# CSCI 370 Final Report

Lunaris

Musad Alam
Sara Gampher
Megan Kulshekar
Dishita Sharma

Revised December 11, 2025

CSCI 370 Fall 2025

Prof. Donna Bodeau

Table 1: Revision history

| Revision | Date | Comments |
|---|---|---|
| New | 09/05/2025 | Initial draft of document |
| Rev – 2 | 09/21/2025 | Addition of Team Profiles and Software Quality section |
| Rev – 3 | 10/11/2025 | Addition of Design Architecture and Ethical Considerations sections, revised Functional Requirement and Software Quality sections |
| Rev - 4 | 10/26/2025 | Addition of Project Completion Status, Future Work, and Lessons Learned sections |
| Rev - 5 | 11/28/2025 | Updated Requirements, Definition of Done, System Architecture, Software Test and Quality, Project Completion Status and Results, Future Work sections. Addition of Acknowledgments, References, and Appendix A sections |
| Rev - 6 | 12/11/2025 | Revised sections based on peer and client feedback |

## Table of Contents

I. Introduction........................................................................3
II. Functional Requirements........................................................................3-6
III. Non-Functional Requirements........................................................................6
IV. Risks........................................................................6
V. Definition of Done........................................................................6-7
VI. System Architecture........................................................................7-10
VII. Software Test and Quality........................................................................10-14
VIII. Project Ethical Considerations........................................................................14-15
IX. Project Completion Status and Results........................................................................15-18
X. Future Work........................................................................18-19
XI. Lessons Learned........................................................................19
XII. Acknowledgments........................................................................19-20
XIII. Team Profile........................................................................20-22
References........................................................................22
Appendix A – Key Terms........................................................................22

## I. Introduction

Lunar Outpost is an industry leader in lunar surface mobility, commercial space robotics, and space resources. The company is also a contractor in various sectors known for building and creating rovers that can trek lunar surfaces. Lunar rovers are subject to different factors such as the impact of its electrical load or the weather, which can affect its energy capacity and performance. The goal of our project is to train astronauts for missions by simulating different conditions on rovers so that astronauts are prepared to act appropriately in emergencies. The project consists of two main models, a power and communication model, with smaller models within each to simulate and control these conditions on the rover.

The power model calculates the power draw of different electrical loads, which is then used for calculating the remaining battery capacity of the rover. The simulation also allows for fault injection from the user or triggered faults like the battery running down, which will cause certain power rails containing specific loads to turn off. The communications model allows for data communication with radios within a certain proximity of the rover. Both models should be compatible with a digital twin of the lunar rover so that we can predict, calculate, and act according to the effects on the rover's energy capacity. The models should also be extensible so that we can simulate these factors and perform accurate calculations for different lunar rovers.

## II. Functional Requirements

The power model consists of a battery, electrical load, and solar panel/umbilical power connection model that work together simultaneously to operate the rover. The communication model consists of a radio model. We specified requirements for each model of our system as well as requirements that encompassed all models. We decided upon this structure to have a clear understanding of the goals we were working towards. The requirements were also created according to Lunar Outpost's style for outlining requirements. Thus, the requirements were broken down into sublevels with Level 1 being a general idea of the requirement and Levels 2 and 3 giving specific detail as to how to fulfill the requirement. It is also important to note that the order of the requirements do not have any significance. The number of the requirement was only used for listing the requirement in this report. Below is a description of these requirements.

### Power Model Requirements

**Requirement 1** - Battery Model Requirements

Applies to the battery model.

**Level 1:** The battery model should be able to accept inputs and parameters before calculating the outputs.

**Level 2:** As an input, the battery model should accept power draw from all the different loads on each of their voltage rails.

**Level 3:** The battery model should subscribe to each of the loads power draw topics. The power draw is published in watts independently for each load.

**Level 2:** As a parameter, the battery model should accept the battery max capacity in Joules.

**Level 3:** The ROS launch file for the battery model shall include the battery max capacity.

**Level 2:** As a parameter, the battery model should accept the voltage .vs. capacity curve of the battery.

**Level 3:** The ROS launch file for the battery model shall include the parameters for the battery curve.

**Level 2:** As a parameter, the battery model should accept the internal resistance.

**Level 3:** The ROS launch file for the battery model shall include the internal resistance in Ohms.

**Level 1:** The battery model must display the remaining energy capacity of the rover based on different factors.

**Level 2:** The remaining energy capacity power must be reported as a builtin ros2 message (e.g. std_msgs float64 in watts*hours or Joules) on a ROS topic.

**Level 2:** The instantaneous voltage and current must also be reported as a builtin ros2 message.

**Level 1:** The battery model should track the energy usage from the battery.

**Level 2:** The battery model shall integrate the net power from all the loads and solar panels.

**Level 1:** The battery model must be able to calculate the state of charge in Joules, the voltage, and the current of the battery when given the power draw of the rover.

**Level 2:** The battery node must accept the total power draw from a list of topics which are to be determined.

## **Requirement 2** - Calculate Power Draw

Applies to the load and solar panel models.

**Level 1:** The program must be able to calculate the total power draw from electrical loads.

**Level 2:** Each load must publish its instantaneous power draw in watts using a builtin ros2 message.

**Level 2:** The solar panel array must publish its instantaneous power production in watts using a builtin ros2 message.

**Level 2:** Each load must have an assigned voltage rail.

**Level 2:** The solar panels must have their own voltage rail independent from the other loads.

**Level 2:** Each solar panel must accept the angle of the sun as an input.

## **Requirement 3** - Handle Multiple Parameters

Applies to all models.

**Level 1:** The program must be able to handle the input of multiple parameters that could impact the rover's performance.

**Level 2:** Each parameter that defines the vehicle’s functionality must be defined in a launch file for the relevant ros2 nodes.

**Level 3:** The battery node must have a launch file parameter for nominal battery capacity.

**Requirement 4** - Make Program Extensible

Applies to all models.

**Level 1:** The program must be extensible and run successfully for different lunar rovers ranging from the Mapp vehicle to LTVS.

**Level 2:** All vehicle definition parameters must be definable in launch files.

**Requirement 5** - Show Result of Fault Injections

Applies to all models.

**Level 1:** The program must be able to show the impact of external fault injections on the solar panels, battery, or electrical load of the rover.

**Level 2:** The solar panels model must accept reasonable fault conditions and model them appropriately.

**Level 3:** The solar panel node must accept an amount of dust coverage as a builtin ros2 message.

**Level 2:** The battery model must accept reasonable fault conditions and model them appropriately.

**Level 3:** The battery must report when its temperature limit has been exceeded.

**Level 2:** The system should show change when fault conditions occur.

**Level 3:** The system should disable power to loads when the battery is non-operational due to a fault.

**Level 2:** The electrical loads model must accept reasonable fault conditions and model them appropriately.

**Level 3:** Electrical loads should model a “short” fault, where they draw a large amount of current (low resistance).

## Communications Model Requirements

**Requirement 1** - Display Proximity of Rover to Radio

Applies to the radio model.

