L-Band Satellite Tracking and Characterization System

Detailed Design Document

Authors:

* Charles Kente Van Horn Jr. (Team Manager) - RF and Microwave Engineering (EE)
* Samuel Walker Taliaferro VIII Velasquez (Software Backend and Software Architecture lead)- Machine Learning (CPE)
* Justin Ward Daigle- Communications (EE)
* Ethan Christopher Duval - Networking and Cyber Security (CPE)
* Jared Lane Jamison - Controls Robotics and Autonomy (EE)

Date: Nov 12, 2023

Abstract

Satellite Tracking and Characterization System. The system aims to provide efficient and accurate tracking and data collection of L-Band satellites using a Raspberry Pi 4 as the main processing platform. The design consists of several interconnected subsystems, including the computer system, operations software, mount subsystem, enclosure, receiver, power system, and antenna. The document provides a comprehensive overview of each subsystem, its specifications, connections, and implementation. Additionally, the document includes an acceptance test plan to evaluate the system's performance and ensure it meets the design requirements.

Table of contents:

[**1.1 Computer Introduction 2**](#_heading=h.bwna42gbwl4v)

[**1.2 Computer System Specifications 3**](#_heading=h.mexu0yu24nl1)

[**1.3 Component Connections 3**](#_heading=h.8n42ayi75136)

[**1.4 Computer Implementation 4**](#_heading=h.qv3v75kijtj2)

[**2.1 Operations Software Introduction 4**](#_heading=h.9z8bb6e8h9dz)

[**2.2.1 User Interface (GUI) 6**](#_heading=h.uo7khwftfhkh)

[**2.2.2 Client-Server Communication (CSComm) 7**](#_heading=h.965ts4or3zcn)

[**2.2.3 Server-Client Communication (SCComm) 8**](#_heading=h.scibhy6uo7i2)

[**2.2.4 SatelliteTracker 8**](#_heading=h.u9yorqynl4w3)

[**2.2.5 TLEParser 9**](#_heading=h.dybskwq70444)

[**2.2.6 Scheduler 10**](#_heading=h.qn3h6g9lmz8a)

[**2.2.7 RSPDuo Interface 11**](#_heading=h.6q9wv2da1mx)

[**2.2.8 Antenna Rotator Interface 11**](#_heading=h.i53meb305ztd)

[**2.2.9 Operations Software Implementation 13**](#_heading=h.jw325wb6rnov)

[**2.3.0 Required Libraries 13**](#_heading=h.ayh4dhz88rau)

[**2.3.1 Executable Creation 13**](#_heading=h.67r2kxezkigg)

[**2.3.2 Raspberry Pi Linux Service for 5G Wi-Fi Hotspot and automatic python script initialization on boot 14**](#_heading=h.a6xhbmnljoi8)

[**2.3.3 Mounting a usb drive to use as a database for backend system 17**](#_heading=h.j322hd24gp3h)

[**3.1 Mount Subsystem 18**](#_heading=h.z8rngon7qk6p)

[**3.2.1 Mount System Requirements 18**](#_heading=h.itnax4jsp7wb)

[**3.2.2 Mount System Derived Requirements 19**](#_heading=h.4zx2rp9z97yu)

[**3.3 Mount Schematic 20**](#_heading=h.wnfgmjmrt641)

[**3.4 Rotator Design 21**](#_heading=h.8xjdb35o6n0c)

[**3.5 Additional Mount Components 23**](#_heading=h.24wcbounfs4n)

[**3.6 Enclosure 25**](#_heading=h.uow16uffk1n7)

[**3.6.1 Enclosure Size 25**](#_heading=h.8wmhqmn02ru3)

[**4.1 Receiver Subsystem 26**](#_heading=h.oue6jmt560na)

[**4.1.1 Filtering Stage 27**](#_heading=h.psbvjm6tzyng)

[**4.1.2 SDR Stage 29**](#_heading=h.qm4xndrq2dvt)

[**5.1 Power System Introduction 29**](#_heading=h.5txif5hhhxar)

[**5.2 Power system specifications 30**](#_heading=h.hlexfsc0eadm)

[**5.2.1 Battery Voltage 31**](#_heading=h.g9r5ogh0r8v4)

[**5.2.2 DC-DC Converters for Voltage Regulation 31**](#_heading=h.gqckh18ybncr)

[**5.2.2.1 Computer power 31**](#_heading=h.ujim1d4259g6)

[**5.2.2.2 Mount Power 32**](#_heading=h.wg86eodnda7e)

