Final Design Report

Submitted to:

Matthew Muller – NREL

Bill Sekulic – NREL

Submitted by:

F22 - 78

MechE Minority

Engineering, Design, & Society

Colorado School of Mines

Golden, Colorado 80401

Logo, circle

Description automatically generated

Capstone Design@Mines

Team Members:

Steffen Damm steffendamm@mines.edu

Adam Houghtaling ahoughtaling@mines.edu

Matt Lange mlange@mines.edu

Mark McNulty markmcnulty@mines.edu

Victoria Polda vpolda@mines.edu

Zachary Steenson zsteenson@mines.edu

Faculty Adviser: Dr. Arkadan [aaarkadan@mines.edu](mailto:aaarkadan@mines.edu)

May 2nd, 2023

**Table of Contents**

|  |
| --- |
|  |
| [Project Review 3](#_Toc131064700)  [Design Approach 4](#_Toc131064701)  [Engineering Analysis 9](#_Toc131064702)  [Final Deliverables 9](#_Toc131064703)  [Project Management 9](#_Toc131064704)  [Budget 9](#_Toc131064705)  [Schedule 11](#_Toc131064706)  [Path forward 11](#_Toc131064707)  [Lessons Learned 11](#_Toc131064708)  [Appendices 11](#_Toc131064709) |

# 

# Project Review

Our project was to develop a fully functional solar tracker controller. This included motor control, remote user interface, and self-sufficient operation of the controller outside. Since the initiation of our project, many aspects of our design have changed.

Initially, we were planning on delivering a complete solution to the design problem which included motor control through a microcontroller along with a wireless connection from the microcontroller to a python-based GUI. Since then, we have revised our list of deliverables. These will be detailed later in the Final Deliverables section of this document. Below is a table showing the requirements given to us by the client. Many of these have since changed.

*Figure 1: Client Requirements*

Table

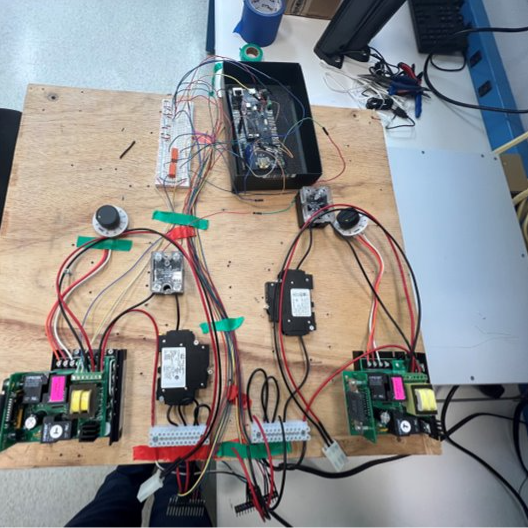
Description automatically generated

During IDR, we focused on deciding what hardware was needed. We decided on the Portenta H7 as our microcontroller of choice. This was later questioned if we should have instead gone with a Raspberry Pi but decided to stick with the Portenta due to it satisfying the project's temperature range requirement, while the Raspberry Pi does not.

During this time as well, the team needed to select the hardware that would drive the 90 VDC motors. We determined that the hardware we selected needed to drive the 90 VDC motors, allow for speed control, and allow for rotational direction changing of the motor. There were two solutions that we found satisfied these criteria a speed controller and a homemade H bridge configuration solution. Looking through each of these solutions we determined that the speed controller solution would work best for the project due to the ease of use and being less complex to deploy compared to the H bridge.

Since the IDR, we completed building of our two-axis prototype using two isolated circuits to drive motors for both azimuth and elevation. We tested each subsystem including all relays, limit switches, speed controller output, and hall sensor feedback. Basic test scripts were written to see how changing the input parameters affected the outputs and gave us an understanding of how to control each subsystem. Once this was completed, we wired each subsystem together to make a complete system and then shifted our focus to writing our software functions. The complete hardware prototype is shown in the figure below.

*Figure 2: Hardware Prototype*



From a software standpoint, we have revised our deliverables to not include fully functioning hardware but instead the framework for future developments. This included an unconnected user interface along with motor control software that allows for basic serial interaction via a command line. During IDR and PDR, running a web server utilizing JSON and HTTP on the Portenta seemed like the path forward. Our design takes this into account and is built to make integration into a design like that easier.

