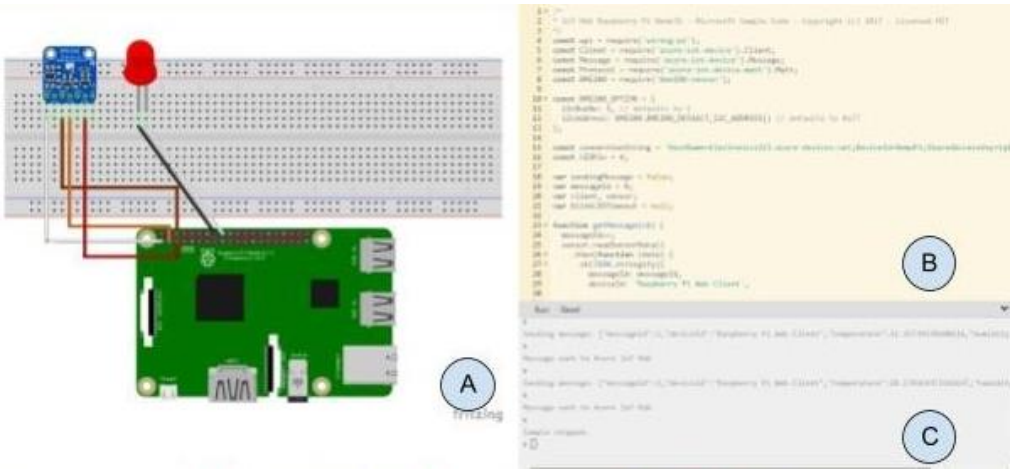



*Figure 33: Raspberry pi Azure IoT Online Simulator Window.*

There are three areas in the web simulator.

I. Assembly area - A BME280 sensor and an LED are connected to a Pi by default. Since the section is locked in the preview version, personalization is not presently possible.

II. Coding area - An online code editor for the code for Raspberry Pi. The default sample application helps to collect sensor data from BME280 sensor and sends it to your Azure IoT Hub. The application is fully compatible with real Pi devices.

III. Integrated console window - It shows the output of your code, i.e., the messages that are being sent to the IoT hub.

We then linked the Raspberry pi to IoT Hub to receive its message. Next step was to create a device identity in the identity registry in our IoT hub. A device can't connect to a hub unless it has an entry in the identity registry.

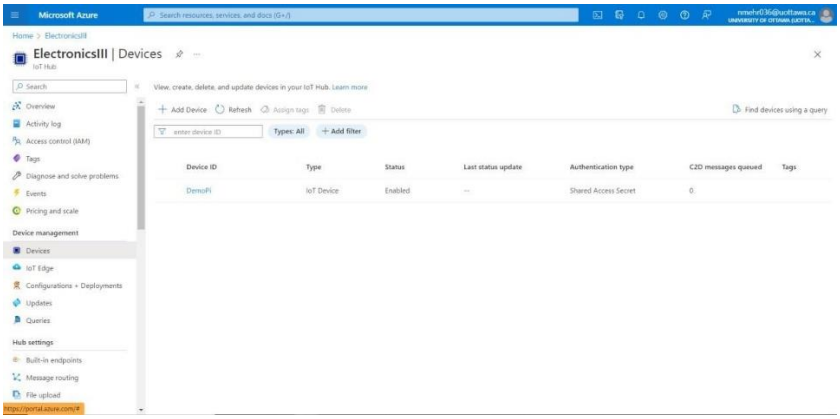



*Figure 34: IoT Hub Devices Window.*

We then display the message received by the IoT hub in Visual Studio Code Terminal. This is done through the Azure IoT Hub Extension available in VS Code.