**Level 1:** The program must display whether the rover is within range of a given radio or not.

> **Level 2:** The proximity is a boolean value (1 - in proximity, 0 - not in proximity) that should be reported as a builtin ros2 message.
>
> > **Level 3:** The proximity is calculated by using the ellipse equation.
> >
> > **Level 3:** The coordinates of the radio will be defined in a launch file.

## III. Non-Functional Requirements

These requirements were created after meeting with the client and discussing their needs. Since our simulation model will be integrated into our client's pre-existing dashboard, these requirements focus more on the implementation of our software.

**Requirement 1** - The program should implement the ros2 framework.

**Requirement 2** - The program should be written in C++.

**Requirement 3** - The program should run in a Linux Ubuntu container.

**Requirement 4** - The project should use Bitbucket for VCS.

**Requirement 5** - The program should have a CI/CD pipeline built in.

**Requirement 6** - The program should be verified using static code analysis.

## IV. Risks

While working through the project, the team faced both technical and skills risks. Below is a description of these risks.

### Technical Risks

- The members of the team do not have Linux Ubuntu machines so they will need to install Docker and create their container properly to run the program.
- Different models may overwrite messages when publishing to a specific topic. These instances will require implementing additional conditions on a case-by-case basis.

### Skills Risks

- The team will need to get familiar using the ros2 framework as this is their first time using it.
- Only two members of the team have prior knowledge about circuits. Thus, the rest of the team will need to learn more about these topics and how it relates to the project.

## V. Definition of Done

The main outcome of our project is to have a power and communications model simulation that can take in data given by Lunar Outpost and run calculations to output expected values. The power model should contain functionality for the battery, electrical loads, solar panels, and voltage rail controls/fault injections. The power model should be able to output the voltage, state of charge, and current drop. The communications model should be able to output whether the rover is in proximity to a radio.

## VI. System Architecture

Figures 1 and 2 below show a high-level overview of the inputs and outputs as well as the launch files required to launch the specific model:

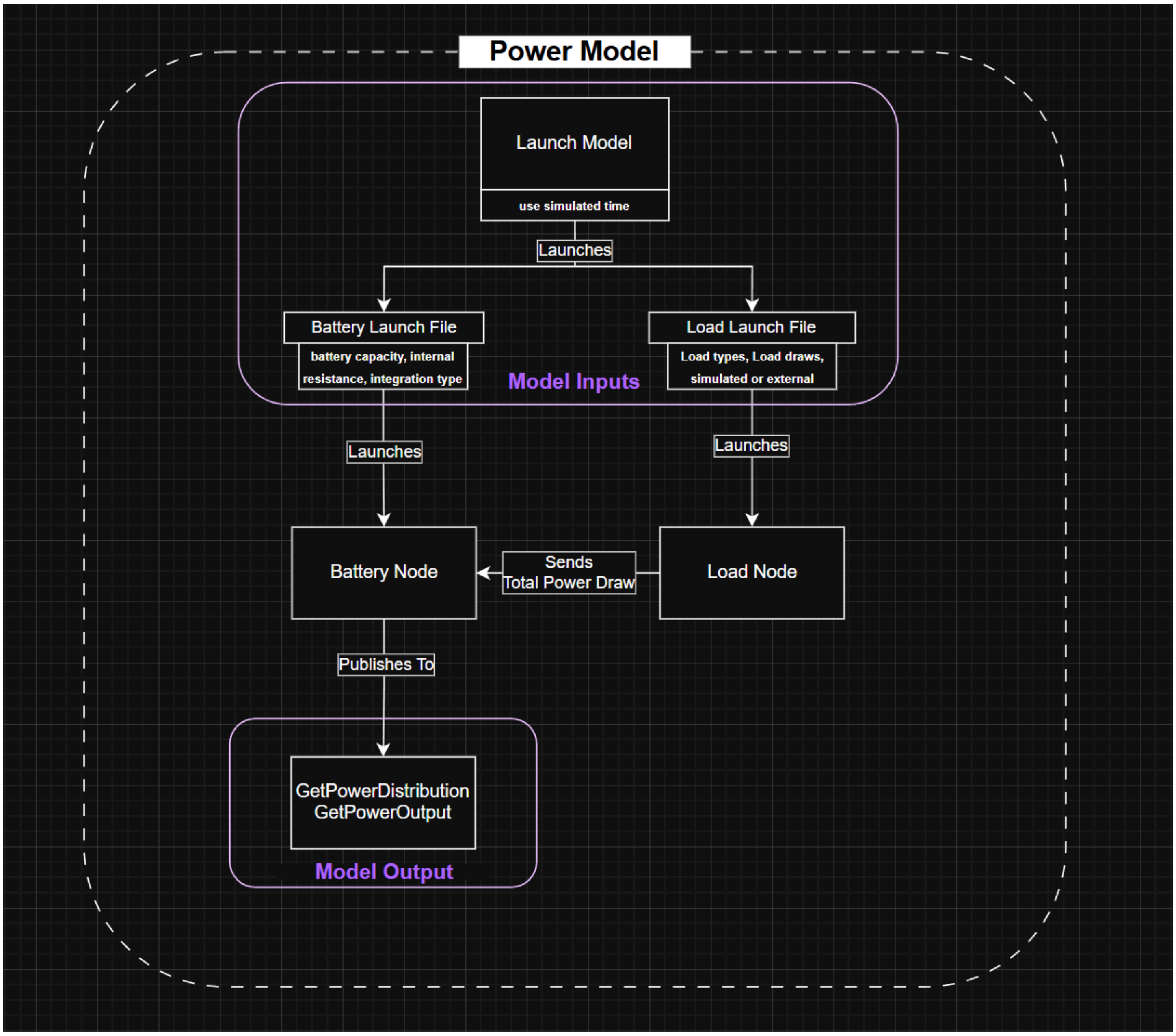


*Figure 1: Flow of Data in Power Model*

Figure 1 shows how the Power Model requires two launch files to be launched simultaneously. The battery launch file contains information such as battery capacity in Joules, internal resistance of the battery in Ohms, and integration type for calculations. The load launch file consists of information like the load names, max, nominal, and standard deviation power draw values in Watts, as well as the external topics list to subscribe to. When both launch files are launched, power draw calculations for each load occur in the load node. These values for power draw are then published to a topic that the battery node listens to. With this information, the battery capacity decreases based on a calculation that happens within the battery node. Finally, relevant information such as if the rover does not have enough battery capacity or if the loads are drawing more power than allotted, can be found by running the getPowerDistribution and getPowerOutput functions.

