ECE 44X Capstone Document

Group 10 - Lost Without Direction

Using GPS to improve communications and tracking for the OSU Robotics Club Mars Rover

Team Members

Angel Huang
Kira Kopcho
Sean Bullis
Austin Grubowski

Table of Contents

1. Overview.......... 7

1.1. Executive Summary.......... 7

1.2. Team Contacts and Protocols.......... 7

1.3. Gap Analysis.......... 8

1.4. Timeline.......... 9

1.5. References and File Links.......... 9

1.5.1. References.......... 9

1.5.2. File Links.......... 9

1.6. Revision Table.......... 9

2. Impacts and Risks.......... 11

2.1. Design Impact Statement.......... 11

2.2. Risks.......... 12

2.3. References and File Links.......... 14

2.4. Revision Table.......... 14

3. Top-Level Architecture.......... 16

3.1. Block Diagram.......... 16

3.2. Block Descriptions.......... 16

3.3. Interface Definitions.......... 20

3.4. References and File Links.......... 23

3.5. Revision Table.......... 23

4. Block Validations.......... 24

4.1. Tracking Algorithm.......... 24

4.1.1. Description.......... 24

4.1.2. Design.......... 24

4.1.3. General Validation.......... 26

4.1.4. Interface Validation.......... 27

4.1.5. Verification Process.......... 31

4.1.6. References and Files Links.......... 32

4.1.7. Revision Table.......... 32

4.2. Rover GPS.......... 34

4.2.1. Description.......... 34

4.2.2. Design.......... 34

4.2.3. General Validation.......... 37

4.2.4. Interface Validation.......... 37

4.2.5. Verification Process.......... 39

4.2.6. Revision Table.......... 39

4.3. Transmitter Hardware.......... 40

4.3.1. Description..40
4.3.2. Design...40
4.3.3. General Validation..44
4.3.4. Interface Validation.. 45
4.3.5. Verification Process... 50
4.3.6. References and Files Links...52
4.3.7. Revision Table..52
4.4. Receiver Hardware...54
4.4.1. Description..54
4.4.2. Design...54
4.4.3. General Validation..55
4.4.4. Interface Validation.. 55
4.4.5. Verification Process... 58
4.4.6. References and Files Links...59
4.4.7. Revision Table..59
4.5. Turntable Controller... 60
4.5.1. Description..60
4.5.2. Design...60
a) Interfaces and Black Box..60
b) Design Documents.. 61
4.5.3. General Validation..69
a) TB6600 Sub-PCB.. 70
b) Arduino Carrier Board..71
c) 5V, 9V Regulator..71
d) UART Receiver..71
e) RS422 Transceiver.. 72
f) Clock Generation.. 72
4.5.4. Interface Validation.. 72
4.5.5. Verification Process... 77
4.5.6. References and Files Links...80
4.5.7. Revision Table..81
4.6. Turntable... 82
4.6.1. Description..82
4.6.2. Design...83
4.6.3. General Validation..86
4.6.4. Interface Validation.. 88
4.6.5. Verification Process... 90
4.6.6. References and Files Links...92
4.6.7. Revision Table..93
4.7. User Interface.. 93
4.7.1. Description..93

4.7.2. Design....94
4.7.3. General Validation....95
4.7.4. Interface Validation.... 96
4.7.5. Verification Process.... 99
4.7.6. References and Files Links....100
4.7.7. Revision Table....100
4.8. RX Code.... 101
4.8.1. Description....101
4.8.2. Design....101
4.8.3. General Validation....103
4.8.4. Interface Validation.... 103
4.8.5. Verification Process.... 104
4.8.6. References and Files Links....104
4.8.7. Revision Table....105
4.9. TX Code.... 105
4.9.1. Description....105
4.9.2. Design....106
4.9.3. General Validation....106
4.9.4. Interface Validation.... 106
4.9.5. Verification Process.... 107
4.9.6. References and Files Links....108
4.9.7. Revision Table....108
5. System Verification Evidence.... 109
5.1. Universal Constraints.... 109
5.1.1. The system may not include a breadboard....109
5.1.2. The final system must contain a student designed PCB....109
5.1.3. All connections to PCBs must use connectors....111
5.1.4. All power supplies in the system must be at least 65% efficient....113
5.1.5. The system may be no more than 50% built from purchased 'modules.'....114
5.2. Requirements.... 115
5.2.1. System Accuracy.... 115
5.2.1.1. Project Partner Requirement....115
5.2.1.2. Engineering Requirement.... 115
5.2.1.3. Testing Method....115
5.2.1.4. Verification Process....116
5.2.1.5. Pass Condition.... 116
5.2.1.6. Testing Evidence.... 116
5.2.2. System Tracking Speed.... 116
5.2.2.1. Project Partner Requirement....116
5.2.2.2. Engineering Requirement.... 116
5.2.2.3. Testing Method....116

5.2.2.4. Verification Process....116
5.2.2.5. Pass Condition....117
5.2.2.6. Testing Evidence....117
5.2.3. System Field of View....117
5.2.3.1. Project Partner Requirement....117
5.2.3.2. Engineering Requirement....117
5.2.3.3. Testing Method....117
5.2.3.4. Verification Process....117
5.2.3.5. Pass Condition....117
5.2.3.6. Testing Evidence....118
5.2.4. Noise Tolerance....118
5.2.4.1. Project Partner Requirement....118
5.2.4.2. Engineering Requirement....118
5.2.4.3. Testing Method....118
5.2.4.4. Verification Process....118
5.2.4.5. Pass Condition....118
5.2.4.6. Testing Evidence....118
5.2.5. System Isolation....118
5.2.5.1. Project Partner Requirement....118
5.2.5.2. Engineering Requirement....118
5.2.5.3. Testing Method....118
5.2.5.4. Verification Process....119
5.2.5.5. Pass Condition....119
5.2.5.6. Testing Evidence....119
5.2.6. System Portability....119
5.2.6.1. Project Partner Requirement....119
5.2.6.2. Engineering Requirement....119
5.2.6.3. Testing Method....119
5.2.6.4. Verification Process....119
5.2.6.5. Pass Condition....119
5.2.6.6. Testing Evidence....120
5.2.7. System Setup....120
5.2.7.1. Project Partner Requirement....120
5.2.7.2. Engineering Requirement....120
5.2.7.3. Testing Method....120
5.2.7.4. Verification Process....120
5.2.7.5. Pass Condition....120
5.2.7.6. Testing Evidence....120
5.2.8. System Update Rate....121
5.2.8.1. Project Partner Requirement....121
5.2.8.2. Engineering Requirement....121

5.2.8.3. Testing Method.....121
5.2.8.4. Verification Process.....121
5.2.8.5. Pass Condition.....121
5.2.8.6. Testing Evidence.....121
5.3. References and File Links.....121
5.4. Revision Table.....121
6. Project Closing.....123
6.1. Future Recommendations.....123
6.1.1. Technical Recommendations.....123
6.1.2. Global Impact Recommendations.....123
6.1.3. Teamwork Impact Recommendations.....124
6.2. Project Artifact Summaries with Links.....124
6.3. Presentation Materials.....125
Appendix.....126
File Links.....126
Code Snippets.....128

1. Overview

1.1. Executive Summary

While tracking exploration vehicles on Mars, the rover must have a communication link to a ground station to send and receive data signals to and from the rover. When the rover moves too far away from the ground station or when the line of sight becomes obstructed the ability to receive these signals can be lost. This project is aimed toward improving the communication link of the Oregon State University Robotics Club's Mars Rover by implementing a radio frequency communication link between the ground station and the rover whose sole purpose is to optimize the data signal strength. These improvements will be tested at the Canadian International Rover Challenge.

The project will incorporate a transmitter on the rover which will transmit a signal on a frequency of 915MHz using the LoRa protocol. Originally, the plan was to use an antenna array which would be multiplexed with a microcontroller development board to generate a pseudo-doppler effect to determine the direction of the beacon signal [1]. Upon further research, it was determined that this was not going to be a viable option for directional finding. We have decided to replace the pseudo-doppler antenna with a small GPS module which will send data to the transmitter on the rover. The transmitter on the rover will then broadcast the GPS data to the ground station. The ground station will also have a GPS module with a fixed location. The difference in the coordinates will determine the heading of the rover relative to the ground station.

The transceiver itself will use an RP2040 microcontroller and a LoRa based modem module to send and receive data to and from the rover and base station. A GPS module mounted on the transceiver will pick up location information from the satellite and send it to the microcontroller via the PCB. The PCB itself will have a USB connection that powers it at 5V, and the receiver will have a communication line to the base station. The RP2040 will be supported by various peripherals including a clock, flash memory and a voltage regulator. In the case of rover failure a battery backup system is implemented. This is done with a power mux and a 5V boost converter connected to a 3.7V LiPo battery.

1.2. Team Contacts and Protocols

The team will primarily communicate using the emails listed below in addition to in-person or virtual meetings.

Sean Bullis - bulliss@oregonstate.edu
Kira Kopcho - kopchok@oregonstate.edu
Austin Grubowski - grubowsa@oregonstate.edu
Angel Huang - huangang@oregonstate.edu

Below is the team protocols table.

Topic	Protocol	Standard
Task Management	Team will use Jira and an Agile Development structure when organizing tasks	At each team meeting we will review the kanban board and assess where each team member is on their tasks. When working on a task, it should be in the “In Progress” column. When complete, it should be in the “Done” column
Team meetings	Team will meet at least bi-weekly, once in class and once outside of class	Meeting formats outside of class will be held on Discord during Fall Term. As we move into the design and prototype phases, it is expected that team members can attend in person meetings.
Mars Rover Meetings	Team members will take advantage of Mars Rover workdays	While joining Mars Rover isn’t a requirement, the lab space is open for members of the team to work in on Saturdays from 10AM - 4PM. As we move into the manufacturing phase, members can utilize that time as they see fit
Documentation	All documentation should be stored in the RDF Capstone folder in the Google Drive	Documents should be placed in the Google Drive in a timely manner once they’re created. Each team member is responsible for keeping the folder organized

Table 1.2.1 - Team Protocols

1.3. Gap Analysis

The purpose of this project is to enhance and upgrade the communications capabilities of the Oregon State University Robotics Club's Rover through the creation of an antenna suite that can accurately track the rover and facilitate high gain communications between the rover and the base station. In order to improve communications, our plan is to send GPS data via LoRa packets to a receiver on the ground station [2]. Based on the difference in coordinates of where the antenna is located and where the rover is, we will rotate our antenna to match where the signal is coming from. An individual in our group will create a custom PCB to act as the rover transmitter. There will also be a similar receiver module on the antenna mount. The current communications equipment lacks the ability to maintain contact when the rover is traversing hilly terrain. Our communications and tracking suite will be used by the OSU Robotics Club Rover team in future competitions.

1.4. Timeline

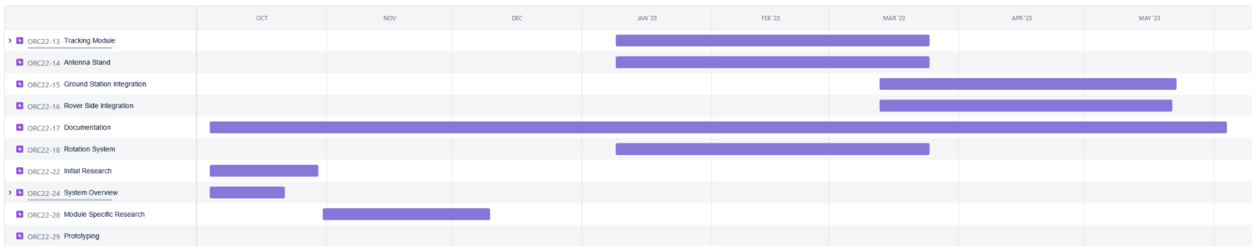


Figure 1.4.1 - Jira Timeline

1.5. References and File Links

1.5.1. References

[1] W. Hofman. “Whats a (psuedo-) Doppler Radio Direction Finder”. PA8W Amateur Radio. http://www.paluidsprekers.nl/pa8w/dopplerRDF.html (Accessed Nov. 4, 2022)
[2] J. Halstead. “LoRa Localization” Link Labs. https://www.link-labs.com/blog/lora-localization (Accessed Nov. 11 2022)

1.5.2. File Links

1) Jira Timeline

1.6. Revision Table

Author	Date	Description
Sean Bullis	10/14/22	Section Creation
Austin Grubowski	10/14/22	Added Gap Analysis
Kira Kopcho	10/14/22	Add Timeline and Protocols Table
Austin Grubowski	11/4/22	Updated Gap Analysis
Kira Kopcho	11/4/22	Minor revisions to executive summary and added references and link to timeline
Sean Bullis, Austin Grubowski, Kira Kopcho, Angel Huang	11/18/22	Revision to latest developments Clarification of GPS and LoRA radio system

		Added link to Jira project site Added references Added datasheet links to appendix
Sean Bullis	1/25/23	Updated Executive Summary and added section about the transceiver hardware
Sean Bullis	5/1/23	Minor changes to make paragraphs more concise
Angel Huang	5/14/23	Added image captions Updated/fixed formatting

2. Impacts and Risks

2.1. Design Impact Statement

a) Public Safety and Welfare

The rover tracking system has been designed with these facts in mind. First and foremost, the design must abide by United States and Canada radio regulations. The LoRa radio modules used are low power (10 mA of typical current) and transmit on the 900 MHz band which is a common commercial and industrial frequency [2]. We foresee no health issues from the antenna's radiant energy given the prevalence of the 900 MHz band and low-power components in use.

For licensing purposes, LoRa radios are considered frequency hopping devices which means they are subject to certain transmission requirements. In the United State, the LoRa modems are compliant under FCC Part 15.247 which specifies that frequency hopping devices in the 902-928MHz band shall use no less than 50 frequencies and the maximum time they can transmit on each frequency cannot be more than 0.4 seconds in a 10 second period [3].

In Canada, LoRa radios fall under RSS-247 which is similar to the US FCC specification except that the limit for timing is no more than 0.4 seconds of transmission per frequency in a 20 seconds period [4]. Devices under RSS-247 and Part 15 are considered license exempt devices meaning that individuals operating them do not need to hold a special license [1][4]. Because of this, the legality of the use of these LoRa radios is not an issue.

b) Cultural and Societal

Currently the 900 MHz band is used for amateur TV and radio, along with other small profile communication systems such as baby monitors, RFID readers, and SCADA (data acquisition) systems [5]. The low transmission power this project uses will only occur during competition tasks so any interference generated will be quick, 1 - 1.5 hour maximum, and minimal, following the inverse-square law. Additionally, this 900 MHz band hosts many non-essential systems so interference should not be affecting life or safety systems. Accordingly, the societal impact of the radio interference of this project is minimal.

c) Environmental

Waste generated during the fabrication of this project has been considered, especially regarding the enclosure which is 3D printed using PLA (Polylactic Acid). PLA is non-toxic to biological organisms and is biocompatible and will naturally degrade into lactic acid. But, it should be industrially composted or recycled [6]. Still, we see no significant impact from the use of PLA on the environment.

Other waste such as aluminum or steel structural elements can be disposed of through metal recycling programs provided by Oregon or OSU. Similarly, electrical components can be safely recycled/properly disposed of through the same programs.

d) Economic

Lastly, the economic state of the semiconductor industry and supply is concerning and this project is, of course, going to use semiconductor devices. The current World Semiconductor Trade Statistics [7] indicates the value of various semiconductors to be in the billions of USD:

the smallest market, sensors, amounts to > 22 billion. Using this as a metric of device production, we do not anticipate this project having an adverse effect on the semiconductor supply due to the small scale of parts, dozens at most, required.

2.2. Risks

Table 2.2.1 outlines the anticipated risks for the project. Each category has a risk assessment indicating the severity of the risk toward the completion of the project and indicators for identifying when the risk may have higher likelihood of occuring.

ID	Risk	Category	Probability	Impact	Performance Indicator	Action
1	Team Member Absence	Organizatio n	Medium	Medium	Short notice of absence / Informed by Member	Determine period of absence Transfer time critical work
2	Severe Team Member Illness	Public Safety	Low	High	Informed by Member	Redistribute work Rescope and reprioritize tasks Follow up with Instructors
3	Battery / Wire / PCB Fire	Safety	Low	High	Fire!	Acquire electric fire rated extinguisher Extinguish Treat injuries / 911 if needed
4	Antenna Collapse	Public Safety	Low	High	Mount on Ground / Tower Collapse	Provide medical assistance Rebuild antenna if still serviceable Follow up with CIRC officials Treat injuries / 911 if needed
5	Loose Clothes or Body Parts Caught in Mechanism	Public Safety	Low	Medium	Mechanism Jam / Calls for Assistance	Disable antenna Reverse mechanism(s) to release foreign object Treat injuries /

						911 if needed
6	Part Shortage	Technical	Medium	High	Out of Stock	Redesign / find replacement
7	Shipping Delay	Technical	Medium	Medium	Behind Deadline / Delivery Date	Communicate with Team Members / Instructors Attempt to change to expedited shipping
8	PCB Delay	Technical	Medium	High	Behind Deadline / Delivery Date	Communicate with Team Members / Instructors Attempt to change to expedited shipping
9	Lack of Electrical / Mechanical Support on Rover	Technical	Low	Med	Lack of Mounting / Battery Power Fuse Box Full	Work with OSURC Rover Club Reallocate Rover resources Purchase / fabricate additional resources
10	Software Incompatibility	Technical	Med	Med	Version / Dependency Mismatch	Research replacement software package(s) Implement
11	Weather Damage	Environme nt	High	High	Mechanism Failure / Misbehavior	Identify failure point(s) Attempt repair Replace with spare if needed
12	Transport / Setup Damage (Significant Emotional Event)	Environme nt	Low	High	Mechanism Damage	Identify failure point(s) Attempt repair Replace with spare if needed

13	Significant Tracking Inaccuracies	Technical	Med	High	Visual Misalignment	Recalibrate / Reboot Check antenna mount

Table 2.2.1 - Risk and Mitigation Table

2.3. References and File Links

2.3.1. References

[1] Canadian Space and Technology Advancement Group. “CIRC 2023 Summer Rules and Guidelines- Communications.” circ.cstag.ca. https://circ.cstag.ca/2023/rules/ (accessed Nov. 3, 2022).
[2] SX1276/77/78/79 - 137MHz to 1020MHz Low Power Long Range Transceiver, SX1276, Rev. 7, Semtech, 2020. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-connect/sx1276#diagrams
[3] Operation within the bands 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz, 47 CFR 15.247, Federal Communications Commission, Apr. 1989. [Online] Available: https://www.law.cornell.edu/cfr/text/47/15.247
[4] Digital Transmission Systems (DTSs), Frequency Hopping Systems (FHSs) and Licence-Exempt Local Area Network (LE-LAN) Devices, RSS-247, Industry Canada, Feb. 2017. [Online]. Available: https://ised-isde.canada.ca/site/spectrum-management-telecommunications/en/devices-and-equipment/radio-equipment-standards/radio-standards-specifications-rss/rss-247-digital-transmission-systems-dtss-frequency-hopping-systems-fhss-and-licence-exempt-local
[5] A. Milne, “Let's talk about 900,” RF Spectrum Tools and Antennas for Wireless Microphones. [Online]. Available: https://www.rfvenue.com/blog/2014/12/14/lets-talk-about-900. (accessed Nov. 4, 2022).
[6] DeStefano, Vincent, et al. “Applications of PLA in Modern Medicine.” Engineered Regeneration, The Authors. Publishing Services by Elsevier B.V. on Behalf of KeAi Communications Co. Ltd., 2020, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7474829/ (accessed Nov. 4, 2022).
[7] H. Kevin, L. Elise, T. Akemi, C. Gabriel, B. Katherine, “WSTS Semiconductor Market Forecast August 2022.” Accessed: Nov. 4 2022. [Online]. Available: https://www.wsts.org/esraCMS/extension/media/f/WST/5636/WSTS_nr-2022_08.pdf

2.3.2. File Links

2.4. Revision Table

Author	**Date**	**Description**
Sean Bullis	11/03/2022	Added risk table
Austin Grubowski	11/04/2022	Added references, file links, and

		revision table sections
Sean Bullis	11/04/2022	Filled out Risk Table
Angel Huang	11/18/2022	Expanded Risk Table Organized by risk
Sean Bullis	4/28/2023	Added Design Impact Statement Section and content
Angel Huang	5/14/23	Updated/fixed formatting

3. Top-Level Architecture

3.1. Block Diagram

This section showcases the project system at a high level using two different diagrams. The first diagram is a black box diagram that lists all the interfaces of our system that communicate with the outside environment. The second diagram is a system wide block diagram that shows how each block of our project communicates with each other.

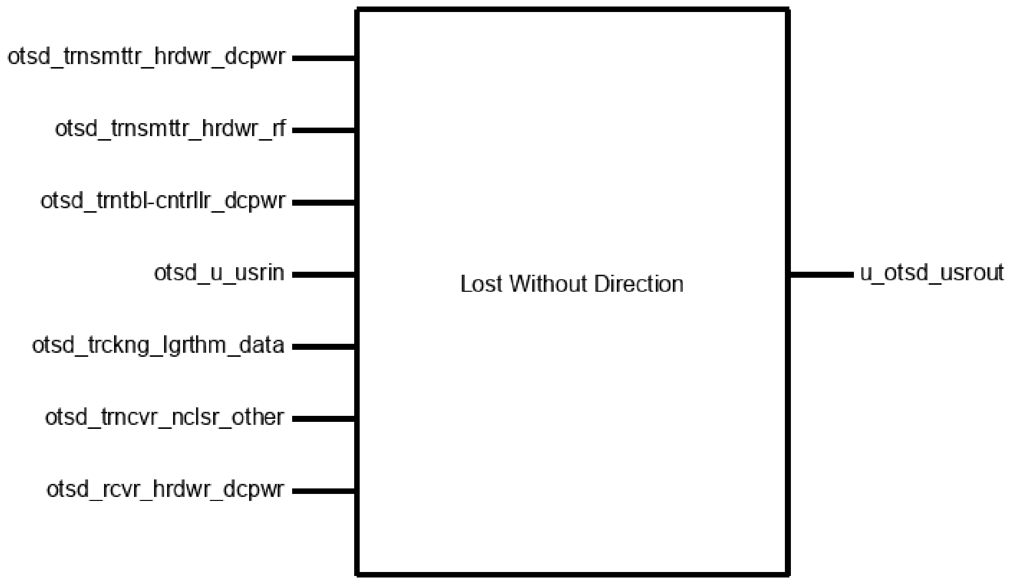


Figure 3.1 - Top Level Black-Box View

3.2. Block Descriptions

The following table defines each block, champion and provides a technical and/or functional description of functionality.

Block Name, Champion	**Block Description**
Transmitter Hardware Champion: Sean Bullis	This block is the hardware that receives the GPS location of the rover via satellite and sends it to the base station. The transmitter receives GPS data using a Ublox NEO-7 based GPS module breakout board from GoouuuTech. The microcontroller on the transmitter is an RP2040 which will uptake the GPS data from the GPS module via UART and transport it into the TX code block. After the TX code block performs its operations the transmitter will receive formatted GPS data from the TX code block via SPI through the RP2040 and will transmit it to the receiver PCB via LoRa protocol. The inputs to the system are 5V input from either a battery boost circuit or 5V from USB power, the GPS data being received by the GPS submodule, and input from the transmitter code block The output is GPS data from the TX code to the receiver hardware

	via RF.
TX Code Champion: Sean Bullis	Code that processes GPS information from the Rover GPS and sends it to the LoRa modem.
Rover GPS Champion: Austin Grubowski	The Rover GPS Block is a library of code that is responsible for sampling UART data received on one of the RX UART pins of the RP2040 Microcontroller chip, parsing said data, storing said data in the corresponding member variable of the GPS data struct, and then passing this data into the main transceiver code (TX Code) when called to do so. The data fed to the RP2040 Microcontroller chip from the GPS module via the UART connection is in the form of NMEA sentences which contain values for the latitude and longitude of the module, the time of the reading, the number of satellites the module is receiving information from, and the altitude, as well as several other received values that are not needed for our purposes. The Rover GPS code library contains a parsing function that first looks for the specific NMEA sentence that begins with “$GPGGA.” The parsing function then goes value by value, saving only the latitude, longitude, and time of receipt to the corresponding members of the NMEA data struct. The main transceiver code, then calls on the members of this struct, reading out the contained data to be transmitted to the next transceiver via the LoRa modem on the transceiver hardware. The GPS module on the other side of the UART connection, is the Gouuu Tech GT-U7 GPS module. This GPS module was sourced from Amazon and selected for its ease of use as well as the ability to interface with it in a number of ways that are beneficial to our project. As well as being directly mounted to the transmitter hardware, a separate Gouuu Tech GT-U7 GPS module is connected to the rover base station over a USB connection.
RX Code Champion: Austin Grubowski	Code that receives condensed GPS data sent, via the LoRa modem, from the rover’s transmitter and passes that data to the ground station, via the transceiver’s USB port, to be processed by the tracking algorithm.

Turntable Champion: Angel Huang	The Turntable is an Actuator based block designed to mount a Ubiquity directional antenna: this antenna is different from the GPS antennas in other blocks. While being an actuator, the block does not output any motion: all motion is contained within the block. All internal motion is strictly rotational in the horizontal plane. Overall physical structure comprises three subassemblies is based on extruded aluminum L-channel, structural framing, and plastic for high strength and low weight: the majority of weight will be contributed by the Ubiquity antenna! A foundation sub-assembly provides an electrical box and adjustable feet for unlevel terrain. The tower sub-assembly resembles a horizontal "U" to provide support for the antenna turntable from above and below for rigidity. Finally, the turntable itself is a cradle built around the Ubiquity that mounts to two Lazy Susan mechanisms on the tower sub-assembly. To provide angular positioning feedback, the Turntable mounts an encoder sub-system based around the AS5048A magnetic encoder. It is mounted on the tower sub-assembly and the magnet is on the cradle sub-assembly. Data between Turntable and Turntable-Controller is sent through an RS422 network to reduce negative effects of EMI (Electromagnetic Interference) based on the SN75C1168N full duplex (simultaneous send and receive) component. 5V power is provided from the Turntable-Controller block.
Turntable-Controller Champion: Angel Huang	The Turntable-Controller is a PCB based block designed to process heading commands from the Tracking Algorithm and UI (user interface) blocks, receive encoder data and issue motor commands to the Turntable block. In the context of the system, the Turntable-Controller acts as motion controller and a data processor: not that it does not handle GPS data from the transmit/receive blocks or perform bearing computations. Importantly, this block allows automatic Turntable positioning with just 12 or 24 VDC power and heading commands from power-on to off. Built around an Arduino Micro microcontroller, the block is designed for flexibility and upgradeability. Each major sub-system receives a sub-PCB to support circuits: there will be one carrier, and at most two motor drivers. This is done to reduce costly reworks if all circuits were on a single PCB and allow field upgrading or repairs of subsystems. Flexible cabling will connects sub-PCBs together within the block. All functions such as initialization, data processing, C&C (command and control) are automatic after power up. Initialization sets the Arduino up then puts control variables and commands the Turntable block to point North. Several loops periodically check Turntable position (10 times a second), processes commands sent via UART (once a second), and issues motor control commands (1000 times a second).

UI Champion: Kira Kopcho	The purpose of the user interface is to provide visualization of the data sent and received by the tracking algorithm as well as provide a method of manually controlling the rotation of the turntable if needed. Both the rover and base station are equipped with GPS modules that send geographic coordinates via USB to the hardware the UI runs on. The idea is that these reported coordinates can then be displayed on the UI so the end user can observe where both the base and the rover are located at all times. At minimum, these coordinates should be updated every 5 seconds to ensure that the data that's being sent to the user is as accurate as possible. In addition the bearing angle calculated by the tracking algorithm is also displayed on the UI. This allows the user to verify the output of the tracking algorithm that is being sent via serial to the turntable module. This angle should update at a minimum rate of 1 update every 5 seconds as well. The UI also provides the end user with a way to send manually defined angles to the turntable. This is meant as a failsafe in case the tracking algorithm is not properly operating. This is important because the rules of the competition the rover compete in prohibit team members from directly interacting with the rover or its communications equipment while competition tasks are underway. By including an interface for manually controlling the angle of rotation of the turntable, the end user can tweak the positioning of the turntable without having to manually interact with it.
Tracking Algorithm Champion: Kira Kopcho	The idea behind the tracking algorithm is to keep track of both the geographic position of the main communications antenna for the rover and the geographic position of the rover itself. In turn, these positions will be used to orient the main communication antenna towards the rover by providing an angle of rotation to the turntable module. Both the rover and the antenna mount have GPS modules mounted on them. The tracking algorithm takes latitude and longitude coordinates from both of these sources and then computes the azimuth (forward bearing) based on them. Basically, azimuth is the horizontal angle on a compass, with true north being 0 degrees and then the rest of the cardinal directions being determined by moving clockwise around a circle. The calculation of this azimuth is produced by feeding both the rover and base coordinates into a series of trigonometric equations. The resulting azimuth from the tracking algorithm is fed to the motor controllers on the turntable module to rotate the antenna in the direction of the rover. Keeping with the engineering requirements of the system, the resulting azimuth will be bound between 0 and 360 degrees. This is to ensure compatibility with the motor driver code on the turntable module. Additionally, the tracking algorithm code itself will provide an updated azimuth at the slowest acceptable speed of 1 update every 5 seconds. For verification purposes, the tracking algorithm will display time stamps alongside the data it receives and each azimuth update it sends.

Tranciever Enclosure Champion: Austin Grubowski	This block is the enclosure that stores the transceiver hardware. This will be 3D printed and have a switch and LED to turn on and indicate whether the battery back up system is on. In addition there will be USB port with a removable panel which can be used for flashing or powering the device. A rubber grommet will secure the USB cable to ensure waterproofing.
Receiver Hardware Champion: Sean Bullis	This block is the hardware that receives the GPS location of the rover from the transmitter and is located on the base station. It uses an Adafruit RFM95 breakout LoRa module to receive LoRa packets from the transmitter and an RP2040 microcontroller to store them in the RX code. The RP2040 will pick up the data over SPI. The inputs to the system are 5V input from either a battery boost circuit or 5V from USB power and the GPS data being received via LoRa. The output is GPS data to the RX code via SPI.

Table 3.2.1 - Block Description Table

3.3. Interface Definitions

The following table includes a complete list of every interface in our system. Each interface is given a name that indicates which blocks it is connected to. Each interface has properties assigned to it that are used to prove if the system is operating nominally.

Interface Name	Properties
otsd_trnsmttr_hrdwr_dcpwr	● **Inominal:** 75mA ● **Ipeak:** 500mA ● **Vmax:** 5.25V ● **Vmin:** 4.75V ● **Vnominal:** 5.0V
otsd_trnsmttr_hrdwr_rf	● **Messages:** NMEA GPS Sentences ● **Other:** 3.3V Logic ● **Protocol:** NMEA
otsd_trntbl-cntrllr_dcpwr	● **Inominal:** 1A ● **Ipeak:** 2A ● **Vmax:** 13V ● **Vmin:** 11V ● **Vnominal:** 12V

otsd_u_usrin	● **Other:** Manually defined angle is only sent when user presses "send" button ● **Type:** Button Click ● **Type:** Text- Angle: 0.0-360.0
otsd_trckng_lgrthm_data	● **Datarate:** Rate Max: 1 update per second ● **Messages:** NMEA Sentences ● **Protocol:** USB
otsd_trncvr_nclsr_other	● **Other:** Battery Switch ● **Other:** Mounting ● **Other:** Battery LED ● **Other:** Weatherproofing
otsd_rcvr_hrdwr_dcpwr	● **Inominal:** 75mA ● **Ipeak:** 500mA ● **Vmax:** 5.25V ● **Vmin:** 4.75V ● **Vnominal:** 5V
tx_cd_trnsmttr_hrdwr_data	● **Datarate:** 12KHz ● **Other:** gps data ● **Protocol:** SPI
trnsmttr_hrdwr_rvr_gps_data	● **Datarate:** 9600 bps ● **Messages:** NMEA Sentences ● **Protocol:** UART
trnsmttr_hrdwr_rcvr_hrdwr_rf	● **Datarate:** Data Rate Min: 1 message per 5 seconds ● **Datarate:** Data Rate Max: 1 message per second ● **Messages:** GPS Latitude and Longitude data, time data ● **Protocol:** LoRa
rvr_gps_tx_cd_data	● **Datarate:** 1 Message Per Second ● **Messages:** Longitude ● **Messages:** UTC Time ● **Messages:** Latitude

rx_cd_trckng_lgrthm_data	● **Datarate:** Rate Min: 1 message per 5 seconds ● **Datarate:** Rate Max: 1 message per second ● **Messages:** Latitude, Longitude, UTC Time ● **Protocol:** USB
trntbl_trntbl-cntrllr_comm	● **Messages:** Data: 16-bit data package ● **Other:** RS422 Logic High Vmin: 2V ● **Protocol:** 4 Wire SPI (data protocol)
trntbl-cntrllr_trntbl_acpwr	● **Inominal:** 0.4A (per stepper positive channel) ● **Other:** Imax: 1A ● **Vmax:** 14V ● **Vnominal:** 12V (per stepper positive channel)
trntbl-cntrllr_trntbl_dcpwr	● **Inominal:** 21mA ● **Ipeak:** 210mA ● **Vmax:** 5.5V ● **Vmin:** 4.5V ● **Vnominal:** 5V
trntbl-cntrllr_trntbl_comm	● **Messages:** Command: 16-bit data package ● **Other:** RS422 Logic High Vmin: 2V ● **Protocol:** 4 Wire SPI (data protocol)
u_otsd_usrout	● **Type:** Numbers - Base Station Coordinates ● **Type:** Numbers - Angle (in degrees) ● **Type:** Numbers - Rover Coordinates
u_trntbl-cntrllr_data	● **Datarate:** 1 message every button click ● **Messages:** Angle: Must be from 0.0 - 360.0 degrees ● **Protocol:** UART Serial (9600 Baud)

trckng_lgrthm_trntbl-cntrllr_data	• **Datarate:** Message Rate Max: 1 message every 1 second • **Datarate:** Serial Transmission Rate: 9600 baud • **Datarate:** Message Rate Min: 1 message every 5 seconds • **Messages:** Angle: 0.0 - 360.0 degrees • **Protocol:** UART Serial
trckng_lgrthm_u_data	• **Datarate:** Rate Min: Updates once every 5 seconds • **Datarate:** Rate max: Updates once per second • **Messages:** Angle: 0.0-360.0 degrees
rcvr_hrdwr_rx_cd_data	• **Datarate:** 12KHz from onboard clock • **Messages:** GPS position and time data • **Protocol:** SPI

Table 3.4.1 - Interface Parameters

3.4. References and File Links

3.4.1. References

3.4.2. File Links

3.5. Revision Table

Author	**Date**	**Description**
Kira Kopcho	3/12/23	Inserted black box diagram from student portal
Kira Kopcho	3/12/23	Inserted generated list of interfaces from the student portal and added small introductory sentence to section 3.3
Angel Huang	5/14/23	Formatting fixes

4. Block Validations

4.1. Tracking Algorithm

4.1.1. Description

The idea behind the tracking algorithm is to keep track of both the geographic position of the main communications antenna for the rover and the position of the rover itself. These positions will be used to orient the main communication antenna towards the rover by providing an angle of rotation to the turntable module.