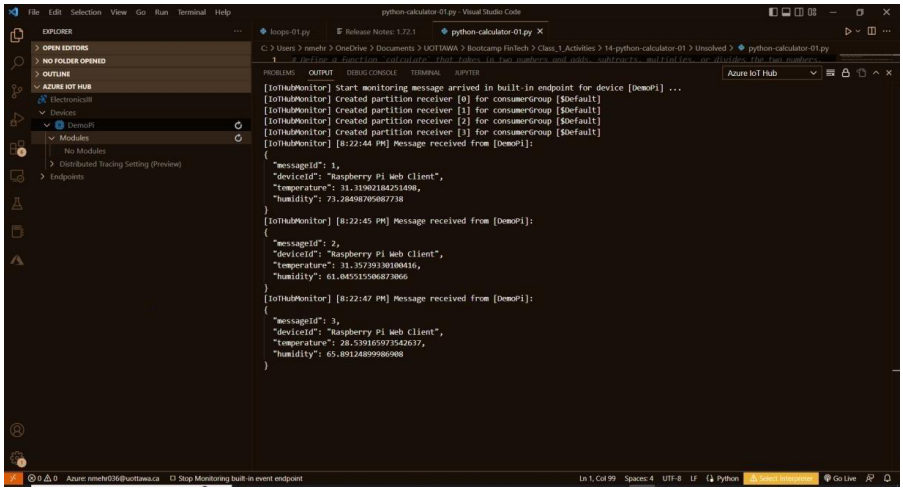



*Figure 35: VS Code Terminal.*

Once we register the device using its *Secondary Key Connection String* which is provided by IoT Hub, we can receive and display the live data being sent by the Raspberry Pi. Figure 