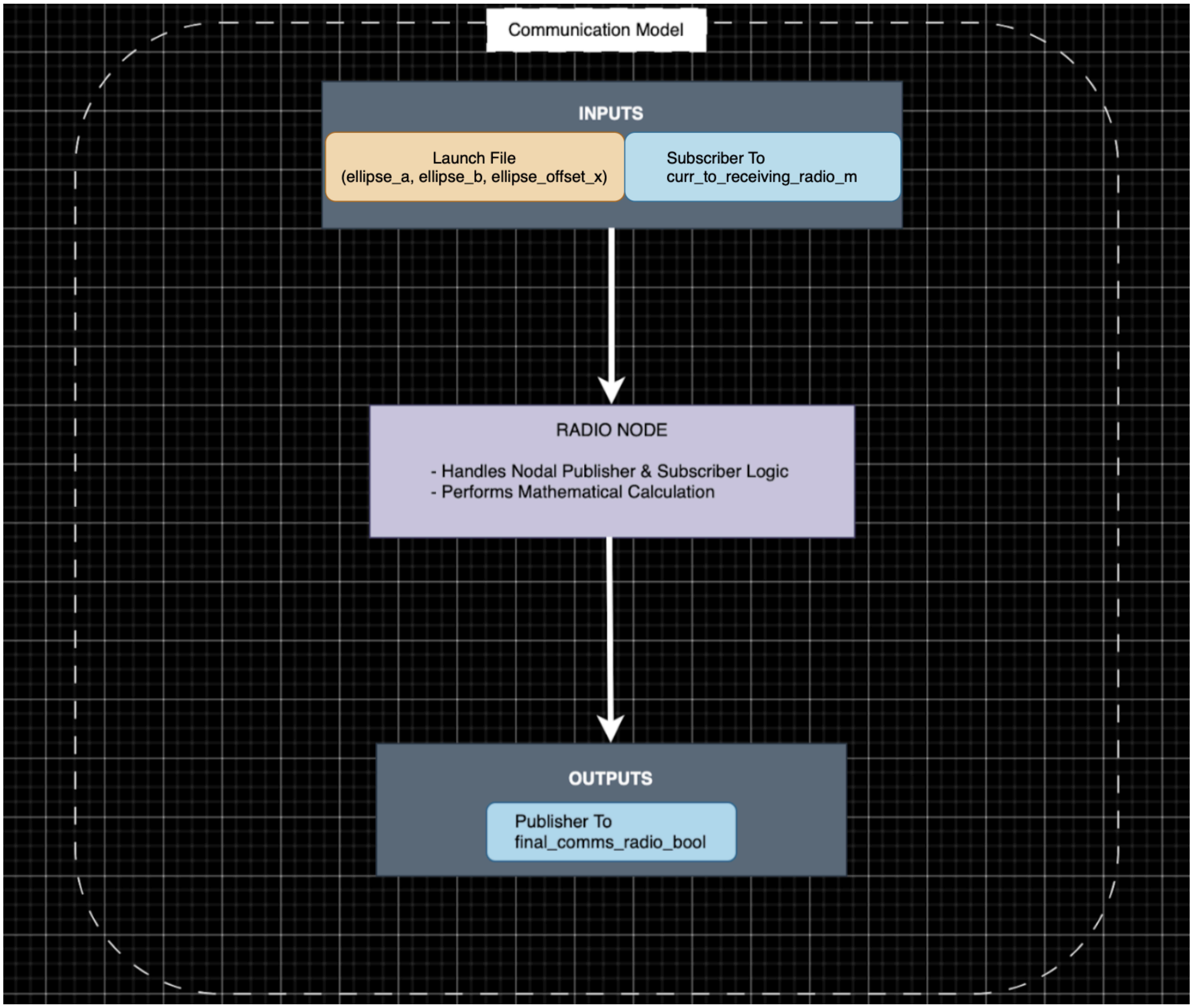

Communication Model
INPUTS
Launch File
(ellipse_a, ellipse_b, ellipse_offset_x)
Subscriber To
curr_to_receiving_radio_m
RADIO NODE
- Handles Nodal Publisher & Subscriber Logic
- Performs Mathematical Calculation
OUTPUTS
Publisher To
final_comms_radio_bool

*Figure 2: Flow of Data in Communications Model*

Similarly, Figure 2 shows how the Communications Model takes in a launch file that specifies parameters such as the ellipse size and ellipse offset. It also listens to an external topic called curr_to_receiving_radio_m for the angle in degrees and current distance in meters of the rover from the radio. Within the radio node, calculations are made to check if the rover is within proximity of the radio. The output, which is a 0 (not in proximity) or 1 (in proximity), boolean value is then published to the final_comms_radio_bool topic. This topic can be printed to the command line so the user or client can check whether the rover is close enough to the radio every second.