Both the rover and the antenna mount have GPS modules mounted on them. The tracking algorithm takes latitude and longitude coordinates from both of these sources and then computes the azimuth (forward bearing) based on them. Basically, azimuth is the horizontal angle on a compass, with true north being 0 degrees and then the rest of the cardinal directions being determined by moving clockwise around a circle. The calculation of this azimuth is produced by feeding both the rover and base coordinates into a series of trigonometric equations.

The resulting azimuth from the tracking algorithm is fed to the motor controllers on the turntable module to rotate the antenna in the direction of the rover. Keeping with the engineering requirements of the system, the azimuth will be bound between 0 and 360 degrees. This is to ensure compatibility with the motor driver code on the turntable module. Additionally, the tracking algorithm code itself will provide an updated azimuth at the slowest acceptable speed of 1 update every 5 seconds. For verification purposes, the tracking algorithm will display time stamps alongside the data it receives and each azimuth update it sends.

4.1.2. Design

The tracking algorithm itself is part of a larger tracking subsystem that consists of the Transceiver Hardware Module, the Turntable Controller, and the User Interface on the ground station. Originally, the GPS mounted on the antenna module that the tracking algorithm gathers data from was going to be considered as a separate block. It was later absorbed into the tracking algorithm itself since it is an off-the-shelf piece of hardware and works out of the box with fairly little hardware or software overhead. Additionally the majority of verifiable properties it has involve the tracking algorithm anyways, so it made more sense to simply encompass it into the tracking algorithm block. A full overview of how the tracking algorithm is interconnected with other systems is displayed in the diagram in Figure 2 below.

The tracking algorithm runs on an Intel NUC 11 mini PC using Ubuntu 20.04. The block itself takes input from two sources, the GPS on the base station (otsd_trckng_lgrthm_data) and the receiver (rx_cd_trckng_lgrthm_data). The GPS on the base station is the same module that is on the rover: a GT-U7. This module is a prebuilt board that includes a u-blox Neo-6 GPS receiver and an 15db antenna. The base GPS is connected to the NUC over USB, and the data sent is parsed by the GPS daemon utility. The transciever module is also connected to NUC via USB. The messages sent by the transciever will be decoded using the Pyserial library.

There are two outputs to the tracking algorithm, the UI output (trckng_lgrthm_u_data) and the output to the turntable controller. The turntable controller (trckng_lgrthm_trntbl-cntrllr_data) simply takes the angle returned by the tracking algorithm. This data is sent using a USB to Serial interface. The UI itself runs on the same NUC the tracking algorithm runs on. Separate code will handle the transport of the results of the tracking algorithm (angle display, rover latitude/longitude, base latitude/longitude) to the UI display. All of the inputs and outputs to the tracking algorithm are displayed in Figure 1 below.

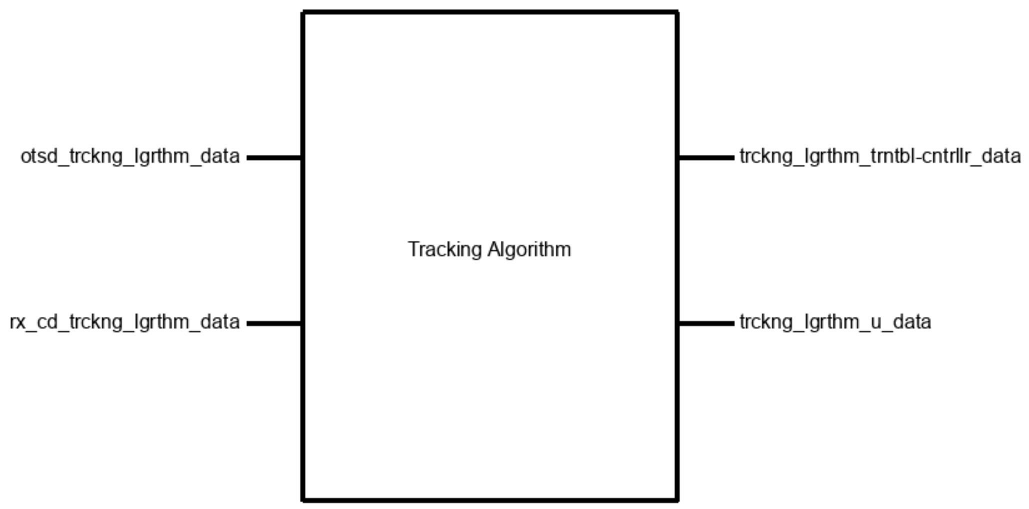


Fig. 4.1.3.1 - Black Box Diagram for the Tracking Algorithm

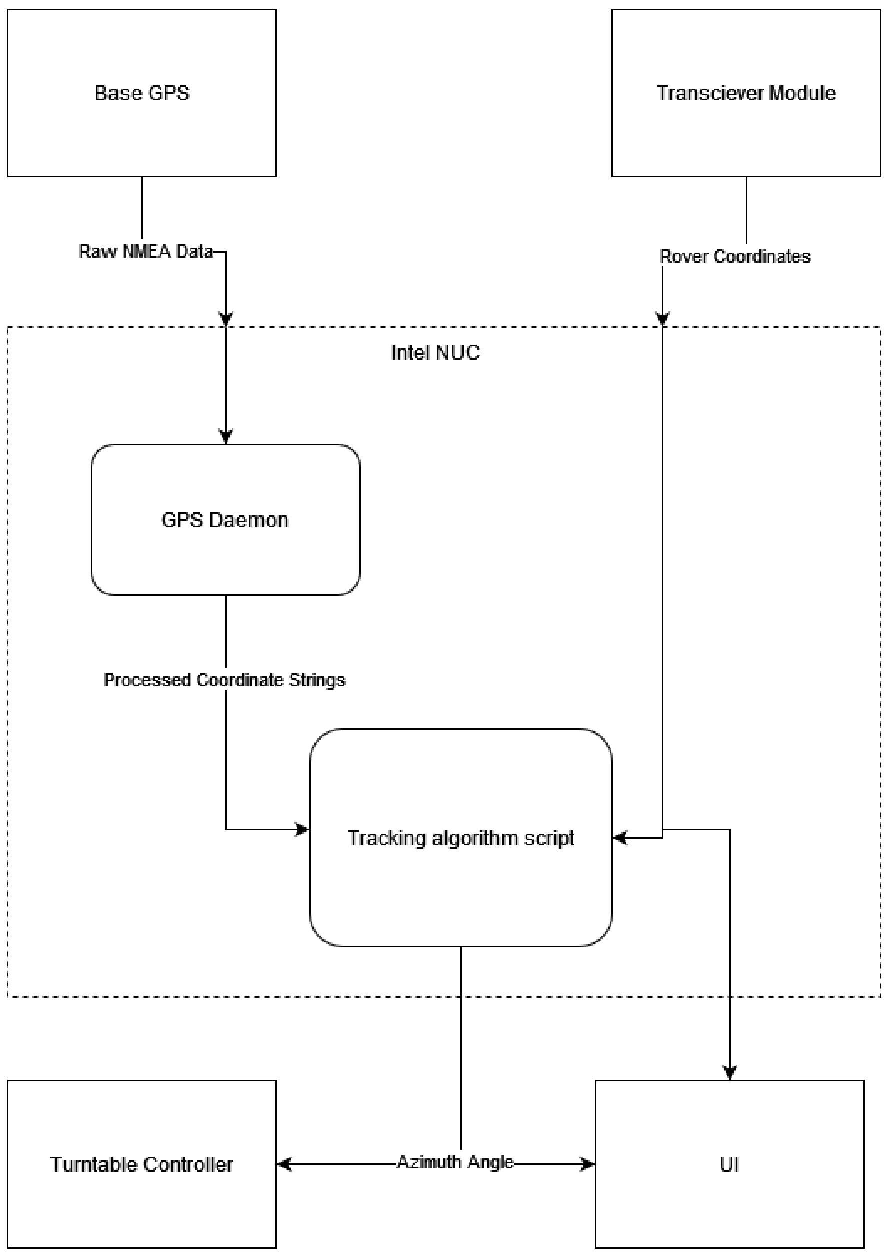


Fig. 4.1.3.2: Full Tracking Subsystem Diagram

4.1.3. General Validation

There are three main parts of the tracking algorithm script. The first is the GPS daemon (GPSd) client, which listens for NMEA 0183 packets from GPS modules over USB or serial [1]. There are many proprietary protocols for receiving GPS data including the UBX protocol that's proprietary to GPS receivers like the NEO-6 [2]. We choose to use NMEA 0183 however, due to its wide use in industry and availability of libraries. The base GPS is connected to the NUC 11 via USB, due to the fact that the NUC does not have exposed serial ports. The client side code is written in Python for both ease of use and to maintain compatibility with other libraries used in the tracking algorithm [3]. The system has a requirement that the angle of rotation (or azimuth) will be updated at least every 5 seconds. This means that all inputs to the tracking algorithm must be processed in under 5 seconds. Using GPSd to process the data from the base GPS helps satisfy this requirement because it has relatively low latency. The optimal line speed of the GPS used is 9600 bits per second, which yields an update rate of about 0.5 seconds [4]. GPSd itself polls the GPS for updates twice per second. This update rate is well within the 5 second requirement.

The second part of the tracking algorithm is the functions for processing the data from the transceiver. The transceiver takes GPS data received from the rover and then sends it to the tracking algorithm code in the form of character streams. It is connected to the NUC via USB and the information from it will be read using the PySerial library since it provides convenient API for reading serial data off a USB port [5]. In order to use the data received from the rover GPS in calculating bearing, we convert the character strings to floating point. The conversions to floating point from characters onboard the NUC because it has a more robust floating point processor than the RPI2040 on the transceiver module. Similar to the coordinates received from the base GPS, the coordinates from the transceiver must be read by the tracking algorithm in less than 5 seconds in order to meet the engineering requirements. To meet this requirement, the tracking algorithm reads from the serial interface once per loop, which gives a rough update rate of once per second (barring any hardware based communication issues).

The final part of the tracking algorithm is the math functions to find the bearing angle from the received coordinates. A general equation for initial bearing or azimuth is used to produce the angle to send to the antenna controller [6]. This angle is bound between 0 and 360 degrees to accommodate for a wide range of positions the rover could drive to. The precision of the final angle is 1 decimal place. The azimuth finding function in the tracking algorithm is hardcoded rather than relying on libraries such as GeoPy. GeoPy relies on geocoders such as Google Maps, and while we used to have access to the Google Maps API for free, it no longer is [7]. In order to validate the accuracy of the angle produced by the tracking algorithm, it is compared to known accurate azimuth calculators. To keep the accuracy of the system within the engineering requirements, the angle returned from the tracking algorithm is required to be within +/- 5 degrees of the angle produced by the bearing calculator.

4.1.4. Interface Validation

The interfaces below aid in the validation that the system meets the engineering requirements. The table lists what each interface is, how it relates to the block and the system around it, and why the design meets the property of each interface. Each interface has values that correspond to meeting certain engineering requirements- namely the engineering requirements for system accuracy, system field of view, and system update rate. Each interface has been designed to meet specifications of these particular requirements.

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**

otsd_trckng_lgrthm_data: Input

Datarate: 1 update per second	Our system has a requirement that tracking is fast and comprehensive, which means nominally the antenna angle should be updated every 5 seconds. In order to make these updates every 5 seconds, the coordinates from the base gps need to be transmitted quicker than the update rate.	The nominal update rate of most GPS modules connected to GPSd is about every half of a second Assuming worst case scenario latency wise, expecting a message every second is reasonable.
Messages: Raw NMEA Data	The Neo-6 GPS chip on the GT-U7 module outputs raw NMEA packets in its default configuration, which is what is used in our system.	The tracking algorithm links into the GPSd daemon, which listens to the serial port the gps is connected to. It collects the raw NMEA data and then the GPSd Python library used in the tracking algorithm interprets this raw data into a human readable format.
Protocol: USB	The GT-U7 GPS module can communicate over either UART or USB. USB is used to communicate to the ground station computer because it does not have any exposed serial ports.	The GT-U7 module provides a micro usb 2.0 connector. In addition, GPSd (the daemon used to collect the raw NMEA data from the base gps) listens for incoming NMEA data on the USB/Serial interfaces provided by the Linux operating system

rx_cd_trckng_lgrthm_data : Input

Datarate: Rate Max: 1 message per second	Our system has a requirement that tracking is fast and comprehensive, which means nominally the antenna angle	The code polls the transceiver for data once per second.

	should be updated every 5 seconds. In order to make these updates every 5 seconds, the coordinates from the rover gps need to be fed to the tracking algorithm in less than 5 seconds.	
Datarate: Rate Min: 1 message per 5 seconds	Nominally the angle of the antenna should be updated every 5 seconds. However, there is the possibility of latency when sending coordinates from the receiver. So worst case scenario, the minimum update rate should be within 5 seconds.	The tracking algorithm currently polls the receiver module for data once per loop which equates to roughly once a second.
Messages: Latitude, Longitude, UTC Time	The tracking algorithm operates by taking the latitude/longitude of the base gps and the latitude/longitude of the rover gps to calculate a bearing angle. UTC time is included to check the speed of updates.	The tracking algorithm uses Pyserial to read incoming data from the receiver, which has the ability to read data strings. The received data is then stored in a string that can be parsed to determine each individual value needed for other program functions.
Protocol: USB	USB is used because the NUC does not have exposed serial ports.	Pyserial, the python library chosen to read the data send by the receiver supports reading from both USB and serial ports.

trckng_lgrthm_trntbl-cntrllr_data: Output

Datarate: Serial Transmission Rate: 9600 baud	9600 is a standard baud rate when working with UART	The arduino microcontroller on the antenna controller is compatible with 9600 baud serial UART transmissions
Datarate: Message Rate Min: 1 message every 5 seconds	5 second intervals are chosen for updates due to potential latency between modules in the tracking sub-system	Assuming minimum latency over UART at 9600 baud, we should reasonably be able to send a message once every five seconds

Datarate: Message Rate Max: 1 message every 1 second	An update every second is the best case scenario, but due to hardware limitations this might not always be possible	Assuming minimum hardware latency between all parts of the tracking system, a message rate of once per second should be achievable.
Messages: Angle: Must be from 0.0 - 360.0 degrees	We do not want to drive the stepper motors at negative angles, so going from 0-360 degrees will ensure we catch wrap-around angles.	The code will have validation for whether the output angle is between 0-360 degrees.
Protocol: UART Serial	The arduino on the turntable controller is expecting angles sent via UART serial.	To achieve the sending of serial data, a FTDI USB/Serial converter is plugged into the NUC. Pyserial is then used the send serial data with UART specifications (9600 baud, 8 bits, 1 stop bit) over the USB port the FTDI cable is plugged into.

trckng_lgrthm_u_data : Output

Datarate: Rate Max- updates once per second	We want the angle to update on the UI synchronously with the update sent to the antenna controller so users can know what position the antenna is currently rotated to at all times.	Since we use a python based module for UI directly on the NUC as well, the latency is negligible between sent data and updates in the UI
Datarate: Rate Min: updates once every five seconds	We want the angle to update on the UI synchronously with the update sent to the antenna controller so users can know what position the antenna is currently rotated to at all times.	Since the code is running directly on the groundstation, latency between the code and the UI should be negligible. However in case there is latency on the hardware end, at minimum the angle should be able to be updated every 5 seconds.

Datarate: Rate Max: updates once per second	We want the angle to update on the UI synchronously with the update sent to the antenna controller so users can know what position the antenna is currently rotated to at all times.	The function to produce the bearing angle is called once per loop, which if the program is running nominally means an update of once per second.
Messages: Angle 0-360 degrees	Since we're binding the angle from 0-360 for sending to the antenna controller, in order to show accurate angles on the UI it also has to be 0-360 degrees.	The code will not send an angle that is not between 0-360

Table 4.1.4.1 - Tracking Algorithm Interface Validation

4.1.5. Verification Process

The verification plan listed below details basic steps to ensure functionality of the tracking algorithm before it is tested with other elements of the larger system. Due to not having access to the transceiver module while writing the initial code, an Adafruit feather is used in place of the transceiver module to send “rover coordinates” to the tracking algorithm. How this is achieved is shown in the code links in the file references section below. Edits to the code will likely have to be made when the entire system integration process begins, but for now the feather provides an adequate simulation of what interfacing with the transceiver module will be like. Timing will be verified based on calculating the difference between the times that each angle is reported. This ensures greater accuracy than using a stop watch and watching the terminal. The verification plan below aims to be simple in its implementation, so even users unfamiliar with the tracking algorithm can verify that is running correctly.

1. Start the GPS daemon on the NUC (if not already set to start on powerup) and wait for the base GPS to get a fix (nominal 1 second)
2. Ensure that the transceiver module is plugged into the NUC and started. If the transceiver is not present at block check off use a similar microcontroller to provide false coordinates over serial to the NUC.
3. Start the tracking algorithm code
4. Let the tracking algorithm run for 1-2 minutes to gather sufficient datapoints, then ctrl-c to stop the process.
5. Plug the reported rover latitude and longitude as well as the base latitude and longitude into a known azimuth calculator to produce the known accurate bearing angle
6. Convert the angle produced by the known azimuth calculator to an angle between 0-360 to compare with the output of the tracking algorithm
7. Verify that the produced angle is within 5 degrees of the angle reported the calculator
8. Record the times reported when the rover coordinates are printed and when the base coordinates are printed.
9. Record the time each angle is printed.

10. Take the difference between the times each angle is printed to determine how fast the tracking algorithm is operating. The goal is 1 update every 5 seconds at maximum.
11. Take the difference between each time each set of coordinates is reported. The goal is that both the rover and base coordinates should update every 5 seconds max, but ideally in less than 5 seconds.

4.1.6. References and Files Links

4.1.6.1. References

[1] E. Raymond, "GPSD" in *The Architecture of Open Source Applications Volume II: Structure, Scale and a Few More Fearless Hacks,* A. Brown and G. Wilson. [Online] Available: https://www.aosabook.org/en/gpsd.html [Accessed: Jan. 20, 2023]
[2] u-Blox, "NEO-6 Data Sheet" Aug. 2009 [Revised Dec. 5, 2011] [Online] Available: https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf [Accessed Jan. 20, 2023]
[3] G.E. Miller, "gpsd Client Example Code", *gpsd.io*, July 24, 2022. [Online] Available: https://gpsd.io/gpsd-client-example-code.html [Accessed Jan. 20, 2023]
[4] E.S Raymond, "Where's the Latency? Performance analysis of GPSes," Version 2.4, Jan. 20, 2020 [Online] Available: https://gpsd.io/performance/performance.html [Accessed Feb. 11, 2023]
[5] C. Lechti, "pySerial API", *pyserial.readthedocs.io*, 2021. [Online] Available: https://pyserial.readthedocs.io/en/latest/pyserial_api.html [Accessed Jan. 20, 2023]
[6] C. Veness "Calculate distance, bearing and more between Latitude/Longitude points", *moveable-type.co.uk* Feb. 2019 [Online] Available: https://www.movable-type.co.uk/scripts/latlong.html [Accessed Jan. 20, 2023]
[7] "Geocoders", *geopy.readthedocs.io* [Online] Available: https://geopy.readthedocs.io/en/stable/#module-geopy.geocoders [Accessed Jan. 20, 2023]

4.1.6.2. File Links

1) RX Module PySerial Test Code
2) Tracking Algorithm Code

4.1.7. Revision Table

Author	**Date**	**Description**
Kira Kopcho	1/20/23	Created Document
Kira Kopcho	1/20/23	Wrote Description and Design sections
Kira Kopcho	1/20/23	Added general verification, interface verification, and verification plan information
Kira Kopcho	2/03/23	Revised block description to include less technical details
Kira Kopcho	2/03/23	Minor revisions to interfaces to

		match what is currently on the portal
Kira Kopcho	2/03/23	Added links to the GitHub repository where the latest version of the Tracking Algorithm code is stored
Kira Kopcho	2/03/23	Minor revisions to the the description to explicitly reference the applicable engineering requirements for this block
Kira Kopcho	2/11/23	Updated Figure 2 in the Design section to be more legible
Kira Kopcho	2/11/23	Minor revisions to the design section paragraphs to reflect interface/design changes made on the student portal
Kira Kopcho	2/11/23	Revised the general verification section to provide more direct references to the engineering requirements. Also reordered the in text citation numbers so they're not out of order
Kira Kopcho	2/11/23	Added introductory paragraph to verification plan and added additional steps to better prove that each requirement is met
Kira Kopcho	2/11/23	Replaced the Black Box Diagram image with an updated version that shows the latest changes to the input/output interfaces
Kira Kopcho	2/11/23	Updated the interface table to match the interfaces that are in the student portal
Kira Kopcho	2/13/23	Merged separate verification document into full project document
Kira Kopcho	5/14/23	Minor grammar/spelling revisions and formatting changes

4.2. Rover GPS

4.2.1. Description

The Rover GPS Block is a library of code that is responsible for sampling UART data received on one of the RX UART pins of the RP2040 Microcontroller chip, parsing said data, storing said data in the corresponding member variable of the GPS data struct, and then passing this data into the main transceiver code (TX Code) when called to do so. The data fed to the RP2040 Microcontroller chip from the GPS module via the UART connection is in the form of NMEA sentences which contain values for the latitude and longitude of the module, the time of the reading, the number of satellites the module is receiving information from, and the altitude, as well as several other received values that are not needed for our purposes. The Rover GPS code library contains a parsing function that first looks for the specific NMEA sentence that begins with “$GPGGA.” The parsing function then goes value by value, saving only the latitude, longitude, and time of receipt to the corresponding members of the NMEA data struct. The main transceiver code, then calls on the members of this struct, reading out the contained data to be transmitted to the next transceiver via the LoRa modem on the transceiver hardware.

The GPS module on the other side of the UART connection, is the Gouuu Tech GT-U7 GPS module. This GPS module was sourced from Amazon and selected for its ease of use as well as the ability to interface with it in a number of ways that are beneficial to our project. As well as being directly mounted to the transmitter hardware, a separate Gouuu Tech GT-U7 GPS module is connected to the rover base station over a USB connection.

4.2.2. Design

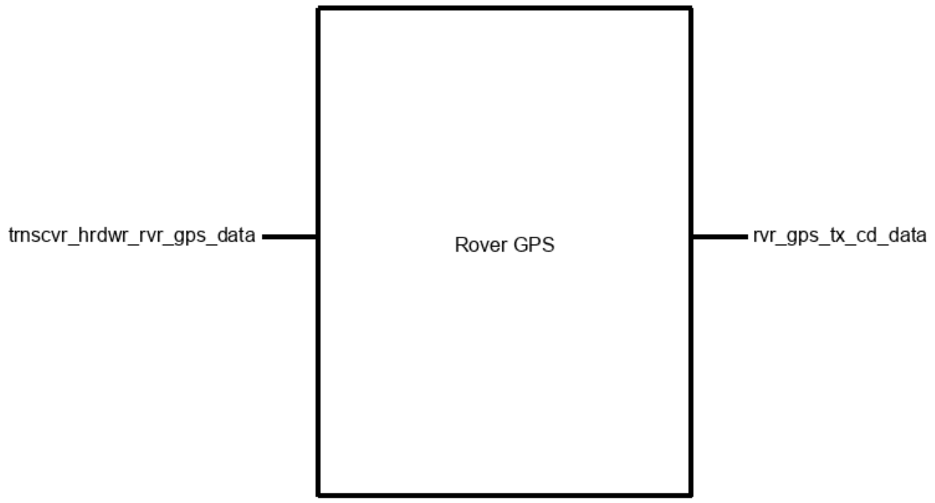


Fig. 4.2.2.1 - Rover GPS Block

The code below first defines the UART port utilized to communicate with the GPS module, the baud rate, LED pin, the UART TX and RX pins, the NMEA sentence buffer size, and the specific NMEA sentence we are interested in parsing. The NMEA_data struct is then defined with the latitude and longitude members being doubles and each of the individual time members being integers. Following these definitions is the parsing function, which is defined as an NMEA_data structure. This function takes in a pointer to an NMEA sentence and returns an NMEA data struct. The parsing function checks the beginning of the sentence that was passed into it to confirm that it matches the NMEA_SENTENCE_GPGGA definition. If there is not a match, no parsing is done and the function returns an NMEA_data structs with zeros filled in for all members. If there is a match between the passed in NMEA sentence and the defined sentence type, the parsing function creates a pointing character which will be used to point to each character of the passed in sentence as we parse it. This pointer is moved to the first comma which precedes the time information in the $GPGGA NMEA sentence. This time information is read into the three time members of the NMEA_data structure; the first two integers are the hour, the next two are the minutes, and the last two are the seconds at the time the location data within this sentence was received by the GPS module. In the $GPGGA NMEA sentence, seconds are delivered as a double, but the GPS module we are using does not record the second data past the decimal point; therefore, this function only reads the seconds as a two digit integer and does not record the zeros displayed after the decimal point in this sentence. After reading the time data, the pointer is moved to one character after the comma succeeding the time data and preceding the latitude data. The first two values of the latitude data are saved in a temporary variable; the remainder of the values are read into a latitude_minutes variable, divided by 60 to obtain the needed value and then added to the latitude variable with is then saved to latitude struct member. After reading the latitude data, the function moves onto the N/W indication field in the NMEA sentence. If in the southern latitude, a minus sign is added to the front of the latitude struct member. The function does the same thing for the longitude and the E/W indication fields as it did for the two previous section. The function concludes by returning the NMEA_data struct.

The main section of the code first sets all of the corresponding I/O pins using the previously defined values. A while loop is then used to continuously run the reading, saving, and writing of GPS data. The NMEA sentences from the GPS module are read and the desired sentence is saved into the NMEA data structure utilizing the parsing function. The data is then read out in the following order: latitude,longitude,time(UTC). The LED is turned on and then off to indicate the returning of data and then the NMEA data is cleared from memory.

```
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/uart.h"
#include "pico/time.h"
#include "pico/multicore.h"

// UART definitions
#define UART_ID uart0
#define BAUD_RATE 9600

// Configure LED pin
```

```
#define LED_PIN 25

// Pins GP16 and GP17 are being used for testing purposes with a
pico;
// pins 0(TX) and 1(RX) will be used when this code is uploaded to
the
// transceiver.
#define UART_TX_PIN 16
#define UART_RX_PIN 17

// Definitions used by the parse_nmea_sentence function
#define NMEA_BUF_SIZE 100
#define NMEA_SENTENCE_GPGGA "$GPGGA"

// Define the NMEA_data struct
typedef struct {
  double latitude;
  double longitude;
  int hour;
  int minute;
  int second;
} NMEA_data;

// NMEA_data parsing function which takes in an NMEA sentence and
fills in the
// longitude, latitude, and time members of the NMEA_data struct.
NMEA_data parse_nmea_sentence(const char *sentence) {
  NMEA_data data = {0};

  if (strstr(sentence, NMEA_SENTENCE_GPGGA) == sentence) {
    char *p = (char *)sentence;

    // move pointer to time
    p = strchr(p, ',') + 1;

    // parse time
    int hour, minute, second;
    sscanf(p, "%2d%2d%2d", &hour, &minute, &second);
    data.hour = hour;
    data.minute = minute;
    data.second = second;

    // move pointer to latitude
    p = strchr(p, ',') + 1;
    // parse latitude
    double latitude, latitude_minutes;
    sscanf(p, "%2lf%lf", &latitude, &latitude_minutes);
    data.latitude = latitude + latitude_minutes / 60.0;

    // move pointer to N/S indicator
    p = strchr(p, ',') + 1;

    // check N/S indicator and adjust latitude
    if (*p == 'S') {
      data.latitude = -data.latitude;
    }

    // move pointer to longitude
    p = strchr(p, ',') + 1;

    // parse longitude
    double longitude, longitude_minutes;
    sscanf(p, "%3lf%lf", &longitude, &longitude_minutes);
    data.longitude = longitude + longitude_minutes / 60.0;

    // move pointer to E/W indicator
    p = strchr(p, ',') + 1;

    // check E/W indicator and adjust longitude
    if (*p == 'W') {
      data.longitude = -data.longitude;
    }
  }

  return data;
}
```

4.2.3. General Validation

The code that will be uploaded to the RP2040 microcontroller chip will be written in the C programming language for purposes of familiarity amongst the relevant members of the group and for the best utilization of the limited resources available. The GPS module used in conjunction with this code is the Gouuu Tech GT-U7 GPS module. This particular module outputs 7 different NMEA sentences, one of which contains all of the desired information; the “$GPGGA” sentence. Parsing this sentence is on this first transceiver’s hardware saves resources on other sets of hardware in our project.

4.2.4. Interface Validation

Interface Property	**Why is this interface this value?**	**Why do you know that your design details for this block above meet or exceed each property?**
trnscvr_hrdwr_rvr_gps_data : Input		
Datarate: 9600 bps	This is the standard operating baud rate of the Gouuu Tech GT-U7 GPS module.	When utilizing the serial monitor to view output of the code, output is only visible in the serial monitor when it is set to 9600 baud. An oscilloscope has been used to view this baud rate and will be used again if need to show definitively that transmissions are at this rate.
Messages: NMEA Sentences	Without explicit configuration, the GPS module sends 7 sentences formatted in the standardized NMEA format.	My code reads the sentence identifier at the beginning of each sentence, determines the required sentence, and parses this sentence.
Protocol: UART	Although the Gouuu Tech GT-U7 GPS module has a simple USB plug-and-play interface, the UART TX and RX pins are utilized and connected to the transceiver hardware.	The appropriate UART parameters are defined and enabled to allow for UART communications.
rvr_gps_tx_cd_data : Output		
Datarate: 1 Message Per Second	One message per second was determined to provide ample time for the tracking of the rover via the rotating antenna.	The functions and code included have a delay of 500 milliseconds with an added 500 milliseconds added with the blinking of the LED, resulting in the transmission of GPS data once ever second. This is visible with the printout of the time with each output in the serial monitor used.
Messages: Longitude,Latitude,UTC Time	This simple string of characters representing the latitude, longitude, and UTC	The code returns these struct members. The latitude, longitude, and UTC time data

	time is utilized due to the ease of handling by the tracking algorithm on the base station	have been viewed as outputs in the serial monitor.

4.2.5. Verification Process

1. Connect the TX and ground pins of the Gouuu Tech GT-U7 GPS module up to an oscilloscope to demonstrate the baud rate of the signal.
2. Use the serial monitor to display the raw NMEA sentences from the GPS module.
3. Use the serial monitor to display the latitude, longitude, and UTC time data after the NMEA sentences have been parsed. The time displayed will show one sentence being transmitted at least once per second
4. Plug the new latitude and longitude sentence into google maps to show the GPS coordinates are accurate and live.

4.2.5.1. References

[1] Semtech, "SX1276/77/78/79 Datasheet," [Online] Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE [Accessed: 11-Feb-2023].
[2] Raspberry Pi, "RP2040 Datasheet - A Microcontroller by Raspberry Pi" [Online] Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf [Accessed: 11-Feb-2023].

4.2.5.2. Files Links

1) GPS Test Code
2) GPS Code Integrated into the Transmitter Code

4.2.6. Revision Table

Author	Date	Description
Austin Grubowski	01/12/23	Created Block Validation document.
Austin Grubowski	01/20/23	Updated interfaces, filled in all remaining sections.
Austin Grubowski	02/11/23	Changed what block is being validated. Filled in all sections.
Austin Grubowski	03/12/23	Corrected references. Clarified verification plan.
Kira Kopcho	5/14/23	Fix revision table to match the style of the rest of the document

4.3. Transmitter Hardware

4.3.1. Description

This block is the hardware that receives the modulated LoRa packet from the transmitter on the rover. The receiver will transfer GPS location data received from the LoRa modem and send that data to the receiver code block via the RP2040 microcontroller via SPI. The system uses the same hardware design to receive the transmitted messages as it is used to send messages.

The RP2040 is driven by a 12MHz clock and uses 128Mb flash storage to store its code. The boot mode is determined using a jumper on the PCB controlling either code upload or code execution depending on the state when powered on.

The inputs to the system are 5V input from either a battery boost circuit or 5V from USB power, and input from the transmitter code block via radio frequency. The output is GPS data to the receiver code block over SPI.

4.3.2. Design

This block takes three inputs and produces two outputs. The input *otsd_trnsmttr_hrdwr_dcpwr* is the power coming from the USB or battery to the transmitter. The transmitter has a built-in power switching and power regulating circuit that requires a 5V input with the capability to draw up to 500mA. The *otsd_trnsmttr_hrdwr_rf* input is data being transferred from outside the system to the on-board GPS submodule. This data is in the form of NMEA sentences and contains GPS data. The *tx_cd_trnscvr_hrdwr_data* describes the data that is transferred from the code block on the transmitter's on-board processor onto the hardware to be sent via LoRa to the receiver.

The output *trnsmttr_hrdwr_rvr_gps_data* is the data transfer from the transceiver hardware's GPS module to the GPS code block on the transmitter's microcontroller. The output *trnsmttr_hrdwr_rcvr_hrdwr_rf* is the transfer of data from the LoRa sub-module on the PCB through radio frequency to the receiver's LoRa sub-module. Figure 1 shows the black box diagram for the block.

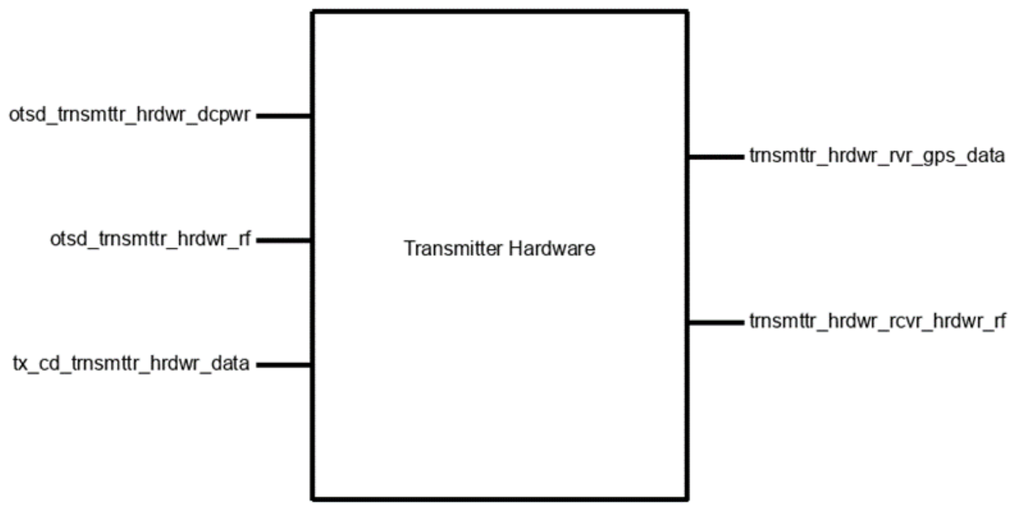

otsd_trnsmttr_hrdwr_dcpwr
otsd_trnsmttr_hrdwr_rf
tx_cd_trnsmttr_hrdwr_data
Transmitter Hardware
trnsmttr_hrdwr_rvr_gps_data
trnsmttr_hrdwr_rcvr_hrdwr_rf

Fig. 4.3.2.1 - Black Box for Transmitter

Figure 2 shows the schematic of the sub-block for the microcontroller. The microcontroller has connections to the GPS module where it receives GPS data from satellite, the LoRa submodule where it sends GPS data to transmit and where it receives GPS data from the transmitter, and the flash memory and power sub-blocks.