[**5.3 Safety 32**](#_heading=h.3io8o6mqetju)

[**5.3.1 Fuses 33**](#_heading=h.ld76t1z9wycm)

[**5.3.2 Emergency Cutoff Switch 33**](#_heading=h.zd3llg2sra4)

[**5.4 Wire Gauge 33**](#_heading=h.2qhhlo3v1aor)

[**6.1 Antenna Subsystem 35**](#_heading=h.aofa13v7r2a4)

[**7 Acceptance Test Plan 36**](#_heading=h.4roj4hhmf3bt)

[**8 Actions Table 37**](#_heading=h.4qp1kivj8909)

# 1.1 Computer Introduction

The computer system for the L-Band Satellite Tracking and Characterization System is responsible for providing the necessary processing power, storage, and connectivity to support the operations software and manage the system's hardware components. This document provides a detailed design explanation for the computer system, including the choice of platform, specifications, and connections.

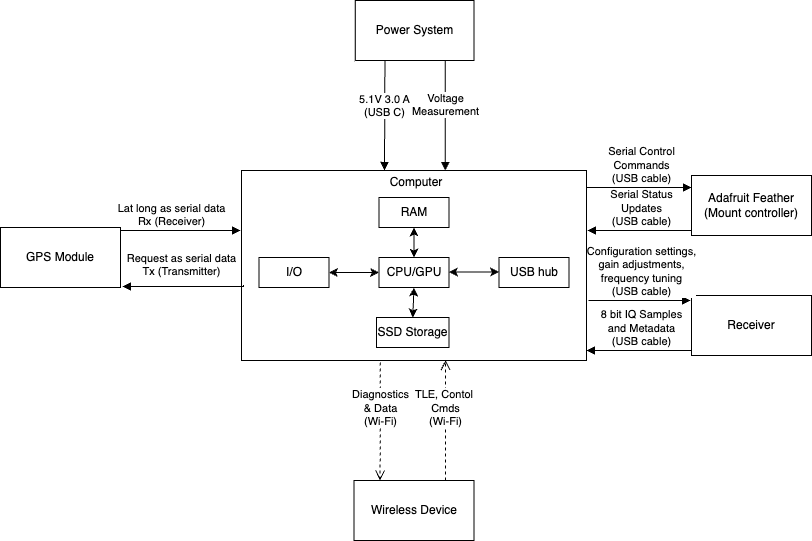
# 1.2 Computer System Specifications

The chosen computer platform is the Raspberry Pi 4 due to its processing power, built-in connectivity options, and support for multiple programming languages. The specifications for the computer system are as follows:

* Platform: Raspberry Pi 4
* RAM: 4GB
* CPU: 1.5GHz
* USB Ports: 4 USB 4.0 ports (for ADC connectivity, USB drive and mount control)
* Storage: 128GB Micro SD card internal, 128GB removable external drive
* Wireless Communication: Built-in 5G Wi-Fi
* Additional Features: SD card port, GPS connection, magnetometer
* Total Cost for computer subsystem: Raspberry Pi 4 with all components estimated at $220.69

# 

# 1.3 Component Connections



* GPS module: Connected to the Raspberry Pi via TX and RX pins, providing time and position information.
* Magnetometer module: Connected to the Raspberry Pi via I2C (SDA and SCL pins), providing magnetic field measurements to determine the heading and orientation of the rotor
* Power: Raspberry Pi is powered via a USB-C connection, supplying 5.1V at 3.0A.
* Mount Control: Raspberry Pi is connected to the antenna rotator via the adafruit feather and controller circuit interface, enabling USB serial control of Yaesu antenna rotators.
* ADC Connectivity: Raspberry Pi is connected to the RSPduo via a USB B to USB A cable for data acquisition and processing.
* USB drive: a usb drive is connected to the Raspberry Pi to allow for faster data offloading, the software will detect if the usb drive is inserted at boot and will automatically start using it as a database if it is detected.