5.11 shows VS Code Terminal displaying the messages sent by the Raspberry Pi simulation in figure 5.9 displayed in its integrated console window.

Work Breakdown Structure:

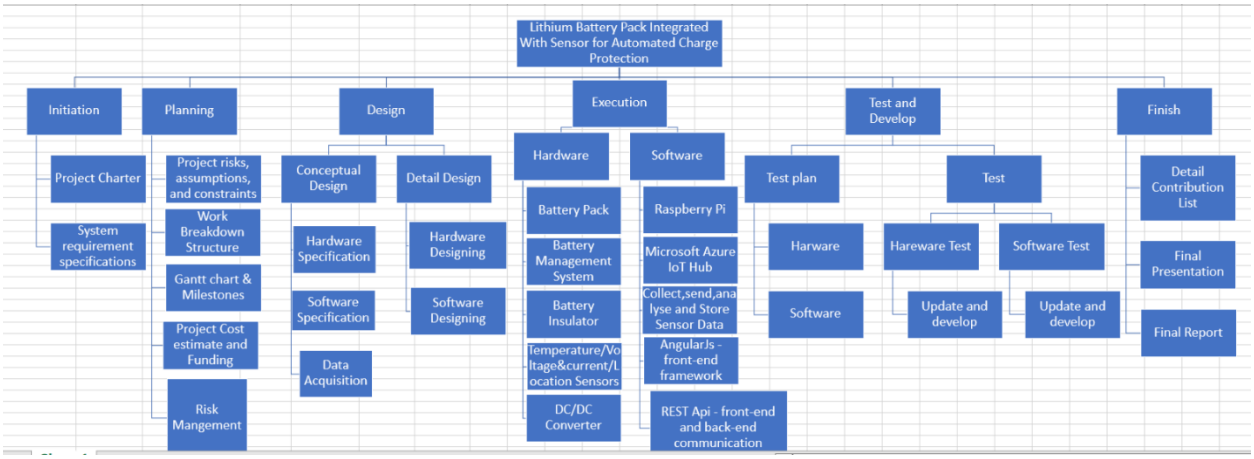
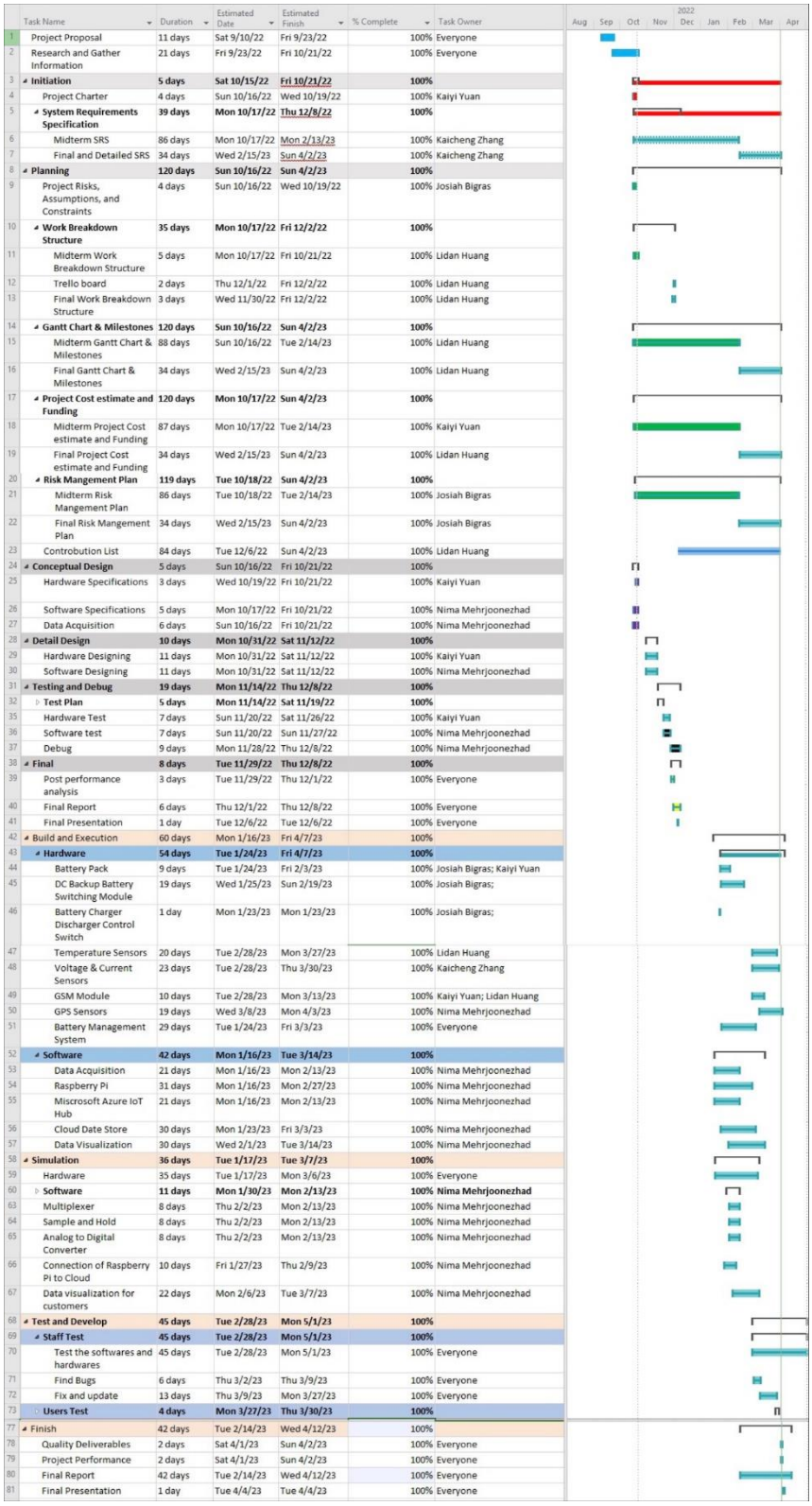