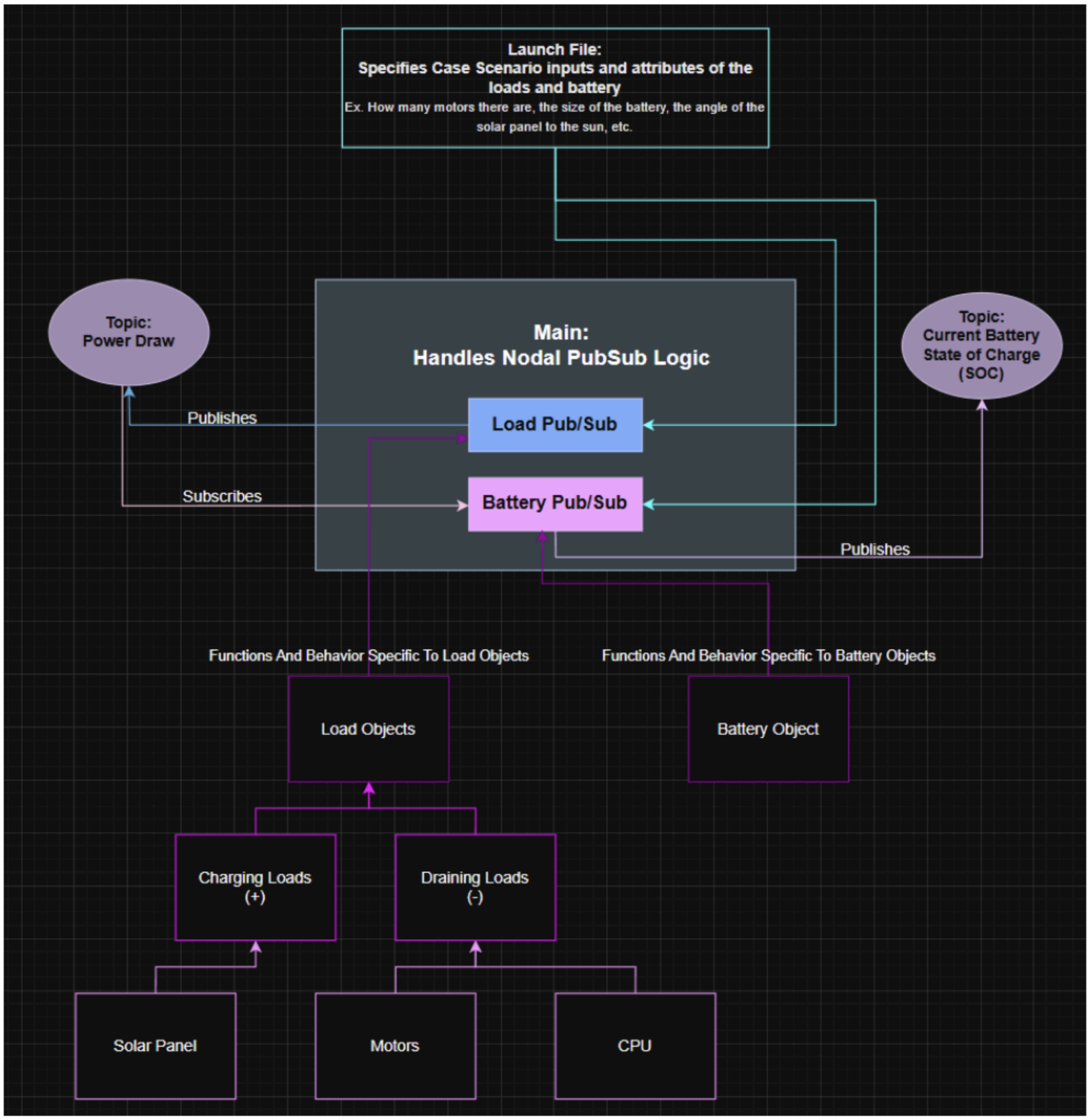

Launch File:
Specifies Case Scenario inputs and attributes of the loads and battery
Ex. How many motors there are, the size of the battery, the angle of the solar panel to the sun, etc.
Topic: Power Draw
Main: Handles Nodal PubSub Logic
Topic: Current Battery State of Charge (SOC)
Publishes
Load Pub/Sub
Subscribes
Battery Pub/Sub
Publishes
Functions And Behavior Specific To Load Objects
Functions And Behavior Specific To Battery Objects
Load Objects
Battery Object
Charging Loads (+)
Draining Loads (-)
Solar Panel
Motors
CPU

*Figure 3: Nodal Communication Diagram*

Figure 3 displays our more detailed implementation plan for each node. It contains the communication plan between nodes, how the predefined node objects will contribute, and the launch file defining certain parameters. This diagram helps us understand how to begin setting up the concrete filesystem to create our project. The launch file is where we can change and add parameters to run different simulations while keeping the rest of the code untouched. Our architecture also contains a "Main" box where the launch file is read in and parsed to acquire the load data, and where the publisher and subscriber logic is handled.

## VII. Software Test and Quality

We agreed upon some rules to ensure software quality throughout the development process. First, each team member will create their own local working version of the code while regularly pulling changes from the main Bitbucket repository. Team members will utilize VSCode with a built in C++ linter to ensure all code through the project remains consistent. Before writing code, the team member will create unit tests and think of edge cases. When the team member has completed their code for a task, they will run their unit tests to verify accurate functionality. They will also test their code with data provided by the client. Since our project does not require a user interface, the data will be checked against the output given in the command line. Additionally, the team member will write documentation for each method and variable they have used in the header file of the model they are working on. The documentation will be written using Doxygen. After that, the team member will submit a pull request in Bitbucket, which will notify all other members that a code review is pending. Everyone will review the code visually and functionally, requesting changes to the code as necessary. Team members may also add and run additional unit tests at this stage. A built in CI/CD pipeline will start automatically after a pull request has been submitted to ensure that the codebase will still build and run. Following a successful CI/CD pipeline and approval from all team members, the code is then pushed into the main branch.

Unit testing and integration testing are crucial for our project given that we are integrating our models into the client's pre-existing dashboard. Thus, we decided to come up with some general unit testing guidelines described below to guarantee consistency when writing tests.

- Write unit tests before writing functional code
- Write unit tests in Google's GTest framework for C++
- Ensure the execution of tests are independent from one another (one test does not affect the other and one test for each function)
- Be able to test different nodes/functions independently
- Write unit tests that will run and cover 90% of the codebase
- Include descriptive comments before each defined unit test, explain what is being tested, and what the intended output is

Additionally, we have created a software test procedure document to verify the output of each of our sub-models. The document can be found here: Software Test Procedure.

We also decided to break down the unit tests based on functionality within each model. Below are test tables outlining the name of the test, the expected result, and the actual result.

**Power Model Tests**

**Battery**

| Test | Expected Result | Actual Result |
|---|---|---|
| fault_battery_depleted | Fault code 5 should be published when battery is below threshold | The JointState message in position 0 has a value of 5.0 |
| fault_battery_normal | Fault code 0 should be published when battery is above threshold | The JointState message in position 0 has a value of 0.0 |
| interpolate_at_zero_percent | The battery voltage should be 3.6 V when there is 0% state of charge (this is an edge case for our model) | Running our interpolate_cell_voltage() method results in 3.6 V when the battery capacity is initially set to 1000.0 J and the remaining battery capacity is set to 0.0 J |
| interpolate_at_100_percent | The battery voltage should be 4.2 V when there is 100% state of charge (this is an edge case for our model) | Running our interpolate_cell_voltage() method results in 4.2 V when the battery capacity is initially set to 1000.0 J and the remaining battery capacity is set to 1000.0 J |
| interpolate_between_points | The battery voltage should be 3.854 V when there is 50% state of charge according to our mathematical calculations | Running our interpolate_cell_voltage() method results in 3.854 V when the remaining battery capacity is set to half of the initial battery capacity |
| initial_battery_capacity | The initial battery capacity of 2000.0 J should be published to the message properly | The message in position 0 is not empty and contains a value of 2000.0 J |
| bus_voltage_calc | The bus voltage should be 39.999 V and the current should be 0.050 Amps based off calculations | The JointState message in position 0 has a value of 39.999 V and in velocity position 0 has a value of 0.050 Amps |
| bus_voltage_calc_special_case | The current should be 0 Amps if the bus voltage is 0 V when the battery voltage is initially set to 0.0 V, the battery resistance is set to 0.000 Amps, and the power is set to 1.0 W | The JointState message in position 0 has a value of 0.0 V and in velocity position 0 has a value of 0.0 Amps |

## Electric Loads

| Test | Expected Result | Actual Result |
|---|---|---|
| mock_load_power_sum | Summation of load power should be 8.6 W when the solar and umbilical power are initially set to 1.5 W and the rest of the load draws are set to 2.5, 0.1, 1.2, 3.3 W respectively | The JointState message in position 0 has a value of 8.6 W |
| normal_dist_power | The normal distribution function should be greater than 0.0 W and less than the max power of 4.0 W | The JointState message containing the value calculated from the normal distribution function is not empty and contains a value that meets both criterion of being greater than 0.0 W and less than 4.0 W |

## Solar Panel and Umbilical Power Connection

| Test | Expected Result | Actual Result |
|---|---|---|
| solar_left_panel_power | If there is 20% dust on the left solar panel, the left dust message should output a value of 0.2 | The JointState message at position 0 is 0.2 |
| solar_right_panel_power | If there is 10% dust on the right solar panel, the right dust message should output a value of 0.1 | The JointState message at position 0 is 0.1 |
| mock_generate_umbilical_power_dust | The dust message should have a value of 0.2 (20%) and the total umbilical power should be 36.0 W based off calculations | The JointState dust message in callback receives a value of 0.2, the JointState power message that is published has a value of 36.0 in position 0 |
| mock_generate_umbilical_power_sun_angle | The sun angle message should have a value of 0.0 degrees and the total umbilical power should be 27.0 W based off calculations | The JointState sun angle message in callback receives a value of 0.0, the JointState power message that is published has a value of 27.0 in position 0 |