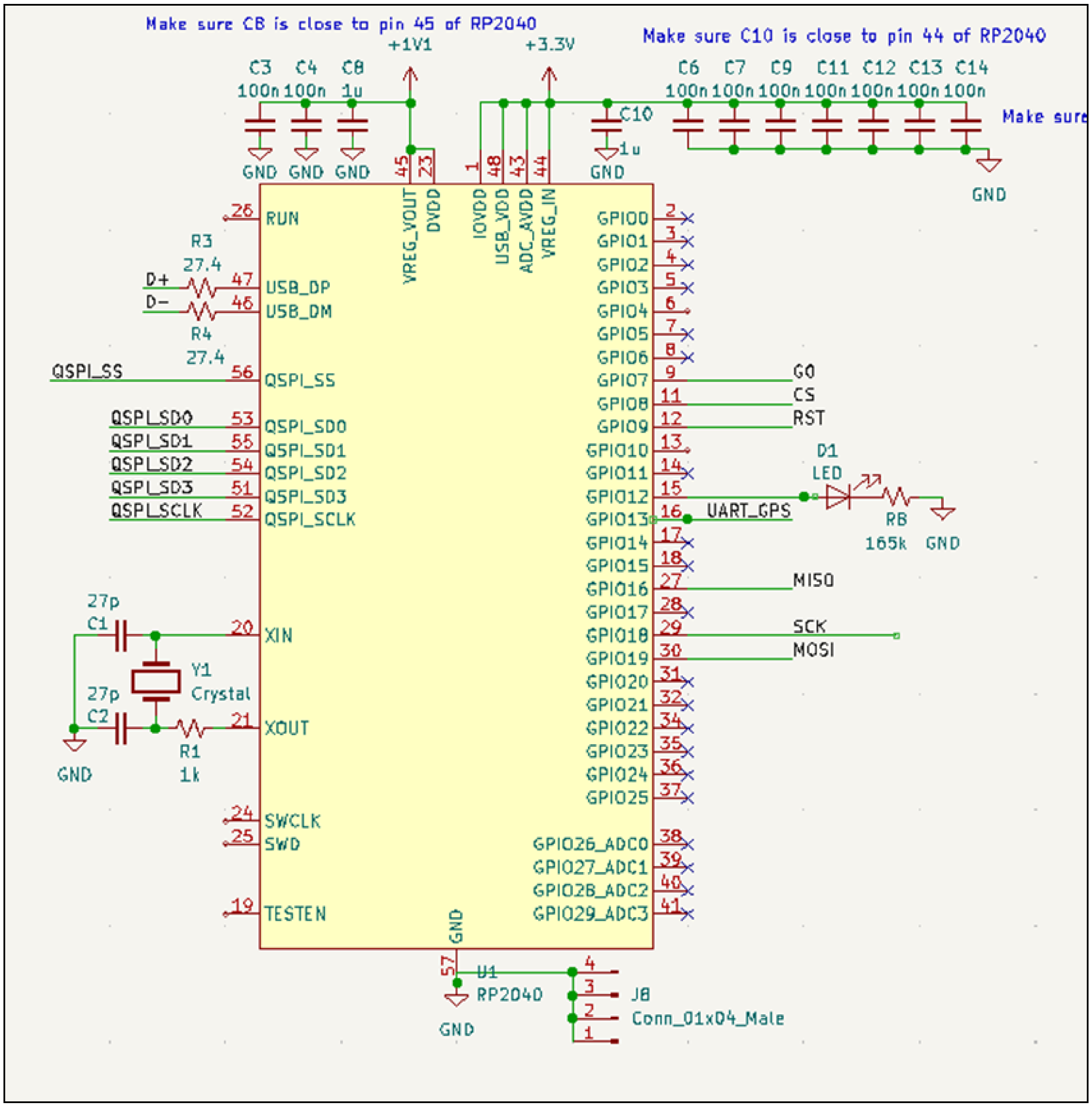


Fig. 4.3.2.2 - RP2040 is shown with connections to the GPS module and LoRa submodule, flash memory submodule, and the clock.

Figure 3 shows the power distribution circuit for the block. Battery connection for a 3.7v LiPo battery goes into a 3.7-5V boost converter to provide 5V supply. The USB and boosted battery power go into a power switcher which selects which power source based on the logic on pins D0 and D1. Jumpers are provided to control the default power source.

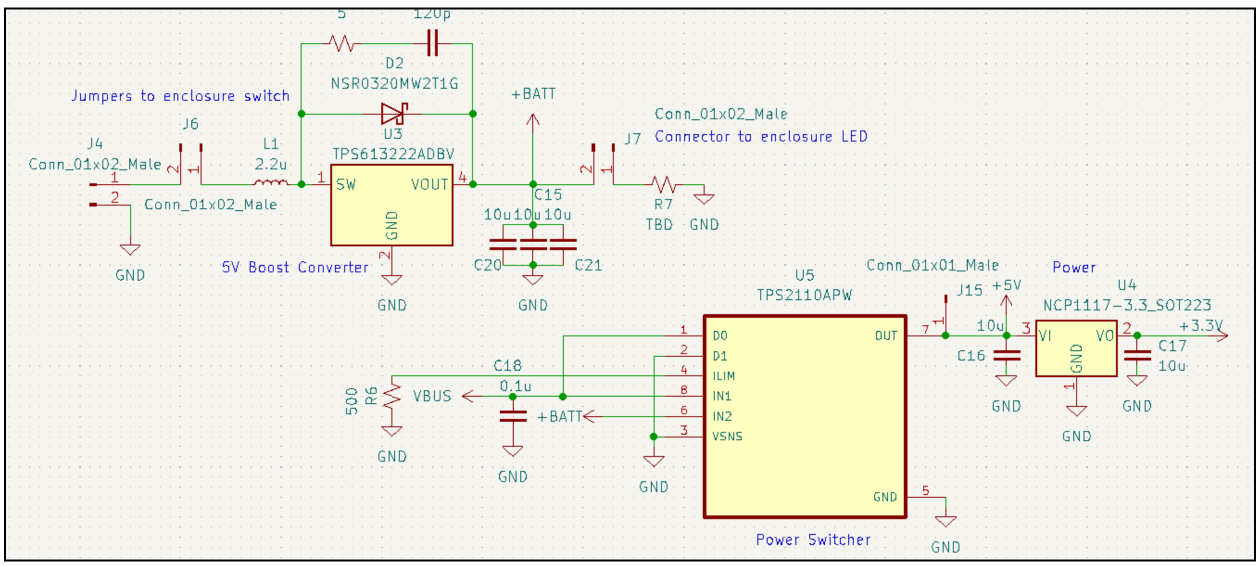


Fig. 4.3.2.3 - The power distribution circuit for the block.

A Winbond W25Q128JVS is used for flash memory which can hold up to 128Mb of data. This is connected via the QSPI ports to the RP2040. The jumper J1 is a connection to control the boot mode upon start-up. If USB_BOOT is held low during power on the device will initialize as USB mass storage, while if the USB_BOOT is held high on power up which is the default case, the device will power on to run code. This is shown in Figure 4.

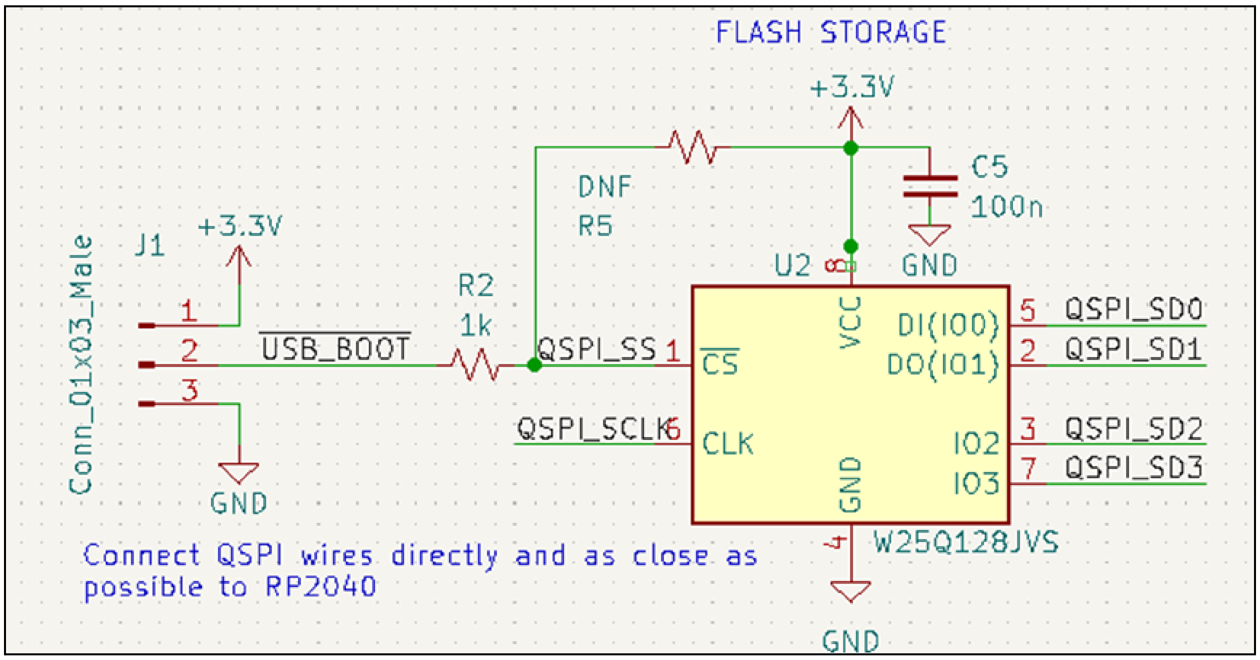


Fig. 4.3.2.4 - Flash storage circuit with USB_BOOT switch to enable code execution mode or code upload.

External connections to sub-modules are also provided. The device does not have a USB port but rather break-out pins for the individual data and power lines for a USB cable. In addition, the GPS module and LoRa modem are connected through these external connections. The external connections are shown in Figure 5.

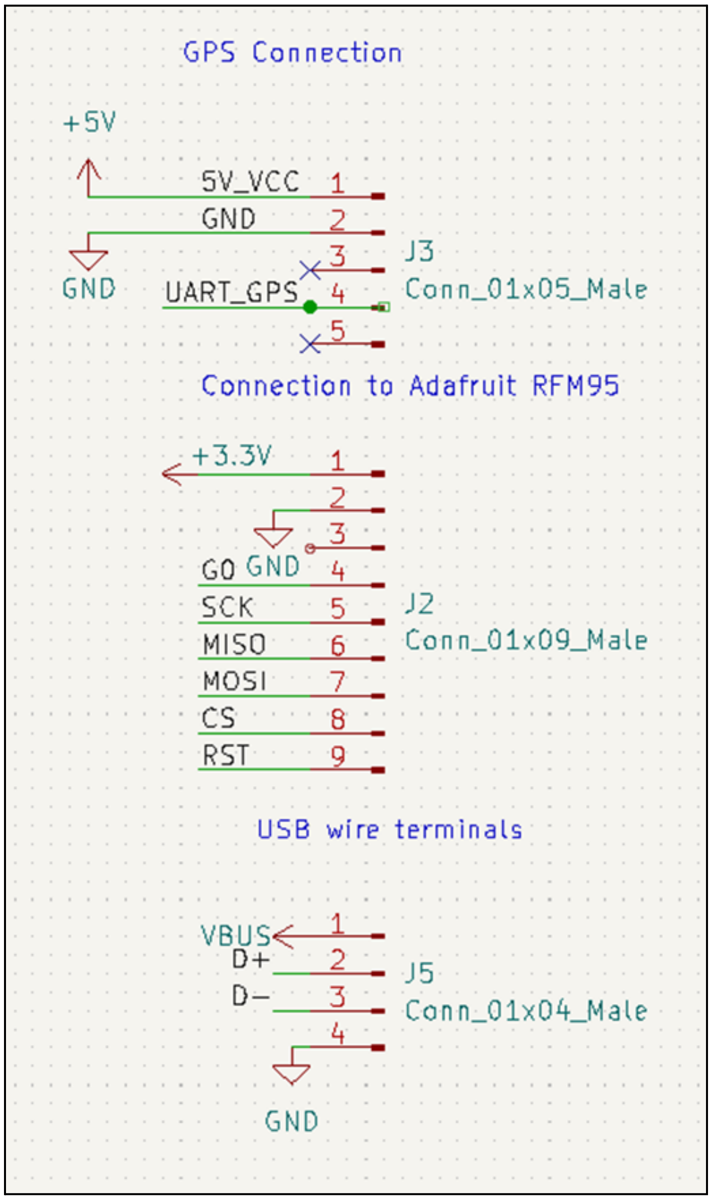


Fig. 4.3.2.5 - Pin connections for external sub-modules and USB

4.3.3. General Validation

This hardware design is based on the minimum design from Raspberry's RP2040 hardware design guide. The chip requires 3.3V input voltage and uses 3.3V logic on its input and output pins [1]. The chip is clocked by a *ABLS-12.000MHZ-B4-T* 12MHz crystal oscillator, and utilizes *W25Q128JVSIM* 128Mb flash storage. Both parts are recommended from the hardware example. The RP2040 has SPI and UART channels and both will be utilized in this design to transfer data to and from the microcontroller.

The 3.3V required for the RP2040 and other peripherals is generated using the *AZ1117IH-3.3TRG1*, a 3.3V voltage regulator from Diodes Incorporated. The voltage regulator has a 1.4V dropout voltage while pulling 1A. This means that the voltage supplied to the voltage regulator must be at least 1.4V higher than its output to generate a stable output voltage at that current. The 5V USB power supplied to the voltage regulator will be sufficient to satisfy this requirement [2].

In the case of USB failure, a battery connection which can support a 3.7V LiPo battery or three AA batteries is provided. The battery is boosted to a 5V output via *TPS61322*. The battery is enabled by a switch and power output is controlled by a *TPS2110APWR* power multiplexer. The multiplexer is configured such that USB power drives the circuit when it is present, however when there is no USB power and the battery switch is enabled the circuit will be driven by the battery. The state of the logic pins D0 and D1 are 1 and 0 by default which selects the output as IN1. D0 is connected to USB, and when USB goes to 0 the D0 and D1 signals will both be 0 which selects for IN2. This is shown by the datasheet for the power multiplexer on page 7 [3].

The wireless transmitting and receiving of the LoRa packets is handled using an *Adafruit RFM95* module. This is connected using pins and pads to bind the module to the board. The module takes 3.3V input and will draw an absolute maximum current of 140mA while transmitting [4]. This module communicates with the RP2040 over the SPI interface with a data transfer rate of 12KHz.

Pins and pads for the NEO-7 based GPS module bind the module to the board. This module takes 3.6V to 5V input and uses 3.3V for its logic level. This module communicates with the RP2040 over the UART interface. GPS data will be retrieved from the GPS module via UART transferring at 9600 bps [5].

4.3.4. Interface Validation

Interface Property	Why is this interface this value?	Why do you know that your design details <u>for this block</u> above meet or exceed each property?
otsd_trnsmttr_hrdwr_dcpwr		

$I_{nominal}$ = 75mA	This value is chosen based on the sum of the typical current values for each of the heavy lifting components on the board; the RP2040, the LoRa modem, and the GPS module.	The RP2040 in active mode will pull about 20mA on its own, and up to 50mA driving all its GPIO pins [1]. The LoRa modem typically draws 134mA during transmission and the GPS module typically draws 32mA during acquisition [4][5]. The memory module draws 25mA during reads/writes. The power switch pulls 55uA typical during power delivery [2]. The boost converter draws 6.5uA while operating. The transmitter will transmit periodically, and read/writes will not be frequent so the RP2040 and GPS module will be the main power sources giving approximately 55mA.
I_{peak} = 500mA	This value was chosen with the absolute maximum current ratings of each of the components in the block in mind. During a heavy load test for the RP2040 running video drives, audio drivers and SD card access (popcorn test) the uC draws 91mA. The LoRa modem pulls 140mA maximum during transmission. The GPS module pulls 40mA maximum. The memory module pulls 25mA during read/write. The power switch pulls 90uA max when delivering power. The	Each of the values for absolute maximum current draws come from the data sheets of the individual components. Summing them all and rounding up with conservative estimates for max values will provide enough for each component individually and then some. The linear voltage regulator is rated to pull 1A of current with a drop off voltage of 1.4V above the output voltage. The power switch is rated to deliver up to 1A as well. The battery boost

	boost converter pulls 10uA max while operating. This sums to 297mA but a conservative estimate of 200mA of headroom is provided to ensure power delivery is applied.	IC is rated to deliver up to 1.8A which are well above the required nominal current.
V_{max} = 5.25	This value was chosen because it is the maximum possible input from the USB source or battery source.	This will be met because the highest output from the battery boost converter is 5.15V according to the TPS61322 datasheet and the highest output from USB is 5.25V according to standard USB specs.
V_{min} = 4.75	This value was chosen because it is the lowest possible input from the USB or battery source.	This will be met because the lowest output from the battery boost converter is 4.85V according to the TPS61322 datasheet and the lowest output from USB is 4.75V according to standard USB specs.
$V_{nominal}$ = 5	This is the nominal input from battery or USB.	This will be met because the typical voltage output of both the battery boost converter and USB is 5V.

otsd_trnsmttr_hrdwr_rf

Other = 3.3V Logic	This is chosen because the power for the RP2040 is 3.3V and operates on 3.3V logic.	This will work because the GPS module outputs a 3.3V logic signal and is compatible with the RP2040.

Protocol = NMEA	This was chosen because it is out of our control and is the protocol used by the GPS submodule to retrieve satellite data.	This is the format of the data coming from the satellite and is out of our control.
Messages = NMEA Sentences	This is what the GPS submodule will provide. It is essentially controlled by the GPS submodule.	The GPS submodule controls this based on what the satellite sends and is a design constraint.

trnsmttr_hrdwr_rvr_gps_data

Data rate = 9600 baud rate	This is the default value for UART.	This works because it is the default baud rate for UART, and this interface uses UART. The RP2040 and the GPS module both support UART [1][5].
Messages = NMEA Sentences	This is what the GPS submodule will output. It is controlled by the GPS submodule.	This will work because it is what is sent by satellite to the GPS module. There is no control over the data until we receive it.
Protocol = UART	This was chosen because the RP2040 and GPS module support UART and it is a simple interface.	This will work because there are UART libraries for managing data transfers for the RP2040 in the C/C++ SDK and both modules support UART.

trnsmttr_hrdwr_rcvr_hrdwr_data

Data rate Min: 1 message per 5 seconds	This value was chosen because it is easily quantifiable and a reasonable update frequency for changing the position of the tracking antenna	This will work because the GPS module gets a position fix quicker than once per second and software can be used to slow the speed down beyond that which can be verified either using time data or LED blinking frequency.
Data rate Max: 1 message per 1 second	This value was chosen because it is easily quantifiable and a reasonable update frequency for changing the position of the tracking antenna	This will work because the GPS module gets a position fix quicker than once per second and software can be used to slow the speed down beyond that which can be verified either using time data or LED blinking frequency.
Messages = GPS Lat/Long and time data	This was chosen because the GPS module will transmit Lat/Long and time data.	This will work because the data will be formatted as C strings and can be transmitted as bytes over the SPI interface.
Protocol = LoRa	This was chosen because the RP2040 and LoRa module support SPI and it is a simple interface.	This will work because there are SPI libraries for managing data transfers for the RP2040 has libraries in the C/C++ SDK and both modules support SPI.

tx_cd_trnsmttr_hrdwr_data

Data rate = 12Khz	This value was chosen because the clock that is driving the RP2040 oscillates at 12KHz. This	This will work because SPI uses the SCLK to drive the data transfer so since the clock driving the RP2040

	will be the SCLK for the SPI protocol and will dictate how quickly data can be picked up.	will be the same clock SCLK is using it will work.
Messages = GPS Lat/Long and time data	This was chosen because the GPS module will transmit Lat/Long and time data. The data will come through the LoRa module and be pushed into the MISO port.	This will work because both the LoRa module and the RP2040 support SPI and the LoRa module can be a SPI Slave while the RP2040 will be the SPI Master and will use the RP2040 SCLK to clock the data transfer [1][4].
Protocol = SPI	This is chosen because it is the easiest way to connect to the ground station. The RP2040 has libraries in the C/C++ SDK to allow for communication through its USB port.	Libraries exist to allow the RP2040 to transmit data across USB. The board will have a USB connection.

4.3.5. Verification Process

otsd_trnsmttr_hrdwr_dcpwr

To show this interface definition is met the following tests will be undertaken,

1. Connect the VBUS pin of a RP2040 based microcontroller to a power supply.
2. Set the power supply input voltage to 4.75V.
3. During normal operation ensure that the current stays at around 75mA and does not exceed 500mA and that the device operates properly for 30 seconds.
4. Set the voltage to 5.25V and ensure that the current stays at around 75mA and does not exceed 500mA and that the device operates properly for 30 seconds.

trnsmttr_hrdwr_rvr_gps_code

To show this interface definition is met the following tests will be undertaken,

1. Connect an oscilloscope to the UART data line connecting the GPS module and the RP2040.
2. Measure the shortest time difference between bit changes over a data exchange period. The inverse of that time will be the frequency in bits per second.
3. Connect the microcontroller via USB to a computer and run PuTTY.
4. Open a connection to the port that the transceiver is connected to on PuTTY.
5. Print the contents of the GPS message to USB via stdout and observe the message contents in the terminal to ensure proper output.

tx_cd_trnsmttr_hrdwr_data

To show this interface definition is met the following tests will be undertaken,

1. Connect an oscilloscope to the SCLK line in the SPI interface and measure the frequency of the clock. The clock should be oscillating at 12MHz. The rate at which data gets transferred across the SPI interface is determined by the clock speed.
2. Connect the microcontroller via USB to a computer and run PuTTY.
3. Open a connection to the port that the transceiver is connected to on PuTTY.
4. Print the contents of the MISO message to USB via stdout and observe the message contents in the terminal.

trnsmttr_hrdwr_rcvr_hrdwr_rf

To show this interface definition is met the following tests will be undertaken,

1. Connect the receiver to the PuTTY via USB and print out the contents of what is being received from the LoRa packets.
2. View the timestamps of each message and when it is sent and ensure that the data rate falls within the proper boundaries.
3. View the contents of each message and ensure that the data is GPS with latitude and longitude information as well as time.

otsd_trnsmttr_hrdwr_rf

To show this interface definition is met the following tests will be undertaken,

1. Provide USB power to the GPS submodule.
2. Connect the IPEX Active Antenna.
3. Use PuTTY to monitor the communications over the COM port the GPS module is plugged into. This will show that the IPEX antenna is compatible and that the data is structured in NMEA sentences using the NMEA protocol.

4.3.6. References and Files Links

4.3.6.1. References

[1] “Raspberry Pi Documentation,” RP2040, 30-Nov-2022. [Online]. Available: https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html. [Accessed: 20-Jan-2023].
[2] “Voltage Regulators, 1.0 A Low-Dropout Positive, Fixed and Adjustable,” ON Semiconductor, Jan-2020. [Online]. Available: https://www.onsemi.com/pdf/datasheet/ncp1117lp-d.pdf. [Accessed: 20-Jan-2023].
[3] “Autoswitching Power Mux,” Texas Instruments, Mar-2010. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps2111a.pdf. [Accessed: 20-Jan-2023].
[4] “915MHz LoRa Transceiver Module,” www.hoperf.com, 2018. [Online]. Available: https://cdn.sparkfun.com/assets/1/5/d/6/6/RFM95CW_Specification_EN_V1.0.pdf. [Accessed: 20-Jan-2023].
[5] “GT-U7 GPS Modules.” [Online]. Available: https://images-na.ssl-images-amazon.com/images/I/91tuvtrO2jL.pdf. [Accessed: 21-Jan-2023].

4.3.6.2. Files Links

1) Transmitter Hardware Master Block Diagram: https://ibb.co/0c2jTL5
2) Transmitter Hardware KiCad Schematics/PCB Design

4.3.7. Revision Table

Author	**Date**	**Description**
Sean Bullis	1/20/23	Section creation
Sean Bullis	2/9/23	Added details on organization of how the block works and described the interfaces more thoroughly
Sean Bullis	2/9/23	Removed the large schematic and broke it down into smaller schematics of the block for clarity. Added explanations for each subcircuit.
Sean Bullis	2/9/23	Revised section 3 and 4 to match updates to circuit

Sean Bullis	2/11/23	Updated interfaces to match current flow for interfaces.
Sean Bullis	2/11/23	Revised section 2, 4, and 5 to match new block organization, added *otsd_trnscvr_hrdwr_rf* and *trnscvr_hrdwr_rvr_gps_code* and removed *trnscvr_hrdwr_tx_cd* interface.
Sean Bullis	2/11/23	Split references section into a reference and a file link and uploaded the master block diagram to the file link section
Sean Bullis	3/13/23	Cut content about receiver
Sean Bullis	3/13/23	Changed transceiver to transmitter and added section about the connections to sub-modules and Figure 5
Sean Bullis	3/13/23	Changed linear voltage regulator from NCP1117 to AZ1117
Sean Bullis	3/13/23	Change transceiver to transmitter and cut receiver interfaces
Sean Bullis	3/13/23	Updated section 5 to match new interfaces and cut receiver interfaces
Sean Bullis	5/1/23	Updated the battery schematic and added borders around images
Kira Kopcho	5/14/23	Fix revision table to match the style of the rest of the document

4.4. Receiver Hardware

4.4.1. Description

This block is the hardware that receives the modulated LoRa packet from the transmitter on the rover. The receiver will transfer GPS location data received from the LoRa modem and send that data to the receiver code block via the RP2040 microcontroller via SPI. The system uses the same hardware design to receive the transmitted messages as it is uses to send messages.

The RP2040 is driven by a 12MHz clock and uses 128Mb flash storage to store its code. The boot mode is determined using a jumper on the PCB controlling either code upload or code execution depending on the state when powered on.

The inputs to the system are 5V input from either a battery boost circuit or 5V from USB power, and input from the transmitter code block via radio frequency. The output is GPS data to the receiver code block over SPI.

4.4.2. Design

This block takes two inputs and produces one output. The input *otsd_rcvr_hrdwr_dcpwr* is the power coming from the USB or battery to the receiver. The transmitter has a built-in power switching and power regulating circuit that requires a 5V input with the capability to draw up to 500mA. The input *trnsmttr_hrdwr_rcvr_hrdwr_rf* is the transfer of data from the LoRa sub-module on the transmitter PCB through radio frequency to the receiver's LoRa sub-module. The *rcvr_hrdwr_rx_code_data* describes the data that is transferred from the LoRa modem on the receiver to the code block running on the RP2040. Figure 1 shows the black box diagram for the block.

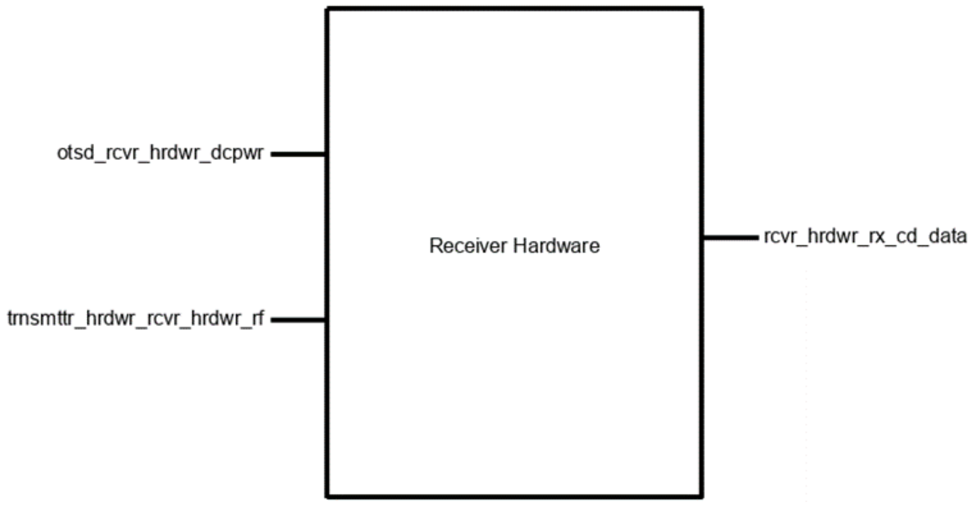


Fig. 4.4.2.1 - Black Box for Transmitter

For breakdowns of the individual pieces of this block please reference the section labeled *Design* of the transmitter hardware validation section. The hardware and its capabilities are identical.

4.4.3. General Validation

For general validation of this block please reference the section labeled *General Validation* of the transmitter hardware validation section.

4.4.4. Interface Validation

Interface Property	**Why is this interface of this value?**	**Why do you know that your design details for this block above meet or exceed each property?**

otsd_rcvr_hrdwr_dcpwr

$I_{nominal}$ = 75mA	This value is chosen based on the sum of the typical current values for each of the heavy lifting components on the board; the RP2040, the LoRa modem, and the GPS module.	The RP2040 in active mode will pull about 20mA on its own, and up to 50mA driving all its GPIO pins [1]. The LoRa modem typically draws 134mA during transmission and the GPS module typically draws 32mA during acquisition [4][5]. The memory module draws 25mA during reads/writes. The power switch pulls 55uA typical during power delivery [2]. The boost converter draws 6.5uA while operating. The transmitter will transmit periodically, and read/writes will not be frequent so the RP2040 and GPS module will be the main power sources giving approximately 55mA.

I_{peak} = 500mA	This value was chosen with the absolute maximum current ratings of each of the components in the block in mind. During a heavy load test for the RP2040 running video drives, audio drivers and SD card access (popcorn test) the uC draws 91mA. The LoRa modem pulls 140mA maximum during transmission. The GPS module pulls 40mA maximum. The memory module pulls 25mA during read/write. The power switch pulls 90uA max when delivering power. The boost converter pulls 10uA max while operating. This sums to 297mA but a conservative estimate of 200mA of headroom is provided to ensure power delivery is applied.	Each of the values for absolute maximum current draws come from the data sheets of the individual components. Summing them all and rounding up with conservative estimates for max values will provide enough for each component individually and then some. The linear voltage regulator is rated to pull 1A of current with a drop off voltage of 1.4V above the output voltage. The power switch is rated to deliver up to 1A as well. The battery boost IC is rated to deliver up to 1.8A which are well above the required nominal current.
V_{max} = 5.25	This value was chosen because it is the maximum possible input from the USB source or battery source.	This will be met because the highest output from the battery boost converter is 5.15V according to the TPS61322 datasheet and the highest output from USB is 5.25V according to standard USB specs.
V_{min} = 4.75	This value was chosen because it is the lowest possible input from the USB or battery source.	This will be met because the lowest output from the battery boost converter is 4.85V according to the TPS61322 datasheet and the lowest output from USB

		is 4.75V according to standard USB specs.
$V_{nominal}$ = 5	This is the nominal input from battery or USB.	This will be met because the typical voltage output of both the battery boost converter and USB is 5V.

trnsmttr_hrdwr_rcvr_hrdwr_data

Data rate Min: 1 message per 5 seconds	This value was chosen because it is easily quantifiable and a reasonable update frequency for changing the position of the tracking antenna.	This will work because the GPS module gets a position fix quicker than once per second and software can be used to slow the speed down beyond that which can be verified either using time data or LED blinking frequency.
Data rate Max: 1 message per 1 second	This value was chosen because it is easily quantifiable and a reasonable update frequency for changing the position of the tracking antenna	This will work because the GPS module gets a position fix quicker than once per second and software can be used to slow the speed down beyond that which can be verified either using time data or LED blinking frequency.
Messages = GPS Lat/Long and time data	This was chosen because the GPS module will transmit Lat/Long and time data.	This will work because the data will be formatted as C strings and can be transmitted as bytes over the SPI interface.
Protocol = LoRa	This was chosen because the RP2040 and LoRa module support SPI and it is a simple interface.	This will work because there are SPI libraries for managing data transfers for the RP2040 has libraries in

		the C/C++ SDK and both modules support SPI.

rcvr_hrdwr_rx_code_data

Data rate = 12Khz	This value was chosen because the clock that is driving the RP2040 oscillates at 12KHz. This will be the SCLK for the SPI protocol and will dictate how quickly data can be picked up.	This will work because SPI uses the SCLK to drive the data transfer so since the clock driving the RP2040 will be the same clock SCLK is using it will work.
Messages = GPS Lat/Long and time data	This was chosen because the GPS module will transmit Lat/Long and time data. The data will come through the LoRa module and be pushed into the MISO port.	This will work because both the LoRa module and the RP2040 support SPI and the LoRa module can be a SPI Slave while the RP2040 will be the SPI Master and will use the RP2040 SCLK to clock the data transfer [1][4].
Protocol = SPI	This is chosen because it is the easiest way to connect to the ground station. The RP2040 has libraries in the C/C++ SDK to allow for communication through its USB port.	Libraries exist to allow the RP2040 to transmit data across USB. The board will have a USB connection.

4.4.5. Verification Process

otsd_rcvr_hrdwr_dcpwr

To show this interface definition is met the following tests will be undertaken,

1. Connect the VBUS pin of a RP2040 based microcontroller to a power supply.

2. Set the power supply input voltage to 4.75V.
3. During normal operation ensure that the current stays at around 75mA and does not exceed 500mA and that the device operates properly for 30 seconds.
4. Set the voltage to 5.25V and ensure that the current stays at around 75mA and does not exceed 500mA and that the device operates properly for 30 seconds.

rcvr_hrdwr_rx_code_data

To show this interface definition is met the following tests will be undertaken,

1. Write code to flash an on-board LED when each message is sent from the code to the hardware.
2. Timing the frequency of flashes will determine how many messages are sent per second.
3. Since the data string sent will be formatted by the RX code block it can be ensured that it is of the proper contents and data type. The data can be checked on the tracking algorithm to ensure that it is received properly.

trnsmttr_hrdwr_rcvr_hrdwr_rf

To show this interface definition is met the following tests will be undertaken,

1. Connect the receiver to the PuTTY via USB and print out the contents of what is being received from the LoRa packets.
2. View the timestamps of each message and when it is sent and ensure that the data rate falls within the proper boundaries.
3. View the contents of each message and ensure that the data is GPS with latitude and longitude information as well as time.

4.4.6. References and Files Links

4.4.6.1. References

For references please see the section labeled *References and File Links* in the transmitter hardware verification section.

4.4.6.2. Files Links

For file links please see the section labeled *References and File Links* in the transmitter hardware verification section.

4.4.7. Revision Table

Author	Date	Description

Sean Bullis	3/13/23	Section Creation

4.5. Turntable Controller

4.5.1. Description

The Turntable-Controller is a PCB based block designed to process heading commands from the Tracking Algorithm and UI (user interface) blocks, receive encoder data and issue motor commands to the Turntable block. In the context of the system, the Turntable-Controller acts as motion controller and a data processor: not that it does not handle GPS data from the transmit/receive blocks or perform bearing computations. Importantly, this block allows automatic Turntable positioning with just 12 or 24 VDC power and heading commands from power-on to off.

Built around an Arduino Micro microcontroller, the block is designed for flexibility and upgradeability. Each major sub-system receives a sub-PCB to support circuits: there will be one carrier, and at most two motor drivers. This is done to reduce costly reworks if all circuits were on a single PCB and allow field upgrading or repairs of subsystems. Flexible cabling will connect sub-PCBs together within the block.

All functions such as initialization, data processing, C&C (command and control) are automatic after power up. Initialization sets the Arduino up then puts control variables and commands the Turntable block to point North. Several loops periodically check Turntable position (10 times a second), processes commands sent via UART (once a second), and issues motor control commands (1000 times a second).