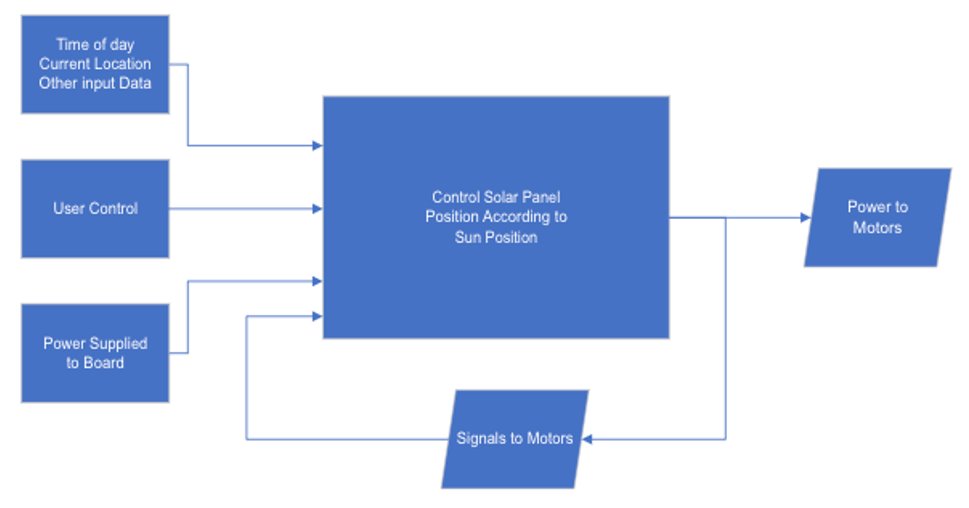
In our initial design, our controller software was expected to replicate all aspects of the previous solution. Since then, we have decided to focus on implementing only one function that has the tracker move to a specified azimuth and elevation. It features the bare bones of other functions and capabilities for future developments. This has been implemented using C++ by controlling the power of the motor until the desired location has been reached. To make this easier for future development all the motor controls have been implemented into a Motor class that gives a future user full control over motor power, motor position, and motor end-stops.

The broader impacts of our design have since changed from delivering a fully functioning solution to setting up later developments for success and ease of implementation. The scope of who the project impacts remained like our preliminary design which includes our current clients, NREL as an organization, and future senior design integrations of the project.

# Design Approach

To solve this problem, several analysis tools were used at different stages to further the design toward the product. First, since there was an existing solution to reverse engineer and analyze, the Black Box Model was used to understand the problem and the necessary inputs and outputs.

*Figure 3: “Black Box” Diagram*

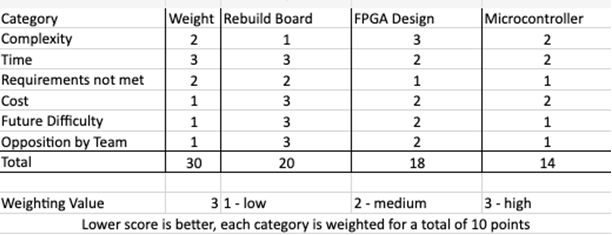
**

From this, the goal was to redesign the central box “Control Solar Panel Position According to Sun Position” while essentially maintaining the relationship between inputs, central control and outputs. The relationships would not be entirely unchanged. For example, the arrow going from “User Control” to central control would need to change in accordance with the project needs, with a new graphical user interface and remote communication. The project requirements can be seen clearly from this analysis. The system must be able to transform user control, time of day/current location data, power and signals to motors into powering the motors to make the module track the sun.

The next step was to identify the ideal hardware/software solution in order to address the project needs. For this stage of component selection and design of the assembly, decision matrices and flow diagrams were used to conceptualize the solution and select the most ideal components.

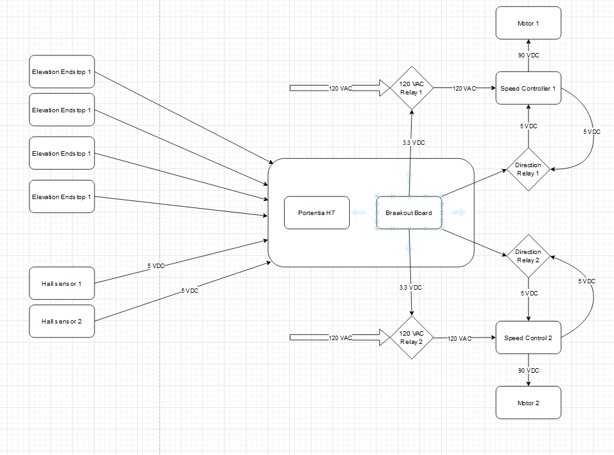
Many solutions were brainstormed, then they were narrowed down to three solutions to solve the problem. These included a board rebuild, an FPGA design and a microcontroller solution. The process was to explore each of the three solutions and consult with the client to select which of the three would be most fitting for the project. The solution selected from the three solutions was the microcontroller solution using a decision matrix.