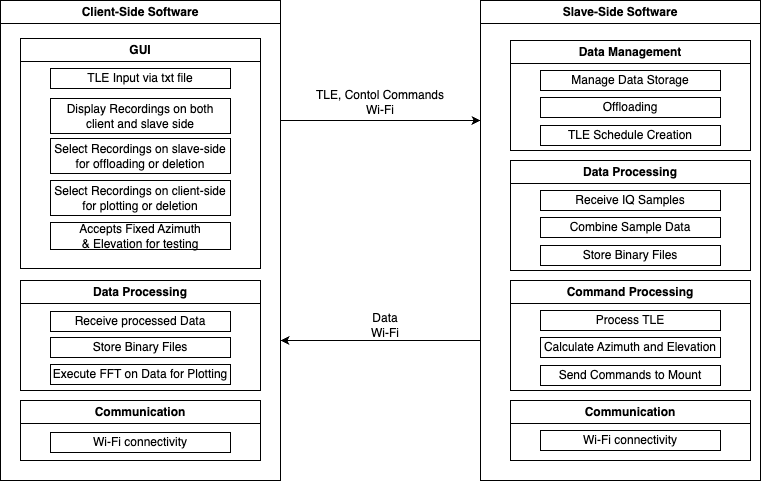
# 1.4 Computer Implementation

The computer system will run the operations software, which is implemented using Python and PyQt for the GUI. The Raspberry Pi's extensive online resources and ease of use with Python will facilitate the development and integration of the operations software. The GPIO pins on the Raspberry Pi will be used to interface with the GPS module as well as the magnetometer, while USB ports will handle connections to the RSPduo and the antenna rotator.