4.5.2. Design

Functionality requires both hardware and software and the goal of easy integration and flexibility of block internals and interfaces drove the design of the Turntable-Controller PCB. Integration is primarily achieved through components and values already used by the OSU Robotics Club (OSURC) and secondarily through design choices such as distributing block sub-systems onto separate PCBs to accommodate sub-system requirements. Flexibility is added through components with specifications beyond basic requirements.

a) <u>Interfaces and Black Box</u>
- **trntbl_trntbl-cntrllr_comm:** accepts absolute (unique data per position) angular position of the Turntable block over SPI.
- **u_trntbl-cntrllr_data:** accepts manual bearing commands over UART.
- **trckng_lgrthm_antnn_systms_cntrllr_data:** accepts automatic bearing commands over UART.
- **otsd_trntbl-cntrllr_dcpwr:** accepts 24 VDC power from a commercial-off-the-shelf (COTS) power supply.
- **trntbl-cntrllr_trntbl_comm:** issues configuration commands to angle sensor in the Turntable block over SPI.
- **trntbl-cntrllr_trntbl_acpwr:** 12 V, 0.4 A stepper motor power to Turntable block motors.

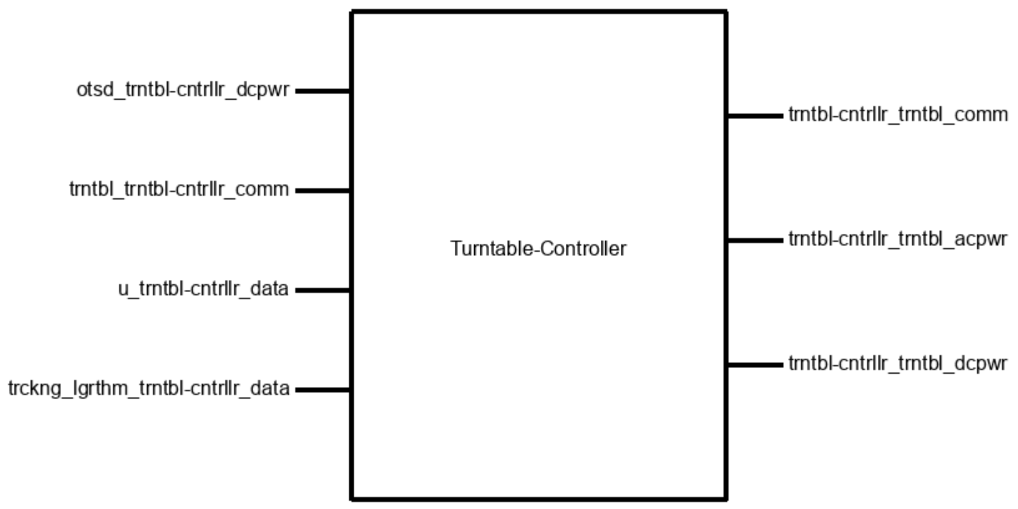

otsd_trntbl-cntrllr_dcpwr
trntbl_trntbl-cntrllr_comm
u_trntbl-cntrllr_data
trckng_lgrthm_trntbl-cntrllr_data
Turntable-Controller
trntbl-cntrllr_trntbl_comm
trntbl-cntrllr_trntbl_acpwr
trntbl-cntrllr_trntbl_dcpwr

Fig. 4.5.2.1 - Black Box

b) Design Documents

Below are, in order of appearance, are stepper motor driver (RDF_E_TB6600), Arduino carrier (RDF_E_Carrier), and carrier board sub-schematics.

The decision for separate schematics and PCBs per sub-system is elaborated on in General Validation but, as a general introduction, done to reduce risk, costs, and ease assembly, upgrades, and repairs.

Block interfaces in each schematic will be labeled with an **INTERFACE** label and will be located next to the associated connector(s). For instance, in the RDF_E_TB6600 schematic, **INTERFACE trntbl-cntrllr_trntbl_acpwr** is beside connector J3.

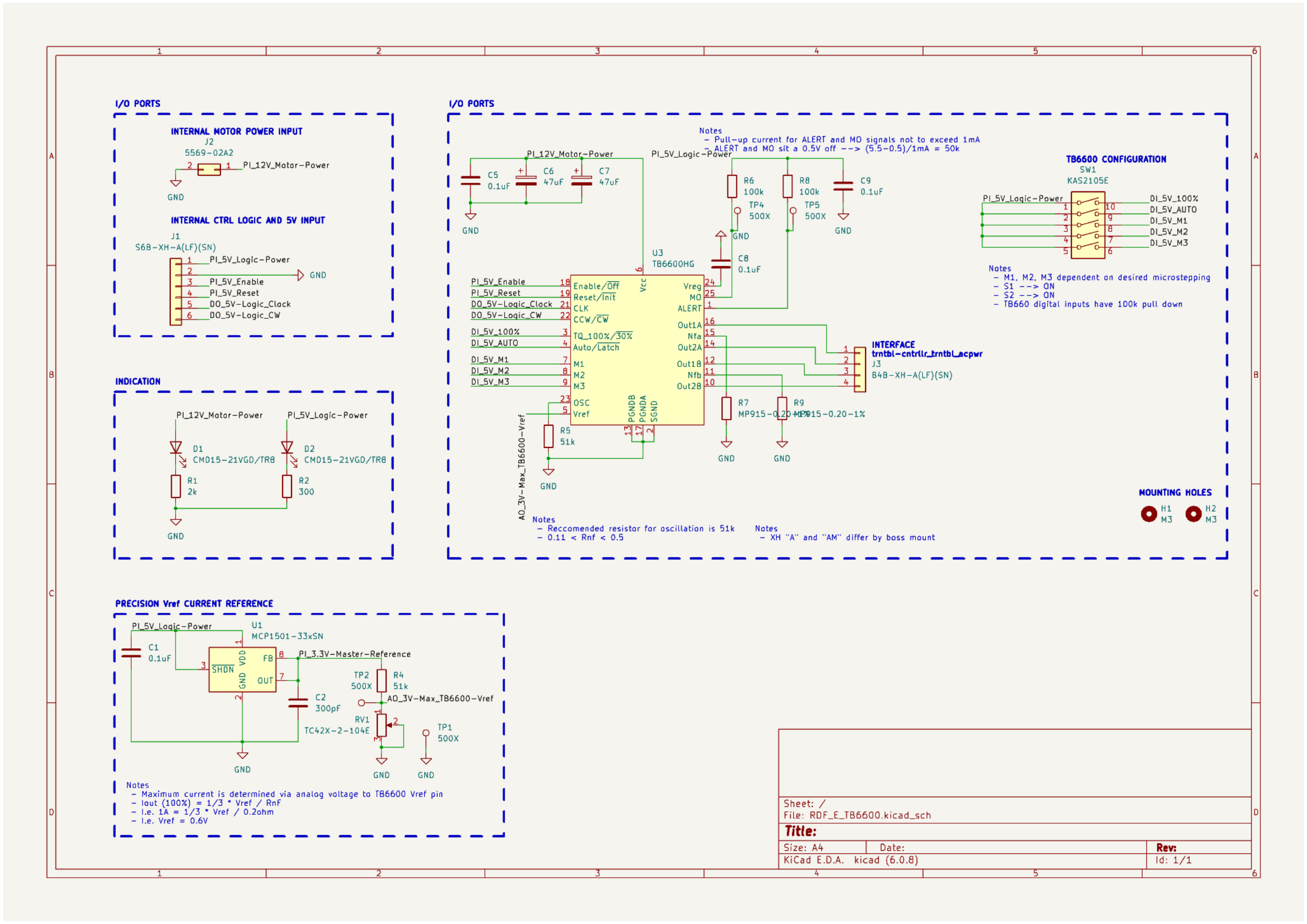


Fig. 4.5.2.2 - TB6600 Sub-PCB Schematic

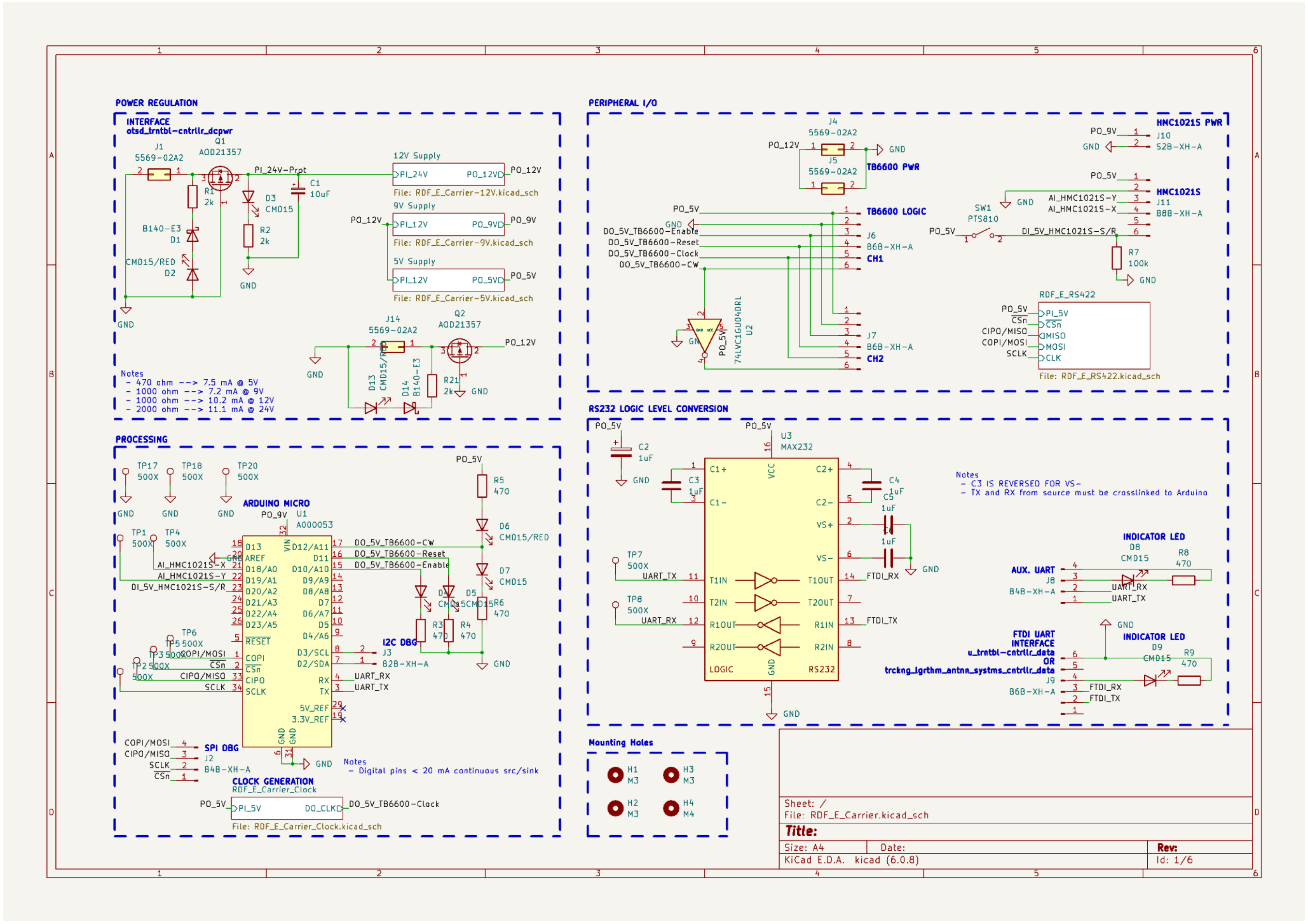

POWER REGULATION
INTERFACE
otsd_trntbl-cntrllr_dcpwr
12V Supply
9V Supply
5V Supply
Notes
- 470 ohm --> 7.5 mA @ 5V
- 1000 ohm --> 7.2 mA @ 9V
- 1000 ohm --> 10.2 mA @ 12V
- 2000 ohm --> 11.1 mA @ 24V
PERIPHERAL I/O
TB6600 PWR
TB6600 LOGIC
CH1
CH2
HMC1021S PWR
HMC1021S
RDF_E_RS422
PROCESSING
ARDUINO MICRO
I2C DBG
SPI DBG
CLOCK GENERATION
Notes
- Digital pins < 20 mA continuous src/sink
RS232 LOGIC LEVEL CONVERSION
Notes
- C3 IS REVERSED FOR VS-
- TX and RX from source must be crosslinked to Arduino
INDICATOR LED
AUX. UART
FTDI UART INTERFACE
Mounting Holes
Sheet: /
File: RDF_E_Carrier.kicad_sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (6.0.8)
Rev:
Id: 1/6

Fig. 4.5.2.3 - Arduino Carrier Board Schematic

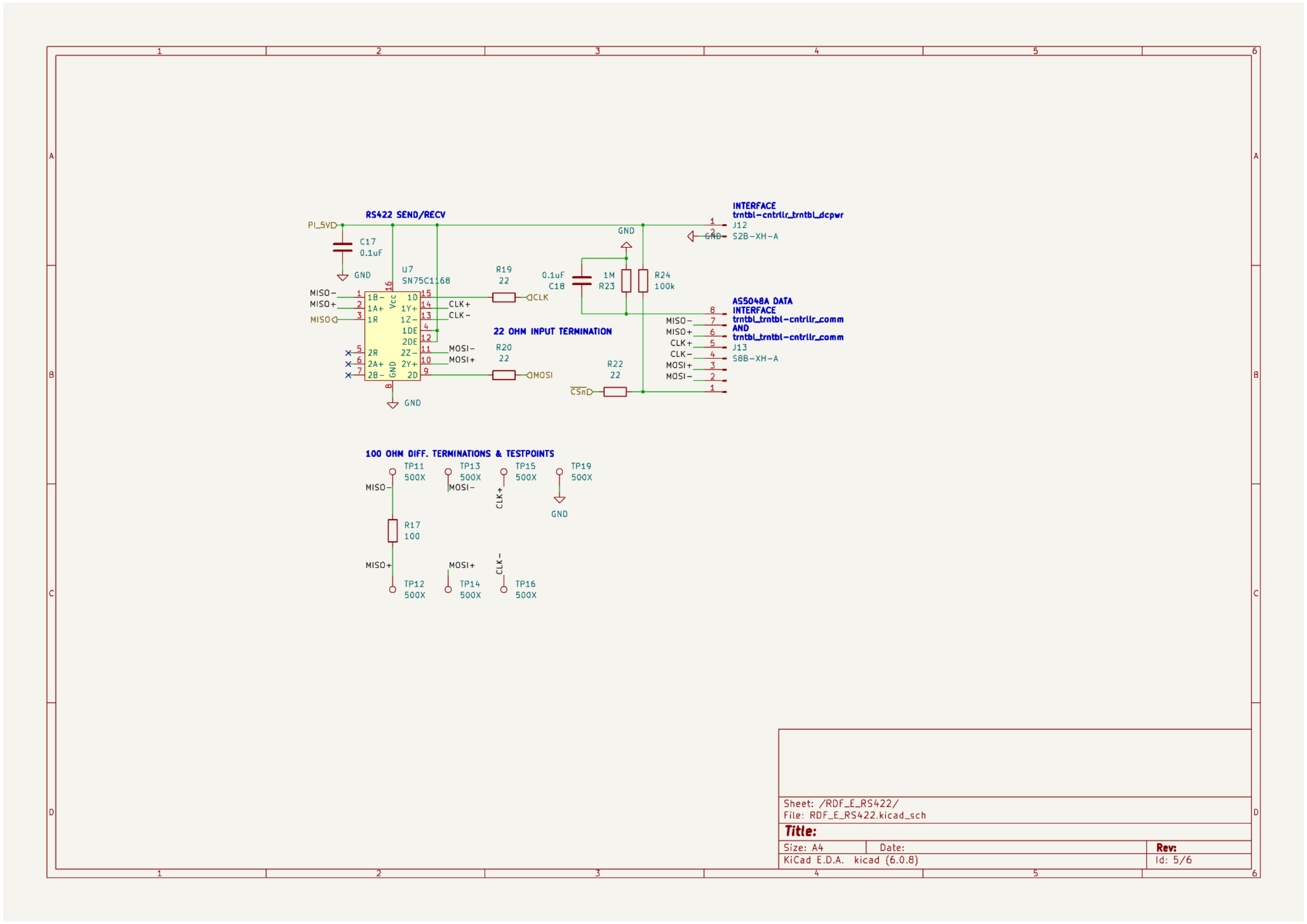


Fig. 4.5.2.4 - RS422 Sub-Circuit Schematic

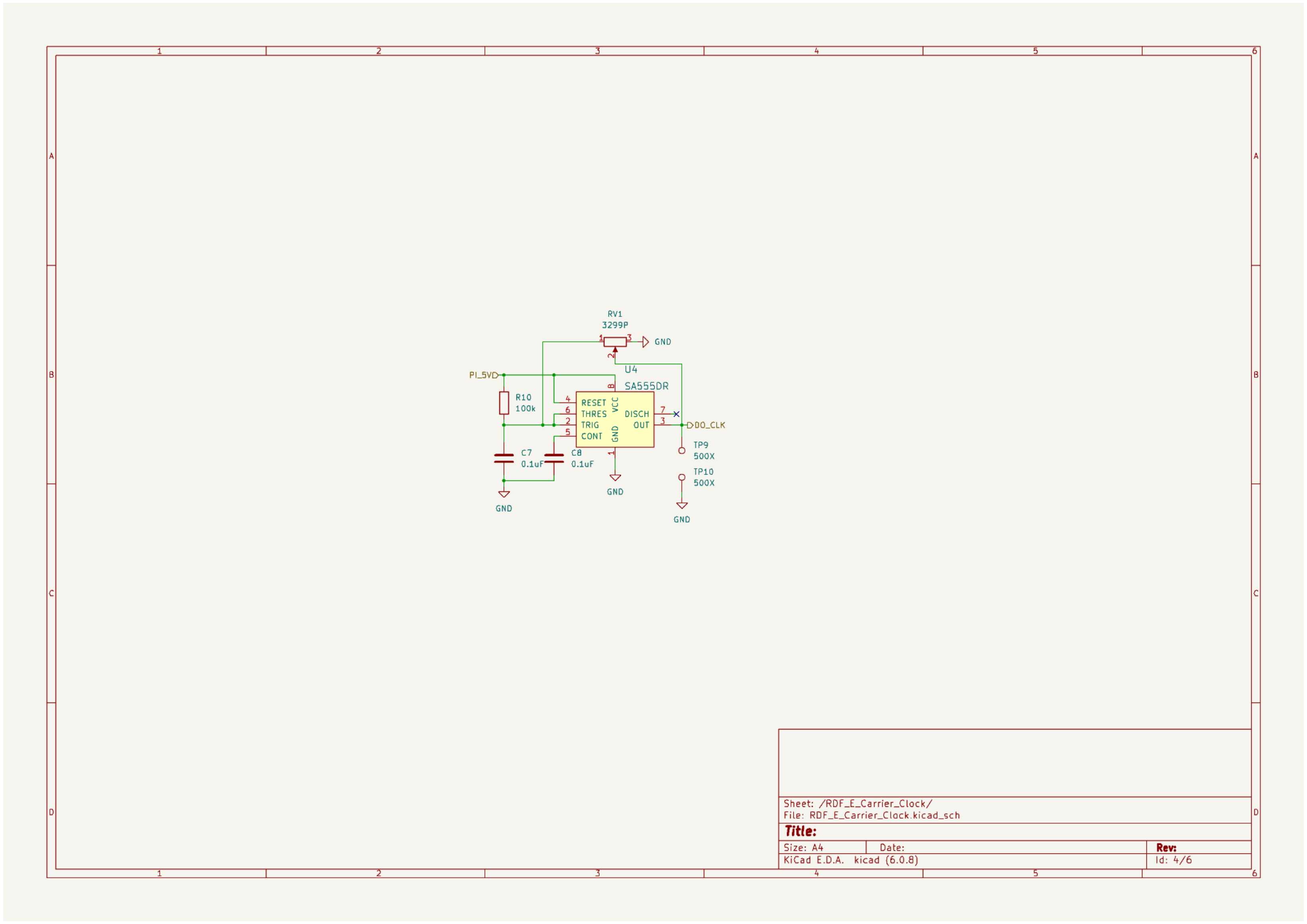
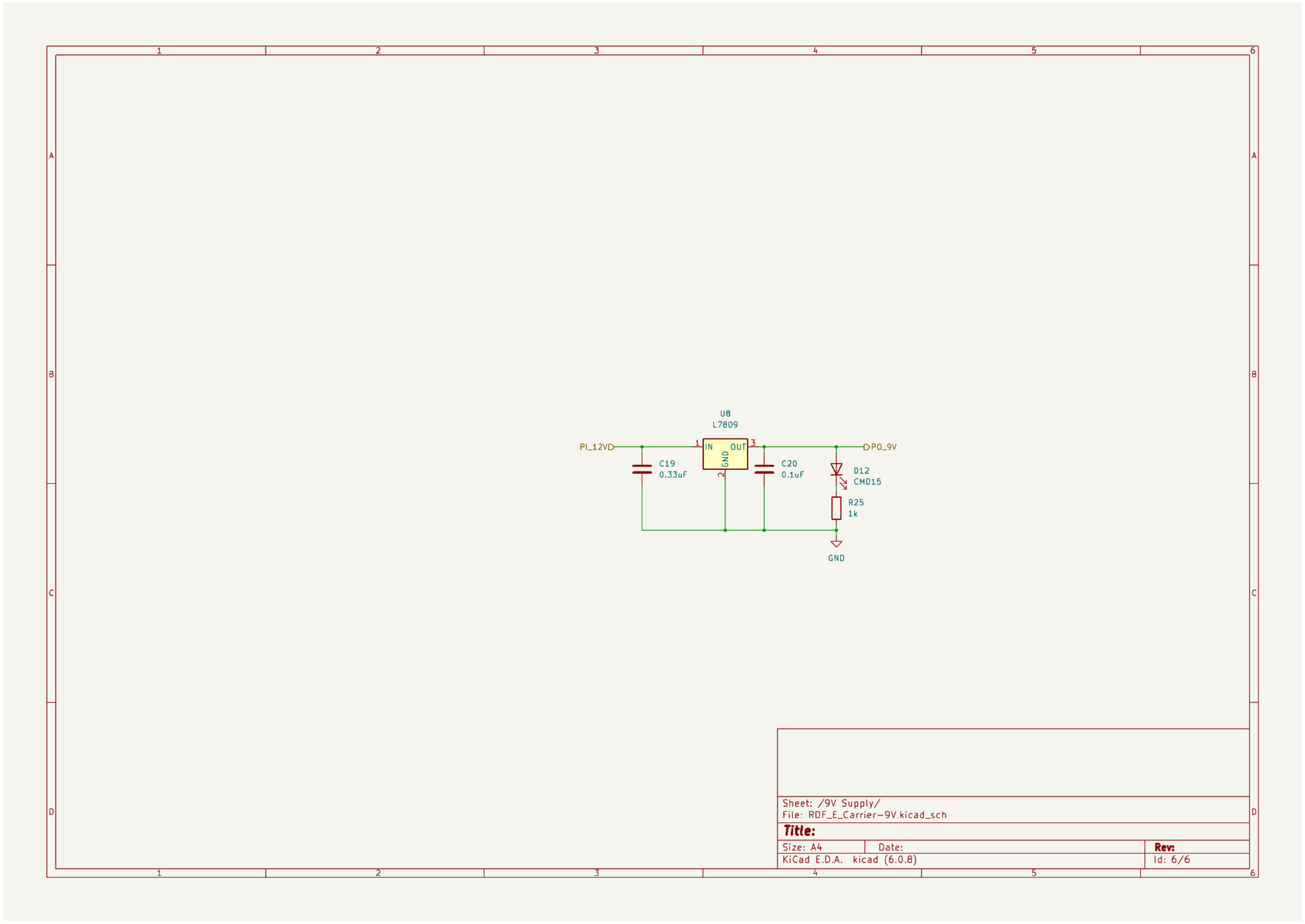
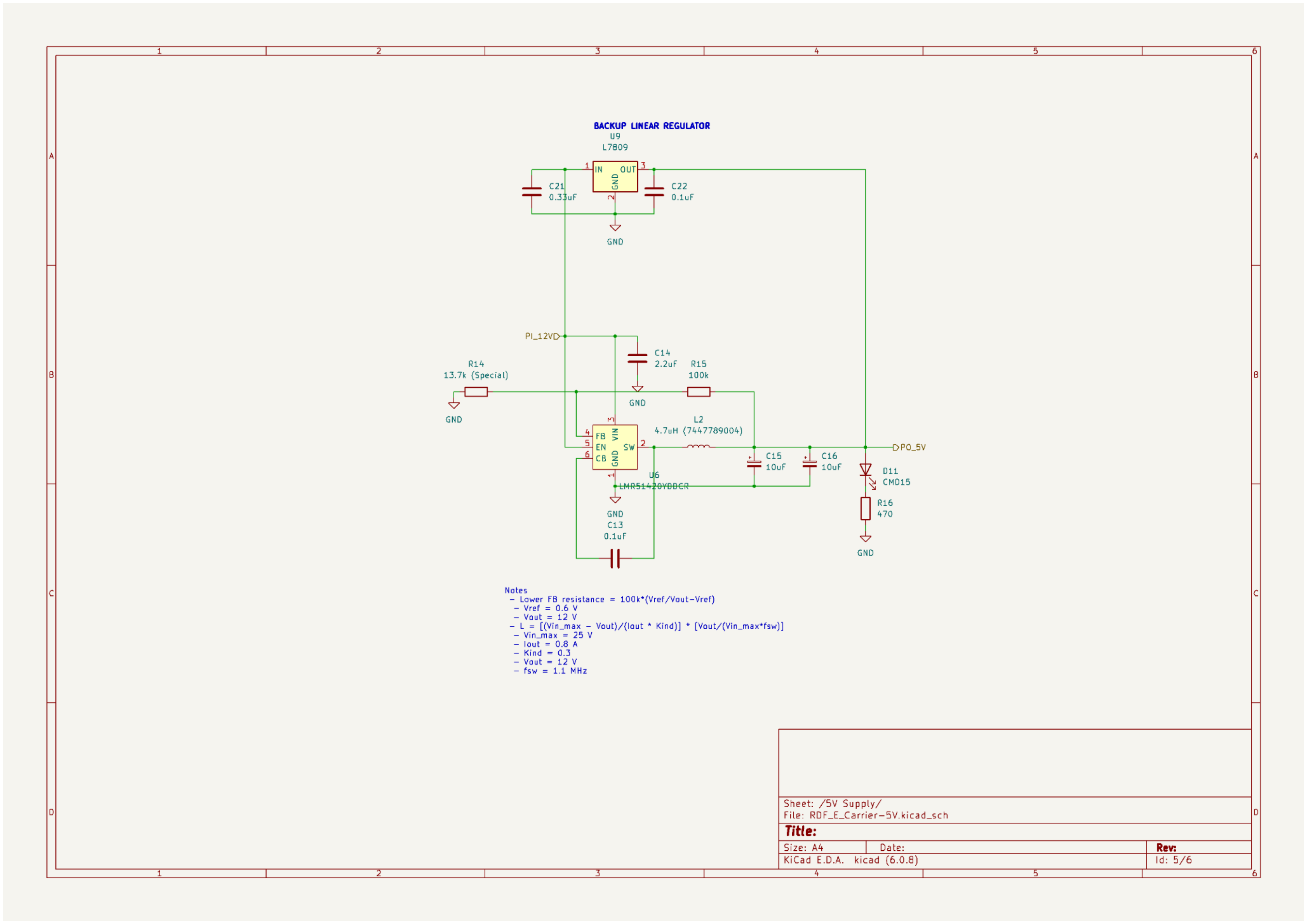


Fig. 4.5.2.5 - 555 Timer Sub-Circuit Schematic



Sheet: /9V Supply/
File: RDF_E_Carrier-9V.kicad_sch
Title:
Size: A4 Date:
KiCad E.D.A. kicad (6.0.8)
Rev:
Id: 6/6

Fig. 4.5.2.6 - 9 V Regulator



Notes
- Lower FB resistance = 100k*(Vref/Vout-Vref)
  - Vref = 0.6 V
  - Vout = 12 V
- L = [(Vin_max - Vout)/(Iout * Kind)] * [Vout/(Vin_max*fsw)]
  - Vin_max = 25 V
  - Iout = 0.8 A
  - Kind = 0.3
  - Vout = 12 V
  - fsw = 1.1 MHz

Fig. 4.5.2.7 - 5 V Regulator

There are two potentiometers in the block. One controls the maximum output current of the TB6600 sub-PCB and is calculated using using equation $I_{out} = (⅓ * V_{ref} / R_{nF})$ [TB6600, pg. 7]. Turntable motors will be operated at 0.4 A, to minimize heating, so $I_{out} = (⅓ * V_{ref} / R_{nF})$ is solved using $I_{out} = 0.4$ A and $R_{nF} = 0.2\ \Omega$ to obtain $V_{ref} = 0.24$ V. V_{ref} must be set before applying 12 V power to the sub-PCB.

The other controls the clock frequency of the Arduino carrier board PCB 555 timer. This can be adjusted during operation with an oscilloscope.

Additionally, there is a DIP switch with 5 toggles controlling TB6600 configuration [TB6600, pg. 4, 5, 13]. Circuits 1 and 2 should be set on for 100% torque output and automatic return from over current or temperature exception. Remaining three control microstepping ratio and configured according to TB6600 datasheet and desired Turntable block performance.

All wiring will use Molex and JST (Japan Solderless Terminal) crimp connectors: crimp (contact is physically pressed onto wire) connectors are used to allow repair and rework. Wires without a specified type will use general purpose hookup wire of the specified AWG.

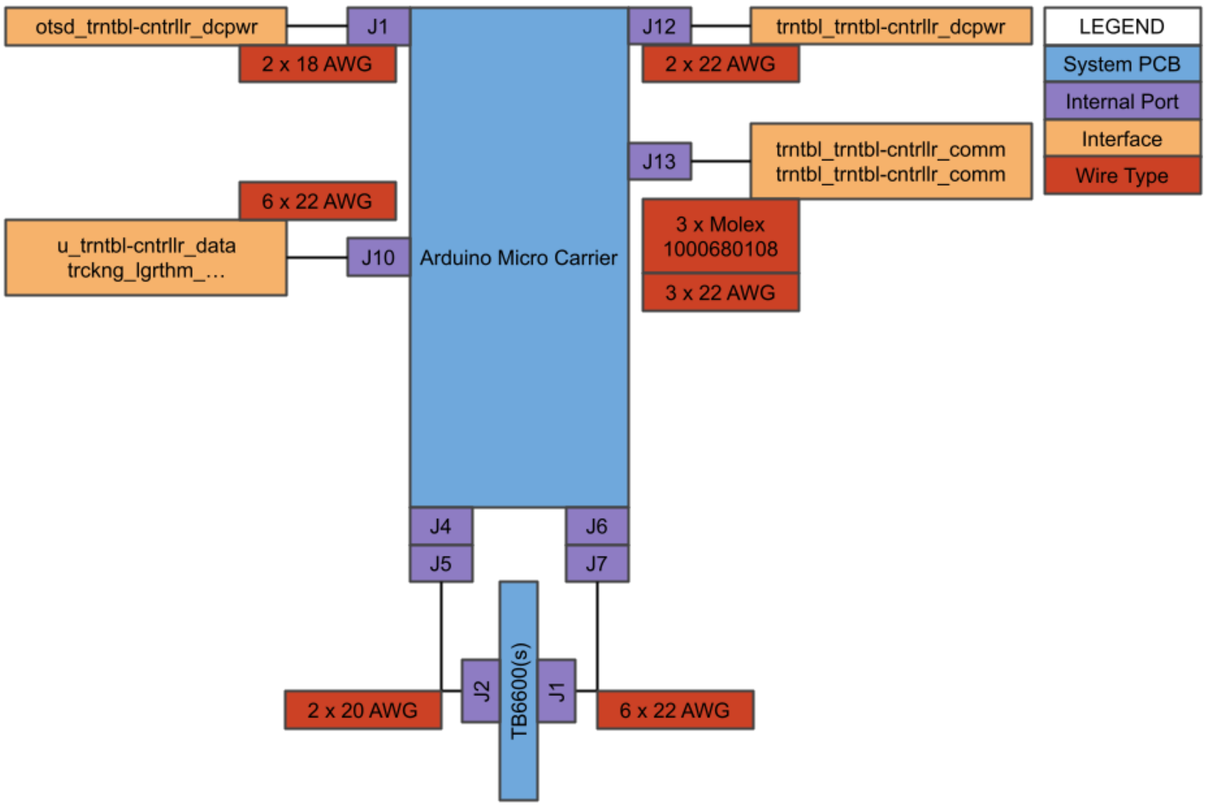


Fig. 4.5.2.8 - lock Internal Wiring Harness

While this block is an electrical block, software is needed to handle data input and control outputs. It is written in Arduino C/C++ via the Arduino IDE and deployed to an Arduino Micro via USB Micro cable. The software performs all setup, processing, and shutdown functions without manual control.

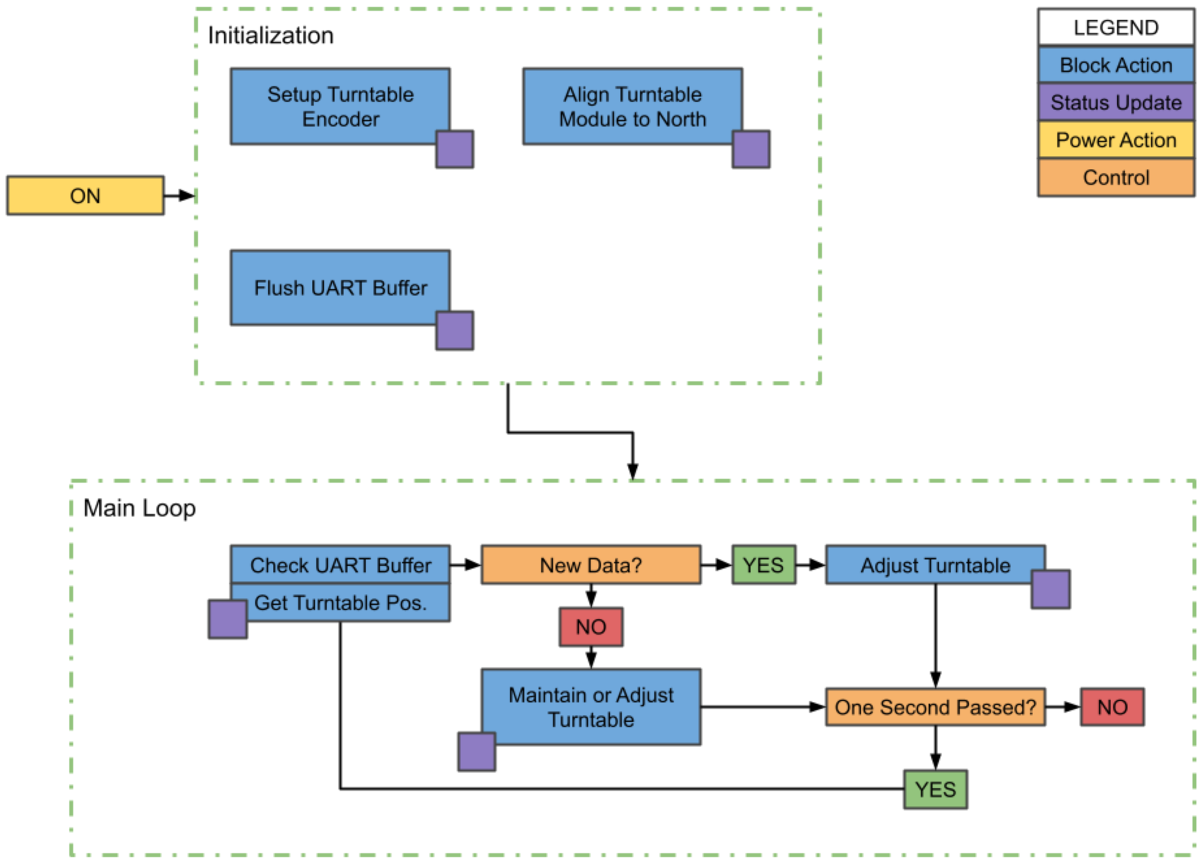


Fig. 4.5.2.9 - Processing and Decision Making Flowchart

4.5.3. General Validation

A distinct design choice is the usage of sub-PCBs per each sub-system. This layout does increase individual PCB count within the block but is done to reduce risk, cost, and improve block performance, reworkability. Crucially, a single PCB is a single failure point: if the motor driver circuit is defective then the entire board needs to be reordered. For performance, separating motor drivers from SPI data lines prevents EMI (electromagnetic interference) issues.