*Figure 36: Work Breakdown Structure.*

Gantt Chart:



| | Task Name | Duration | Estimated Date | Estimated Finish | % Complete | Task Owner |
|---|---|---|---|---|---|---|
| 1 | Project Proposal | 11 days | Sat 9/10/22 | Fri 9/23/22 | 100% | Everyone |
| 2 | Research and Gather Information | 21 days | Fri 9/23/22 | Fri 10/21/22 | 100% | Everyone |
| 3 | **Initiation** | **5 days** | **Sat 10/15/22** | **Fri 10/21/22** | **100%** | |
| 4 | Project Charter | 4 days | Sun 10/16/22 | Wed 10/19/22 | 100% | Kaiyi Yuan |
| 5 | **System Requirements Specification** | **39 days** | **Mon 10/17/22** | **Thu 12/8/22** | **100%** | |
| 6 | Midterm SRS | 86 days | Mon 10/17/22 | Mon 2/13/23 | 100% | Kaicheng Zhang |
| 7 | Final and Detailed SRS | 34 days | Wed 2/15/23 | Sun 4/2/23 | 100% | Kaicheng Zhang |
| 8 | **Planning** | **120 days** | **Sun 10/16/22** | **Sun 4/2/23** | **100%** | |
| 9 | Project Risks, Assumptions, and Constraints | 4 days | Sun 10/16/22 | Wed 10/19/22 | 100% | Josiah Bigras |
| 10 | **Work Breakdown Structure** | **35 days** | **Mon 10/17/22** | **Fri 12/2/22** | **100%** | |
| 11 | Midterm Work Breakdown Structure | 5 days | Mon 10/17/22 | Fri 10/21/22 | 100% | Lidan Huang |
| 12 | Trello board | 2 days | Thu 12/1/22 | Fri 12/2/22 | 100% | Lidan Huang |
| 13 | Final Work Breakdown Structure | 3 days | Wed 11/30/22 | Fri 12/2/22 | 100% | Lidan Huang |
| 14 | **Gantt Chart & Milestones** | **120 days** | **Sun 10/16/22** | **Sun 4/2/23** | **100%** | |
| 15 | Midterm Gantt Chart & Milestones | 88 days | Sun 10/16/22 | Tue 2/14/23 | 100% | Lidan Huang |
| 16 | Final Gantt Chart & Milestones | 34 days | Wed 2/15/23 | Sun 4/2/23 | 100% | Lidan Huang |
| 17 | **Project Cost estimate and Funding** | **120 days** | **Mon 10/17/22** | **Sun 4/2/23** | **100%** | |
| 18 | Midterm Project Cost estimate and Funding | 87 days | Mon 10/17/22 | Tue 2/14/23 | 100% | Kaiyi Yuan |
| 19 | Final Project Cost estimate and Funding | 34 days | Wed 2/15/23 | Sun 4/2/23 | 100% | Lidan Huang |
| 20 | **Risk Mangement Plan** | **119 days** | **Tue 10/18/22** | **Sun 4/2/23** | **100%** | |
| 21 | Midterm Risk Mangement Plan | 86 days | Tue 10/18/22 | Tue 2/14/23 | 100% | Josiah Bigras |
| 22 | Final Risk Mangement Plan | 34 days | Wed 2/15/23 | Sun 4/2/23 | 100% | Josiah Bigras |
| 23 | Controbution List | 84 days | Tue 12/6/22 | Sun 4/2/23 | 100% | Lidan Huang |
| 24 | **Conceptual Design** | 5 days | Sun 10/16/22 | Fri 10/21/22 | 100% | |
| 25 | Hardware Specifications | 3 days | Wed 10/19/22 | Fri 10/21/22 | 100% | Kaiyi Yuan |
| 26 | Software Specifications | 5 days | Mon 10/17/22 | Fri 10/21/22 | 100% | Nima Mehrjoonezhad |
| 27 | Data Acquisition | 6 days | Sun 10/16/22 | Fri 10/21/22 | 100% | Nima Mehrjoonezhad |
| 28 | **Detail Design** | **10 days** | **Mon 10/31/22** | **Sat 11/12/22** | **100%** | |
| 29 | Hardware Designing | 11 days | Mon 10/31/22 | Sat 11/12/22 | 100% | Kaiyi Yuan |
| 30 | Software Designing | 11 days | Mon 10/31/22 | Sat 11/12/22 | 100% | Nima Mehrjoonezhad |
| 31 | **Testing and Debug** | **19 days** | **Mon 11/14/22** | **Thu 12/8/22** | **100%** | |
| 32 | **Test Plan** | **5 days** | **Mon 11/14/22** | **Sat 11/19/22** | **100%** | |
| 35 | Hardware Test | 7 days | Sun 11/20/22 | Sat 11/26/22 | 100% | Kaiyi Yuan |
| 36 | Software test | 7 days | Sun 11/20/22 | Sun 11/27/22 | 100% | Nima Mehrjoonezhad |
| 37 | Debug | 9 days | Mon 11/28/22 | Thu 12/8/22 | 100% | Nima Mehrjoonezhad |