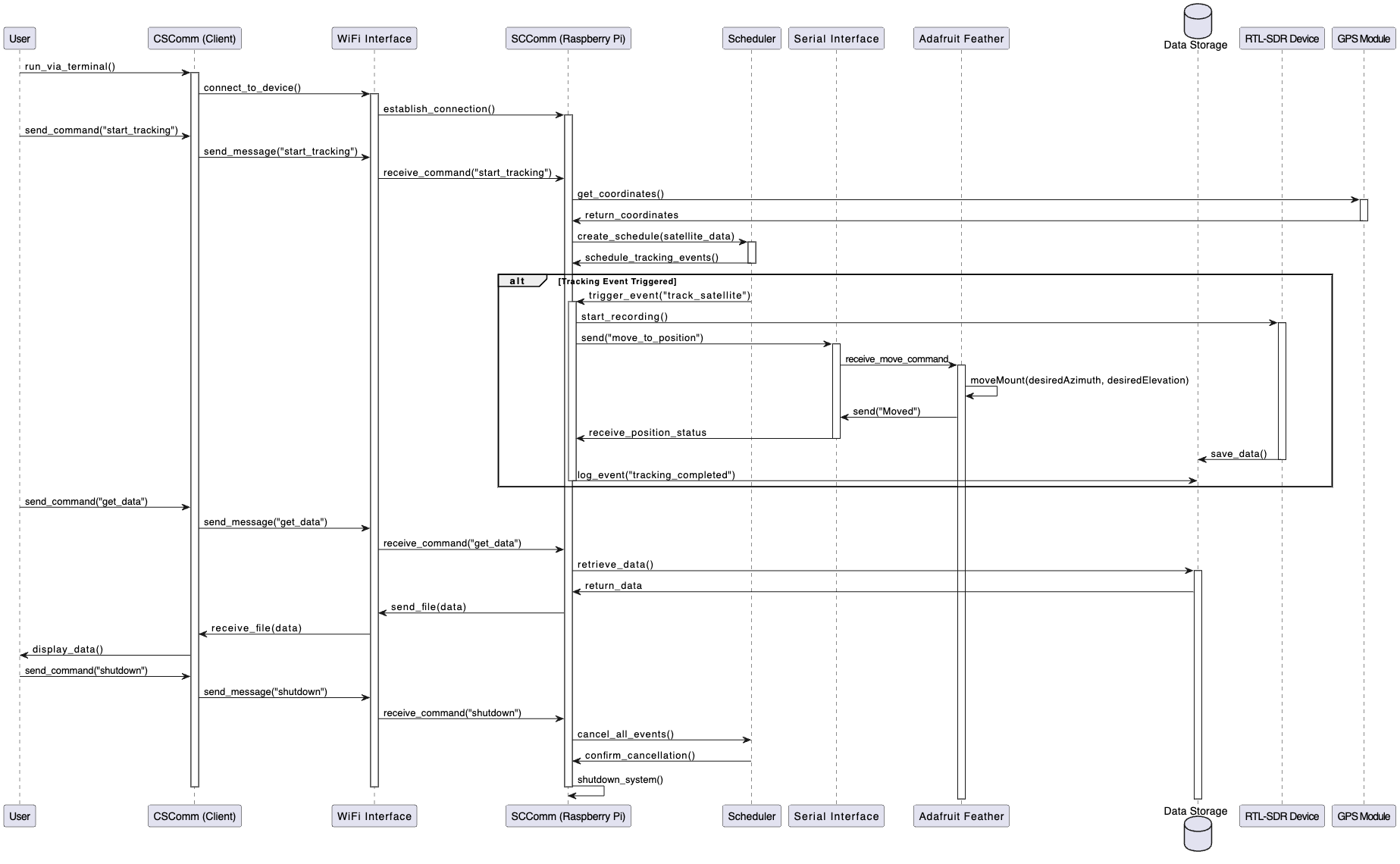
# 2.1 Operations Software Introduction

The operations software for the L-Band Satellite Tracking and Characterization System is responsible for providing an interface between the user and the system's hardware components. It allows the user to input commands, receive system status updates, and manage recorded data.

The software consists of several modules that communicate with each other to provide a seamless experience for the user. The software was entirely custom built with the use of some open source libraries.

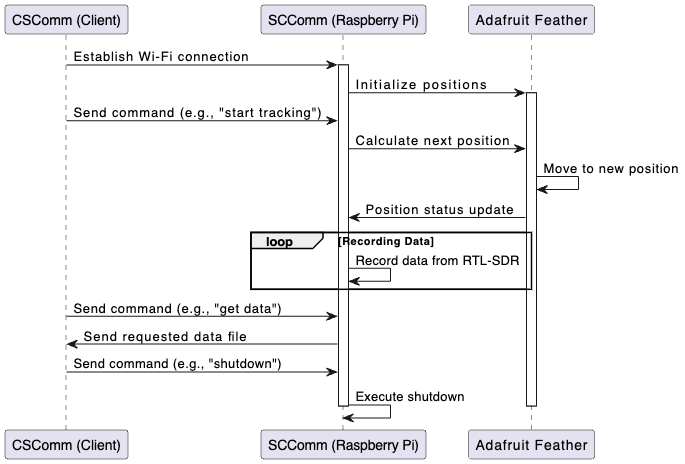


Software Architecture:

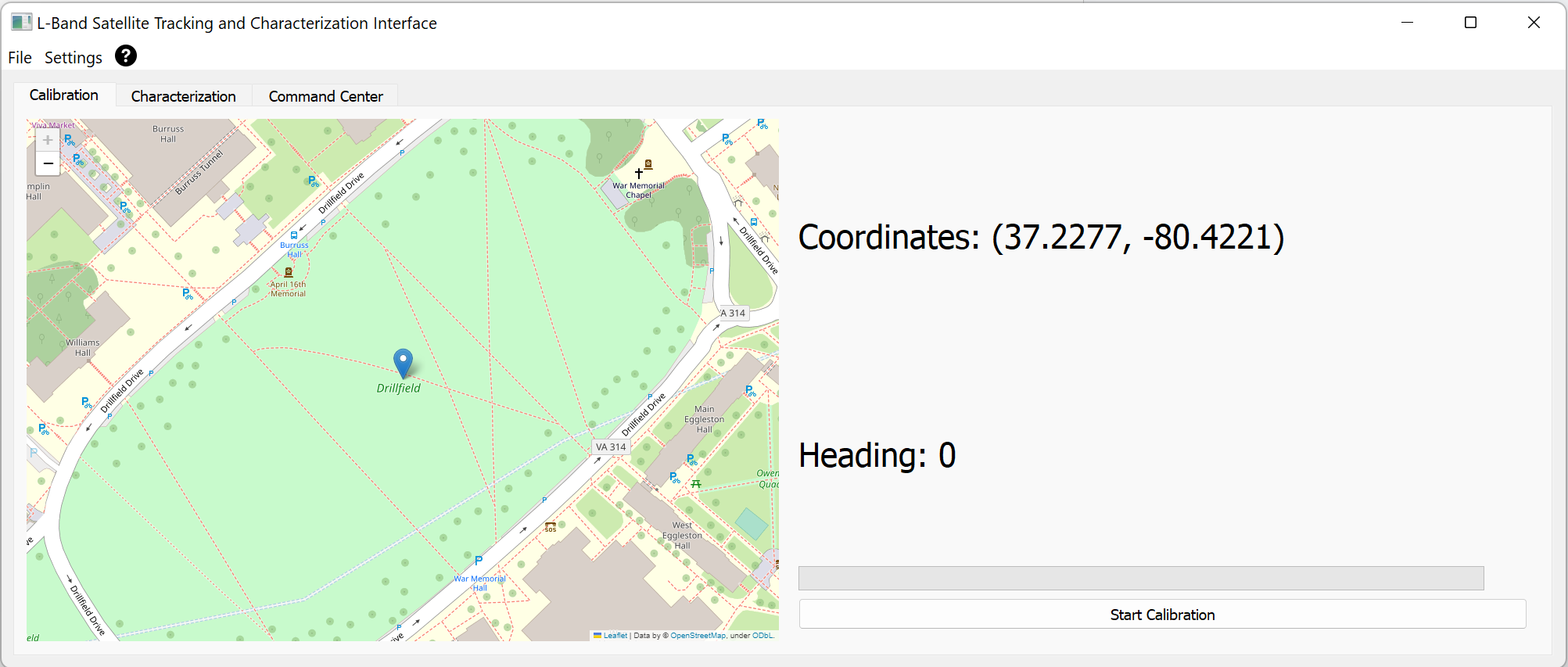


The software architecture consists of the following main components:

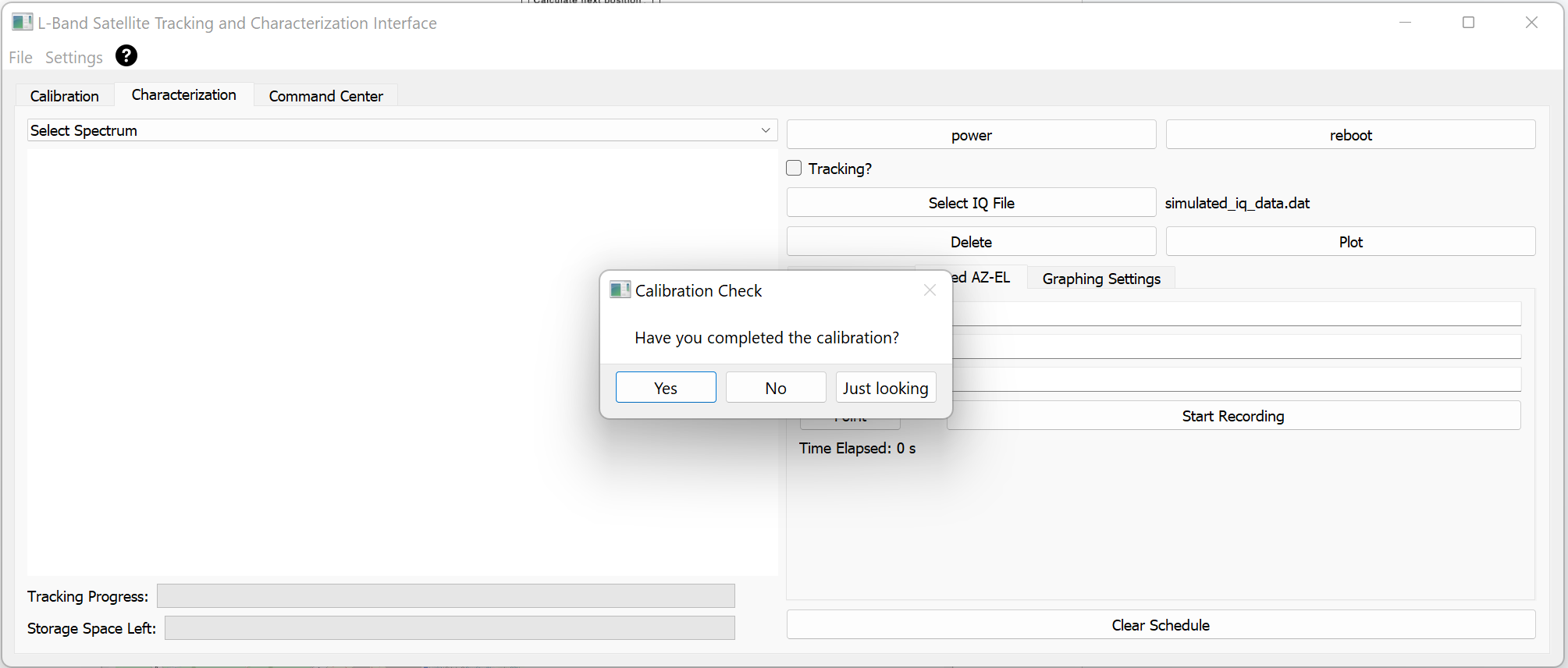
1. User Interface (GUI)
2. Client-Server Communication (CSComm)
3. Server-Client Communication (SCComm)
4. SatelliteTracker class (SatelliteTracker)
5. TLEParser
6. Scheduler
7. RSPDuo Interface
8. YaesuRotator/Adafruit Feather Interface