## <u>Communications Model Tests</u>

### Radio

| Test | Expected Result | Actual Result |
|---|---|---|
| PointAtOriginIsInside | The output should be true, the point at origin should be inside the ellipse when the distance is initially set to 0.0 m, the angle is 0.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is 0.0 m | TRUE - the point at origin is inside the ellipse |
| PointInsideEllipseOnXAxis | The output should be true, the point (5,0) should be inside the ellipse when the distance is initially set to 5.0 m, the angle is 0.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is 0.0 m | TRUE - the point (5,0) is inside the ellipse |
| PointOutsideEllipseXAxis | The output should be false, the point (15,0) should be outside the ellipse when the distance is initially set to 15.0 m, the angle is 0.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is 0.0 m | FALSE - the point (15,0) is outside the ellipse |
| PointOutsideEllipseYAxis | The output should be false, the point (0,10) should be outside the ellipse when the distance is initially set to 10.0 m, the angle is 90.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is 0.0 m | FALSE - the point (0,10) is outside the ellipse when b=5.0 |
| NegativeOffsetShiftsCenter | The output should be true, the offset center should be inside the ellipse when the distance is initially set to -5.0 m, the angle is 180.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is -5.0 m | TRUE - the point at offset center is inside the ellipse |

| | | |
|---|---|---|
| DiagonalPointInsideEllipse | The output should be true, the diagonal point should be inside the ellipse when the distance is initially set to 4.0 m, the angle is 45.0 degrees, the a value is 10.0 m, the b value is 5.0 m, and the offset at x is 0.0 m | TRUE - the diagonal point is inside the ellipse |
| CircularCase | The output should be true, the point should be inside the circular ellipse when the distance is initially set to 5.0 m, the angle is 45.0 degrees, the a value is 10.0 m, the b value is 10.0 m, and the offset at x is 0.0 m | TRUE - the point is inside the circular ellipse |

## VIII. Project Ethical Considerations

Ethics are an important part of our project for many reasons from protecting our client's proprietary information to following good standards when writing code. Some of the ACM/IEEE ethics principles that we find most pertinent to our project are Principles 2 and 3 which are described below.

**ACM Principle 2:** CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interest of their client and employer consistent with public interest.

**2.01.** Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education

The importance of Principle 2 is due to the nature of our work being used for space exploration by the client. It is critical that our work provides helpful and accurate assistance in the simulation of space exploration vehicles. We especially want to focus on 2.01 because our code will be used in production and there should not be any inaccuracies in our system that have occurred because a team member did not have the knowledge to perform their duties exceptionally. This could potentially cause harm to people using the system in the future if not addressed properly. Thus, in the absence of educational limitation, team members are expected to proceed only when they have researched enough such as by reading documentation, understanding how equations work, or asking relevant questions when meeting with the client POC.

**ACM Principle 3:** PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

**3.06.** Work to follow professional standards when available, that are most appropriate for the task at hand, departing from these only when ethically or technically justified

This principle is important because incorrect simulation could lead to the loss of multimillion dollar machinery or the endangerment of astronauts. This could impact our client significantly and the consequences could be drastic. Therefore, it is important that we follow requirements set by the client and their clients like NASA carefully. Team members are expected to familiarize themselves with relevant sections such as usability, safety, and security of standard documents.

The principle we believe is most in danger of being violated is **IEEE Principle III:** To strive to ensure code is upheld by colleagues and coworkers. This principle is highly dependent on members properly documenting code and environment setup. With all the dependencies in the ros2 architecture, there are various problems that could arise through a lack of proper documentation which could make the project unusable or relay inaccurate information. Neglecting to continuously document could slow development and/or prevent expandability or development of future project iterations.

The two Michael Davis tests that can be applied to our project are the **Reversibility** and **Mirror Tests**.

1. **Reversibility Test** asks the individual “Would this choice still look good if [we] traded places? (i.e. if [we] were one of those adversely affected by it?)”.

In our case, this would be trading places with astronauts doing space exploration or stakeholders who have invested tremendous amounts of money on lunar vehicles. If we were the astronauts, we would be counting on the simulation to be accurate since it will guide actions relevant to the mission and our safety. If we were a stakeholder, we want an accurate simulation because if not we could potentially lose millions in investments. Putting ourselves in their shoes, it becomes clear as to why this test is critical.

2. **Mirror Test** asks the individual “Would I feel proud of myself when I look in the mirror?”.

This test is relevant because our software will be used by large scale organizations who depend on research and development to ensure the safety of employees and success of missions. We should feel that our work is meaningful and contributes positively to the field of innovation. Having pride and feeling satisfied with the work we perform will help us meet expectations, work collaboratively, and deliver an exceptional product.

There are some ethical considerations we must also take into account if our software quality plan is not implemented properly. If thorough testing is not completed, then the product may miss an edge case. If this edge case occurs, it could mean failure of the mission which could result in catastrophic disaster either physically or financially for stakeholders. Another ethical consideration we must consider is the relationship between our university, Colorado School of Mines, and the client. Our reputation with this project matters and if we fail to follow our software quality plan, it could result in a damaged perspective of Mines students not upholding quality work.

Some security concerns we must focus on are ensuring the launch files we are given with parameter data come from a reliable source and that our code repository is secure and protected as it contains proprietary information from the client. Additionally, since the simulation is written in C++, we must pay close attention to memory manipulation. This is because there are a variety of ways in which to manipulate memory that could yield inaccurate results causing our simulation not to work. Therefore, it is fundamental that the team understands the functions they are using and implements memory safe functions when applicable.

## IX. Project Completion Status and Results

Our goal was to create a power model of a digital twin that simulates the battery capacity and power of a generic vehicle traversing a lunar surface. Our model also allows for control of fault conditions to avoid disasters before they happen. Additionally, we were tasked with creating a communications model that lets the user or client know if the rover is within range of a radio. We have achieved these goals by developing a C++

simulation that uses the ros2 architecture. Both of our main models use a publisher/subscriber architecture pattern to collect and send data.

Our power model uses the power of different loads to calculate the remaining battery capacity of the given rover. We have implemented a method for listening to messages consisting of the power drawn from different loads. This method is dynamic allowing the user of the system or the client to specify how many different loads there are. We listen to the power coming in from the solar panels and umbilical power connection and then run a calculation that outputs the total power draw and the remaining battery capacity after each second. We are also able to manage fault conditions by turning off voltage rails both manually and automatically when certain faults occur such as when the battery capacity has reached 0 Joules. For the communications model, we have calculated the proximity of the rover to the radio using a link closure equation, which is the equation of an ellipse.