*Figure 4: Concept Selection Decision Matrix*

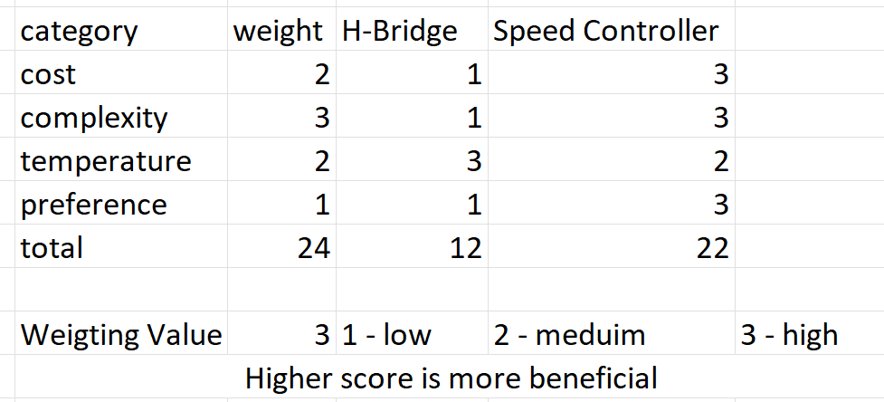
**

The exploration of this solution utilized a flow diagram and decision matrices for the components that were needed. The components were selected from potential candidates that generally fit the project requirements, but they had different benefits and drawbacks. The following figures represent that exploration.

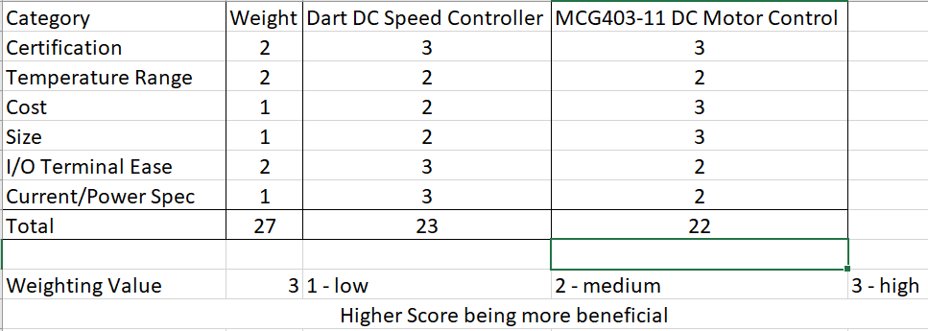
*Figure 5: Initial Flow Diagram for Design*