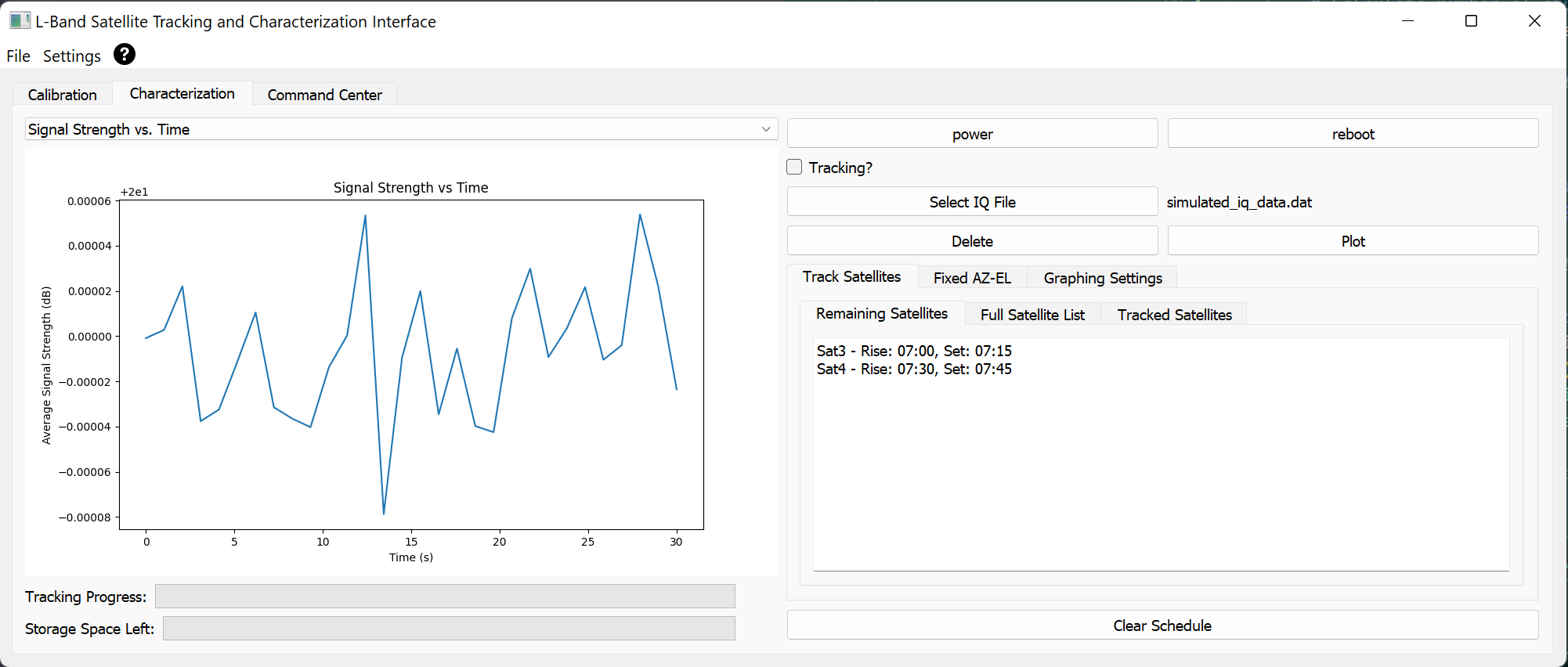
# 2.2.1 User Interface (GUI)