Reworking sub-PCBs is much easier. Flexible crimped cables link each sub-PCB to the Arduino carrier board and can be removed as needed. Sub-PCBs can be replaced, upgraded, removed, or added without block downtime.

Each PCB is designed in KiCAD which sets defaults for trace widths, via diameters, et cetera. Each parameter was evaluated and adjusted to meet IC, block interna, and interface parameters: generally, the default case is sufficient. Of particular note is the usage of 1 ounce copper content even in high power situations which is validated by ensuring the narrowest point does not exceed the minimum trace required for a desired current.

Components are SMD (surface mounted device) mounting type when possible though current supply chain issues do occasionally require through hole. SOIC (small outline integrated circuit) and 0603 (0.060” x 0.030”) packages are used to reduce component footprint and shrink PCB sizes. Through hole capacitors are constrained to a 5 mm diameter and 2 mm lead pitch (separation between pins). All passive components have a 50 V minimum to prevent destruction during brief overloads.

Connectors are standardized to the JST (Japan Solderless Terminal) XH and Molex 5569 series of rectangular crimp connectors because both are currently or recently used by OSURC. This eliminates the need to train future members and provides a source of stock if supply issues arise.

With supply, the majority of passive components have in-stock alternatives. Specialized components such as the MCP1501-33xSN, 555 timer, trim potentiometers have compatible alternatives but some such as the TB6600 driver cannot be replaced.

a) TB6600 Sub-PCB

Driving the stepper motors of the Turntable block is achieved using the Toshiba Semiconductor and Storage TB6600HG. It is selected for its range of 8 - 42 VDC input and 4.5A output maximum in order to support steppers of all sizes [TB6600, pg. 26]. The HZIP-25 through hole package is a non-issue because of its heat dissipation (3.2 W) [TB6600, pg. 26] and the through holes simplifying sub-PCB layout. Finally, the TB6600 uses few external components [TB6600, pg. 31], and is controlled through just direction and clock inputs [TB6600, pg. 30]: this makes design and control easier.

Noise output, microstepping (division of one step into N steps) were not important. Noise is expected to be covered up by outdoor ambience and no noise requirements are design parameters. A minimum microstepping ratio of 1/16 is available [TB6600, pg. 7] and allows a 0.1125° step which sufficiently satisfies the “System Accuracy” Engineering Requirement (ER).

For the TB6600, a sub-PCB allows advantageous positioning to maximize cooling and EMI onto other block sub-systems. PCB costs were reduced with horizontal positioning of the HZIP-25.

Control of output maximum current is defined by an analog voltage at the TB6600 V_{ref} pin, current sensing resistor, and requires no additional communication circuitry or software. For I_{out} adjustment, the Microchip Technology MCP1501-33xSN generates a precision 3.3 V [MCP1501-33xSN, pg. 3] reference that is divided by 51 kΩ resistor and 100 kΩ potentiometer network for a range of 0 - 2.185 V. V_{ref} . This does exceed the TB6600 V_{ref} 1.95 V limit [TB6600, pg. 26] but is verified in a post-assembly calibration procedure with the TB6600 off. To prevent change during block install , the potentiometer will be fixed using temporary adhesives.

Components associated with power handling i.e. the current sense resistors and motor power input plug are rated at or above operational conditions. The sense resistors are MP915-0.20-1% TO-126 resistors with a maximum dissipation of 15 W [MP915-0.20-1%, pg. 2]. Intended current is 0.4 A per channel so by $P = I^2R$ each resistors will see $P = 0.4^2*0.2$ for 0.32

W of heat. The TB6600 indicates a total maximum on-resistance of 0.6 Ω which, at 0.4 A, is 0.096 W: maximum no heatsink dissipation is 3.2 W.

For connectors, the Molex 5569 in 2 pin configuration supports 600 V and 13 A per contact [Molex 5569, pg. 2]. Lower voltage and current connectors use the Japan Solderless Terminals XH series of connectors which support 250 V and 3 A per contact [JST XH, pg. 1]. These components are used by OSURC which can provide component stock.

On the PCB, solid power and GND (0 V or ground) planes are used to support high currents and reduce EMI and since high impedances (resistance to changing signals) could degrade TB6600 performance. Large power planes also distribute heat under high power scenarios. All lower power control signals use a standard 0.25 mm width trace to handle 0.8 A of current while motor power lines use a 0.7 mm width trace to handle 1.8 A of current. A 1 ounce of copper content will be used to save costs and copper resources and will sustain 1.8 A currents.

b) Arduino Carrier Board

The choice for Arduino Micro is for its simplicity and capability. A low power draw over USB or 7 - 12 V simplifies power circuitry [1]. For inputs and outputs, the Arduino supports 12 ADC inputs with 5 V range, 2 SPI, 1 I^{2}C, and 20 digital input/output with 5 V logic levels which is sufficient for all block internal and interface connections [1].

A 28 KB program and 2.5 KB SRAM memories provide sufficient space for program and variables [1]. Largest variables will be IEEE floats with size of 4 bytes: the 2.5 KB SRAM can handle 625 float variables. The 28 KB program memory limit will not be reached because loaded code will only perform decision making without storing large arrays.

As an Arduino product, the Arduino IDE, built-in and 3rd party libraries, and easy deployment simplify the software workflow. For instance, libraries for reading angle data from an AS5048A (on the Turntable block) exist [2] and eliminate the need for custom functions.

c) 5V, 9V Regulator

The low voltage, low current requirements of all sub-PCBs allows usage of linear regulators instead of complex switching regulators. STMicroelectronics's L78S family is chosen for output currents of 2 A in an easy to use TO-220 package [L78SXX, pg. 9, 11, 13]. Output ripple and deviation, which may be larger with linear regulators, is not critical since accurate voltages (i.e. the 3.3 V references) are generated with dedicated reference ICs.

Higher power buses such as the 12 V line are stabilized using bulk capacitance. These will reduce ripple and transients during sudden demands in power such as rapid commands to the stepper motor drivers or during system startup

.

d) UART Receiver

Data from the OSURC groundstation is sent through UART (Universal Asynchronous Receiver/Transmitter) using the RS232 standard.

UART is natively supported by the Arduino thus requiring minimal software overhead to receive data. A UART baud rate of 9600 bits per second is sufficient for one float variable of 32 bits every one second at maximum command speed. This correlates to 3.3 mSec per command so no communication bottlenecks will occur.

However, the RS232 electrical parameters use voltages in the range 5 - 15 V [3] but are level shifted using the Texas Instruments MAX232. This IC is chosen for its built-in charge pumps (a device that increases voltage) [MAX232, pg. 1], minimal external component count, and ease of design [MAX232, pg. 10].

e) RS422 Transceiver

Angle data reported from the Turntable block is sent over SPI that is physically transmitted with the RS422 electrical standard. The RS422 standard is implemented to reduce the effects of EMI from motors and the overall system antenna through differential signals [4] . To drive and receive, the Texas Instrument SN75C1168N provides two drivers and receivers for the two output and one input signals of SPI [SN75C1168N, pg. 3].

However, the SN75C1168N is designed to use transmission cables of 100 Ω impedance (resistance to AC signal). This is satisfied by the Molex 1000680108 which is a twinaxial (two side-by-side cables) cable of 28 AWG per conductor [1000680108, pg. 1]. For harness assembly, the 27.5 AWG will fit into the JST SXH-001T-P0.6 connector that accommodates 22 - 28 AWG following the JST standard of OSURC [SXH-001T-P0.6, pg. 2].

A manufactured validated application circuit is combined with industry practices to create the RS422 sub-circuit and reduce design errors and risk. This includes the series 22 Ω on output signals and parallel 100 Ω resistors to terminate differential signals [5]. Transmission speed will be constrained by the Arduino to exceed SN75C1168N performance which is computed to be about 50 MHz maximum from 20 ns total for rise and fall [SN75C1168N, pg. 7]. Reducing clock speed is acceptable since the AS5048A supports lower clock speeds [AS5048A, pg. 12].

f) Clock Generation

Clock generation is through the ubiquitous 555 timer. A simple 50 % duty cycle (ratio of on to off) is achieved through resistors, capacitors, and potentiometer to adjust output frequency . Validation was done through falstad circuit simulator [Sim File].

4.5.4. Interface Validation

Interface: **trntbl_trntbl-cntrllr_comm**

Note that references to AS5048A are purely for determining values. The AS5048A is part of the Turntable actuator block.

Messages: Data: 16-bit data package containing 14-bit angle data	Received message size is referenced from AS5048A SPI interface characteristics.	- Turntable block AS5048A reports data in a 16-bit data package (AS5048A, SPI Read Package, fig. 20, pg. 15). - Bit 13:0 is the 14-bit data - Other bits are for control
Other: RS422 Differential Logic High	This is the positive and negativeV_{OH} threshold of the SN75C1168N.	- SN75C1168N reports a minimum required V_{OH} of 2V inverting (SN75C1168N, Driver Section, Electrical Characteristics, pg. 6).

Vmin: +/-2V		
Protocol: 4 Wire SPI (data protocol)	The AS5048A variant uses an SPI for commands and outputs.	- SPI is chosen for control features at the cost of additional input/output hardware. - Dedicated input and output prevents collisions. - Chip select allows multiple peripherals on one SPI network.

Interface: **trntbl_trntbl-cntrllr_comm**

Note that references to AS5048A are purely for determining values. The AS5048A is part of the Turntable actuator block.

Messages: Data: 16-bit data package containing address and read/write command	Received message size is referenced from AS5048A SPI interface characteristics.	- Turntable block AS5048A receives commands in a 16-bit data package (AS5048A, SPI Command Package, fig. 19, pg. 14). - Bit 15 (MSB) is the even parity bit - Bit 14 is read (1) or write (0) command. - Bit 13:0 is the 14-bit address to read from or write to.
Other: RS422 Differential Logic High Vmin: +/-2V	This is the positive and negativeV_{OH} threshold of the SN75C1168N.	- SN75C1168N reports a minimum required V_{OH} of 2V inverting (SN75C1168N, Driver Section, Electrical Characteristics, pg. 6).
Protocol: 4 Wire SPI (data protocol)	The AS5048A variant uses an SPI for commands and outputs.	- SPI is chosen for control features at the cost of additional input/output hardware. - Dedicated input and output prevents collisions. - Chip select allows multiple peripherals on one SPI network.

Interface: **trntbl-cntrllr_trntbl_acpwr**

Inominal: 0.4A (per stepper positive channel)	Turntable motor is rated for 0.4 A per coil.	- Turntable stepper motor 17HS15-1504S is rated for 1.5 A [6]. - TB6600 can output 5 A of maximum current (TB6600, Electrical Characteristics, pg. 27)
Vnominal: 12V (per stepper positive channel)	Turntable motor is rated for 12 V but can accept higher and lower voltages.	- Turntable stepper motor 17HS15-1504S has been tested at 24 V. [7]. - Will use operational voltage of 12V and coil current 0.4A for safe and low heat operation.

Vmax: 14V	Turntable	- LRS-50-12 has a maximum output of 13.8 V (LRS-50-12, Specification, pg. 2) which could be sent 1:1 through the TB6600.
Other: Imax: 1A	Derated value from 17HS15-1504S maximum of 1.5 A continuous.	- Turntable stepper motor 17HS15-1504S is rated for 1.5 A [6].

Interface: **trntbl-cntrllr_trntbl_dcpwr**

Note that references to AS5048A are purely for determining values. The AS5048A is part of the Turntable actuator block.

Inominal: 21mA	Computed by summing operational current requirements of circuits connected to this interface.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - SN75C1168N in DIP-16 with max draw of 6mA for just logic (SN75C1168N, Electrical Characteristics, pg. 6)
Ipeak: 215mA	Computed by summing operational current requirements of circuits connected to this interface.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - SN75C1168N in DIP-16 with max draw of 200mA for logic and output (SN75C1168N, Absolute Maximum Ratings, pg. 5)
Vmax: 5.5V	Determined from AS5048A datasheet.	- AS5048A reports a maximum of 5.5V to V_{CC} (AS5048A, Operating Conditions, fig.8, pg 7). - SN75C1168N reports a maximum of 5.5V to V_{CC} (SN75C1168N, Recommended Operating Conditions, pg. 5)
Vmin: 4.5V	Determined from AS5048A datasheet.	- AS5048A reports a minimum of 4.5V to V_{CC} (AS5048A, Operating Conditions, fig.8, pg 7). - SN75C1168N reports a minimum of 4.5V_{CC} (SN75C1168N, Recommended Operating Conditions, pg. 5)
Vnominal: 5V	Determined from AS5048A datasheet.	- AS5048A accepts a range of 4.5V to 5.5V (AS5048A, Operating Conditions, fig.8, pg 7). - SN75C1168N accepts a range of 4.5V to 5.5V (SN75C1168N, Recommended Operating Conditions, pg. 5)

Interface: **u_trntbl-cntrllr_data**

Datarate:	The user interface is	- Each message will be held in the UART

Message Rate Max: Unlimited messages every 1 second	controlled by a human being who may be impatient or need rapid changes.	buffer. - Processing occurs every 1 seconds and flushes the buffer to prevent overloading.
Messages: Angle: Must be from 0.0 - 360.0 degrees	Bearing data is typically represented as a numerical value. The user interface is more intuitive if decimal numbers are used.	- System will operate using degrees relative to longitude lines. - Degrees is an easily understandable representation of angular position. - Arduino control algorithm rejects out-of-bounds values.

Interface: **trckng_lgrthm_antnn_systms_cntrllr_data**

Datarate: Message Rate Max: 1 message every 1 second	Command rate is limited to save groundstation and Arduino resources.	- The Arduino will be programmed to expect a new message 1 seconds after the last message.
Datarate: Serial Transmission Rate: 9600 baud	A data rate defines command transfer rate.	- 9600 bits per second is 1200 bytes per second. Thus, a 32-bit float takes 3.3 milliseconds to transmit.
Datarate: Message Rate Min: 1 message every 5 seconds	Command rate has a minimum rate to allow reasonably fast tracking of the OSURC rover.	- The Arduino will hold the last provided heading until a new message is received.
Messages: Angle: Must be from 0.0 - 360.0 degrees	Bearing data is typically represented as a numerical value. The user interface is more intuitive if decimal numbers are used.	- System will operate using degrees relative to longitude lines. - Degrees is an easily understandable representation of angular position. - Arduino control algorithm rejects out-of-bounds values.
Protocol: UART Serial	Transferred commands need a common protocol between groundstation and Arduino.	- The Arduino supports native supports UART as part of the SoftwareSerial() library. - The groundstation will use a USB - UART converter with configurable baud rate [USB-RS232-WE-1800-BT_5.0, pg. 5].

Interface: **Otsd_trntbl-cntrllr_dcpwr**

Inominal: 1A	Computed by summing the typical operation rating of all integrated circuits.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - TB6600 in HZIP-25 package with draw of 0.4 A per motor coil over four coils for a total of 1.6 A. Additional draw of 4.2mA with RESET enabled in 1/1 step mode (TB6600, Electrical Characteristics, pg. 27). - LMV324 in SOIC-14 with max draw of 1.160 mA for logic (LMV324, Electrical Characteristics, pg. 6). - Typical supply current not listed so will use maximum - Two SN75C1168N in DIP-16 with max draw of 12 mA for just logic (SN75C1168N, Electrical Characteristics, pg. 6) - Arduino Micro with max draw of 500 mA (USB standard) [1] - Quiescent currents of L78S regulators sum to 24 mA (L78SXX, Electrical Characteristics, pg. 9 - 13)
Ipeak: 2.2	Computed by summing the “worst-case” ratings.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - TB6600 in HZIP-25 is limited to 0.4 per coil over four coils for 1.6 A (TB6600, Electrical Characteristics, pg. 27). - LMV324 in SOIC-14 with max draw of 1.160 mA for logic (LMV324, Electrical Characteristics, pg. 6). - Typical supply current not listed so will use maximum - Two SN75C1168N in DIP-16 with max draw of 200 mA each (SN75C1168N, Absolute Maximum Ratings, pg. 5) - Arduino Micro with max draw of 500 mA (USB standard) [1]
Vmax: 13	Determined from the maximum output of selected AC-DC converter	- LRS-50-12 has a maximum output of 13.8 V (LRS-50-12, Specification, pg. 2)
Vmin: 11	Determined from the lowest acceptable input voltage in the block.	- TB6600 lists 8V as the absolute minimum input (TB6600, Operating Range, pg. 26) - LRS-50-12 has minimum output of 10.2 V (LRS-50-12, Specification, pg. 2)
Vnominal:	Common supply voltage.	Stepper motors are driven at 12V nominally

12		Common supply voltage

4.5.5. Verification Process

Interface under test: **otsd_trntbl-cntrllr_dcpwr**

Testing Materials:

- Turntable-Controller Block
- Electronic Load
- DC Supply
- Multimeter
- Power Leads
- Multimeter Leads

1. Connect DC Supply to Turntable-Controller
2. Adjust DC Supply to 12 V output
    a. Verify that motor control, encoder data, UART function
    b. If operations are successful then this verifies nominal input range
3. Adjust DC Supply to 11 V output
4. Adjust DC Supply to 13 V output
    a. Verify that motor control, encoder data, UART function
    b. If operations are successful then this verifies voltage input range
5. Observe current draw at all conditions
    a. If below 2.2 A then this verifies operation without exceeding maximum current
    b. If below or around 1 A then this verifies operation at nominal current

---

Interface under test: **trntbl_trntbl-cntrllr_comm** and **trntbl-cntrllr_trntbl_comm**

Testing Materials:

- Turntable-Controller Block
- AS5048A Encoder Board of Turntable Block
- DC Supply
- Power Leads
- Oscilloscope
- Probe Leads
- Computer with Arduino Serial Monitor
- USB A to USB Micro B

1. Connect Computer to Arduino
2. Apply 12 V to block
3. Watch Serial Monitor
    a. 1st message will be command to AS5048A in 16-bit string
    b. 2nd message will be data received from AS5048A in 16-bit string
    c. If messages are displayed and received then 16-bit data package size is verified

4. Attach oscilloscope leads to SPI + and SPI - signals (CLI, MISO, MOSI)
    a. If magnitude of voltages are > 2 V then RS422 logic level is verified
5. Attach oscilloscope leads to SPI test points (CLK, MISO, MOSI, ~CSn)
6. Configure oscilloscope to decode SPI
    a. 16-bit length
    b. Read on rising of clock
    c. Inverted CSn
7. Observe decoded data
    a. If MOSI is 0x1111111111111111 then this verifies command 16-bit string
    b. If MSIO is 16-bit (data varies) then this verifies data 16-bit string
    c. If both observed then this also verifies SPI interface in use

---

Interface under test: **trntbl-cntrllr_trntbl_acpwr**

Testing Materials:

- Turntable-Controller Block
- 17HS15-1504S Stepper Motor and Terminal Blocks
- DC Supply
- Multimeter
- Oscilloscope
- Multimeter Leads
- Oscilloscope Leads

1. Connect 17HS15-1504S to block
2. Set multimeter to current mode
3. Attach multimeter leads between 17HS15-1504S Channel A+ and TB6600 subsystem J1, Pin 1
    a. Use test terminal block as breakout
4. Apply 12 V
    a. If current on channel is nominally 0.4 A then this verifies nominal channel current
5. Adjust JV1 of TB6600 sub-system
    a. Adjust until the current reaches 1.0 A.
    b. Allow current to remain steady for 1 minute
    c. If current holds at 1.0 A then this verifies maximum channel current
6. Stop 12 V
7. Remove multimeter leads and reattach cabling between Channel A+ and TB6600 sub-system J1.
8. Attach Oscilloscope (+) lead to Channel A+, (-) lead to Channel A-
9. Apply 12 V
    a. If voltage is averaged 12 V then this verifies nominal channel voltage
    b. If voltage is < 14 V then this verifies maximum channel voltage

---

Interface under test: **trntbl-cntrllr_trntbl_dcpwr**

Testing Materials:

- Turntable-Controller Block
- Electronic Load
- DC Supply
- Multimeter
- Multimeter Leads
- Power Leads

1. Attach DC load to 5 V output.
2. Attach multimeter leads to 5 V output.
3. Apply 12 V to block
4. Set DC load to 21 mA
    a. If voltage remains 5 V nominally then nominal current verified
    b. If voltage is between 4.5 and 5.5 V then voltage verified
5. Set DC load to 215 mA
    a. If voltage remains 5 V nominally then maximum current verified

---

Interface under test: **u_trntbl-cntrllr_data** and **trckng_lgrthm_trntbl-cntrllr_data**

Testing Materials:

- Turntable-Controller Block
- Computer with Arduino Serial Monitor and PUTTY
- Timer
- USB A to USB Micro B
- FTDI Cable (USB-RS232-WE-1800-BT_5.0)

1. Connect Computer to Arduino
2. Connect Serial Monitor
3. Send numerical decimal (i.e. 100.0) value
    a. Arduino Serial Monitor will display 1st message details
    b. Send two numerical values 1 second apart
    c. Arduino Serial Monitor will display 1st message details
    d. Arduino Serial Monitor will display 2nd message details
    e. If messages correctly update to show 1st and 2nd messages 1 second apart then this verifies nominal data rate of 1 message/1 second
4. Send two numerical values 5 second apart
    a. Repeat 4a, 4b
    b. If both readouts correct then minimum data rate of 1 message/5 seconds verified
5. Send three numerical values by sending 001.0002.0003.0 in one string
    a. Arduino Serial Monitor will display 003.0 (discards “older 001.0 and 002.0).
    b. If 003.0 command received then arbitrary user data rate verified
6. Connect Oscilloscope (+) lead to orange pin, (-) to black pin, of FTDI cable

7. Connect FTDI to test computer
8. Connect using PuTTY over Serial
    a. 9600 baud
    b. 8-bits
    c. No parity
    d. 1-bit stop
9. Send characters using PuTTY
    a. If oscilloscope nominally indicates 9600 Hz across narrowest pulse then this verifies 9600 baud rate
    b. If decoded data matches send character then this verifies UART
    c. Can show that pins 1, 0 of the block Arduino Micro are UART only pins

4.5.6. References and Files Links

4.5.6.1. References

[1] Arduino, “Arduino Micro,” *Arduino*, Available: https://store.arduino.cc/products/arduino-micro [Accessed: Jan. 31, 2023]
[2] Emiliano Borghi, *AS5048A.cpp*. [Source Code]. Github, 2021
[3] Analog Devices, “Fundamentals of RS-232 Serial Communications,” *Analog Devices*, https://www.analog.com/en/technical-articles/fundamentals-of-rs232-serial-communications.html [Accessed: Feb. 08, 2023]
[4] R. Smith, “QUICK REFERENCE FOR RS485, RS422, RS232 AND RS423,” [Online]. Available: http://www.rs485.com/rs485spec.html [Accessed: Feb. 11, 2023]
[5] T Kugelstadt, “Extending the SPI bus for long-distance communication,” *ti.com*, [Online], Available: https://www.ti.com/lit/an/slyt441/slyt441.pdf [Accessed: Feb. 11, 2023]
[6] Stepperonline, “Nema 17 Bipolar 42Ncm(59.49oz.in) 1.5A 42x42x39mm 4 Wires w/ 1m Cable & Connector,” *omc-stepperonline.com*, [Online]. Available: https://www.omc-stepperonline.com/nema-17-bipolar-42ncm-59-49oz-in-1-5a-42x42x39mm-4-wires-w-1m-cable-connector-17hs15-1504s [Accessed: Jan. 31, 2023]
[7] Stepperonline, “1403/17HS15-1504S_Torque_Curve,”*omc-stepperonline.com*, [Online]. Available: https://www.omc-stepperonline.com/index.php?route=product/product/get_file&file=1403/17HS15-1504S_Torque_Curve.pdf [Accessed: Jan. 31, 2023]

4.5.6.2. Files Links

1) TB6600 Datasheet: https://toshiba.semicon-storage.com/info/docget.jsp?did=14683&prodName=TB6600HG
2) AS5048A Datasheet: https://ams.com/documents/20143/36005/AS5048_DS000298_4-00.pdf
3) Molex 5569 Datasheet: https://www.molex.com/webdocs/datasheets/pdf/en-us/0026013114_PCB_HEADERS.pdf
4) JST XH Datasheet: https://www.jst.com/wp-content/uploads/2021/01/eXH-new.pdf

5) MAX232 Datasheet: https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fmax232
6) L78SXX Datasheet: https://www.st.com/content/ccc/resource/technical/document/datasheet/e9/be/53/a3/1f/6f/4f/75/CD00000449.pdf/files/CD00000449.pdf/jcr:content/translations/en.CD00000449.pdf
7) MCP1501-33xSN Datasheet: https://ww1.microchip.com/downloads/en/DeviceDoc/20005474E.pdf
8) MP915-0.20-1% Datasheet: http://www.caddock.com/Online_catalog/Mrktg_Lit/MP9000_Series.pdf
9) SN75C1168N Datasheet: https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fsn65c1168
10) 1000680108 Datasheet: https://www.molex.com/pdm_docs/sd/1000680108_sd.pdf
11) SXH-001T-P0.6 Datasheet: https://www.jst-mfg.com/product/pdf/eng/eXH.pdf
12) USB-RS232-WE-1800-BT_5.0 Datasheet: http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_USB_RS232_CABLES.pdf
13) LRS-50-12 Datasheet: https://www.meanwellusa.com/upload/pdf/LRS-50/LRS-50-spec.pdf
14) falstad 50 % Duty Cycle Clock Generator Circuit: https://drive.google.com/file/d/1y9Z1gmqx6X80IfSl0sj19NV8qep19TuO/view?usp=sharing

4.5.7. Revision Table

Author	Date	Description
Angel Huang	1/20/23	Initial section creation
Angel Huang	2/5/23	Revised TB6600 Info - Removed torque and RPM calculations - Added justification and design of TB6600 circuit Reworked Design - Reexamined focus and moved information too appropriate sections - Added placeholders for images + resources - Removed extraneous and redundant information Organization - Added A, B … Z subsections to General Validation, Design

Angel Huang	2/8/23	Revised HMC1021S Info Organization - Added A, B … Z subsections to General Validation, Design Revised Carrier Board Info Removed redundant placeholder graphics
Angel Huang	2/10/23	Finished writing Carrier board section - Added Clock generation sub-section Cleaned up Design section
Angel Huang	2/11/23	Removed redundant information from Interface Validation Reworked Description General revision, cleanup, redundancy removal Reworked Verification Plan Reorganized Verification Plan Moved reference and file links in from Digikey order and changelog
Angel Huang	2/15/23	Fixed **trntbl-cntrllr_trntbl_comm** duplication with **trntbl_trntbl-cntrllr_comm**
Angel Huang	3/13/23	Removed HMC1021S details Updated reference, file links, to reflect HMC1021S removal
Kira Kopcho	5/14/23	Format revision table to follow style of the rest of the document

4.6. Turntable

4.6.1. Description

The Turntable is an Actuator based block designed to mount a Ubiquity directional antenna: this antenna is different from the GPS antennas in other blocks. While being an actuator, the block does not output any motion: all motion is contained within the block. All internal motion is strictly rotational in the horizontal plane.

Overall physical structure comprises three subassemblies is based on extruded aluminum L-channel, structural framing, and plastic for high strength and low weight: the majority of weight will be contributed by the Ubiquity antenna! A foundation sub-assembly provides an electrical box and adjustable feet for unlevel terrain. The tower sub-assembly resembles a horizontal "U" to provide support for the antenna turntable from above and below for rigidity. Finally, the turntable itself is a cradle built around the Ubiquity that mounts to two Lazy Susan mechanisms on the tower sub-assembly.

To provide angular positioning feedback, the Turntable mounts an encoder sub-system based around the AS5048A magnetic encoder. It is mounted on the tower sub-assembly and the magnet is on the cradle sub-assembly. Data between Turntable and Turntable-Controller is sent through an RS422 network to reduce negative effects of EMI (Electromagnetic Interference) based on the SN75C1168N full duplex (simultaneous send and receive) component. 5V power is provided from the Turntable-Controller block.

4.6.2. Design

The Turntable actuator block is unconventional in that it does not output mechanical force or motion: all motion is contained within the block. Therefore, it needs to integrate mechanical and electrical systems, and their respective issues, together. Below is an introduction to the block via the "black box" input, output diagram.

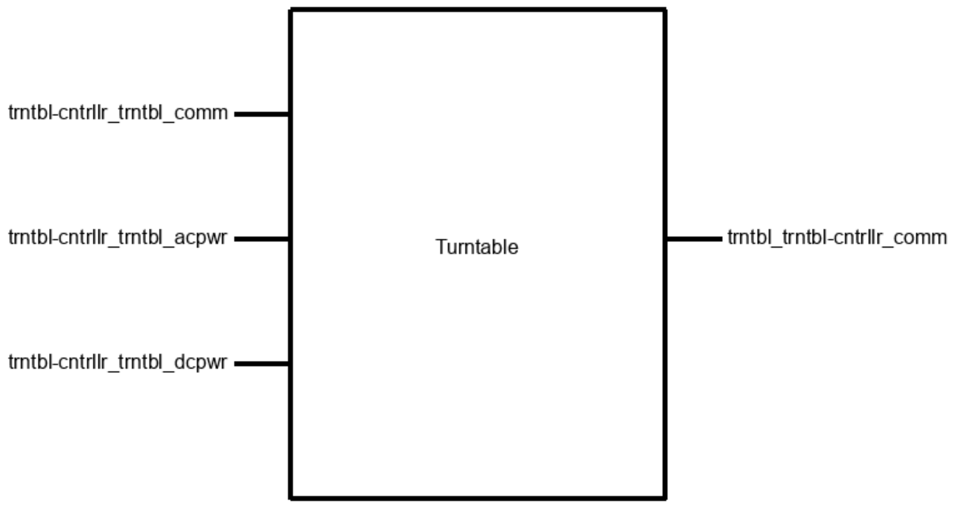

trntbl-cntrllr_trntbl_comm
trntbl-cntrllr_trntbl_acpwr
trntbl-cntrllr_trntbl_dcpwr
Turntable
trntbl_trntbl-cntrllr_comm

Fig. 4.6.2.1 - Black Box I/O Diagram

a) <u>Interfaces</u>

- **trntbl_trntbl-cntrllr_comm:** accepts absolute (unique data per position) angular position of the Turntable block over SPI.
- **trntbl-cntrllr_trntbl_comm:** issues configuration commands to angle sensor in the Turntable block over SPI.
- **trntbl-cntrllr_trntbl_acpwr:** 12 V, 0.4 A stepper motor power to Turntable block motors.
- **trntbl-cntrllr_trntbl_dcpwr:** 5 V peripheral and sensor power to the Turntable block. Primarily used for the aforementioned angle sensor.

A primary design goal is to reduce weight, promote modularity, and provide ease of use. Aluminum framing components such as McMaster 47065T101 combine high strength, low weight (versus steel), and modularity into a reasonably priced component. Similarly, Polycarbonate (McMaster 8574K192) is easily machined into lightweight but strong flat components such as brackets and 3D printed PLA forms cheap but complex and reasonably strong components such as PCB or 90° brackets.

The primary screw choice is #8-32 (3/16” diameter with 32 threads per inch) in an attempt to standardize screw size and required tooling. #¼-20 screws are used for connecting aluminum frame pieces together or peripherals to said frames. #6-32 is used in tight spaces such as the turntables. Finally, M3 and M5 are used to mount PCBs and pulleys respectively: the turntable stepper motors use a 5 mm diameter shaft alongside pulley (3693N11) components.

The overall design consists of three major subassemblies.

A “foundation” subassembly provides supporting and leveling, for uneven terrain, and mounts the Turntable-Controller PCB block in an electrical box. It is built from two frame pieces (2 foot 47065T101 and 2 foot 47065T107) connected with frame brackets (47065T239). Two rubber wheels provide easier transport. Leveling is achieved through three 8” adjustable screws at the extreme ends of the frame pieces: the triangle formation eliminates frustrations associated with four legs. The foundation subassembly electrical box is cut from clear polycarbonate for visual debugging.

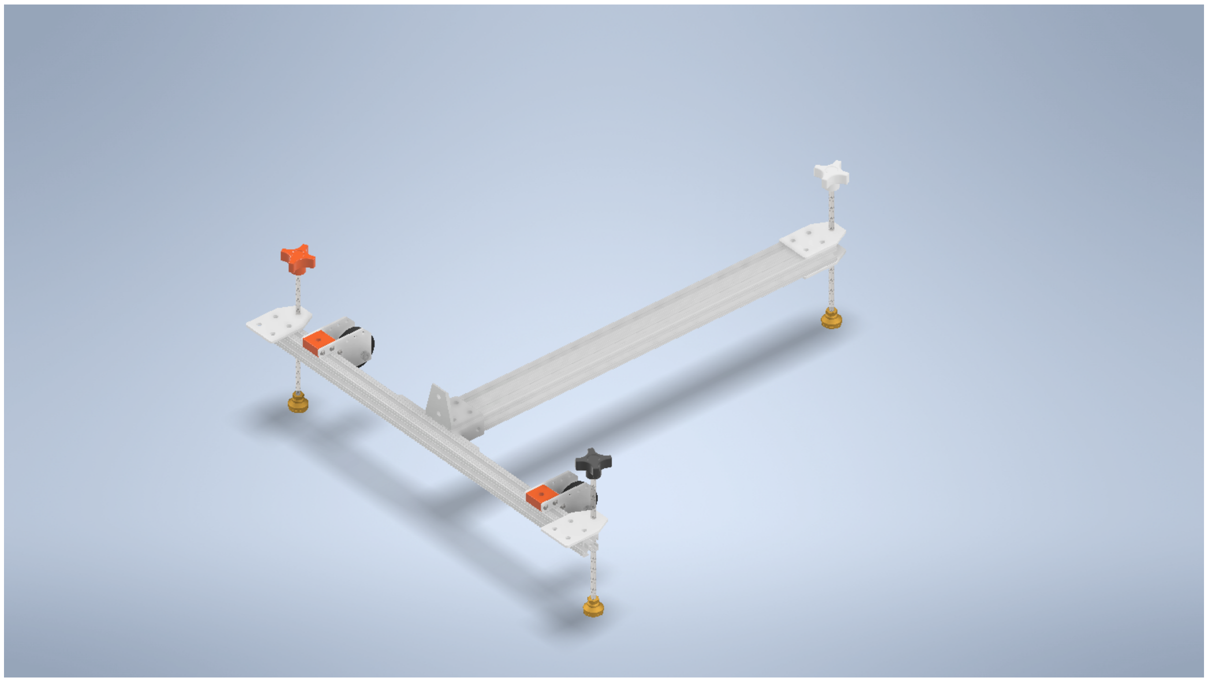


Fig. 4.6.2.2 - Foundation CAD Render

Atop the foundation is the “tower” subassembly. It is built around a 4 foot McMaster 47065T101 with holes drilled to affix L-channel: these connections are made rigid using 97832A216 shoulder screws which provide tighter tolerances than conventional machine screws. Each L-channel supports a McMaster 6031K16 turntable through #6-32, limited by the turntable dimensions, screws. The tower is connected to the foundation by framing brackets: the use of framing allows extensions to height in the future.