**

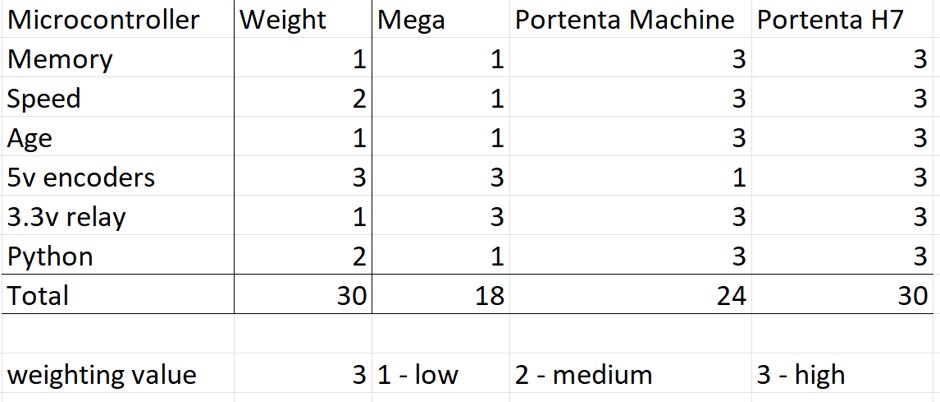
*Figure 6: Motor Drive Decision Matrix*



*Figure 7: Speed Controller Decision Matrix*

**

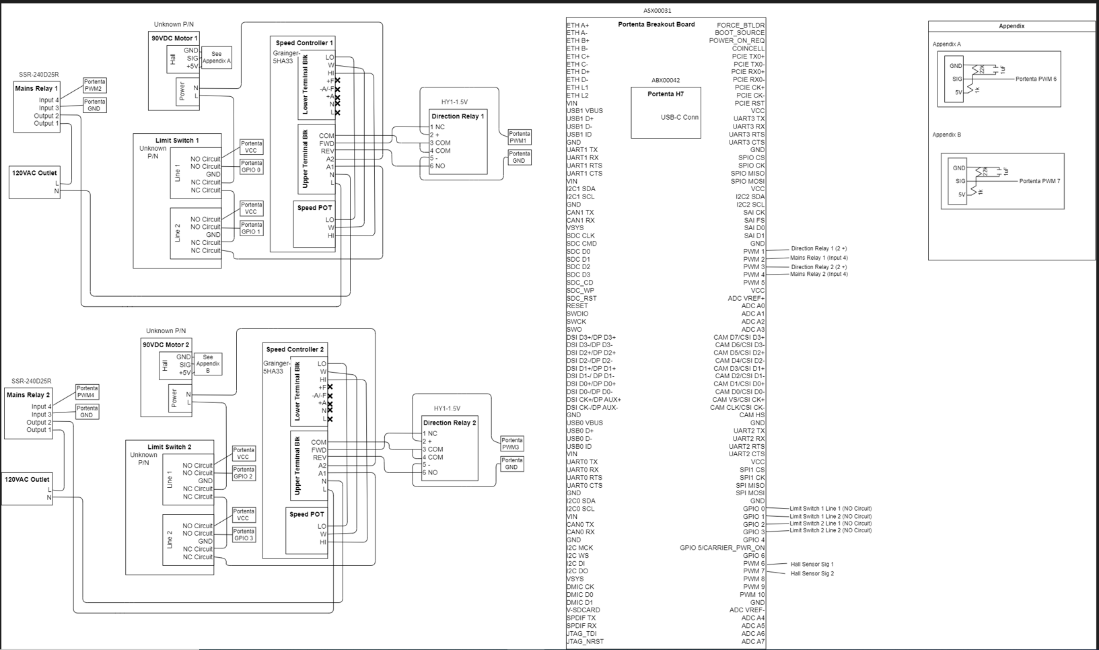
*Figure 8: Microcontroller Decision Matrix*

**

From this point, the microcontroller solution selected, the concept needed to be fully explored and fleshed out. In addition to the hardware selection, software was selected to go along with this. The software solution depended heavily on the selected hardware solution. As python was one of the client’s wants, the native python environment for the Portenta H7 controller called MicroPython was selected for microcontroller code. Additionally, Python was selected to be used for the GUI code for consistency and meeting the clients wants.

Fleshing out the microcontroller solution broke down into the hardware team, API team and front-end software team. From there, the design was iterative, receiving feedback from the clients and refining the solution. The following figures are the full circuit diagram and software architecture solution, the class diagram for the controller software, and the graphical user interface.