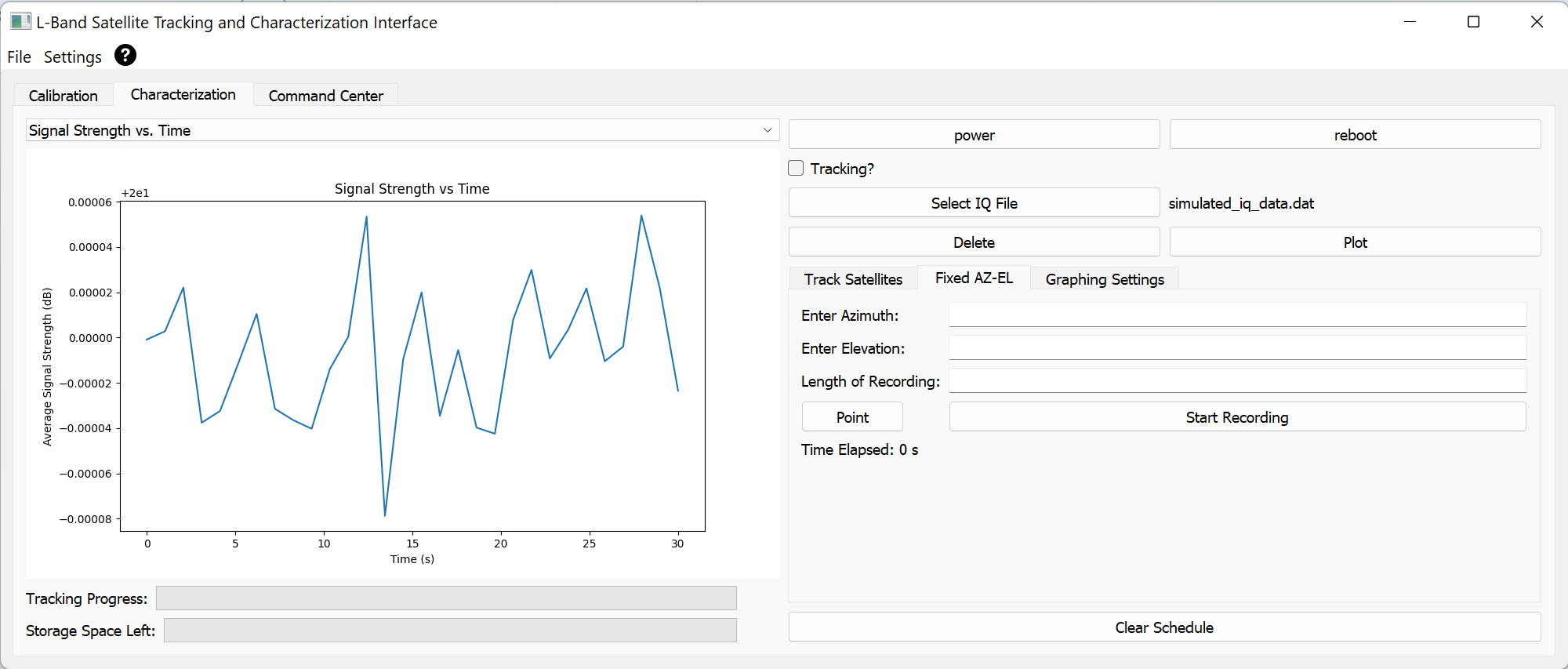
The landing page of the system is extremely simple, ensuring that there is no way that the user can progress further without making sure the system is properly configured and calibrated. Once calibration is complete the users will be able to move onto the next tab, where they can interact with the system. Attempting to move to the characterization tab prematurely will result in a popup, as shown below, ensuring that the user knows that the system has not been properly calibrated and will potentially lead to errors if they proceed.