| 38 | **Final** | **8 days** | **Tue 11/29/22** | **Thu 12/8/22** | **100%** | |
| 39 | Post performance analysis | 3 days | Tue 11/29/22 | Thu 12/1/22 | 100% | Everyone |
| 40 | Final Report | 6 days | Thu 12/1/22 | Thu 12/8/22 | 100% | Everyone |
| 41 | Final Presentation | 1 day | Tue 12/6/22 | Tue 12/6/22 | 100% | Everyone |
| 42 | **Build and Execution** | 60 days | Mon 1/16/23 | Fri 4/7/23 | 100% | |
| 43 | **Hardware** | **54 days** | **Tue 1/24/23** | **Fri 4/7/23** | **100%** | |
| 44 | Battery Pack | 9 days | Tue 1/24/23 | Fri 2/3/23 | 100% | Josiah Bigras; Kaiyi Yuan |
| 45 | DC Backup Battery Switching Module | 19 days | Wed 1/25/23 | Sun 2/19/23 | 100% | Josiah Bigras; |
| 46 | Battery Charger Discharger Control Switch | 1 day | Mon 1/23/23 | Mon 1/23/23 | 100% | Josiah Bigras; |
| 47 | Temperature Sensors | 20 days | Tue 2/28/23 | Mon 3/27/23 | 100% | Lidan Huang |
| 48 | Voltage & Current Sensors | 23 days | Tue 2/28/23 | Thu 3/30/23 | 100% | Kaicheng Zhang |
| 49 | GSM Module | 10 days | Tue 2/28/23 | Mon 3/13/23 | 100% | Kaiyi Yuan; Lidan Huang |
| 50 | GPS Sensors | 19 days | Wed 3/8/23 | Mon 4/3/23 | 100% | Nima Mehrjoonezhad |
| 51 | Battery Management System | 29 days | Tue 1/24/23 | Fri 3/3/23 | 100% | Everyone |
| 52 | **Software** | **42 days** | **Mon 1/16/23** | **Tue 3/14/23** | **100%** | |
| 53 | Data Acquisition | 21 days | Mon 1/16/23 | Mon 2/13/23 | 100% | Nima Mehrjoonezhad |
| 54 | Raspberry Pi | 31 days | Mon 1/16/23 | Mon 2/27/23 | 100% | Nima Mehrjoonezhad |
| 55 | Miscrosoft Azure IoT Hub | 21 days | Mon 1/16/23 | Mon 2/13/23 | 100% | Nima Mehrjoonezhad |
| 56 | Cloud Date Store | 30 days | Mon 1/23/23 | Fri 3/3/23 | 100% | Nima Mehrjoonezhad |
| 57 | Data Visualization | 30 days | Wed 2/1/23 | Tue 3/14/23 | 100% | Nima Mehrjoonezhad |
| 58 | **Simulation** | **36 days** | **Tue 1/17/23** | **Tue 3/7/23** | **100%** | |
| 59 | Hardware | 35 days | Tue 1/17/23 | Mon 3/6/23 | 100% | Everyone |
| 60 | **Software** | **11 days** | **Mon 1/30/23** | **Mon 2/13/23** | **100%** | **Nima Mehrjoonezhad** |
| 63 | Multiplexer | 8 days | Thu 2/2/23 | Mon 2/13/23 | 100% | Nima Mehrjoonezhad |
| 64 | Sample and Hold | 8 days | Thu 2/2/23 | Mon 2/13/23 | 100% | Nima Mehrjoonezhad |
| 65 | Analog to Digital Converter | 8 days | Thu 2/2/23 | Mon 2/13/23 | 100% | Nima Mehrjoonezhad |
| 66 | Connection of Raspberry Pi to Cloud | 10 days | Fri 1/27/23 | Thu 2/9/23 | 100% | Nima Mehrjoonezhad |
| 67 | Data visualization for customers | 22 days | Mon 2/6/23 | Tue 3/7/23 | 100% | Nima Mehrjoonezhad |
| 68 | **Test and Develop** | **45 days** | **Tue 2/28/23** | **Mon 5/1/23** | **100%** | |
| 69 | **Staff Test** | **45 days** | **Tue 2/28/23** | **Mon 5/1/23** | **100%** | |
| 70 | Test the softwares and hardwares | 45 days | Tue 2/28/23 | Mon 5/1/23 | 100% | Everyone |
| 71 | Find Bugs | 6 days | Thu 3/2/23 | Thu 3/9/23 | 100% | Everyone |
| 72 | Fix and update | 13 days | Thu 3/9/23 | Mon 3/27/23 | 100% | Everyone |
| 73 | **Users Test** | **4 days** | **Mon 3/27/23** | **Thu 3/30/23** | **100%** | |
| 77 | Finish | 42 days | Tue 2/14/23 | Wed 4/12/23 | 100% | |
| 78 | Quality Deliverables | 2 days | Sat 4/1/23 | Sun 4/2/23 | 100% | Everyone |
| 79 | Project Performance | 2 days | Sat 4/1/23 | Sun 4/2/23 | 100% | Everyone |
| 80 | Final Report | 42 days | Tue 2/14/23 | Wed 4/12/23 | 100% | Everyone |
| 81 | Final Presentation | 1 day | Tue 4/4/23 | Tue 4/4/23 | 100% | Everyone |