*Figure 9: Full Circuit Diagram*

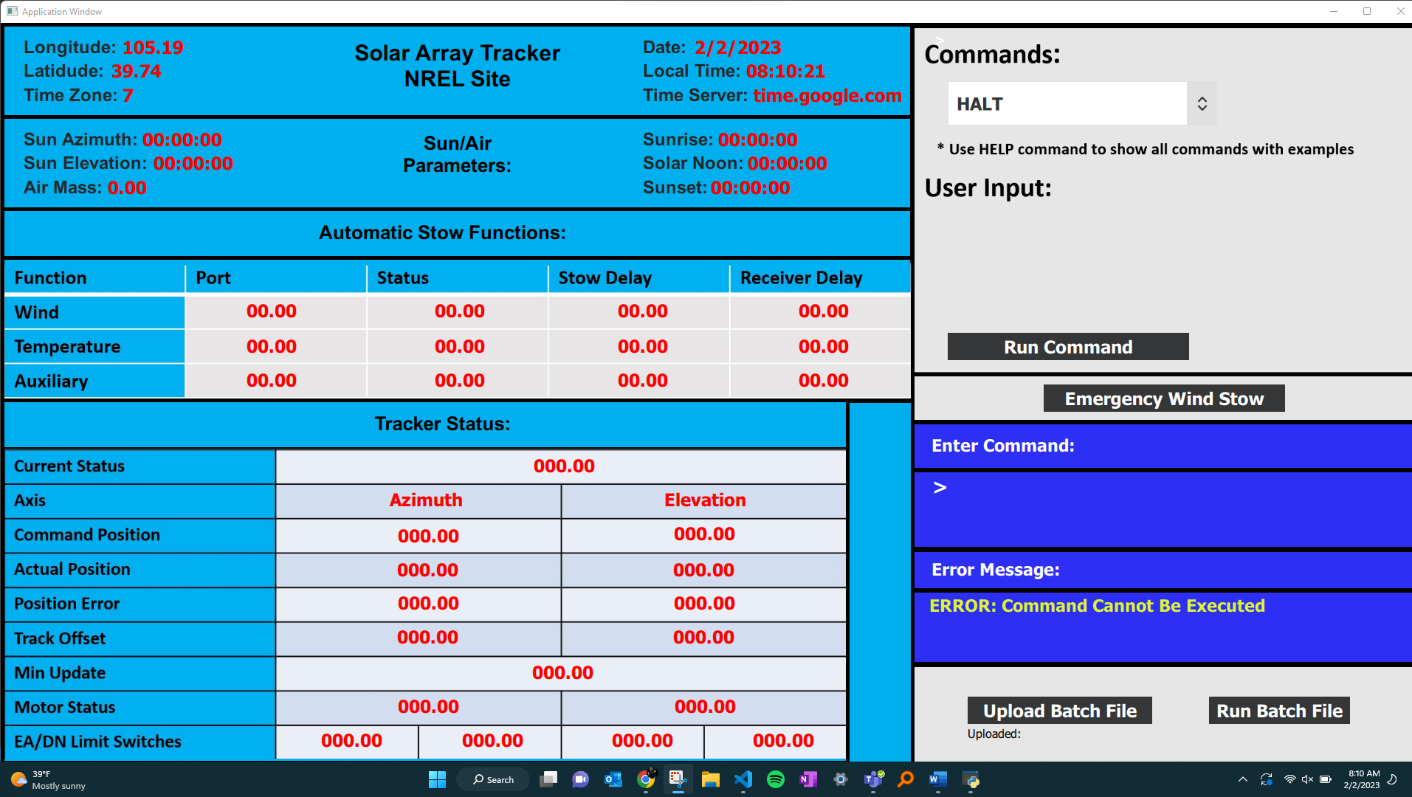


*Figure 10: Software Architecture*

Diagram

Description automatically generated

*Figure 11: Graphical User Interface*



This was translated to the physical prototype via testing, redesign and iteration.

For the API design, many different solutions were considered but due to scheduling and planning we were unable to implement them. We considered hosting the API on the web such as a server or the cloud, along with hosting it physically on the Portenta or a Raspberry Pi. We ended up hosting it on the other, slower core, of the Portenta as this would maximize the utilization of our resources.

Many different solutions were explored, like what data format to have the API communicate with the GUI and Controller software. The main options were JSON, MessagePack, and writing our own custom package.

The diagram below shows a sample interaction between the cores and assuming the user inputs the command “stow”.

*Figure 12: Core Interaction*

Diagram

Description automatically generated

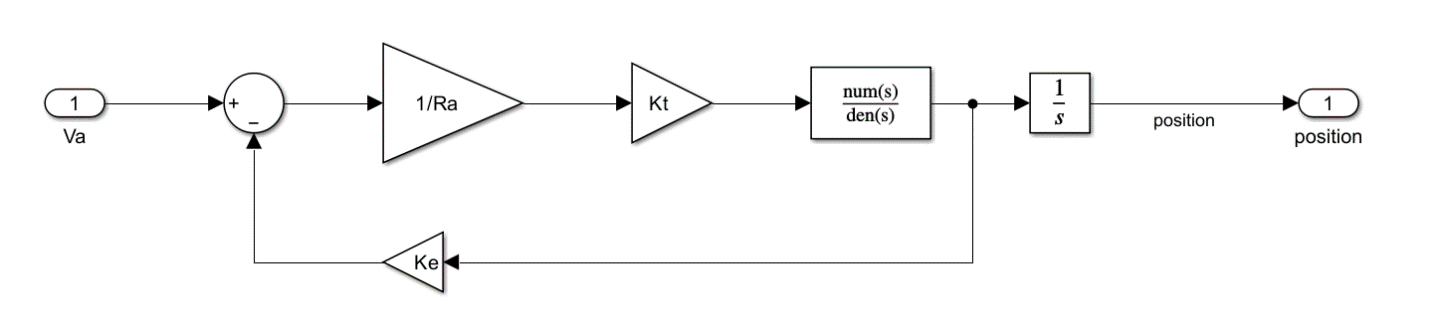
The API also delt with how often to update the GUI with data from the controller. It was decided with the client that every one second was enough. This raised issues of how to have the Portenta accept commands while also printing data.