The tower also mounts a 17HS15-1504S stepper motor which drives a 600 tooth belt (7947K758). To prevent slipping, the belt is tensioned using a 3D printed sliding mechanism and idler pulley and guided around the tower pole with another idler. Between Turntable and

Turntable-Controller, the 17HS15-1504S motor is what accepts the **trntbl-cntrllr_trntbl_acpwr** interface.

Additionally, an encoder board based on the AS5048A is mounted between the top pair of L-channel. The AS5048A is an absolute encoder which removes the need for zeroing: it will provide the same output for the same position. It features a SPI interface, 14-bit position value (AS5048A, pg. 11, SPI Interface), and a generous tolerance in magnet mounting (AS5048A, pg. 32, Fig. 36 and Fig. 37).

To mitigate EMI effects, an SN75C1168N converts the SPI signals into differential pairs: this is only done to CLK, MISO, and MOSI. This IC is what interfaces between the Turntable-Controller and encoder board of Turntable.

Mounting of the motor is offset to provide some counter force to the 4 kg primary Ubiquiti antenna (AM-2G15, page 4, Specifications) attached to the cradle subassembly. However, the design does allow for all torque and gravity forces to be sustained by the L-channels, turntables, and shoulder screws.

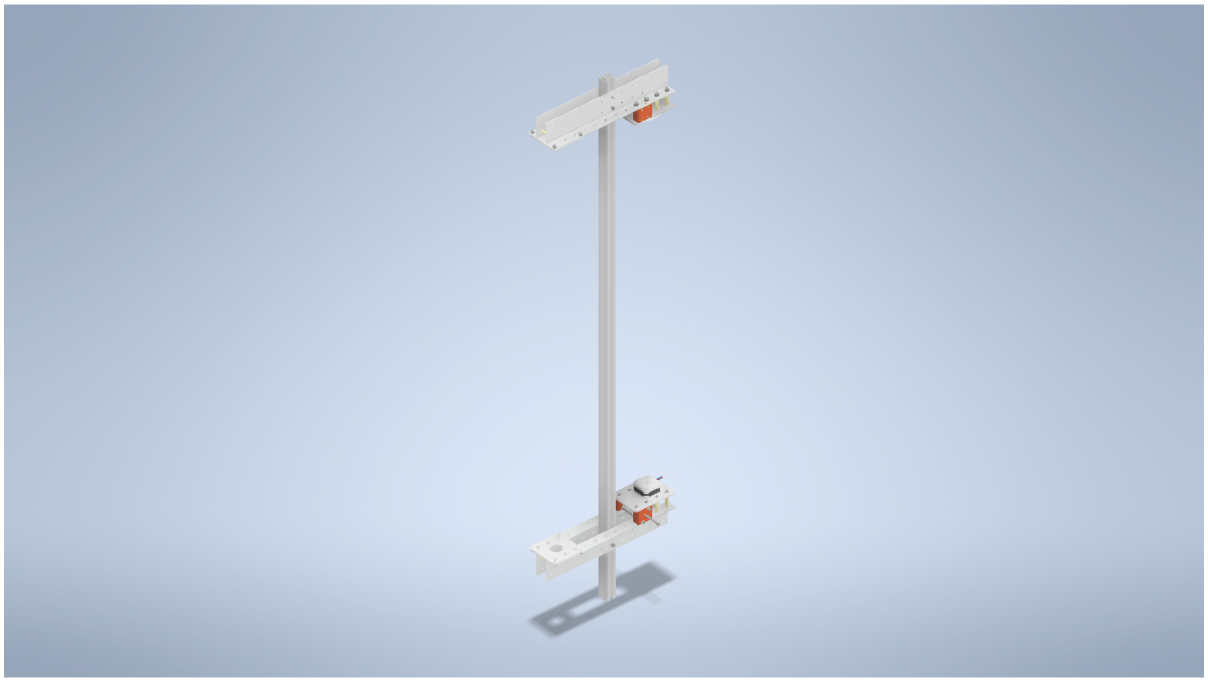


Fig. 4.6.2.3 - Tower CAD Render

Finally, the “cradle” sub assembly holds the Ubiquiti AM-2G15 antenna between two aluminum L-channels. The lower strength of PLA is accounted for by using multiple brackets to distribute stress, providing support to reduce torque forces from the Ubiquity, and adding stiffening plates to relieve stress.

A passthrough for the Ubiquity Ethernet cable is designed into block parts and passed through the center of the turntable mechanisms. This also relieves rotational stresses on the cable and prevents tangles.

The top turntable mechanism is occupied by the magnet needed for the AS5048A encoder board.

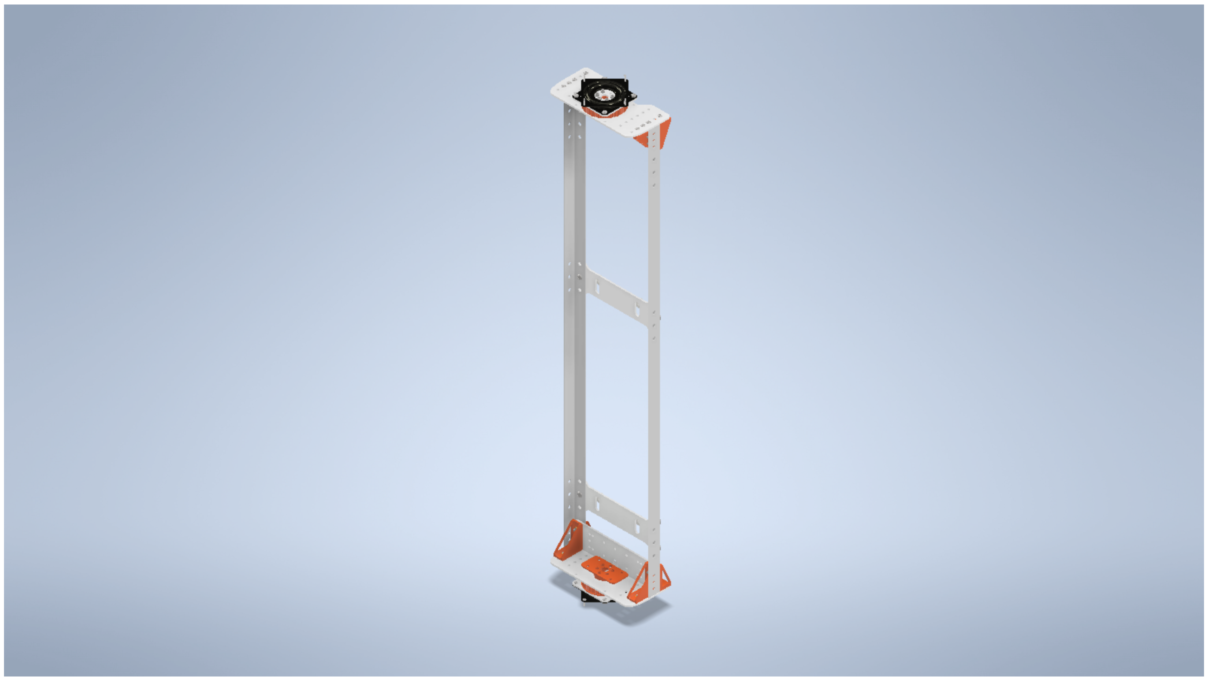


Fig. 4.6.2.4 - Cradle CAD Render

b) Design Documents

While it is industry standard to provide annotated manufacturing files, it will be easier and shorter to link to a Google Drive repository containing the parts themselves and annotated documents [Part Repository].

4.6.3. General Validation

When combining electrical and mechanical systems into one system, negotiating all the intricacies of both can be very challenging. What might be inconsequential in one could have major implications in the other. For instance, simply surrounding electronics in a tough plastic box might cause heat issues if no fans or cooling ports are added or slow down debugging by requiring removing a lid. On the flip side, an all metal construction is more durable but also conductive and, potentially, magnetic. Careful planning is needed to reconcile all these facts.

Starting with structure, the use of 6000 series aluminum is clear because it's lightweight, rigid, and strong [1]. But, polycarbonate and PLA plastics sacrifice some rigidity and strength for lower weight. Examining McMaster 8574K192 polycarbonate, it is rated at an impact strength of 2 ftlb/in and 9,100 PSI tensile strength. Unfortunately, the ECE curriculum does not prepare for mechanical engineering computations nor has the block champion, Angel, taken strength of materials yet. Thus, previous experience with ⅛" polycarbonate indicates that 8574K192 polycarbonate should be sufficient for loads perpendicular and parallel to the material.

For PLA, the intricacies of 3D printing make any computations difficult: infill percent, material behavior, temperature, are changed part properties. Like the polycarbonate, experience has determined PLA to be a cheap and reasonably strong part as long as certain printing orientations are used.

However, certain design choices are used to reduce stress. For flat polycarbonate parts, braces are added to provide rigidity. In the turntable mechanisms, two polycarbonate plates are placed at 90° angles against one another to prevent bending of the lower plate. Additionally, the size of plates are minimized to both reduce costs and torque, which is proportional to distance, forces.

For 3D printed parts, their weaker strength is compensated by adding multiple redundant parts to distribute forces. The same turntable mechanism uses four printed brackets instead of just two to distribute torque forces in all rotational directions.

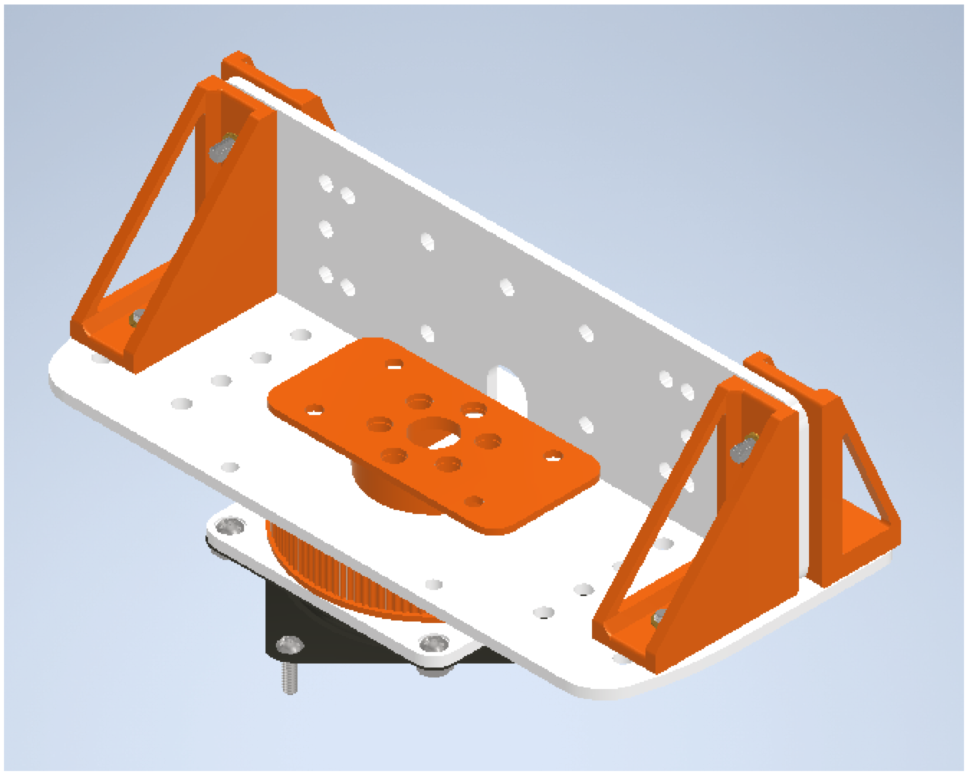


Fig. - 4.6.3.1 - Turntable Cross Bracing and Stress Distribution

As for custom components in general, they were deemed possible to manufacture based on the specifications of available machinery. For polycarbonate components, they were routed using a FoxAlien Masuter CNC router with a working area of 400 x 380 mm and a linear accuracy up to 0.001 mm [2]. 3D prints were done on an Ender 3 with a working volume of 220 x 220 x 250 mm and an accuracy of 0.1 mm [3]. Aluminum parts were kindly machined by a technical school, CTEC of Salem, Oregon, using high accuracy and automated metal mills.

Damage to custom components will be mitigated by providing all necessary design files and required material stock details. 3D printed components can be printed through the 3D Printing Club's print farm. Polycarbonate and aluminum parts will have spares made before deployment during competition.

For the tower design, a concern was the torque force exerted by the AM-2G15 antenna. The antenna weighs 4 kg and is cantilevered 114.476 mm, or 0.114 m, out from the tower pole. This equates to 4.47336 Nm of torque, by T = fd, imparted onto the tower pole and thus into the connecting brackets. The stepper motors opposite the antenna only provide 0.239 Nm of counter torque derived from a 0.280 kg motor weight and 0.087 m distance. To counter over 4 Nm of torque, aluminum brackets were used in conjunction with large #¼-20 screws (McMaster 47065T239 and 3136N202). Additionally, to prevent tipping, the Ubiquiti antenna is cantilevered over the foundation pieces as seen in Figure 5 so any torque force “twists” the foundation down and not upwards i.e. like when someone leans into a ladder instead of outwards.

Finally, the encoder board is designed to interface between two moving subassemblies: a physical or optical encoder would’ve limited adjustments due to manufacturing defects and required tighter tolerances. It also sends data via the SPI protocol and physically transported by RS422 back to the Turntable-Controller block.

Primarily, the absolute positioning capability is useful because the cradle subassembly cannot be expected to be in the same position on system power-on nor be perfectly zeroed during competition. By using absolute positioning, any position can instantly be uniquely determined without additional setup.

Secondarily, the generous tolerances in magnet mounting (AS5048A, pg. 32, Fig. 36 and Fig. 37) of ± 0.25 mm linearly 0.5 - 2.5 mm vertically allows operation even with some manufacturing deviations. Conveniently, the 9049 magnet from Radial Magnet Inc. is a powerful neodymium magnet with low field deviation of 1% maximum (9049, pg. 1) which allows for additional deviation as stated in the AS5048A documentation (AS5048A, pg. 32, Magnet Placement).

Electrically, the AS5048A operates off easily supplied 5 V from the Turntable-Controller and only draws 15 mA (AS5048A, pg. 8, Operating Conditions). It includes non-inverting Schmitt Triggers to filter out low voltage noise on the SPI pins which improves reliability in high EMI environments (AS5048A, pg. 5, Pin Assignments). This is improved with the RS422 differential protocol implemented by a Texas Instruments SN75C1168: a differential protocol that both extends the range of SPI and improves EMI tolerance [4].

4.6.4. Interface Validation

Interface: **trntbl_trntbl-cntrllr_comm**

Messages: Data: 16-bit data package containing 14-bit angle data	Received message size is referenced from AS5048A SPI interface characteristics.	- Turntable block AS5048A reports data in a 16-bit data package (AS5048A, SPI Read Package, fig. 20, pg. 15). - Bit 13:0 is the 14-bit data - Other bits are for control
Other: RS422 Differential Logic High Vmin: +/-2V	This is the positive and negativeV_{OH} threshold of the SN75C1168N.	- SN75C1168N reports a minimum required V_{OH} of 2V inverting (SN75C1168N, Driver Section, Electrical Characteristics, pg. 6).
Protocol: 4 Wire SPI (data protocol)	The AS5048A variant uses an SPI for commands and outputs.	- SPI is chosen for control features at the cost of additional input/output hardware. - Dedicated input and output prevents collisions.

		- Chip select allows multiple peripherals on one SPI network.

Interface: **trntbl_trntbl-cntrllr_comm**

Messages: Data: 16-bit data package containing address and read/write command	Received message size is referenced from AS5048A SPI interface characteristics.	- Turntable block AS5048A receives commands in a 16-bit data package (AS5048A, SPI Command Package, fig. 19, pg. 14). - Bit 15 (MSB) is the even parity bit - Bit 14 is read (1) or write (0) command. - Bit 13:0 is the 14-bit address to read from or write to.
Other: RS422 Differential Logic High Vmin: +/-2V	This is the positive and negativeV_{OH} threshold of the SN75C1168N.	- SN75C1168N reports a minimum required V_{OH} of 2V inverting (SN75C1168N, Driver Section, Electrical Characteristics, pg. 6).
Protocol: 4 Wire SPI (data protocol)	The AS5048A variant uses an SPI for commands and outputs.	- SPI is chosen for control features at the cost of additional input/output hardware. - Dedicated input and output prevents collisions. - Chip select allows multiple peripherals on one SPI network.

Interface: **trntbl-cntrllr_trntbl_acpwr**

Inominal: 0.4A (per stepper positive channel)	Turntable motor is rated for 0.4 A per coil.	- Turntable stepper motor 17HS15-1504S is rated for 1.5 A [6]. - TB6600 can output 5 A of maximum current (TB6600, Electrical Characteristics, pg. 27)
Vnominal: 12V (per stepper positive channel)	Turntable motor is rated for 12 V but can accept higher and lower voltages.	- Turntable stepper motor 17HS15-1504S has been tested at 24 V. [7]. - Will use operational voltage of 12V and coil current 0.4A for safe and low heat operation.
Vmax: 14V	Turntable	- LRS-50-12 has a maximum output of 13.8 V (LRS-50-12, Specification, pg. 2) which could be sent 1:1 through the TB6600.
Other: Imax: 1A	Derated value from 17HS15-1504S maximum of 1.5 A continuous.	- Turntable stepper motor 17HS15-1504S is rated for 1.5 A [6].

Interface: **trntbl-cntrllr_trntbl_dcpwr**

Inominal: 21mA	Computed by summing operational current requirements of circuits connected to this interface.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - SN75C1168N in DIP-16 with max draw of 6mA for just logic (SN75C1168N, Electrical Characteristics, pg. 6)
Ipeak: 215mA	Computed by summing operational current requirements of circuits connected to this interface.	- AS5048A in TSSOP-14 package with max draw of 15 mA (AS5048A, fig. 8, pg. 7) - SN75C1168N in DIP-16 with max draw of 200mA for logic and output (SN75C1168N, Absolute Maximum Ratings, pg. 5)
Vmax: 5.5V	Determined from AS5048A datasheet.	AS5048A reports a maximum of 5.5V to V_{CC} (AS5048A, Operating Conditions, fig.8, pg 7). SN75C1168N reports a maximum of 5.5V to V_{CC} (SN75C1168N, Recommended Operating Conditions, pg. 5)
Vmin: 4.5V	Determined from AS5048A datasheet.	AS5048A reports a minimum of 4.5V to V_{CC} (AS5048A, Operating Conditions, fig.8, pg 7). SN75C1168N reports a minimum of 4.5V_{CC} (SN75C1168N, Recommended Operating Conditions, pg. 5)
Vnominal: 5V	Determined from AS5048A datasheet.	AS5048A accepts a range of 4.5V to 5.5V (AS5048A, Operating Conditions, fig.8, pg 7). SN75C1168N accepts a range of 4.5V to 5.5V (SN75C1168N, Recommended Operating Conditions, pg. 5)

4.6.5. Verification Process

Interface under test: **trntbl_trntbl-cntrllr_comm** and **trntbl-cntrllr_trntbl_comm**

Testing Materials:

- Turntable-Controller Block
- AS5048A Encoder Board of Turntable Block
- DC Supply
- Power Leads
- Oscilloscope
- Probe Leads
- Computer with Arduino Serial Monitor
- USB A to USB Micro B

1. Connect Computer to Arduino
2. Apply 12 V to block
3. Watch Serial Monitor
    a. 1st message will be command to AS5048A in 16-bit string
    b. 2nd message will be data received from AS5048A in 16-bit string
    c. If messages are displayed and received then 16-bit data package size is verified
4. Attach oscilloscope leads to SPI + and SPI - signals (CLI, MISO, MOSI)
    a. If magnitude of voltages are > 2 V then RS422 logic level is verified
5. Attach oscilloscope leads to SPI test points (CLK, MISO, MOSI, ~CSn)
6. Configure oscilloscope to decode SPI
    a. 16-bit length
    b. Read on rising of clock
    c. Inverted CSn
7. Observe decoded data
    a. If MOSI is 0x1111111111111111 then this verifies command 16-bit string
    b. If MSIO is 16-bit (data varies) then this verifies data 16-bit string
    c. If both observed then this also verifies SPI interface in use

---

Interface under test: **trntbl-cntrllr_trntbl_acpwr**

Testing Materials:

- Turntable Block
- Turntable-Controller Block
- TB6600 subsystem PCB (separate from Turntable-Controller Block)
- 17HS15-1504S Stepper Motor and Terminal Blocks
- DC Supply
- Multimeter
- Oscilloscope
- Multimeter Leads
- Oscilloscope Leads

1. Connect Turntable 17HS15-1504S motors to Turntable-Controller
2. Set multimeter to current mode
3. Attach multimeter leads between 17HS15-1504S Channel A+ and Turntable-Controller, TB6600 subsystem J1 connector, Pin 1
    a. Use test terminal block as breakout
4. Apply 12 V to Turntable-Controller Block
    a. If current on channel is nominally 0.4 A then this verifies nominal channel current
5. Adjust JV1 of Turntable-Controller, TB6600 sub-system
    a. Adjust until the current reaches 1.0 A.
    b. Allow current to remain steady for 1 minute
    c. If current holds at 1.0 A then this verifies maximum channel current
6. Stop 12 V

7. Remove multimeter leads and reattach cabling between Channel A+ and TB6600 sub-system J1.
8. Connect Turntable 17HS15-1504S to spare TB6600 subsystem PCB
9. Connect control cable from spare TB6600 J1 to Turntable-Controller J6
10. Connect spare TB6600 to DC Supply
    a. Use the SAME DC Supply as the Turntable-Controller Block
11. Adjust supply to spare TB6600 so oscilloscope indicates 12 V nominally
12. Adjust supply to spare TB6600 so oscilloscope indicates 14 V maximum
    a. This verifies nominal and maximum input voltage

Interface under test: **trntbl-cntrllr_trntbl_dcpwr**

Testing Materials:

- Turntable-Controller Block
- AS5048A Encoder Board of Turntable Block
- DC Supply
- Multimeter
- Multimeter Leads
- Power Leads

1. Attach DC Supply to J1 of AS5048A Encoder Board of Turntable Block
2. Apply 12 V to Turntable-Controller
3. Apply 5 V to AS5048A Encoder Board
    a. Verify that SPI data is being sent and received
    b. Verify that current draw is nominally 21 mA and below 210 mA
    c. If so, this verifies nominal operation at 5 V and 21 mA
4. Apply 4.5 V and 5.5 V to AS5048A
    a. Verify that SPI data is being sent and received
    b. Verify that current draw is below 210 mA
    c. If so, this verifies operation within specified supply voltage range

4.6.6. References and Files Links

4.6.6.1. References

[1] MatWeb, "Overview of materials for 6000 Series Aluminum Alloy", *matweb.com*, [Online]. Available: https://www.matweb.com/search/datasheet.aspx?MatGUID=26d19f2d20654a489aefc0d9c247cebf&ckck=1 [Accessed: Mar. 14, 2023]
[2] Creality, "Ender-3 Pro", *creality.com*, [Online]. Available: https://www.creality.com/products/ender-3-pro-3d-printer [Accessed: Mar. 14, 2023]
[3] FoxAlien, "CNC Router Machine Masuter", *foxalien.com*, [Online]. Available: https://www.foxalien.com/collections/cnc-router/products/cnc-router-machine-masuter [Accessed: Mar. 14, 2023]

[4] T Kugelstadt, “Extending the SPI bus for long-distance communication,” *ti.com*, [Online], Available: https://www.ti.com/lit/an/slyt441/slyt441.pdf [Accessed: Feb. 11, 2023]
[6] Stepperonline, “Nema 17 Bipolar 42Ncm(59.49oz.in) 1.5A 42x42x39mm 4 Wires w/ 1m Cable & Connector,” *omc-stepperonline.com*, [Online]. Available: https://www.omc-stepperonline.com/nema-17-bipolar-42ncm-59-49oz-in-1-5a-42x42x39mm-4-wires-w-1m-cable-connector-17hs15-1504s [Accessed: Jan. 31, 2023]
[7] Stepperonline, “1403/17HS15-1504S_Torque_Curve,”*omc-stepperonline.com*, [Online]. Available: https://www.omc-stepperonline.com/index.php?route=product/product/get_file&file=1403/17HS15-1504S_Torque_Curve.pdf [Accessed: Jan. 31, 2023]

4.6.6.2. Files Links

1) AS5048A Datasheet: https://ams.com/documents/20143/36005/AS5048_DS000298_4-00.pdf
2) SN75C1168 Datasheet: https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fsn65c1168

4.6.7. Revision Table

Author	**Date**	**Description**
Angel Huang	3/13/23	Initial section creation
Angel Huang	3/14/23	Fixed copy-paste and formatting issues Added references and file links
Kira Kopcho	5/14/23	Format revision table to follow style of the rest of the document, changed font from Times New Roman to Arial

4.7. User Interface

4.7.1. Description

The purpose of the user interface is to provide visualization of the data sent and received by the tracking algorithm as well as provide a method of manually controlling the rotation of the turntable if needed.

Both the rover and base station are equipped with GPS modules that send geographic coordinates via USB to the hardware the UI runs on. The idea is that these reported coordinates can then be displayed on the UI so the end user can observe where both the base and the rover are located at all times. At minimum, these coordinates should be updated every 5 seconds to ensure that the data that’s being sent to the user is as accurate as possible. In addition the bearing angle calculated by the tracking algorithm is also displayed on the UI. This allows the user to verify the output of the tracking algorithm that is being sent via serial to the turntable module. This angle should update at a minimum rate of 1 update every 5 seconds as well.

The UI also provides the end user with a way to send manually defined angles to the turntable. This is meant as a failsafe in case the tracking algorithm is not properly operating. This is important because the rules of the competition the rover compete in prohibit team members from directly interacting with the rover or its communications equipment while competition tasks are underway. By including an interface for manually controlling the angle of rotation of the turntable, the end user can tweak the positioning of the turntable without having to manually interact with it.

4.7.2. Design

The user interface is the main way end users can get information about the positioning of the rover, as well as interact with the turntable module. The UI itself was created using the Qt Framework- a cross platform software development library that has bindings in Python and C++. The current UI for the entire rover is written using the Python bindings for Qt, so our project also uses Python to ensure that our UI can later be added into the rover UI as a whole.

The UI has two inputs, one is the bearing angle determined by the tracking algorithm (trckng_lgrthm_u_data) and the other is a user defined angle for manually controlling the turntable (otsd_u_usrin). The user defined input includes text from a text box and a button click to actually send the angle across serial to the turntable module. Since the tracking algorithm and the UI run on the same piece of hardware, both python instances can communicate with each other.

The UI also provides two outputs. The first is a data output to show the end user data about what is being received by the tracking algorithm (u_otsd_usrout). Text displays on the UI dynamically update to show both the geographic position of the rover, the base station, and the updated bearing angle from the tracking algorithm. These interfaces update about once per event cycle, which is about once per second unless the NUC is under heavy processing load. The second output from the tracking algorithm is the serial output to the turntable module for manually setting the angle. Qt provides a serial interface so the UI is able to write messages across the serial link to the controller with UART specifications. Pictured below is a black box diagram that shows the inputs and outputs associated with the UI. For a full diagram of how the tracking subsystems works, see section 4.1.2.

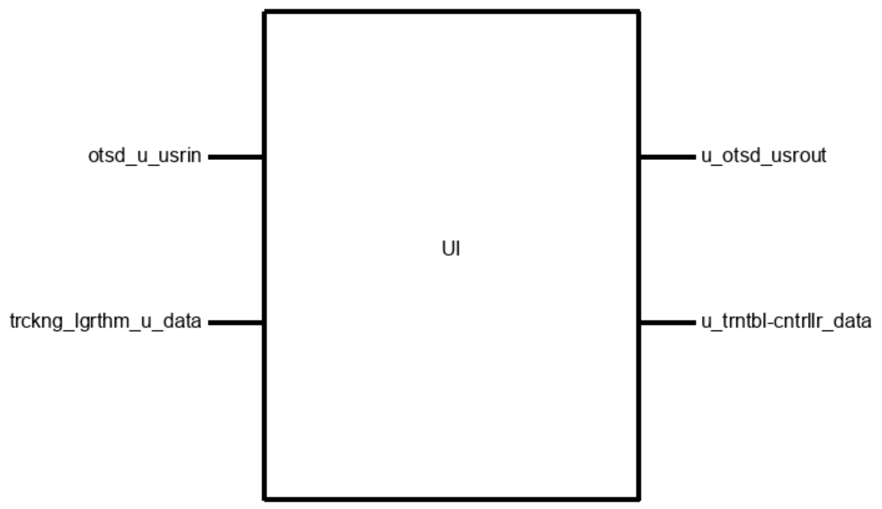


Fig. 4.7.2.1 - Block Diagram

4.7.3. General Validation

The user interface is part of the larger tracking subsystem. Like the tracking algorithm, it runs on the intel NUC 11 that runs the larger rover UI. For verification purposes, a smaller UI widget was created, but eventually, this UI block will be merged into the larger rover UI. As a result, many design decisions for this block are influenced by maintaining compatibility with the larger rover UI.

The UI itself was written using the Qt framework with Python bindings. Qt is a cross-platform development tool primarily used to create graphical user interfaces. Natively, Qt is written in C++, but there is a library with python bindings known as PyQt which shares most of the same functionality as the original C++ version [1][2]. PyQt is what the current user interface for the rover is made with, so to maintain compatibility the user interface for the tracking subsystem is made with PyQt as well. Other Python libraries do exist for creating graphical user interfaces (such as Tkinter), but to ensure compatibility with the existing codebase PyQt is used.

The main goal of the user interface for the tracking subsystem is to provide feedback to the end user about the tracking subsystem. In short, the end user should see the inputs and outputs of the tracking algorithm on the user interface. There are three key pieces of information associated with the tracking algorithm: The coordinates received from the rover GPS, the coordinates of the base station GPS, and the bearing angle produced from the tracking algorithm. Originally, the coordinates from the base station were not planned to be included in the UI since the base GPS is not mobile. However, since the UI can also be used as a diagnostic tool for the tracking algorithm, the base coordinates are included in the updates the UI receives. The rover coordinates and bearing angle change state more frequently than the base coordinates, so it is important to include them to show their updates in real time.

The code for the user interface consists of 3 files: the ui XML file generated by Qt Designer, the Tracking Coordinator, and the UI script itself. Qt Designer is a tool for creating GUI using Qt Widgets [3]. Instead of writing the Qt Widgets from scratch, the tool allows the developer to simply drag and drop the desired widgets into an interface and set their properties. When the developer saves their work, an XML script is compiled that includes the information of each of the widgets. The named widgets in the XML file are the same variables that are used by both the UI script and the Tracking Coordinator. Qt Designer was chosen to cut down on the programming time for the GUI. Since the UI script and the Tracking Coordinator are both more involved pieces of code, setting the layout and properties of each Qt Widget in the designer was much quicker than writing a Python or C++ implementation of each widget. When the UI is eventually absorbed into the larger rover UI, the widgets in this XML file will be added to a larger XML file for the entier rover UI.

The actual pieces of code that run the UI are the Tracking Coordinator and the UI script. The Tracking Coordinator handles receiving input from the tracking algorithm. Since the tracking algorithm sends updates to the the UI frequently, the functions responsible for triggering the UI update must be run in a separate thread from the main UI window. If these updates are run in the main UI script, the UI will freeze until each update is completed [4]. This unfortunately adds a bit more processing overhead to the UI, but to avoid freezing the cost is minimal. Threading the tracking UI will also prevent it from freezing the entire rover UI when it is merged in. To actually receive values from the tracking algorithm, the Tracking Coordinator utilizes a python listener. Essentially, the UI opens a socket that the tracking algorithm can send its data across [5]. The tracking coordinator keeps track of the latitude/longitude values for the base and rover and the bearing angle. These are updated each time a message is received on the listener.

The main UI script is links the updates produced by the tracking coordinator to the widgets specified in the UI XML file. In short, it actually generates the visible UI window. It provides the code responsible for allowing users to interact with the manual control features of the UI as well as defines the Qt Widgets that the Tracking Coordinator sends updates to. In short, it is the main piece of code that ties all separate UI functions together. When our user interface is eventually absorbed into the larger rover UI, the main UI script will be called from a higher python instance that runs the entier rover UI. For now, it produces a standalone window for verification purposes.

4.7.4. Interface Validation

The UI serves as the link between the internals of the tracking subsystem and the end user. Since it is an important diagnostic tool and source of information for the end user, it is of the utmost importance that it works properly. The following table lists interfaces that aid in proving the UI works as intended. The table lists what each interface is, how it relates to the block and the system around it, and why the design meets the property of each interface.

Interface Property	Why is this interface this value?	Why do you know that your design details <u>for this block</u> above meet or exceed each property?

otsd_u_usrin : Input

Other: Manually defined angle is only sent when user presses "send" button	In order to prevent the manual angle sending from blocking output from the tracking algorithm by continuously sending angles, the user input has the requirement that angles should only be sent from the UI when the user requests them to be sent via a button click	The Qt framework only triggers a serial write to the turntable when the flag indicating the “send angle” button has been pressed is set.
Type: Button Click	This interface value exists to avoid the accidental sending of angles from the UI to the turntable controller.	Qt provides bindings for buttons within the UI. Button clicks set certain flags in the code and when those flags are set they can be used to trigger other events/functions in the main UI loop.
Type: Text- Angle: 0.0-360.0	Just like the output of the tracking algorithm, the turntable controller is only set up to accept angles that are between 0-360 degrees	Within the UI code, the option to click the “send angle” button is not available until the code checks that the angle input by the user is between “0.0” and “360.0”

u_otsd_usrout : Output

Type: Numbers - Base Station Coordinates	The coordinates for the base station are floating point numbers. The UI reports these numbers as they are updated, with 1 decimal point of precision.	Labels (textboxes) in QT can be dynamically updated when new information is received. They can also store both numerical and text string values. In this case, numerical values are used.
Type: Numbers - Angle (in degrees)	The bearing angle reported by the tracking algorithm is a floating point value between 0-360 degrees. This angle is sent to both the turntable and the UI.	Labels (textboxes) in QT can be dynamically updated when new information is received. They can also store both numerical and text string values. In this case, numerical values are used.