Our program meets all of the functional requirements specified by the client. We have performed unit and integration testing, as well as system testing with our client. While our unit tests highlighted in this document use dummy data for verification, we have also tested our models with proprietary data given by the client. This is because our program extends the pre-existing dashboard our client has. We have also modeled our data to observe if the trends follow what we expect to happen. Figure 3 is an example of how the different loads and models in our program interact with each other and how they impact the final remaining capacity of the battery over time.

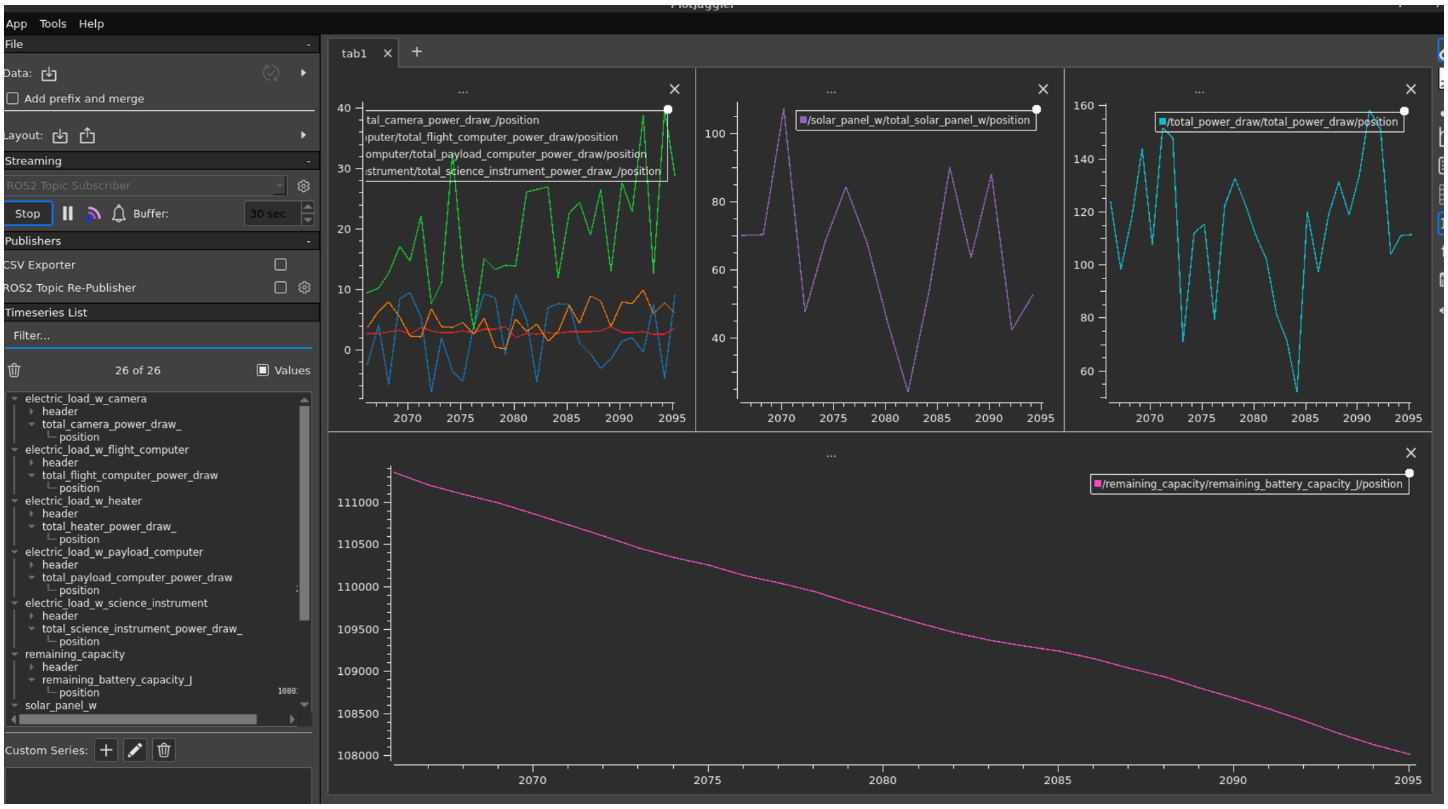

App Tools Help
File
Data:
Add prefix and merge
Layout:
Streaming
ROS2 Topic Subscriber
Stop
Buffer: 30 sec
Publishers
CSV Exporter
ROS2 Topic Re-Publisher
Timeseries List
Filter...
26 of 26
Values
electric_load_w_camera
header
total_camera_power_draw_
position
electric_load_w_flight_computer
header
total_flight_computer_power_draw
position
electric_load_w_heater
header
total_heater_power_draw_
position
electric_load_w_payload_computer
header
total_payload_computer_power_draw
position
electric_load_w_science_instrument
header
total_science_instrument_power_draw_
position
remaining_capacity
header
remaining_battery_capacity_J
position
solar_panel_w
Custom Series:
tab1
tal_camera_power_draw_/position
iputer/total_flight_computer_power_draw/position
omputer/total_payload_computer_power_draw/position
istrument/total_science_instrument_power_draw_/position
/solar_panel_w/total_solar_panel_w/position
/total_power_draw/total_power_draw/position
/remaining_capacity/remaining_battery_capacity_J/position
2070
2075
2080
2085
2090
2095
111000
110500
110000
109500
109000
108500
108000

*Figure 3: Distribution of Power Draw and Remaining Battery Capacity Graphs*

The plot in Figure 3 was created using PlotJuggler as it is customizable and allows us to view statistics for different loads. The left hand list shows the different types of loads and their associated topics which publish the power drawn after every second when the simulation is running. The top left graph shows the power drawing distributions for the various loads such as the camera, flight computers, and science instruments. The top middle graph shows the trend of solar panel power. The top right graph shows the total power drawn after accounting for all the loads. The bottom graph shows the remaining capacity of the battery of the rover which depletes over time. This means our system works as intended because as there are more loads, more power is drawn causing the battery capacity to decrease. Our graph has a negative linear slope as it is only an approximation of real-world scenarios. Figure 4 below also shows results of our calculations based on system testing with the client.