In the end, the API ended up consisting of just a serial interface with the M7 core of the Portenta which could then communicate with the slower M4 to read data. This data was then printed to the terminal for display. The data could be stopped to allow for new commands to be inputted.

# Engineering Analysis

To simulate the two-axis solar tracker, a Simulink file was created to reflect our design. The file takes a sample PWM input and graphs the voltage output. Along with PWM signal, the model also requires the torque constant (Kt), the armature resistance (Ra) and the voltage EMF constant (Ke). The simulation would model the given motor given it receives a PWM input and the voltage output would simulate the hall sensor readings. Upon further investigation, these parameters were not available due to the current motor’s age with no method of documentation. A potential solution would be to replace the tracker’s motors with updated motor hardware with new documentation that provides the necessary constants. Nevertheless, this simulation may yield useful results for implementation in the future but cannot be implemented with the current motors.

*Figure 13: Simulink Model for Motor Analysis*



While constructing our test apparatus there were a few issues that the team ran into. One of these issues had to do with the output voltage of the hall sensors attached to the motors not equaling the initial estimate based on the power supplied to the system. This took some considerable testing to determine the source of error. Consulting the team’s technical advisor, Christopher Coulston, the team determined that there was an internal resistance with our level shifter causing a voltage divider circuit. This resulted in a lower unusable voltage from the hall sensors.

To solve this problem our technical advisor advised using an external resistance to step down the voltage rather than using the level shifter. To determine the value of the resistor, an internal resistance needed to be calculated to determine the internal resistance of the circuit to correctly determine the external resistor to use to output the desired voltage for our microcontroller.

To accomplish this, we ran an experiment using different external resistor values. Then we used the voltage divider equation, equation 1 below. To see how the output voltage changed based on the external resistance.

𝐸𝑞𝑢𝑎𝑡𝑖𝑜𝑛 1: 𝑉𝑜=𝑉𝑠∗𝑅𝑒𝑥𝑅𝑒𝑥+𝑅𝑖𝑛

Moving the equation around you can get an equation for the internal resistance based on the external resistance, source voltage, and output voltage, equation 2.

𝐸𝑞𝑢𝑎𝑡𝑖𝑜𝑛 2:𝑅𝑖𝑛=𝑅𝑒𝑥(𝑉𝑠𝑉𝑜−1)

Given these two equations we then run the experiment to calculate the internal resistance of the hall sensor. Applying equation 2 we calculate the internal resistance of the hall sensor.

*Figure 14: Hall Sensor Internal Resistance Experiment*

Text

Description automatically generated

Looking at the data we can see that the internal resistance of almost all the cases runs right around 9 kiloohms, however there are two data points that look like outliers so we will exclude them in our averaging. We can see that the average internal resistance is 9.14 kiloohms. Based on the desired output voltage of 3.3 VDC and the average internal resistance we can calculate the external resistance needed to compute this, 20.1 kiloohms. We can also fact check this number with one of our experiments seeing the output of 3.31 VDC and an external resistance used of 21.68 kiloohms to achieve this. Given this we can confidently say that the internal resistance of the hall sensor is 9.1 kiloohms.

Preliminary Drawings- Circuit Design

*Figure 15: Circuit Diagram*

Diagram, schematic

Description automatically generated

At this point in the project the circuit diagram of the overall system looks nearly identical to the one presented at the end of last semester. However, our understanding of the end-stops changed slightly. The team assumed that there was only one trigger position of the end-stops, while there are two. The initial trigger is a normally open switch that takes the 3.3V rail and connects it to a GPIO pin when triggered. This will tell the controller that the end-stop has been reached and to stop moving. The second trigger position that we were informed about is a fail-safe that will cut off the power to the motor, halting movement. This is accomplished by a normally closed switch that the motor's power gets routed through. Both end-stops are connected in series with the motor power so when this fail-safe position is triggered, the normally closed switch then becomes open stopping the power delivery to the motor.