Type: Numbers - Rover Coordinates	The coordinates for the rover station are floating point numbers. The UI reports these numbers as they are updated, with 1 decimal point of precision.	Labels (textboxes) in QT can be dynamically updated when new information is received. They can also store both numerical and text string values. In this case, numerical values are used.

u_trntbl-cntrllr_data : Output

Datarate: 1 message per click	The angle should only be sent to the turntable when the button is clicked. Additionally, it is not ideal to send duplicate angles, so the messages should be limited to once per click	When a button is clicked in Qt it sets a flag that is checked and cleared once per UI event loop. So only one angle is sent per button click.
Messages: Angle: Must be from 0.0 - 360.0 degrees	The turntable module only accepts values between 0 and 360 so the output from the UI must meet this specification	The text entered by the user is checked within the UI to ensure it is between 0.0 and 360.0. The option to click the “send angle” button is unavailable until a valid angle is entered
Protocol: UART Serial (9600 Baud)	The arduino on the turntable controller is configured to accept UART serial messages so the data being sent from the UI must match UART specifications (8 data bits, 9600 baud, and 1 stop bit).	The Qt framework provides a serial library that can be configured to meet different serial standards. The serial output from the UI is configured to meet UART specifications.

trckng_lgrthm_u_data : Input

Datarate: Rate Min: Updates once every 5 seconds	The tracking algorithm has a minimum update rate of 1 message every 5 seconds so the UI should also have a minimum update rate of one message every 5 seconds.	When the UI threads are running under normal load conditions it updates at the same frequency as the tracking algorithm.
Datarate: Rate max: Updates once per second	The tracking algorithm has a maximum update rate of 1 message every second so the UI should also have a maximum update rate of one message every second.	When the UI threads are running under normal load conditions it updates at the same frequency as the tracking algorithm.

Messages: Angle: 0.0-360.0 degrees	The tracking algorithm returns an angle between 0-360 on success and a -1 when an error occurs. If the UI reads a -1 from the tracking algorithm, that indicates an error.	The UI is setup to read the bearing angle output directly from the tracking algorithm.

4.7.5. Verification Process

The verification plan below is meant to prove the functionality of the user interface before it is combined with the larger tracking subsystem. The main purpose of the verification plan below is to ensure that the UI module can update at a reasonable rate when presented with new data. To test this, there is a function included in the UI code that generates random numbers for coordinates and the bearing angle. This can be seen in the file links below. Verification also requires a USB-serial cable to verify data is actually being sent from the UI. The following verification plan aims to be simple so users not familiar with the UI can quickly verify the operation of the block. It is split into two sections: verification of manual angle sending and verification of update rates.

Verification of Manual Angle Control:

1. Attach the USB-serial output cable to the NUC and connect the TX (orange) line to an oscilloscope
2. Place the oscilloscope in “single-run/single-shot” mode
3. Start the python UI script
4. Enter a value between “0.0” and “360.0” in the textbox. The “.0” is required for the value to be valid
5. Click away from the textbox. The “send angle” button should become active
6. Click the send angle button
7. Adjust the zoom on the oscilloscope to see the pulsed waveform clearly.
8. Set cursors to the beginning and end of 1 square wave pulse.
9. Calculate the baud rate by using 1/T where T is the elapsed time between the start and the end of the pulse
10. Read off the 8 data bit pulses (minus the first logic high start bit) to determine the binary data sent across the serial. The least significant bit is the first pulse and the most significant bit is the last pulse
11. Convert the 8 data bits to an ASCII value using a binary to ASCII table
12. Validate that the ASCII value matches the value printed in the UI terminal
13. Validate that the ASCII value matches the angle value by using a Text to ASCII converter
14. Type in a negative value to the textbox and verify that the “send angle” button remains greyed out
15. Type in a value over 360.0 and verify that the “send angle” button remains greyed out

Verifcation of UI update rates:

1. Start the UI

2. Wait for the UI to start updating the angle/coordinate values
3. Keep time with the system clock to verify that the update rate of the UI is within 5 seconds
4. Valid numbers for angle is between “0.0” and “360.0”. The bearing angle will be displayed as -1 if it is not updating correctly/a value is received in error
5. The base and rover coordinates will also display -1 if not updating properly.

4.7.6. References and Files Links

4.7.6.1. References

[1] “About Qt.” Qt Wiki. [Online] Available: https://wiki.qt.io/About_Qt [Accessed Mar. 13, 2023]
[2] “Qt for Python.” doc.qt.io. [Online] Available: https://doc.qt.io/qtforpython/index.html [Accessed Mar. 13, 2023]
[3] “Qt Designer Manual.” doc.qt.io [Online] Available: https://doc.qt.io/qtforpython/index.html [Accessed Mar. 14, 2023]
[4] L.P Ramos. “Use PyQt's QThread to Prevent Freezing GUIs.” Real Python. [Online] Available: https://realpython.com/python-pyqt-qthread/ [Accessed Mar. 14, 2023]
[5] “Multiprocessing - Process Based Parallelism: Listeners and Clients.” docs.python.org [Online] Available: https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing.connection [Accessed Mar. 14, 2023]

4.7.6.2. Files Links

1) Full UI Code

4.7.7. Revision Table

Author	Date	Description
Kira Kopcho	2/28/23	Added Verification Steps
Kira Kopcho	3/12/23	Revised description section to better describe the purpose and functionality of the user interface
Kira Kopcho	3/12/23	Added short description of design for the UI and how it communicates with other blocks.
Kira Kopcho	3/12/23	Filled out interface validation table
Kira Kopcho	3/14/23	Added existing sections to full project document
Kira Kopcho	3/14/23	Finished general verification section

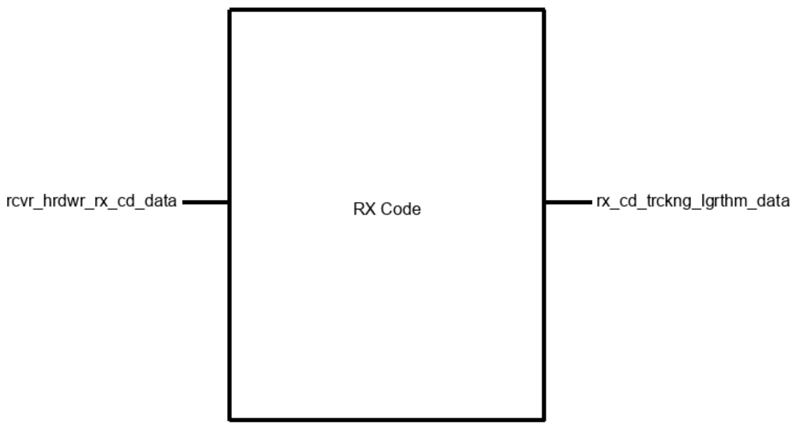
Kira Kopcho	3/14/23	Added references section and link to full UI code
Kira Kopcho	5/14/23	Format revision table to match the style of the rest of the document

4.8. RX Code

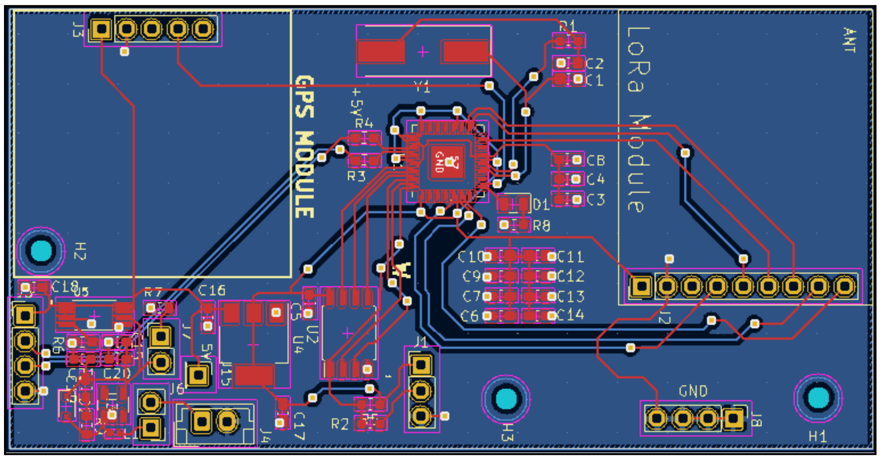
4.8.1. Description

Code that receives condensed GPS data sent, via the LoRa modem, from the rover's transmitter and passes that data to the ground station, via the transceiver's USB port, to be processed by the tracking algorithm.

4.8.2. Design



RX Code Block with Interfaces



General Code Outline:

a) Check if the LoRa modem is starting up and ready to receive transmitted data
b) Print LoRa modem status

c) Receive LoRa packet if modem is operational
d) Parse LoRa packet
e) Check that packet has a size greater than zero
f) Check that there is no erroneous data received within the packet
g) If there is no erroneous data, export data to the USB port at a maximum rate of one packet of longitude, latitude, and UTC time data per second

Main Code and Valid Check Function:

```
void print_if_valid(const char *str) {
    int len = strlen(str);
    int i;
    int has_extra_character = 0;
    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' &&
str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i] != '7' && str[i] !=
'8' && str[i] != '9' && str[i] != ',' && str[i] != ':' && str[i] != '-' &&
str[i] != '.' && str[i] != 'U' && str[i] != 'T' && str[i] != 'C') {
            has_extra_character += 1;
        }
    }
    if (!has_extra_character) {
        printf("%s\n", str);
    }
    has_extra_character = 0;
}
int main(){
      stdio_init_all();
      if (!LoRa.begin(915E6)){
            printf("Starting LoRa failed!\n");
            while(1){
                  printf("Failed!\n");
            }
      }
      printf("LoRa Started\n");
      LoRa.receive();
      int counter=0;
      while(1){
            int packetSize = LoRa.parsePacket();
            if(packetSize > 0){
                    string message = "";
                    while(LoRa.available()){
                    message += (char)LoRa.read();
                    }
            print_if_valid(message.c_str());
            }
        return 1;
}
```

4.8.3. General Validation

The code that will be uploaded to the RP2040 microcontroller chip will be written in the C++ programming language for purposes of familiarity amongst the relevant members of the group and to utilize the limited resources available to the fullest.

Outputting the code via the USB port allows for the transceiver to simply be plugged into the rover ground station.

The exporting at a maximum rate of one packet of information per second facilitates ample tracking resolution with the antenna while also saving memory resources on the RP2040 microcontroller

4.8.4. Interface Validation

The receiver code is the link between the receiver hardware and the tracking algorithm, via the base station, and is tasked with controlling the receiving transceiver hardware and printing the valid received data to the tracking algorithm.

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
rx_cd_trckng_lgrthm_data : Output		
Datarate: Rate Min: 1 message per 5 seconds	An update rate of once per 5 seconds is the minimum rate which still allows the antenna module to track the rover. This minimum padding to account for the possibility of lost or incomplete packets.	The runtime for the code is less than a second, which exceeds the minimum requirement of processing and transmitting the LoRa packet within 5 seconds.
Datarate: Rate Max: 1 message per second	One message per second was determined to provide ample time for the tracking of the rover via the rotating antenna.	The runtime for the code is less than a second, which allows for the printing of the received

		packets at a rate of once per second.
Messages: Latitude, Longitude, UTC Time	This data is required for determining the position of the rover.	The RX code does not edit this information, it only handles the receipt and printing of this data. This message is received and confirmed to not contain erroneous data via the print_if_valid() function.
Protocol: USB	A USB connection is used between the Receiver and the tracking algorithm running on the base station.	The USB output is enabled in the makefile for the code along with the .h file.
rcvr_hrdwr_rx_cd_data : Input		
Datarate: 12KHz from onboard clock	The 12KHz clock is used to facilitate the SPI communication.	It is specified in the RP2040 and LoRa modem datasheets
Messages: GPS position and time data	This data is required for determining the position of the rover.	The RX code does not edit this information, it only handles the receipt and printing of this data. This message is received and confirmed to not contain erroneous data via the print_if_valid() function.
Protocol: SPI	An SPI connection is used to communicate between the RP2040 Microcontroller and the LoRa modem.	The SPI communication is enabled in the makefile for the code along with the .h file.

4.8.5. Verification Process

The purpose of the verification below is to prove the receiver code is capable of communicating with the receiver hardware to receive packets via the LoRa modem and print out the received packets as long as the packets received do not contain erroneous data.

1. Confirm the LoRa modem starts up via a printed message viewed on the terminal of the computer the receiver is plugged into.
2. Confirm no erroneous packets are being printed by viewing the printout on the terminal.
3. Confirm the output of longitude, latitude, and time data packets at a rate of once per second via a terminal on a computer plugged into the transceiver's USB port.
4. Verify the latitude and longitude being received are accurate values by plugging them into google maps.

4.8.6. References and Files Links

4.8.6.1. References

[1] Semtech, “SX1276/77/78/79 Datasheet,” [Online] Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_FkpgP5kzjiNyiAbqcpqh9qSjE [Accessed: 11-Feb-2023].
[2] Raspberry Pi, “RP2040 Datasheet - A Microcontroller by Raspberry Pi” [Online] Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf [Accessed: 11-Feb-2023].

4.8.6.2. Files Links

1) Receiver Code
2) Receiver CMakeLists
3) Receiver CMake File
4) Receiver LoRa.h
5) Receiver LoRa.cpp
6) Receiver Print.h
7) Receiver Print.cpp

4.8.7. Revision Table

Author	Date	Description
Austin Grubowski	1/12/23	Created Block Validation document
Austin Grubowski	1/20/23	Updated interfaces, filled in all remaining sections
Austin Grubowski	3/14/23	Updated interfaces, references, file attachments, and added code
Kira Kopcho	5/14/23	Format revision table to follow style of the rest of the document

4.9. TX Code

4.9.1. Description

Code written in C++ that processes GPS information from the Rover GPS and sends it to the LoRa modem. It combines with the Rover GPS block to format and package the GPS data into LoRa packets with a specified format for parsing on the receiver.

4.9.2. Design

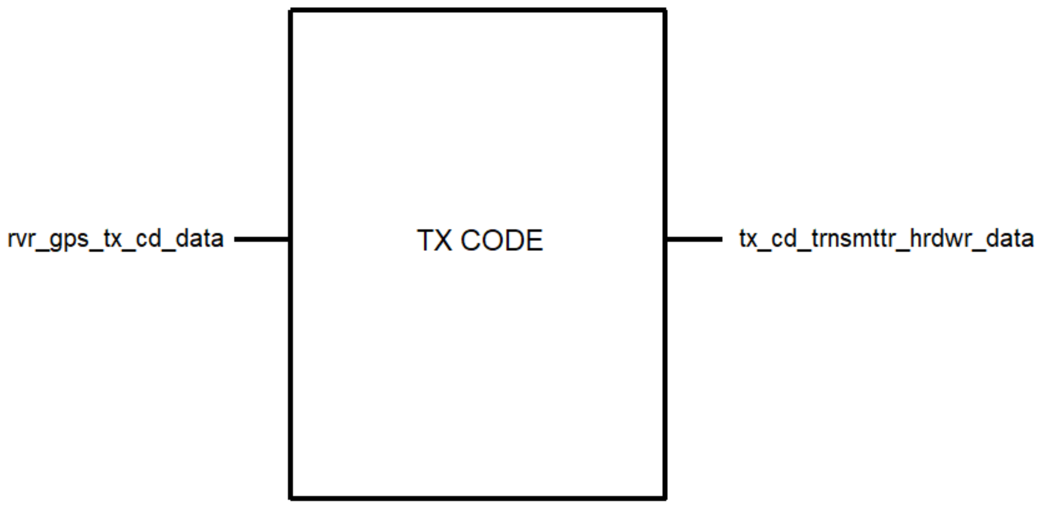


Fig. 4.9.2.1 - Black Box Diagram of TX Code block

4.9.3. General Validation

In order to verify that this block works an input of a formatted string in the following format: "Latitude, Longitude, Timestamp, Packet Number" will be sent to the block. The result must be bit sequences over the SPI interface to the transmitter hardware. This can be verified using an oscilloscope on the SPI line that is connecting the RP2040 to the LoRa module on the transmitter.

4.9.4. Interface Validation

The transmitter code is the link between the transmitter hardware and the incoming GPS data, and is tasked with converting the GPS string into bit sequences on the transmitter hardware.

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
rvr_gos_tx_cd : input		
Datarate: Rate Min: 1 message per 5 seconds	An update rate of once per 5 seconds is the minimum rate which still allows the antenna module to track the rover. This minimum padding to account for the possibility of lost or incomplete packets.	The runtime for the code is less than a second, which exceeds the minimum requirement of processing and transmitting the LoRa packet within 5 seconds.

Datarate: Rate Max: 1 message per second	One message per second was determined to provide ample time for the tracking of the rover via the rotating antenna.	The runtime for the code is less than a second, which allows for the printing of the received packets at a rate of once per second.
Messages: Latitude, Longitude, UTC Time	This data is required for determining the position of the rover.	The TX code does not edit this information, it only converts this information into bits on the SPI bus.
tx_cd_trnsmttr_hrdwr : Output		
Datarate: 12KHz from onboard clock	The 12KHz clock is used to facilitate SPI communication.	It is specified in the RP2040 datasheet.
Messages: GPS position and time data	This data is required for determining the position of the rover.	The TX code does not edit this information, it only converts this information into bits on the SPI bus.
Protocol: SPI	An SPI connection is used to communicate between the RP2040 Microcontroller and the LoRa modem.	The SPI communication is enabled in the makefile for the code along with the .h file.

4.9.5. Verification Process

The purpose of the verification below is to prove the transmitter code is capable of communicating with the transmitter hardware to send packets via the LoRa modem and print out the received packets as long as the packets received do not contain erroneous data.

1. Power on the transmitter via USB to a computer and wait until the GPS module gets a satellite fix indicated by a solid red light on the GPS module.
2. Using PuTTY or another serial port listener, observe if any data is being printed to the terminal.
3. Confirm the output of longitude, latitude, and time data packets at a rate of once per second via a terminal on a computer plugged into the transceiver’s USB port.
4. Connect an oscilloscope to the SPI MOSI terminal of the LoRa module on the transmitter and see if data is coming in with the proper frequency (12KHz).
5. Verify the latitude and longitude being received are accurate values by plugging them into google maps.

4.9.6. References and Files Links

4.9.6.1. References

[1] Semtech, “SX1276/77/78/79 Datasheet,” [Online] Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE [Accessed: 11-Feb-2023].
[2] Raspberry Pi, “RP2040 Datasheet - A Microcontroller by Raspberry Pi” [Online] Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf [Accessed: 11-Feb-2023].

4.9.6.2. Files Links

1) Transmitter Code Github

4.9.7. Revision Table

Author	Date	Description
Sean Bullis	5/4/23	Section Creation and Content Added

5. System Verification Evidence

5.1. Universal Constraints

5.1.1. The system may not include a breadboard

The system, as shown below, does not have a breadboard. All electrical circuits are either in off-the-shelf systems or assembled onto custom made PCBs.

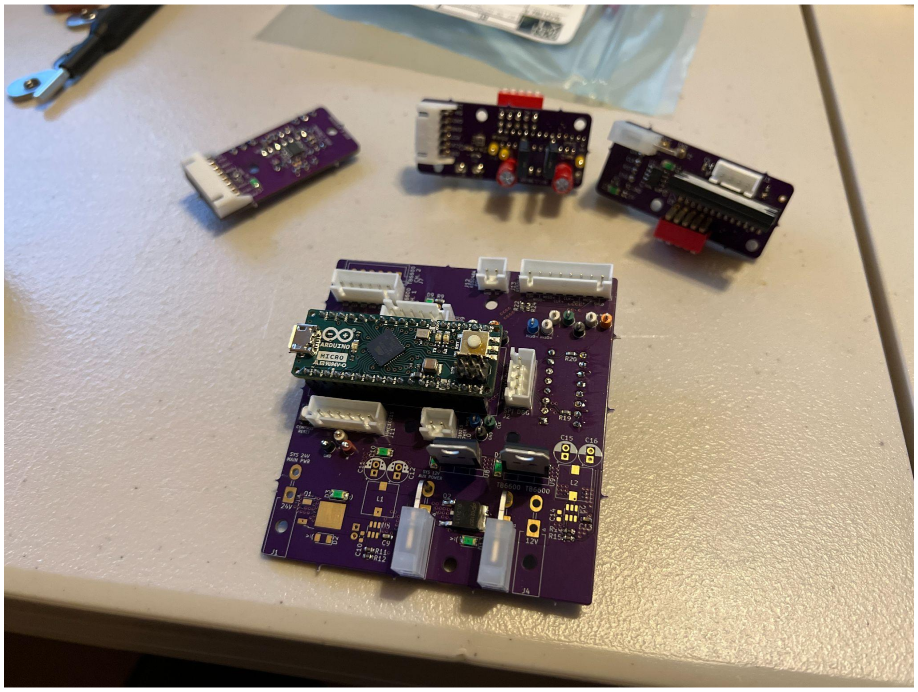


Fig. 5.1.1.1 - Controller Boards
Encoder (Top Left), Stepper Driver (Middle, Right), Arduino Carrier (Middle)

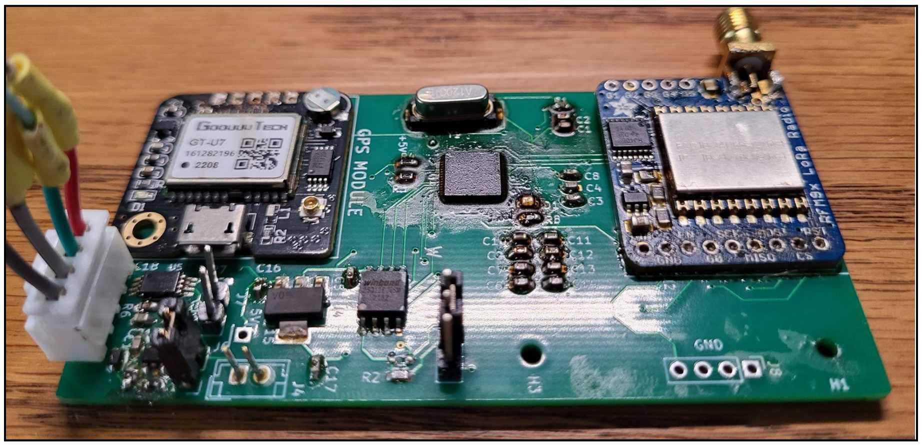


Fig. 5.1.1.2 - Universal Transceiver Board

5.1.2. The final system must contain a student designed PCB.

The system contains multiple student designed PCB's for the transmitter and receiver combination and for the turntable controller.

Figure 5.1.2.1 shows the PCB design from KiCad for the transceiver which depending on the hardware can be operated as a transmitter or receiver. Figure 5.1.2.2 and Figure 5.1.2.3 show the physical PCB in its transmitter configuration and its receiver configuration respectively. According to KiCad, the tool the transceiver was designed in, there are 189 pads on the PCB.

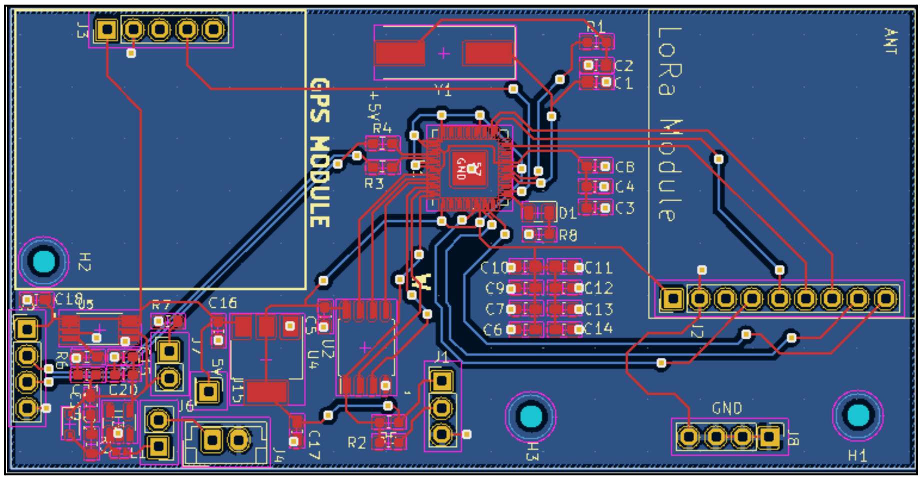


Fig. 5.1.2.1 - Transceiver PCB Design

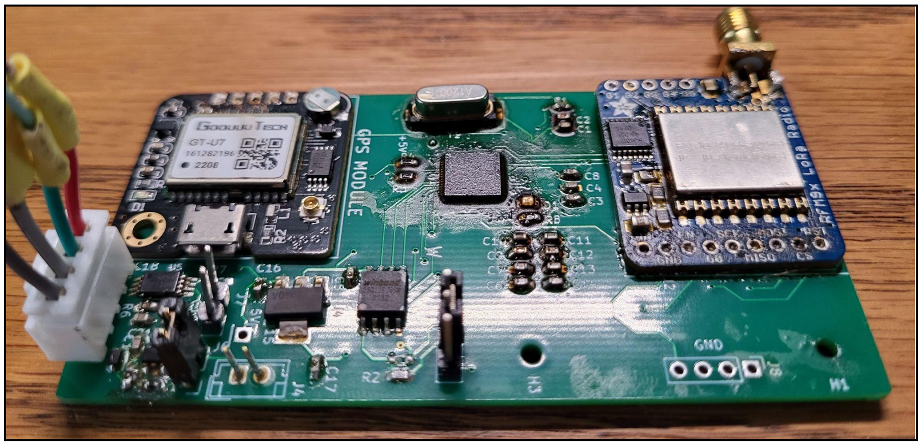


Fig. 5.1.2.2 - Transceiver implemented as a transmitter

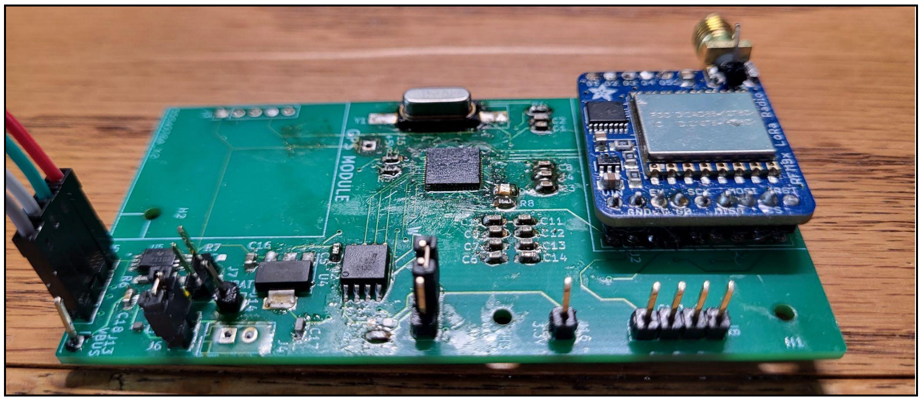


Fig. 5.1.2.3 - Transceiver implemented as a receiver

5.1.3. All connections to PCBs must use connectors

The PCB for the transceiver has four connections, two of which are JST connectors and two of which are male pin headers. One JST connector connects a breakout USB cable to the PCB and the other JST connector connects the external battery. One of the male pin header connections acts as a bridge to the switch on the enclosure that sets whether or not the battery is on, and the other set connects to the LED on the enclosure to indicate that the battery is running. There are no connectors that are directly soldered or otherwise not using connectors on the transceiver PCB.

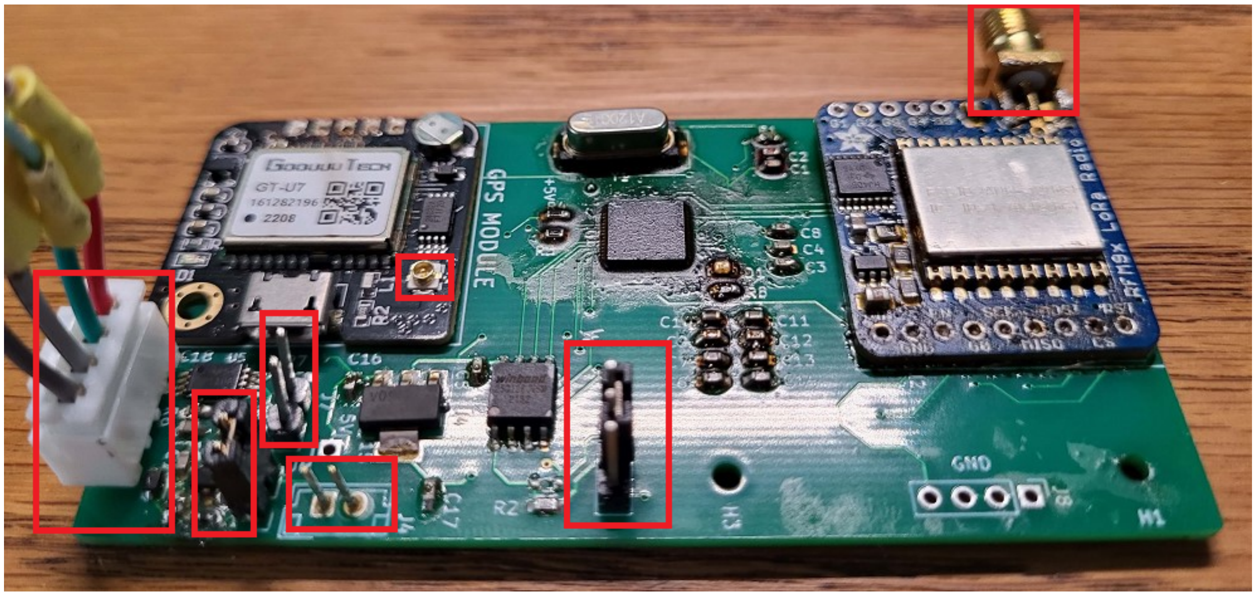


Fig. 5.1.3.1 - Transmitter PCB showing 7 connectors.

From left to right there is a USB cable connected via JST. There are 3 two-pin header connectors for connections to enclosure devices. There is a IPEX antenna mount for the GPS module. There is a 3 pin header to connect to a switch on the enclosure to control the boot-state of the microcontroller. There is an SMA connector to a high gain Molex antenna.

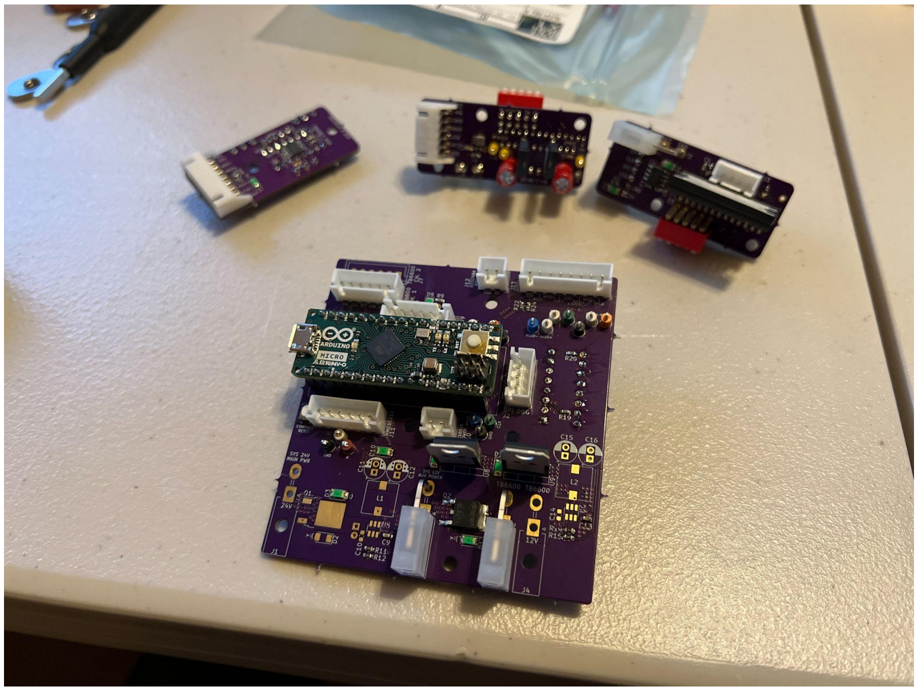


Fig. 5.1.3.2 - Controller Connections

Similarly, the controller hardware uses a mixture of JST XH series connections and Molex Mini-Fit JR for all connections. Below is an image of the wiring used and connectors are clear between cable and boards.

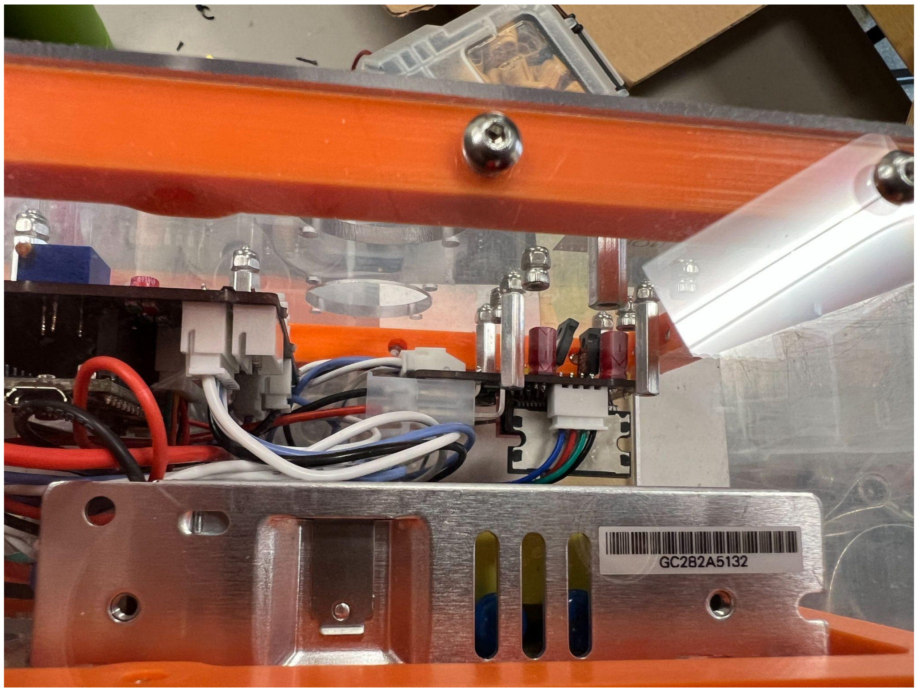


Fig. 5.1.3.3 - Control Box Wiring

5.1.4. All power supplies in the system must be at least 65% efficient.

For the Turntable-Controller block, it is supplied 12 V from an LRS-50-12 AC to DC converter that is quoted from the datasheet to be 86% efficient. For the block PCBs, 9 V is generated by an STMicroelectronics LS7809 linear regulator. To compute efficiency, 9/12 is simply 75%. But, generating 5 V using a linear regulator would be too inefficient so a switching regulator based on the Texas Instruments LMR51420YDDCR is used.

Consulting the chart below, it is known that at least 0.05 A will be drawn during normal operation: 0.05 A is the current drawn by the Turntable block encoder system. Since other systems such as motor control pins, timers, and LEDs will be running, it is likely that nominal current draw on 5 V will exceed 0.05 A.