*Figure 37: Gantt Chart.*

Milestones:

| Project Milestone | Description | Date |
|---|---|---|
| Phase 1: Completed the Project Proposal | Determine the project topic.<br>Gather information. | 2022/09/23 |
| Phase 2: Plan and Design | Battery pack hardware and software conceptual design. System requirement specifications. Project planning, such as estimated costs, risks, planning time. | 2022/10/21 |
| Phase 3: Detail Design | Hardware and software detailed design. | 2022/11/12 |
| Phase 4: Testing and Debugging | Test plan. Staff test the software and debug. | 2022/11/20 |
| Phase 5: Updated Project Report (Design and Plan) | Updated and detailed SRS, Gantt chart, WBS, schedule with milestones, estimated budget, risk management plan, contribution list | 2022/11/30 |
| Phase 6: Final Report and Presentation (Design and Plan) | Detailed plan, conceptual design, schedule, estimated budget and post-performance analysis. Analyze test results and developments. Present about the final report. | 2022/12/08 |
| Phase 7: Construction and Execution of Hardware and Software | Connect hardware devices based on the circuit diagram and test software functionality. | 2023/02/10 |
| Phase 8: Hardware and Software Simulation | Simulate the hardware connection on MATLAB Simulink and run it and try whether the software program can run on the simulated circuit. | 2023/02/13 |
| Phase 9: Updated Project Report (Construction and Execution) | Project charter, System requirements specification, comparison of planed and accrual, detailed design, hardware and software simulation, schedule and budget outlook, demo. | 2023/02/27 |
| Phase 10: Finish (Final Report and presentation) | Project charter, System requirements specification, comparison of planed and accrual, detailed design, hardware and software simulation, updated schedule and budget outlook, demo, detailed contribution list, Post-Performance Analysis | 2023/04/12 |

Budget:

| Items | Price |
|---|---|
| 7.2V 2000mAh NiMH Battery | $0 |
| DC Backup Battery Switching Module | $10.97 |
| Buck/Converter | $7.78 |
| Battery Charger Discharger Control Switch | $19.23 |
| Battery Charger | $50.93 |
| Raspberry Pi | $0 |
| DHT22 Temperature Sensor | $0 |
| GPS | $0 |
| Voltage and Current Sensor | $16.94 |
| Timing Circuit | $11.22 |
| High Precision ADC Development Board for Raspberry Pi | $13.18 |
| SIM7600G-H | $130 |
| Wires & Capacitors | $0 |
| Total | $260.25 |

Risk Management Plan:

To properly understand all the risks involved with our project, we must first identify all the potential electrical hazards associated with small battery electrical circuits, including electrical shock, thermal burns, fire, and explosion. This may involve reviewing the electrical systems, equipment, and processes, as well as assessing the knowledge, skills, and experience of the group members who will work on the hardware.

To manage these potential risks, we've been coordinating with two individuals. Pierre Laflamme, the Health, Safety, and Risk Manager, and Michel Robert, the Electronic Technicians in SITE. Lately, we've been spending most of our time fact-checking all our components with Michel to ensure that all our components are working properly together and that the build of our device goes smoothly. As of this moment, we haven't had any safety issues. However, it's still important to follow a process plan in case a hazard does occur. To avoid risks, we avoid leaving equipment on while it isn't being used, we don't make electrical connections while equipment is on, and we verify all connections before powering on any equipment.

Even with our many changes to the design and components, the hazards remain the same, we are still dealing with an equally dangerous battery and are still at all the same risks as before. In fact, the same emergency procedure we specified in our initial risk assessment remains the same with these Nickel batteries.