UML class Diagram of Controller

*Figure 16: Control Class Structure*

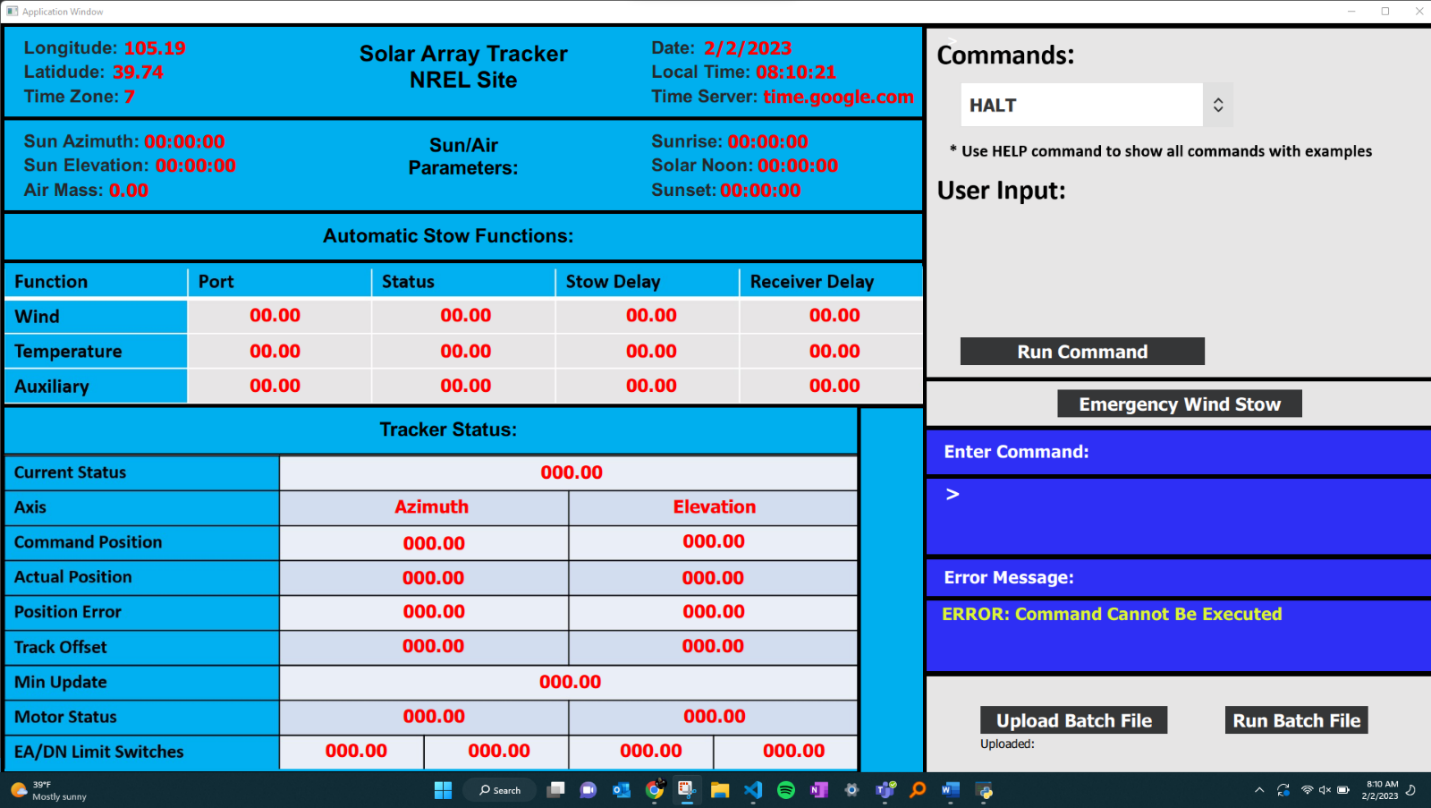
Diagram

Description automatically generated

The team coded the hardware to begin making the system. This requires the team to have a good understanding of how the architecture of the system will interact to efficiently build the code system from the beginning. To accomplish this the team used object-oriented programming to construct a class object of every physical entity on the tracker: hall sensor, end-stops, and motors. Each motor will inherit two end stops and a hall sensor to track its position. Then the overall tracker will have two motor classes which will describe the azimuth and elevation motor of the tracker and SunPot which will get the positional data of the sun.