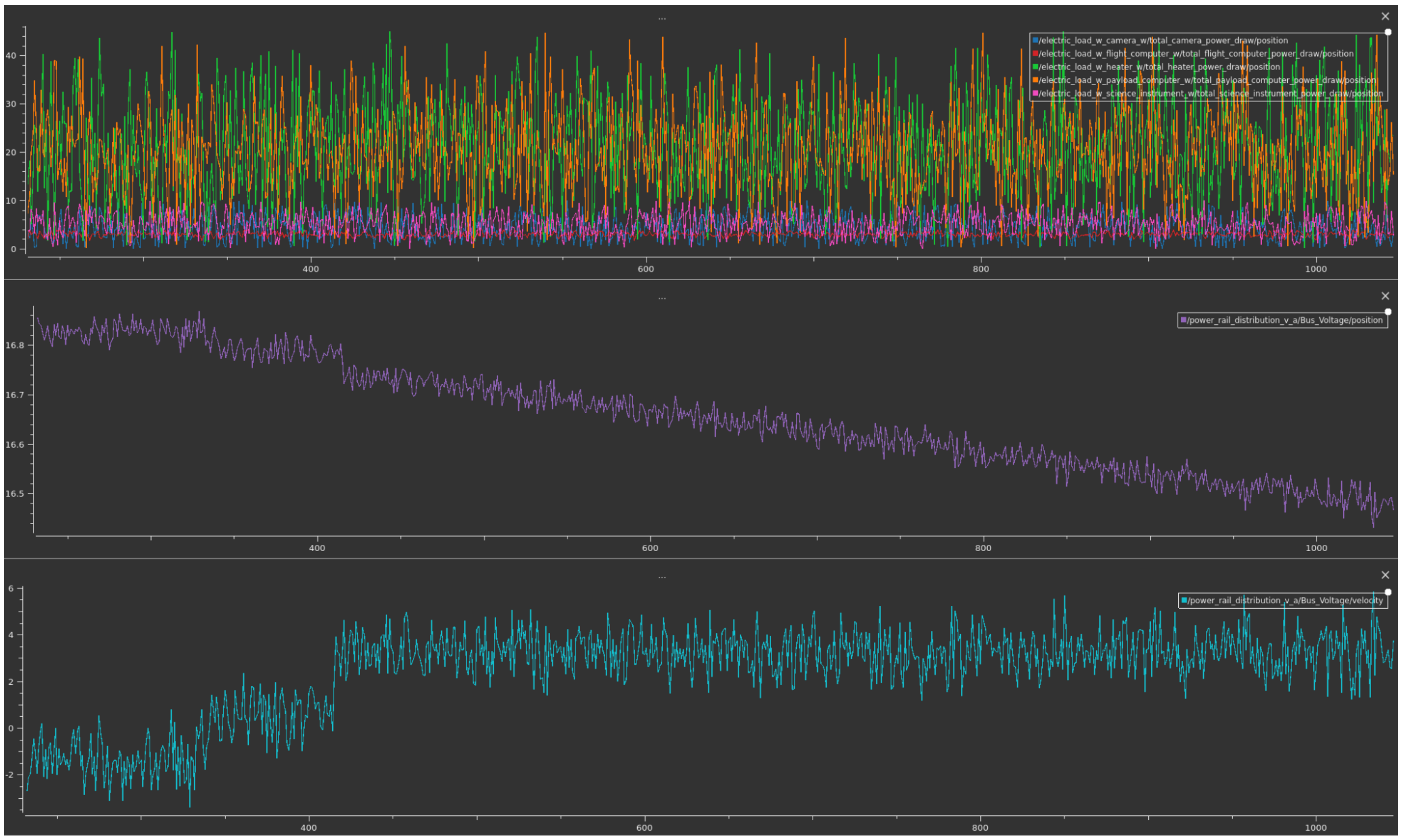
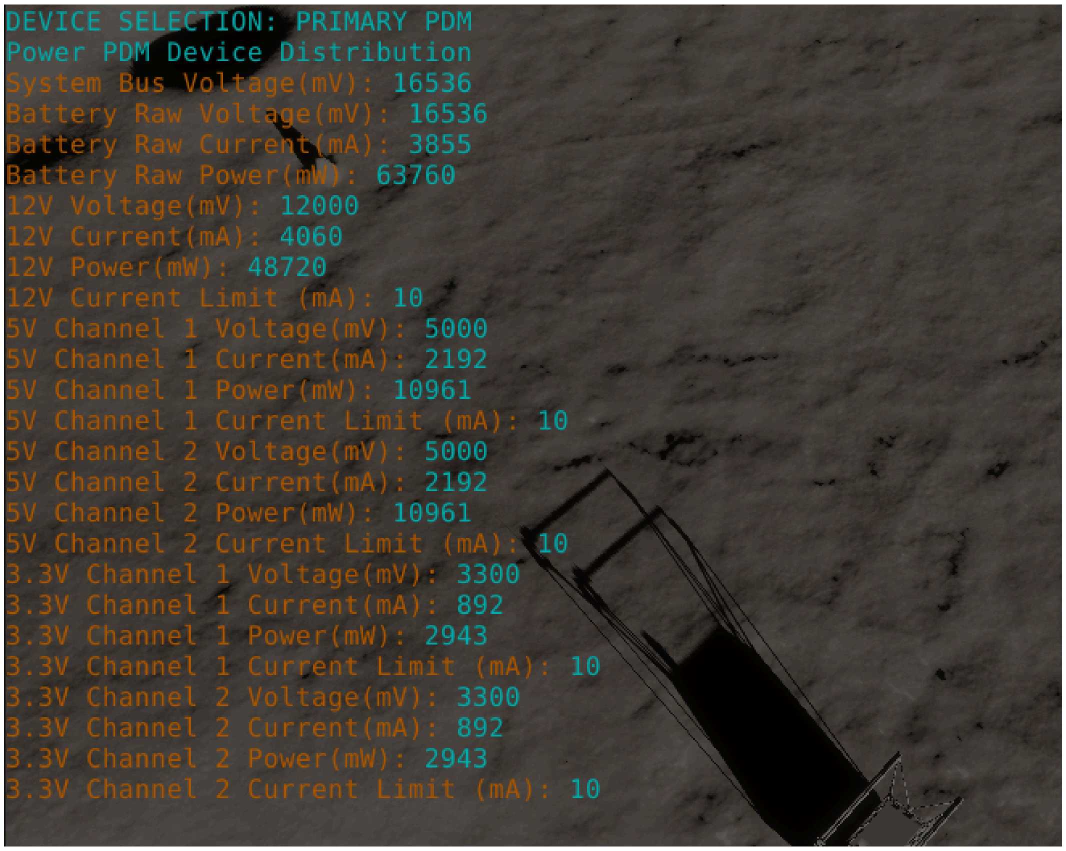


*Figure 4: Graphs of Power Draw of Electrical Loads, Voltage on Bus Voltage Rail, and Current on Bus Voltage Rail*

The top graph in Figure 4 shows the normal distribution power draw values for each of the electrical loads. The middle graph shows the voltage decreasing on the Bus Voltage rail as power continues to be drawn from the system. The bottom graph shows the current increasing on the Bus Voltage rail in relation to the power being drawn and the voltage decreasing. This is as intended because of the power equation P = IV, where P is power in Watts, I is current in Amps, and V is voltage in Volts. Since current and voltage are inversely proportional in the power equation, it is expected that as one of the variables increases, the other decreases. In our case, the voltage is decreasing while the current is increasing as expected. This meant our calculations were considered accurate. Additionally, Figure 5 below shows the values calculated for voltage, power, and current after integrating our simulations with Lunar Outpost's pre-existing dashboard.



```
DEVICE SELECTION: PRIMARY PDM
Power PDM Device Distribution
System Bus Voltage(mV): 16536
Battery Raw Voltage(mV): 16536
Battery Raw Current(mA): 3855
Battery Raw Power(mW): 63760
12V Voltage(mV): 12000
12V Current(mA): 4060
12V Power(mW): 48720
12V Current Limit (mA): 10
5V Channel 1 Voltage(mV): 5000
5V Channel 1 Current(mA): 2192
5V Channel 1 Power(mW): 10961
5V Channel 1 Current Limit (mA): 10
5V Channel 2 Voltage(mV): 5000
5V Channel 2 Current(mA): 2192
5V Channel 2 Power(mW): 10961
5V Channel 2 Current Limit (mA): 10
3.3V Channel 1 Voltage(mV): 3300
3.3V Channel 1 Current(mA): 892
3.3V Channel 1 Power(mW): 2943
3.3V Channel 1 Current Limit (mA): 10
3.3V Channel 2 Voltage(mV): 3300
3.3V Channel 2 Current(mA): 892
3.3V Channel 2 Power(mW): 2943
3.3V Channel 2 Current Limit (mA): 10
```