All these risks can cause a lot of long-term damage to lithium batteries and overall shorten their lifespan. Along with this long-term damage, the batteries could become unstable and cause potential hazards. To mitigate these risks, the battery must be equipped with voltage, current, and temperature sensors. When the voltage or current draw becomes too high, the battery management system of the device will automatically isolate the battery, thus protecting the batteries themselves

and any external device connected to the system. This isolation will protect against the accidental dead shortening of the positive and negative leads. The voltage sensor will monitor the voltage levels of the batteries so that they aren't discharging past their minimum limit and charging past their maximum limit. Along with the voltage sensor, we will be using a designated charging controller. The temperature sensors will monitor if the batteries are overheating or need to be heated in colder temperatures. While charging the battery is too cold or warm temperatures, can negatively impact the battery or potentially damage it. To prevent this, the battery will again isolate itself.

As for potential personal risks from the device, it's important that our safety, and the safety of the consumer, comes first. All hazards should be mitigated, and all risks should be labeled and made clear. When it comes to the actual construction of our device, we must follow our process plan and follow the step-by-step plan that will mitigate human errors and protect us when working with equipment. If we must deviate from our process plan, we will keep note of it in case we need to retrace our steps. It's also important to have an emergency plan in case the battery does become unstable. In all cases, a fire extinguisher should be present. But in the case that the batteries dead short, or experience physical damage and begin expanding or smoking, they should be placed in a bucket of sand, brought outdoors, and left until the batteries stop showing signs of being unstable. The fire extinguisher can also be used in extreme situations.

Along with safety while constructing the device, it's good practice to also be safe in the working environment. PPE or personal protective equipment is necessary when working in an electrical lab. Some personal protection that will be used in our project are gloves, which can prevent shocking hazards and cuts, glasses, which will help protect against sparks when spot welding, and a mask, which can be very important to protect against solder fumes or other dangerous fumes when constructing the battery. When in the lab, the person responsible for the space must be notified, as well as the teaching assistant. No equipment will be operated alone, and no team member will be working in the lab environment alone. In our case, we will always be working in groups. While following the process plan, since we will be working in groups of two people or larger, the supervising group member will assure that all steps are performed correctly and that no corners are cut when the following procedure. Our developments in the lab will be continuously communicated to the teaching assistant.

Since we will be performing lots of work with lithium batteries, heavy metals, and electronic components, it's important to remember the carbon footprint of all these components and to recycle them when possible. Components such as these shouldn't be disposed of in normal waste bins, and instead should be disposed of at appropriate recycling plants. This is good practice since devices like lithium-ion batteries can be very harmful to the environment and may be hazardous to workers when disposed of in the landfill.

With the potential risks and the potential hazards of our project, we must assign the project a risk rating of 'HIGH'. The likelihood of exposure to the hazard is rated at a level of three out of five ('POSSIBLE), while the consequence of exposure to the hazard is at a level of three out of five ('MODERATE'). A hazard assessment has been attached, as well as an explanation of further precautions and hazards regarding this project.

| | | Consequence | | | | |
|---|---|---|---|---|---|---|
| | | Insignificant | Minor | Moderate | Major | Catastrophic |
| Likelihood | Almost Certain | High | High | Extreme | Extreme | Extreme |
| | Likely | Medium | High | High | Extreme | Extreme |
| | Possible | Low | Medium | High | Extreme | Extreme |
| | Unlikely | Low | Low | Medium | High | Extreme |
| | Rare | Low | Low | Medium | Medium | High |

*Figure 38: Hazard Assessment Table.*

Post-Performance Analysis:

The primary purpose of our group this semester is to set up our project plans and software design. This also includes the background of the work conducted, the design methodology, the design diagrams, the analysis of the results obtained, and the conclusion and recommendation for further development. Based on the cooperation of this semester, we can summarize our PPA.

**Mistakes were made**: In the process of uploading and saving the design information, we took it for granted that using the Wi-Fi module is the most common and convenient choice, so we researched and designed some codes and tested them. When we thought the result was perfect, TA suggested we use the BMS module instead of the Wi-Fi module. Because when users use the e-scooters outdoors, we cannot guarantee that they will be covered by Wi-Fi at any time.

**Things worked well**: After our team members searched and investigated the basic information, we successfully decided on the direction of the project through brainstorming. After listing the specific plan, the division of labor is clear, and the team members get along well and help each other at any time. The software is designed successfully and runs smoothly.