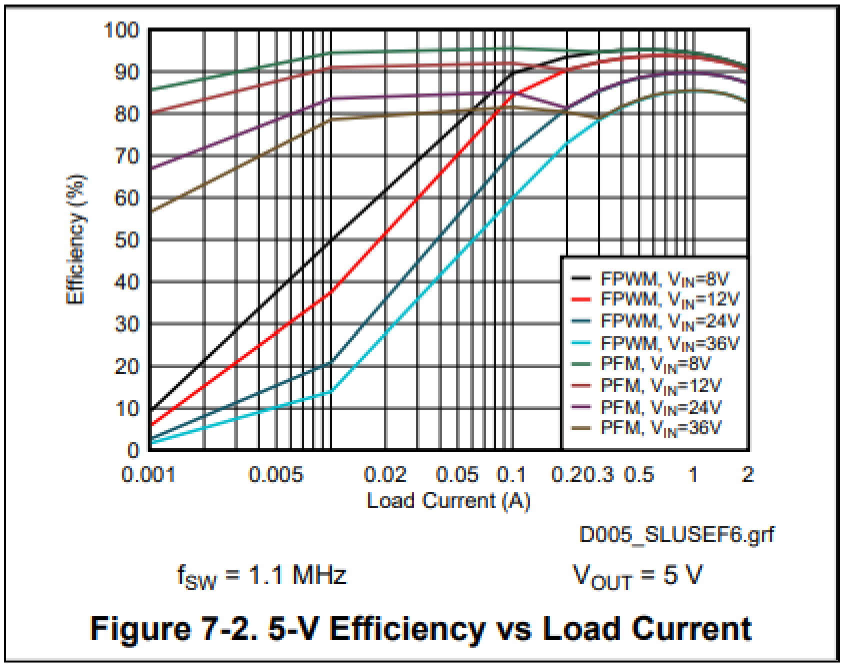


Fig. 5.1.4.1 - LMR51420YDDCR Efficiency Versus Load

Using > 0.05 A of load alongside a 12 V input and an 1.1 MHz FPWM version of the LMR51420YDDCR, the efficiency is at least 70% which fulfills the 65% minimum.

For the transmitter the input USB voltage is 5V and the system voltage is 3.3V using a linear voltage regulator which maintains a 3.3V output. The output to input ratio is 3.3:5 which is 66%. This fulfills the 65% minimum requirement.

It is worth noting that the battery system is not 65% efficient as the boost converter has an efficiency of approximately 93% at 100mA current draw, as shown in Figure 5.1.4.2, from 3.7V to 5V volts and then another 66% efficiency from 5V down to 3.3V which results in 61% efficiency for the battery system. This was approved because the battery is not intended to provide main power for the transmitter and is only present as a backup system in the case of rover failure.

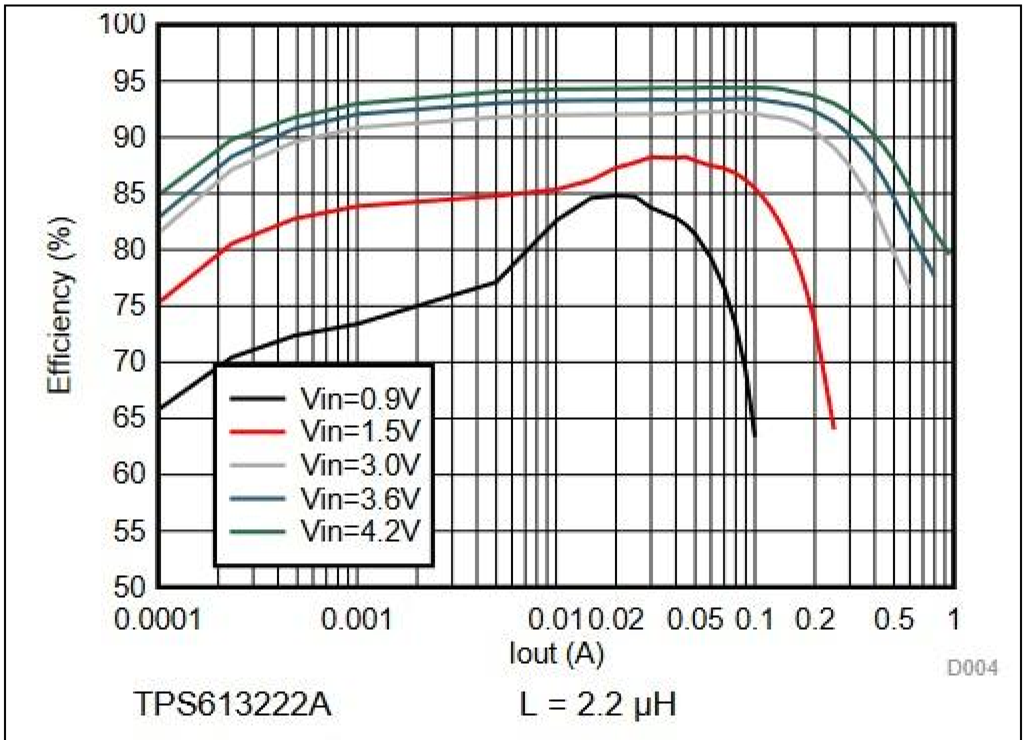


Figure 5. Load Efficiency with Different Inputs

Fig. 5.1.4.2 - Efficiency of the TPS61322 boost converter pulled from the datasheet. With an input voltage of 3.6V drawing 100mA of current the efficiency is approximately 93%.

5.1.5. The system may be no more than 50% built from purchased 'modules.'

Production materials and their sources are detailed below:

Block	**Built or Purchased**	**Comments**
Receiver Hardware	Built	Uses two purchased sub-modules, the LoRa module and the GPS module. Due to the non-trivial nature of incorporating these into the custom microcontroller system built onto the PCB this is considered built but with half purchased components.
Transmitter Hardware	Built	Uses two purchased sub-modules, the LoRa module and the GPS module. Due to the non-trivial nature of

		incorporating these into the custom microcontroller system built onto the PCB this is considered built but with half purchased components.
TX Code	Built	Coded by integrating open source libraries but developed independently.
Rover GPS	Built	Custom embedded C code.
RX Code	Built	Custom embedded C code.
Enclosure	Built	Custom 3D printed enclosure.
Turntable Controller	Built	Custom PCB fabricated by OshPark. Components are off-the-shelf but assembled thus considering them “built”.
Turntable	Built	Aluminum stock was purchased from McMaster and fabricated into components. Plastic components either 3D printed or CNC routed from McMaster purchased stock. Additional wire harnesses are custom made.
Tracking Algorithm	Built	Custom Python code.
UI	Built	Custom Python code.

Table 5.1.5.1 - Component Usage Breakdown

5.2. Requirements

5.2.1. System Accuracy

5.2.1.1. Project Partner Requirement

The system's tracking performance is accurate

5.2.1.2. Engineering Requirement

The system primary antenna sub-system will have a rotation mechanism that shall be accurate to <45° when the rover is 65 meters away.

5.2.1.3. Testing Method

By Inspection.

5.2.1.4. Verification Process

Materials

- OSURC Rover (if available)
- OSURC Groundstation
- System
- > 65 m tape measure
- Protractor

1. Power on system, groundstation: perform setup actions
2. Move the Rover or Rover mounted GPS unit to >= 65 m away from groundstation GPS
3. Move the Rover or Rover mounted GPS unit left or right
4. If possible, move the GPS unit in a circle around the groundstation
5. Visually inspect the tracking alignment
6. Compute the actual angle using reported GPS coordinates
7. Measure the turntable heading using a protractor
8. Compute error (if any)

5.2.1.5. Pass Condition

System never exceeds 45 degrees of error at 65 m of distance.

5.2.1.6. Testing Evidence

1) System Accuracy Evidence: https://drive.google.com/drive/folders/1MT-oWe74SUMO_BPuBn3LY3fwDfsNVtF4?usp=sharing

5.2.2. System Tracking Speed

5.2.2.1. Project Partner Requirement

The system is fast and comprehensive.

5.2.2.2. Engineering Requirement

The system will rotate the primary antenna using the turntable sub-system at a minimum angular velocity of 5°/s.

5.2.2.3. Testing Method

By Test.

5.2.2.4. Verification Process

Materials

- System
- OSURC Rover Ground Station
- Timer

1. Turn on system power
2. Wait for system to align to 0°

3. Turn on OSURC Rover Ground Station and launch control Python scripts
4. Manually command the turntable to 181.0°: antenna will rotate left -179.0° +/- 5°
5. Set timer to 00:00
6. Manually command the turntable to 179.0° AND start timer: antenna will rotate right +179° +/- 5°
7. Stop timer when antenna stops: the rotation mechanism will have traveled at least 348° (-179° + 5° --> zero --> 5° - 179.0°)
8. Divide 348° by time to computer angular velocity

5.2.2.5. Pass Condition

Calculated angular velocity of greater than 5 °/s.

5.2.2.6. Testing Evidence

1) System Tracking Speed Evidence: https://drive.google.com/file/d/1nuDwsXQXzKhutJFxQHdvQpJxFIPpsCyU/view?usp=sharing

5.2.3. System Field of View

5.2.3.1. Project Partner Requirement

Tracking is fast and comprehensive.

5.2.3.2. Engineering Requirement

The system will rotate the primary antenna using the turntable sub-system in an angle range of 0° - 360°.

5.2.3.3. Testing Method

By Inspection.

5.2.3.4. Verification Process

Materials

- System
- OSURC Rover Ground Station

1. Turn on system power
2. Wait for system to align to 0°
3. Turn on OSURC Rover Ground Station and launch control Python scripts
4. Manually command the turntable to 180.0°: antenna will rotate right 180.0° +/- 5° 5. Manually command the turntable to 180.1°: antenna will rotate left 359.9° +/- 5°

5.2.3.5. Pass Condition

Turntable mechanism covers the entire rotation range.

5.2.3.6. Testing Evidence

1) System Field of View Evidence: https://drive.google.com/file/d/1nuDwsXQXzKhutJFxQHdvQpJxFIPpsCyU/view?usp=sharing

5.2.4. Noise Tolerance

5.2.4.1. Project Partner Requirement

Tracking method is unaffected by outside signals

5.2.4.2. Engineering Requirement

The system will have a packet loss of less or equal to 5% over 2 hours of activity.

5.2.4.3. Testing Method

By Inspection.

5.2.4.4. Verification Process

1. Set up the transmitter to begin transmitting packets. A packet number associated with the packet being sent will be appended to the transmission.
2. Set up the receiver to begin receiving packets. The packet number appended in the transmission will be used to calculate how many packets are received versus expected. This will print to the terminal of the computer the receiver is connected to.
3. Letting the transmitter transmit for 2 hours a running total of packet loss will be displayed along with the contents of the transmission.
4. Confirm for a 2 hour period the packet loss rate is less than or equal to 5%.

5.2.4.5. Pass Condition

A print out of the running total for packet loss is less than or equal to 5%.

5.2.4.6. Testing Evidence

1) System Noise Tolerance: https://drive.google.com/file/d/1INAZQhA2z7LImkt6TVPxPrH6uQWOBtPV/view?usp=sharing

5.2.5. System Isolation

5.2.5.1. Project Partner Requirement

Tracking method does not interfere with Rover.

5.2.5.2. Engineering Requirement

The system will be self contained and self powered so that it will track location even when the rover is not operational.

5.2.5.3. Testing Method

By Demonstration.

5.2.5.4. Verification Process

1. Run the transmitter with no connection to USB.
2. Connect the receiver to a computer and wait until packets begin being received.
3. This will confirm that the transmitter is able to transmit even when not powered by the rover.

5.2.5.5. Pass Condition

Reception of packets on the receiver.

5.2.5.6. Testing Evidence

1) System Isolation (Youtube): https://youtube.com/shorts/ao4RAZTD53A?feature=share
2) System Isolation (Google Drive): https://drive.google.com/file/d/1q2ynfjiUNPIv0i5S7O3A2fcvhQMLGA0N/view?usp=sharing

5.2.6. System Portability

5.2.6.1. Project Partner Requirement

Antenna mount should be easy to set up and transport

5.2.6.2. Engineering Requirement

The system will weigh no more than 200lbs and no single module shall exceed 50lbs.

5.2.6.3. Testing Method

By Inspection.

5.2.6.4. Verification Process

Materials:

- System
- 5/32" Hex driver
- Scale with lbs measurement
- Calibration weight

1. Weight calibration weight, note deviations
2. Place foundation subassembly on scale
3. Place tower-cradle subassembly on scale
4. Tabulate weights and add/subtract deviation from step 1

5.2.6.5. Pass Condition

Total system weighs less than 200 lb and individual subassembly modules weigh less than 50 lb.

5.2.6.6. Testing Evidence

1) System Portability Evidence: https://drive.google.com/drive/folders/16Cgb8JZjdL2myaBVj7o2ePytjgV3n7Xe?usp=sharing

5.2.7. System Setup

5.2.7.1. Project Partner Requirement

Antenna mount should be easy to set up and transport.

5.2.7.2. Engineering Requirement

The system will require no more than 15 minutes for 9 out of 10 people (working in a 4 person team) to initialize the system electrical/software sub-systems and level the turntable sub-system to +/-15° relative to level ground.

5.2.7.3. Testing Method

By Demonstration.

5.2.7.4. Verification Process

Materials:

- System
- Timer
- OSURC Ground Station
- 5/32" hex head screwdriver
- Power drill
- > 6 OSURC members

1. From available OSURC members, form at a 4-person team
2. Stage system on unlevel terrain outside (weather permitting) or inside with terrain analogs i.e. metal, plastic, wood blocks
3. Repeat the following for each team
4. Zero and start timer
5. Level system using built-in leveling feet and indicators: indicators show a maximum deviation of +/- 15°
6. Power on system
7. Call and record time for each team

5.2.7.5. Pass Condition

Time logged for 9 out of 10 members should not exceed 15 minutes.

5.2.7.6. Testing Evidence

1) System Setup (Videos): https://drive.google.com/drive/folders/1gf3SyEnM_UIlRvIvwXbMSIc2ew6h5CNv?usp=sharing

2) System Setup (timesheet): https://docs.google.com/spreadsheets/d/1JuAbAHxVd1yYRojl5HFAKLFV65yoe17BtJJQCSATnKk/edit?usp=sharing

5.2.8. System Update Rate

5.2.8.1. Project Partner Requirement

Tracking is fast and comprehensive.

5.2.8.2. Engineering Requirement

The system should nominally update the angle of rotation once per second with a slowest acceptable rate of one update every 5 seconds.

5.2.8.3. Testing Method

5.2.8.4. Verification Process

1. Set up the transmitter and receiver in normal operation.
2. Once the receiver starts receiving packets, check the timestamps on each received packet.
3. Ensure that the time stamps between packets isn't more than 5 seconds or less than 1 second.

5.2.8.5. Pass Condition

Time stamp difference between adjacent received packets to be between 1 and 5 seconds.

5.2.8.6. Testing Evidence

1) System Update Rate: https://drive.google.com/file/d/1Bwjwc1gNeN5Ucdoj0L8YoaWqcvN0SteC/view?usp=share_link

5.3. References and File Links

5.3.1. References

5.3.2. File Links

5.4. Revision Table

Author	Date	Description
Angel Huang	4/28/23	Section Creation and Added Content for Requirements and Constraints
Sean Bullis	5/4/23	Added details for requirements and constraints related to the transmitter hardware and PCB.

Kira Kopcho	5/14/23	Minor formatting changes to make section match the style of the rest of the document

6. Project Closing

6.1. Future Recommendations

6.1.1. Technical Recommendations

It is recommended that subsequent teams:

a) Implement realistic parameters for interfaces, components, requirements, and other project components.

During system and interface requirements and hardware development, and later testing and revision, several parameters were ambitious in scope. An example was the original rotation speed of 22.5 °/s that was reduced to 5 °/s after discussing whether or not the system's motors could achieve 22.5 °/s.

To prevent such occurrence, we recommend incorporating feasibility "studies", or research, during Fall Term alongside thorough design reviews and discussions with associated parties.

b) Test power handling circuitry early.

As part of block and system testing, it was discovered that some power processing circuits were not efficient enough, due to linear regulators, or were problematic. With power being the cornerstone of any PCB, this is recommended to prevent damage and frustration in later project stages.

Like part "a" above, we recommend through design reviews alongside earlier testing via prototypes. Additionally, the team can work with OSURC and either develop new circuits or use, with permission, a pre-existing circuit.

c) Ensure mechanical parts can be fabricated or reproducible though alternative manufacturing techniques.

The system uses 3D printing, precision drilling, and CNC routing to produce the mechanical components in the turntable and foundation sub-systems. The first two are readily available but the last is not due to being supported by a team member and not OSU or OSURC and will become an issue if/once parts break or degrade.

Future teams could find a 3rd party CNC machining service though this is not practical due to processing times and cost. Alternatively, they could revise the parts to simplify geometry and/or use 3D printing or milling processes.

d) Conduct design reviews

During the project, our design reviews were not as thorough as we would've liked and some issues encountered might've been solved with a second or third pair of eyes.

For future projects, design reviews should, ideally, be held after designs are prototyped and at a "ready to build/order" state. Ultimately, the design review schedule should be determined by the future team but the review should be as thorough as possible.

6.1.2. Global Impact Recommendations

It is recommended that subsequent teams:

a) Reduce plastic waste through efficient part design and layout

3D printing and CNC cutting were unavoidable but could be optimized to reduce unrecyclable plastic waste generation.

For 3D prints, parts could have features not requiring support (which also improves quality), be small, or multifunctional. And for CNC cutting, the laying out of flat pieces shall be compacted to reduce excessive trims/edges.

A final option is to use recyclable plastics but the fabrication process tends to remove identifying features and thus ability to recycle said plastic.

b) Reduce PCB waste using thorough design reviews and proper recycling

Like point “a” the development of PCBs is going to require prototypes and extra components and thus potentially result in waste. For instance, the turntable sub-system controller required three revisions each with three PCBs per order due to mistakes and changing requirements.

Reduction of waste is primarily done through design reviews to catch the mistakes or revisions that would cause new PCB and component orders. Additionally, a future team should have requirements for the system and blocks firmly established before Winter Term. Finally, the waste generated should be properly disposed of via OSU or an external service such as state operated waste transfer stations.

6.1.3. Teamwork Impact Recommendations

It is recommended that subsequent teams:

a) Include an individual with mechanical design and have access to machining processes.

While this project is primarily electrical and software, there will undoubtedly be processes requiring mechanical knowledge alongside specialized machinery, and software.

Since mechanical engineering concepts are not taught in the ECE curriculum, we recommend working with OSURC’s mechanical engineering teams or having one team member teach themselves how to use Autodesk Inventor.

b) Establish communications with associated parties, the OSU Robotics Club, and provide consistent updates though a consistent person(s).

Some project blocks will be attached to pre-existing or in-development systems that OSURC is working on. The OSURC team needs to know such things as: power requirements, space and mounting requirements, et cetera.

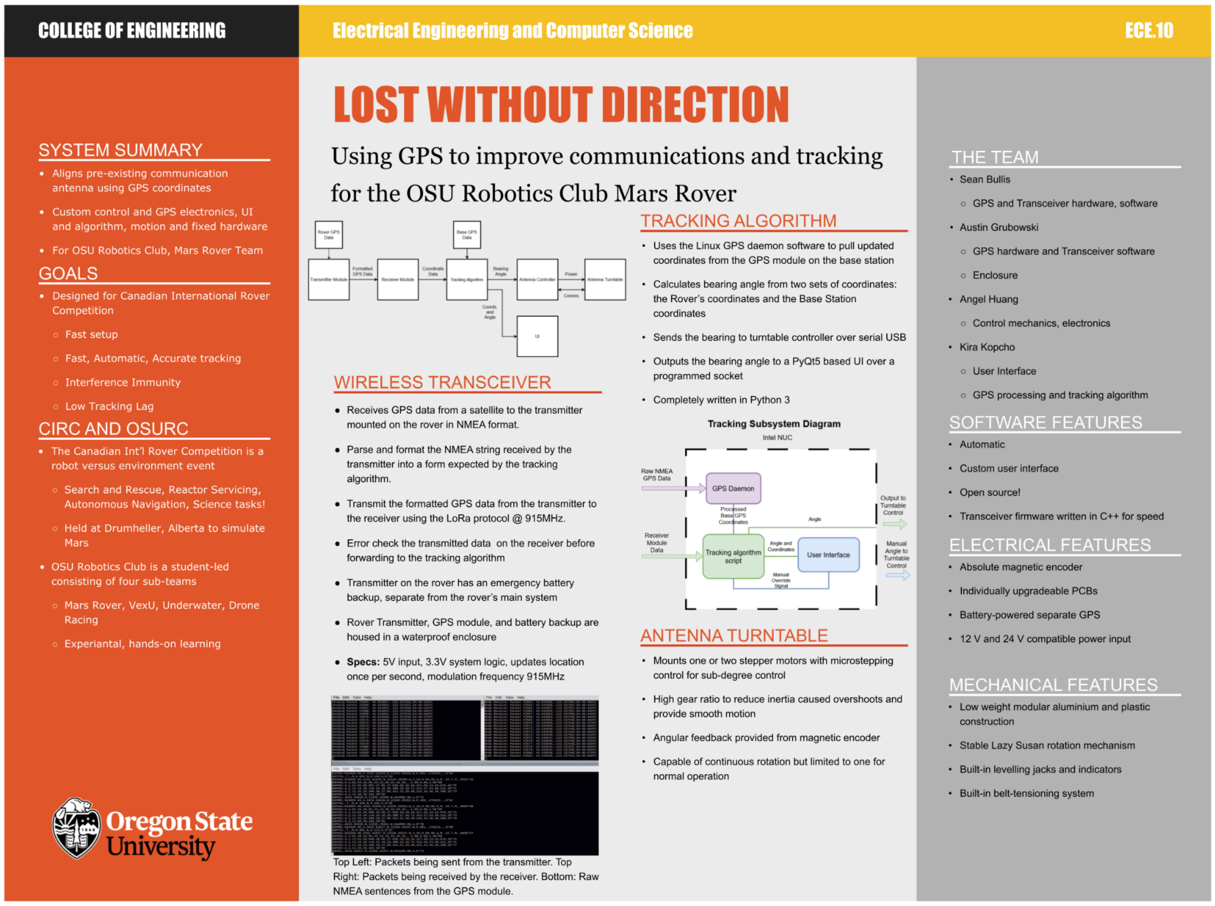
Additionally, communication between this and OSURC teams was spotty and usually between multiple people related/unrelated to the problems at hand.

We simply recommend updating OSURC during their daily meetings through the parties associated with any current problems or tasks.

6.2. Project Artifact Summaries with Links

1) Github: Link
2) Google Drive with schematics, mechanical files: Link
3) Google Drive with manuals, instructions: Link

6.3. Presentation Materials



COLLEGE OF ENGINEERING

Electrical Engineering and Computer Science

ECE.10

LOST WITHOUT DIRECTION

Using GPS to improve communications and tracking for the OSU Robotics Club Mars Rover

SYSTEM SUMMARY

- Aligns pre-existing communication antenna using GPS coordinates
- Custom control and GPS electronics, UI and algorithm, motion and fixed hardware
- For OSU Robotics Club, Mars Rover Team

GOALS

- Designed for Canadian International Rover Competition
  - Fast setup
  - Fast, Automatic, Accurate tracking
  - Interference Immunity
  - Low Tracking Lag

CIRC AND OSURC

- The Canadian Int'l Rover Competition is a robot versus environment event
  - Search and Rescue, Reactor Servicing, Autonomous Navigation, Science tasks!
  - Held at Drumheller, Alberta to simulate Mars
- OSU Robotics Club is a student-led consisting of four sub-teams
  - Mars Rover, VexU, Underwater, Drone Racing
  - Experiantal, hands-on learning

Oregon State University

WIRELESS TRANSCEIVER

- Receives GPS data from a satellite to the transmitter mounted on the rover in NMEA format.
- Parse and format the NMEA string received by the transmitter into a form expected by the tracking algorithm.
- Transmit the formatted GPS data from the transmitter to the receiver using the LoRa protocol @ 915MHz.
- Error check the transmitted data on the receiver before forwarding to the tracking algorithm
- Transmitter on the rover has an emergency battery backup, separate from the rover's main system
- Rover Transmitter, GPS module, and battery backup are housed in a waterproof enclosure
- **Specs:** 5V input, 3.3V system logic, updates location once per second, modulation frequency 915MHz

Top Left: Packets being sent from the transmitter. Top Right: Packets being received by the receiver. Bottom: Raw NMEA sentences from the GPS module.

TRACKING ALGORITHM

- Uses the Linux GPS daemon software to pull updated coordinates from the GPS module on the base station
- Calculates bearing angle from two sets of coordinates: the Rover's coordinates and the Base Station coordinates
- Sends the bearing to turntable controller over serial USB
- Outputs the bearing angle to a PyQt5 based UI over a programmed socket
- Completely written in Python 3

Tracking Subsystem Diagram

ANTENNA TURNTABLE

- Mounts one or two stepper motors with microstepping control for sub-degree control
- High gear ratio to reduce inertia caused overshoots and provide smooth motion
- Angular feedback provided from magnetic encoder
- Capable of continuous rotation but limited to one for normal operation

THE TEAM

- Sean Bullis
  - GPS and Transceiver hardware, software
- Austin Grubowski
  - GPS hardware and Transceiver software
  - Enclosure
- Angel Huang
  - Control mechanics, electronics
- Kira Kopcho
  - User Interface
  - GPS processing and tracking algorithm

SOFTWARE FEATURES

- Automatic
- Custom user interface
- Open source!
- Transceiver firmware written in C++ for speed

ELECTRICAL FEATURES

- Absolute magnetic encoder
- Individually upgradeable PCBs
- Battery-powered separate GPS
- 12 V and 24 V compatible power input

MECHANICAL FEATURES

- Low weight modular aluminium and plastic construction
- Stable Lazy Susan rotation mechanism
- Built-in levelling jacks and indicators
- Built-in belt-tensioning system

Figure 6.3.1: Presentation Poster

Full size poster accessible at: [Link]()
Showcase accessible at:

Appendix

File Links

1) Jira Timeline: https://drive.google.com/file/d/1CoS4GB7AdC4U7T53uIHK8zXw_XgSJR3a/view?usp=sharing
2) RX Module PySerial Test Code: https://github.com/kira-the-engineer/OSURC-RDF-Tracking-Algo/blob/main/rover_gps/serial_test.py
3) Tracking Algorithm Code: https://github.com/kira-the-engineer/OSURC-RDF-Tracking-Algo/blob/main/rover_gps/tracking_algorithm.py
4) Semtech, "SX1276/77/78/79 Datasheet," [Online] Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE [Accessed: 11-Feb-2023].
5) Raspberry Pi, "RP2040 Datasheet - A Microcontroller by Raspberry Pi" [Online] Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf [Accessed: 11-Feb-2023].
6) GPS Test Code: https://drive.google.com/file/d/1V6JCfHaPI86UXx0uz5sLqmQ9VrP5L_-o/view?usp=drivesdk
7) GPS Code Integrated into the Transmitter Code: https://drive.google.com/file/d/1rF8ZZS-Qw9_9L2kERjRCqD2jdvwiPEKP/view?usp=drivesdk
8) Transmitter Hardware Master Block Diagram: https://ibb.co/0c2jTL5
9) Transmitter Hardware KiCad Schematics/PCB Design: https://github.com/bulliss93/Rover_RDF_Capstone/tree/main/Transceiver%20KiCad%20Project
10) TB6600 Datasheet: https://toshiba.semicon-storage.com/info/docget.jsp?did=14683&prodName=TB6600HG
11) AS5048A Datasheet: https://ams.com/documents/20143/36005/AS5048_DS000298_4-00.pdf
12) Molex 5569 Datasheet: https://www.molex.com/webdocs/datasheets/pdf/en-us/0026013114_PCB_HEADERS.pdf
13) JST XH Datasheet: https://www.jst.com/wp-content/uploads/2021/01/eXH-new.pdf
14) MAX232 Datasheet: https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fmax232
15) L78SXX Datasheet: https://www.st.com/content/ccc/resource/technical/document/datasheet/e9/be/53/a3/1f/6f/4f/75/CD00000449.pdf/files/CD00000449.pdf/jcr:content/translations/en.CD00000449.pdf

16) MCP1501-33xSN Datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/20005474E.pdf
17) MP915-0.20-1% Datasheet:
http://www.caddock.com/Online_catalog/Mrktg_Lit/MP9000_Series.pdf
18) SN75C1168N Datasheet:
https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fsn65c1168
19) 1000680108 Datasheet: https://www.molex.com/pdm_docs/sd/1000680108_sd.pdf
20) SXH-001T-P0.6 Datasheet: https://www.jst-mfg.com/product/pdf/eng/eXH.pdf
21) USB-RS232-WE-1800-BT_5.0 Datasheet:
http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_USB_RS232_CABLES.pdf
22) LRS-50-12 Datasheet:
https://www.meanwellusa.com/upload/pdf/LRS-50/LRS-50-spec.pdf
23) falstad 50 % Duty Cycle Clock Generator Circuit:
https://drive.google.com/file/d/1y9Z1gmqx6X80IfSl0sj19NV8qep19TuO/view?usp=sharing
24) AS5048A Datasheet:
https://ams.com/documents/20143/36005/AS5048_DS000298_4-00.pdf
25) SN75C1168 Datasheet:
https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fsn65c1168
26) Full UI Code:
https://github.com/kira-the-engineer/OSURC-RDF-Tracking-Algo/tree/main/UI
27) Receiver Code:
https://drive.google.com/file/d/1DGDJGhgGxPoLZCKzkVGwpo5AFW8kS-nG/view?usp=sharing
28) Receiver CMakeLists:
https://drive.google.com/file/d/1Ah0BCqrQd3v2QfV8YBtBraRIB3RfE6TV/view?usp=sharing
29) Receiver CMake File:
https://drive.google.com/file/d/1HksnLogDHZxTHJ_7zE_J2TICSrPDUSYO/view?usp=sharing
30) Receiver LoRa.h:
https://drive.google.com/file/d/1yR-YXFBH2KXLO9g4hBhGhy65XfYY5vN4/view?usp=sharing
31) Receiver LoRa.cpp:
https://drive.google.com/file/d/1DcJsCEDySoyNGLJYjnMaG6eEvOo7zAxu/view?usp=sharing
32) Receiver Print.h:
https://drive.google.com/file/d/1Ie-IDsvIk0i68eBVy3d64vierDe4t5WC/view?usp=sharing
33) Receiver Print.cpp:
https://drive.google.com/file/d/1tZAurQnRsHmb4oo6_pI3veBBn7MvgOmA/view?usp=sharing

34) Transmitter Code Github: https://github.com/bulliss93/Rover_RDF_Capstone/tree/main/Transmitter_code
35) System Accuracy Evidence: https://drive.google.com/drive/folders/1MT-oWe74SUMO_BPuBn3LY3fwDfsNVtF4?usp=sharing
36) System Tracking Speed Evidence: https://drive.google.com/file/d/1nuDwsXQXzKhutJFxQHdvQpJxFIPpsCyU/view?usp=sharing
37) System Field of View Evidence: https://drive.google.com/file/d/1nuDwsXQXzKhutJFxQHdvQpJxFIPpsCyU/view?usp=sharing
38) System Noise Tolerance: https://drive.google.com/file/d/1INAZQhA2z7LImkt6TVPxPrH6uQWOBtPV/view?usp=sharing
39) System Isolation (Youtube): https://youtube.com/shorts/ao4RAZTD53A?feature=share
40) System Isolation (Google Drive): https://drive.google.com/file/d/1q2ynfjiUNPIv0i5S7O3A2fcvhQMLGA0N/view?usp=sharing
41) System Portability Evidence: https://drive.google.com/drive/folders/16Cgb8JZjdL2myaBVj7o2ePytjgV3n7Xe?usp=sharing
42) System Setup (Videos): https://drive.google.com/drive/folders/1gf3SyEnM_UIlRvIvwXbMSIc2ew6h5CNv?usp=sharing
43) System Setup (timesheet): https://docs.google.com/spreadsheets/d/1JuAbAHxVd1yYRojI5HFAKLFV65yoe17BtJJQCSATnKk/edit?usp=sharing
44) System Update Rate: https://drive.google.com/file/d/1Bwjwc1gNeN5Ucdoj0L8YoaWqcvN0SteC/view?usp=share_link
45) Github: https://github.com/OSURoboticsClub/Rover_RDF_Capstone
46) Google Drive with schematics, mechanical files: https://drive.google.com/drive/folders/1eabaa22pd-X_umyBzblk1s9yCK5jU_xw?usp=share_link
47) Google Drive with manuals, instructions: https://drive.google.com/drive/folders/1AitQD8xjyh8UOPr-0d0mMcfvInuupwlk?usp=share_link

Code Snippets

1) Rover GPS Receiver Code

```
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "hardware/uart.h"
```

```
#include "pico/time.h"
#include "pico/multicore.h"

// UART definitions
#define UART_ID uart0
#define BAUD_RATE 9600

// Configure LED pin
#define LED_PIN 25

// Pins GP16 and GP17 are being used for testing purposes with a
pico;
// pins 0(TX) and 1(RX) will be used when this code is uploaded to
the
// transceiver.
#define UART_TX_PIN 16
#define UART_RX_PIN 17

// Definitions used by the parse_nmea_sentence function
#define NMEA_BUF_SIZE 100
#define NMEA_SENTENCE_GPGGA "$GPGGA"

// Define the NMEA_data struct
typedef struct {
  double latitude;
  double longitude;
  int hour;
  int minute;
  int second;
} NMEA_data;

// NMEA_data parsing function which takes in an NMEA sentence and
fills in the
// longitude, latitude, and time members of the NMEA_data struct.
NMEA_data parse_nmea_sentence(const char *sentence) {
  NMEA_data data = {0};

  if (strstr(sentence, NMEA_SENTENCE_GPGGA) == sentence) {
    char *p = (char *)sentence;

    // move pointer to time
    p = strchr(p, ',') + 1;

    // parse time
    int hour, minute, second;
    sscanf(p, "%2d%2d%2d", &hour, &minute, &second);
    data.hour = hour;
    data.minute = minute;
    data.second = second;

    // move pointer to latitude
    p = strchr(p, ',') + 1;

    // parse latitude
    double latitude, latitude_minutes;
    sscanf(p, "%2lf%lf", &latitude, &latitude_minutes);
    data.latitude = latitude + latitude_minutes / 60.0;

    // move pointer to N/S indicator
    p = strchr(p, ',') + 1;

    // check N/S indicator and adjust latitude
    if (*p == 'S') {
      data.latitude = -data.latitude;
    }

    // move pointer to longitude
    p = strchr(p, ',') + 1;

    // parse longitude
    double longitude, longitude_minutes;
    sscanf(p, "%3lf%lf", &longitude, &longitude_minutes);
    data.longitude = longitude + longitude_minutes / 60.0;

    // move pointer to E/W indicator
    p = strchr(p, ',') + 1;

    // check E/W indicator and adjust longitude
    if (*p == 'W') {
      data.longitude = -data.longitude;
    }
  }

  return data;
}
```

2) Main Code and Valid Check Function:

```
void print_if_valid(const char *str) {
    int len = strlen(str);
    int i;
    int has_extra_character = 0;
    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' &&
str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i] != '7' && str[i] !=
'8' && str[i] != '9' && str[i] != ',' && str[i] != ':' && str[i] != '-' &&
str[i] != '.' && str[i] != 'U' && str[i] != 'T' && str[i] != 'C') {
            has_extra_character += 1;
        }
    }
    if (!has_extra_character) {
        printf("%s\n", str);
    }
    has_extra_character = 0;
}
int main(){
      stdio_init_all();
      if (!LoRa.begin(915E6)){
            printf("Starting LoRa failed!\n");
            while(1){
                  printf("Failed!\n");
            }
      }
      printf("LoRa Started\n");
      LoRa.receive();
      int counter=0;
      while(1){
            int packetSize = LoRa.parsePacket();
            if(packetSize > 0){
                  string message = "";
                  while(LoRa.available()){
                  message += (char)LoRa.read();
                  }
            print_if_valid(message.c_str());
            }
       return 1;
}
```