*Figure 5: Calculated Values for Voltage, Current, and Power on Each Rail of a Rover*

Figure 5 was the final step to verifying that our calculations and controls were working as per Lunar Outpost's requirements. In the example used in Figure 5, the rover has three voltage rails: 12V, 5V, and 3.3V. Each rail has different types of loads, which is why some of the values outputted are lower than others. The values were correct and our overall project and integration into Lunar Outpost's system were successful.

## X. Future Work

Due to time constraints, there were certain features we were unable to add but would like to implement in the future. For example, we want to build a graphical user interface (GUI) to create fault injections. This can currently be simulated via the command line but we believe having a GUI would make it easier for people without a tech background to create and manage injections to the system. Another feature that could be added to our simulation is how other factors like the torque of the tires and the incline of the terrain affect the battery capacity of the rover.

From an architectural perspective, it might be beneficial to change how the voltage rail controls are implemented in the future. This is because the flow of data is non-intuitive right now especially when trying to output fault condition codes and turn off voltage rails. Currently, there is a separate folder for fault code within the battery model but it may be better to abstract the faults to its own model since it requires data from both the battery and electrical loads models. However, this may take additional time as it would require publishing data to topics that can be subscribed to within another model. Furthermore, a new node would need to be created to encapsulate the faults.

Additionally, one thing we could revise is our documentation of methods so it is more consistent with the rest of the client's system. While we have used Doxygen for documentation and written documents like test procedures, it is crucial to keep these up to date with the client's system.

Besides adding and modifying a few features to make the user and developer experience better, we believe we have addressed all of our client's requirements and have created a simulation that works successfully.

## XI. Lessons Learned

We have learned many lessons throughout the development of this project. Below are some of our most significant takeaways.

One lesson we learned was the importance of pair programming a project when the group members have varying levels of background knowledge of the problem at hand. The benefits of pair programming are better communication and explanation of ideas, more efficient progress, and learning to work with a team. We found this to be a very useful technique for our project as all of us came into the project with very little knowledge of the ros2 framework.

Another lesson we learned was how useful it is to use a framework like ros2 which makes it much easier to simulate different aspects of robotic machines like rovers using interprocess communication. Creating nodes for our battery and loads and publishing and subscribing to topics was a lot more seamless when using the ros2 framework versus trying to simulate this from scratch using solely C++.

Additionally we learned that using a Docker container for writing code was helpful especially when all of the team members use different operating systems. By utilizing a Docker container, we were all able to have the same operating system and run into less conflicts when pulling and pushing code to our remote repository.

The fourth lesson we learned was that drawing out our ideas and plans for architecture design was beneficial because it allowed us to visualize how the different components of our system work together and how we should implement our code as a team. Abstraction was a key part of this and we found that separating the battery model from the loads model made it easier to make the necessary changes while also taking into account coupling and cohesion principles.

The final significant lesson we learned is that modern development of hardware related systems are strongly advised to use C++. The flexibility of C++ made niche features such as implementing timers and the centralization of the battery easy. While other languages have strengths, C++ was overall the best choice for our program as it clearly defines structures and paradigms that are simple to use such as instantiating objects dynamically.

## XII. Acknowledgments

Thank you to our client, Lunar Outpost, for your continued support and guidance throughout the software development process. A special thank you to Shelby O'Callaghan and Leah Valencia for helping us test our system.

Thank you to Dr. Austin Oltmanns who taught us so much about circuits and the ros2 architecture! We are grateful for your willingness to always meet with us for help and for your encouragement throughout the process. The success of our project would not have been possible without you!

Thank you to our amazing advisor Donna Bodeau for helping us become better software developers! We are thankful for your help navigating this project throughout the semester.

## XIII. Team Profile

### Musad Alam

My name is Musad Alam and I am a 4th year Computer Science major with a focus in Computer Engineering. I am from Aurora, Colorado. I am an automotive enthusiast and I have joined the Battery Workforce Challenge as a software subsystem team member to fuel my love for cars and engineering. The club is entering its 3rd and final year of competition. When I am not at school or BWC, I am skateboarding, playing video games, or looking at car parts I can't afford.

### Sara Gampher

My name is Sara Gampher, and I am a Senior in Computer Science with an emphasis in Data Science. I am also working towards my Masters in Statistics, which I will be graduating with in May 2026. I am from Olathe, Kansas and moved to Colorado to go to Mines! I love hiking, reading, playing the violin, and learning to snowboard.

## Megan Kulshekar

My name is Megan Kulshekar and I am a Senior studying Computer Science with a minor in Computational and Applied Math. I am originally from Parker, Colorado and I enjoy exploring nature in our beautiful state. On campus, I serve as a Logistics Director for Mines' Society of Women Engineers chapter. I was also the previous President of Mines ACM chapter and have participated in Human Robot Interaction (HRI) undergraduate research in the past. Outside of school, I enjoy traveling, attending hackathons, swimming, and playing the piano.

## Dishita Sharma

My name is Dishita Sharma and I am a 4th year finishing up my undergrad and pursuing my masters, both in Computer Science with a focus in AI/ML. I am originally from Colorado Springs, CO. At Mines, I have been an Undergraduate Research Fellow for the past 3 years and just transitioned to being a graduate researcher at my research lab, Hammerling Research Group. I am also a part of the Vanguard Scholar Community and Society of Women Engineers (SWE). Outside of Mines, I love yoga, swimming, snowboarding, and traveling to new places.

## References

[1] "ROS 2 Documentation — ROS 2 Documentation: Rolling documentation." Accessed: Nov. 28, 2025. [Online]. Available: https://docs.ros.org/en/rolling/index.html

[2] "GTest Framework," GeeksforGeeks. Accessed: Nov. 28, 2025. [Online]. Available: https://www.geeksforgeeks.org/software-testing/gtest-framework/

[3] "Doxygen homepage." Accessed: Nov. 28, 2025. [Online]. Available: https://www.doxygen.nl/index.html

## Appendix A – Key Terms

| **Term** | **Definition** |
|---|---|
| *ROS2* | *An open-source framework that contains software libraries and tools for building robot applications [1].* |
| *GTest* | *A framework for creating unit tests in C++ developed by Google [2].* |
| *Doxygen* | *A built-in documentation generator that can be used with various languages including C++ [3].* |