**We could have done differently**: If we start over this semester, we will think more about the real-world functions of our product. When designing a battery pack, not only consider its function and price, etc. But also pay more attention to its assembly method and the shape and size of the finished product to make it as convenient as possible for customers to use. Currently, our product size may be bigger than the average battery pack size.

**Improvements would suggest if our project was extended**: The biggest challenges for battery design are energy density, power density, charging time, life, cost, and sustainability. If we have more time for this semester, we hope to find ways to improve these performances and add them to our battery pack. Each feature improvement requires a lot of research and design. In the hardware production of the next semester, we will adjust as much as possible to improve the performance of the battery pack.

Contribution List:

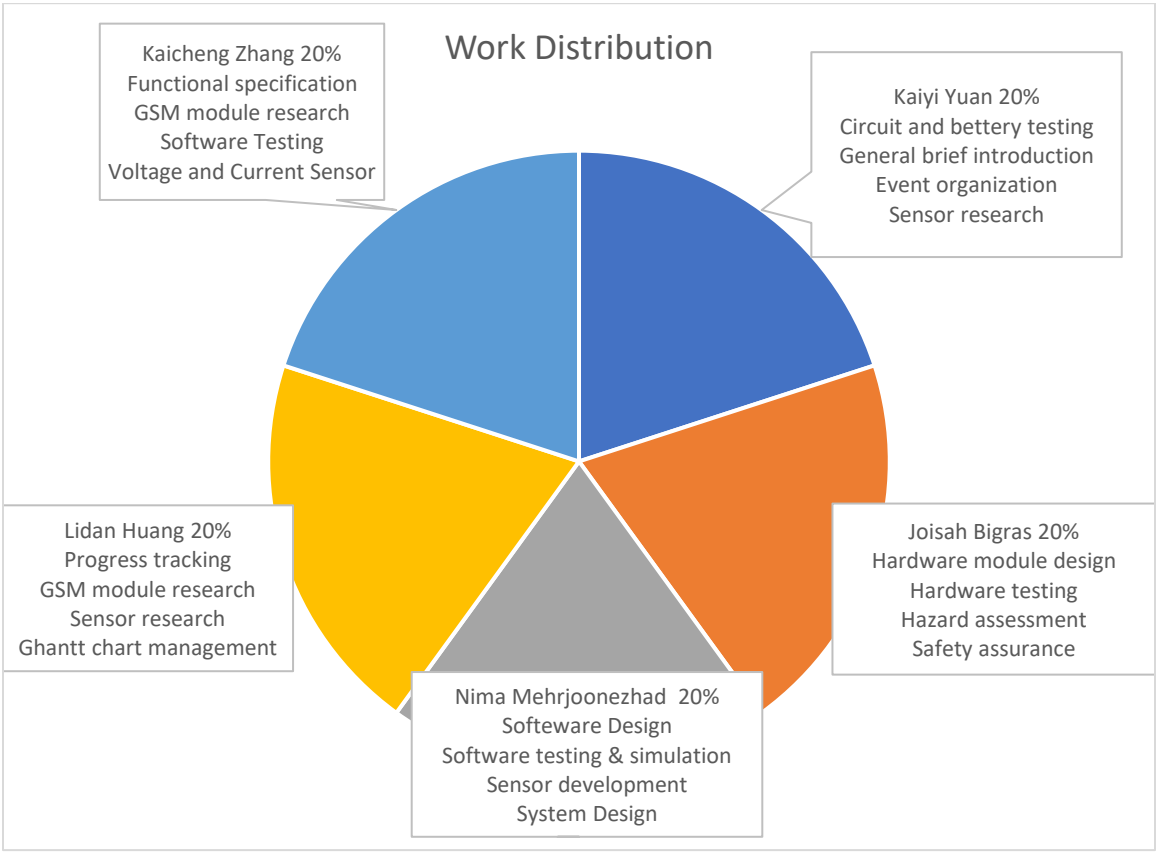



*Figure 39: Contribution List.*

Reference List:

"How to Tether Internet on Raspberry Pi Using GSM SIM800L Module?" *How to Connect Raspberry Pi to Internet Using GSM SIM800L Module?*. https://circuitdigest.com/microcontroller-projects/how-to-tether-internet-on-raspberry-pi-using-sim800l-gsm-module.