Current GUI

*Figure 17: GUI Design*



At this point, the graphical user interface can be built and run successfully from its QML file without errors. The output variables shown in red are now capable of being set as any python variable in the python wrapper around the QML file. This allows the python program to run on a loop and continually update variables. The time and date variables are examples that were demonstrated to work and can be updated as frequently as desired. Additionally, the buttons, dropdown and commands can now be set to any python function. This allows the buttons, dropdown, and typed commands to fully operate the controller by communicating commands to the API.

Sample API interaction with GUI and Controller

*Figure 18: Advanced API design*

Diagram

Description automatically generated This is an outline of the interaction between our project’s GUI, the API and the controller software. The user can interact with the GUI to initiate function calls or view data. These include the functions from the original FORTRAN code such as STOW and HALT. Then, the GUI will send an HTTP request to the API via a wired connection. An initial http packet will be sent from the GUI through the API to the controller. This will return a status code notifying the user if there are any issues with the connection or if the controller is unresponsive. After this initial packet, it will then send whatever request the user has or simply return current data from the tracker. This will be in JSON data format and feature what state and parameters need to be updated, and all of the data collected and calculated from the tracker as well as any errors. This packet will travel from the GUI to the API hosted on Core 2 of the Portenta. The API will then parse the data and send it forward to the controller running on Core 1 of the Portenta via the hardware between the cores.

The GUI will be continually polling every 1 second to the API and controller, asking for new data to be displayed. The controller will send this data using interrupts over to the API for the API to then send it to the GUI for display. This will also allow for logging, which will be discussed with the client and decided on. Once the client disconnects, the API shall take note of this and handle the data appropriately. The controller and API shall continue to run regardless of if the GUI is or not.  This will not be implemented by the team at this time. This is more of a sample reference for future developments.

The current feasible design of the API was made consisting of a user interface via serial connection to the Portenta. This interface will allow user input of different functions and parameters such as Azimuth and Elevation. This will then be handled by Core 2 (M4) of the Portenta and sent to the controller software Core 1 (M7). Core 1 will send back all data, errors, and the current state every second. Core 2 will then display this in the terminal log for viewing.

# Final Deliverables

Regarding the scope of the Two-Axis Tracking project, there were a few deliverables that the team aimed to produce for the clients and future teams. Firstly, the team produced a full schematic for the system. This includes a high-level flow chart for the microcontroller, speed controller, and hall sensors inputs and a lower-level pin layout describing the wiring to the Portenta H7. Along with this, the team delivered base movement functionality with two-axis monitoring that tracker’s the solar position. Additionally, a GUI was coded that displays the status of the tracker, including the type of function being sent with a dropdown menu and text command format, actual position, error, the azimuth and elevation of the panel, and the current date and time. With this, the team also demonstrated communication between the two cores on the H7 microcontroller to efficiently allocate the memory. Finally, a serial output program was made showing basic commands for the tracker with user input.

# Project Management

## Budget

Below is our initial budget compared to our final. The total cost is quite different. We decided on a different microcontroller. We also realized we do not need a power supply as there are some on site. Since our initial estimate, a more accurate site enclosure for mounting and wiring has been done as seen below.

*Figure 19: Initial Budget*Table

Description automatically generated

*Figure 20: Final Budget*

Table

Description automatically generated

*Figure 21: Work Breakdown Structure*Diagram

Description automatically generated

Schedule

Our schedule over the last few weeks is shown below.

*Figure 22: Schedule*

Chart

Description automatically generated

## Path Forward

With the last few weeks of the project fast approaching, we will continue to develop hardware, software, and documentation. Documentation for usage and future developments will be made. A working software and hardware model will be done that can do a basic test run. We will work on communicating with the client to ensure good design choices are made and approved of.

# Lessons Learned

The Solar Tracker Project has allowed for many developments in the team, both professionally and technically.

The most significant are those such as communication within the team but also with a client. Initially, we underestimated how much communication was needed. This was a great lesson as communication is a vital part of anyone’s career development.

In terms of the design process, something we would do differently is try to explore more options early on before committing. This involves communicating not only with the client but also with our Technical Advisor and other resources.

Some aspects of the project were significantly more difficult than expected. This was seen primarily with the software. The communication between the GUI and control software proved to be more difficult than expected.

**Appendix**

As agreed, all important documents such as schematic, GUI, and controller code are in the Github repositories listed below:

<https://github.com/zsteenson/GUI-two-axis-tracker-controller.git>

<https://github.com/zsteenson/two-axis-tracker-controller-code.git>