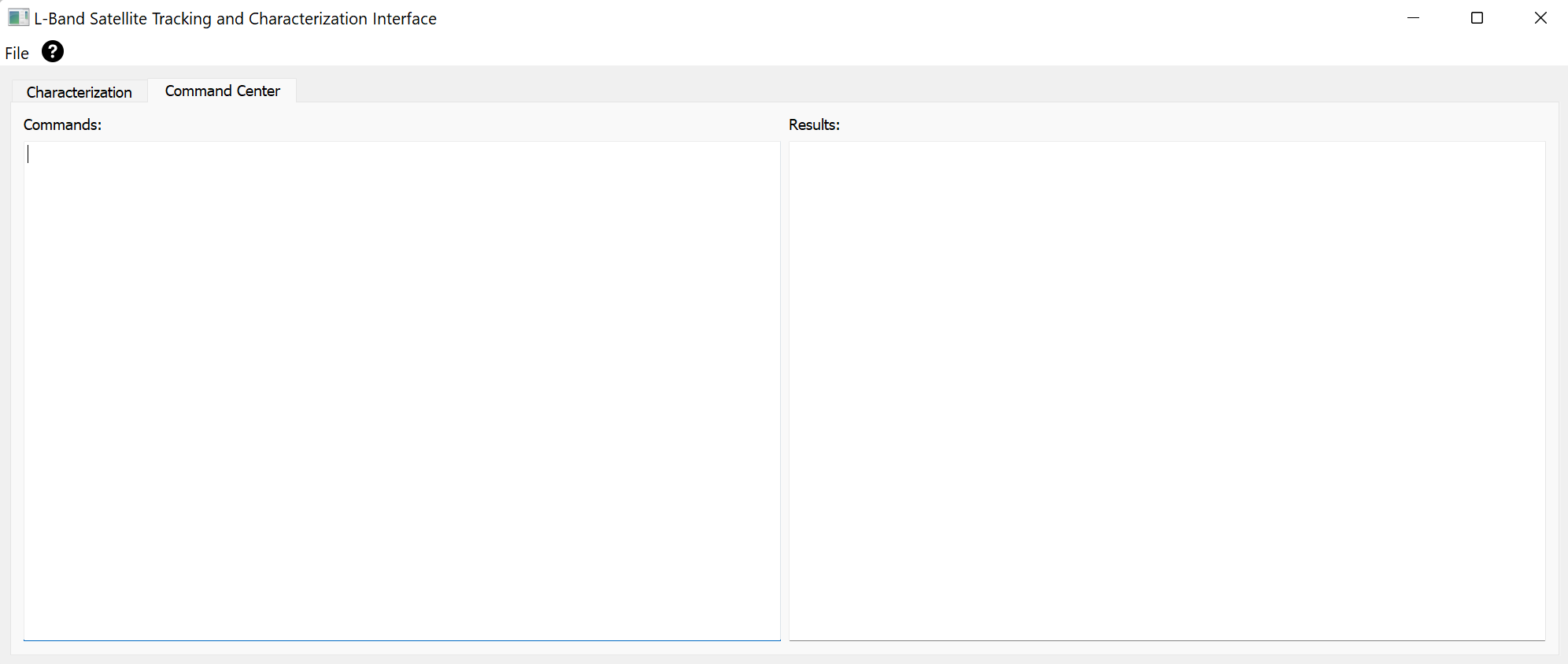
The GUI provides an intuitive interface for users to interact with the system. Users can input TLE data, send control commands, select recordings, input fixed azimuth and elevation for stationary recording, as well as the ability to shut down or reboot the system. We have implemented a way for the user to monitor the progress of the tracking and the amount of storage left, ensuring we have no issues with data loss, whether this be due to corruption or overflow errors. If the user is connected to the system when storage reaches around 80% there will be a warning popup, this will happen again until the issue is resolved.



We have the ability to keep track of which satellites have been tracked as well as which ones are still in the pipeline as shown in the lower left half of the application. This updates in real-time when connected to the system, updating every 2 seconds, unless there is a change, which will cause it to be updated instantaneously.



The GUI has multiple states, depending on what the user intends to do with the system. The images provided above show the two primary states that the user will interact with, autonomous tracking, as well as manual positioning. Another state allows the user to interface directly with the computer, sending commands as strings instead of using buttons and other interfaces within the GUI. This tab is mostly used for debugging and ensuring the system is working as desiredThis is shown below.



# 2.2.2 Client-Server Communication (CSComm)

CSComm is responsible for facilitating communication between the GUI and SCComm. It transmits TLE data, control commands, recording selections, and offload requests from the GUI to SCComm.

The algorithm consists of the following steps:

1. Initialize Wi-Fi connection:
   1. Search for available Wi-Fi networks.
   2. Display the list of networks to the user.
   3. Connect to the selected network (SCComm) using its credentials.
2. Listen for user input in the GUI or via commands in the terminal:
   1. If the user inputs TLE data, send the data to SCComm using a formatted command like "add\_to\_queue:<tle\_data>".
   2. If the user sends a control command, send the command to SCComm using a formatted command like "<control\_command> <arg1> <arg2>".
   3. If the user selects a recording, send the selection to SCComm using a formatted command like "get <recording\_id>".
   4. If the user inputs fixed azimuth and elevation, send the values to SCComm using a formatted command like "move <azimuth> <elevation>".
   5. "calibrate", "start\_tracking", "stop\_tracking", "setViewingWindow", and other predefined commands are sent as is.
3. Monitor the Wi-Fi connection:
   1. If the connection is lost or disconnected, display an error message and attempt to reconnect.
4. Terminate the connection:
   1. When the user shuts down the system, cancel any recordings, close the Wi-Fi socket and release all resources.

# 2.2.3 Server-Client Communication (SCComm)

SCComm receives data and commands from CSComm, processes the received data, and executes the commands. It communicates with SatelliteTracker to manage and process data.

The algorithm consists of the following steps:

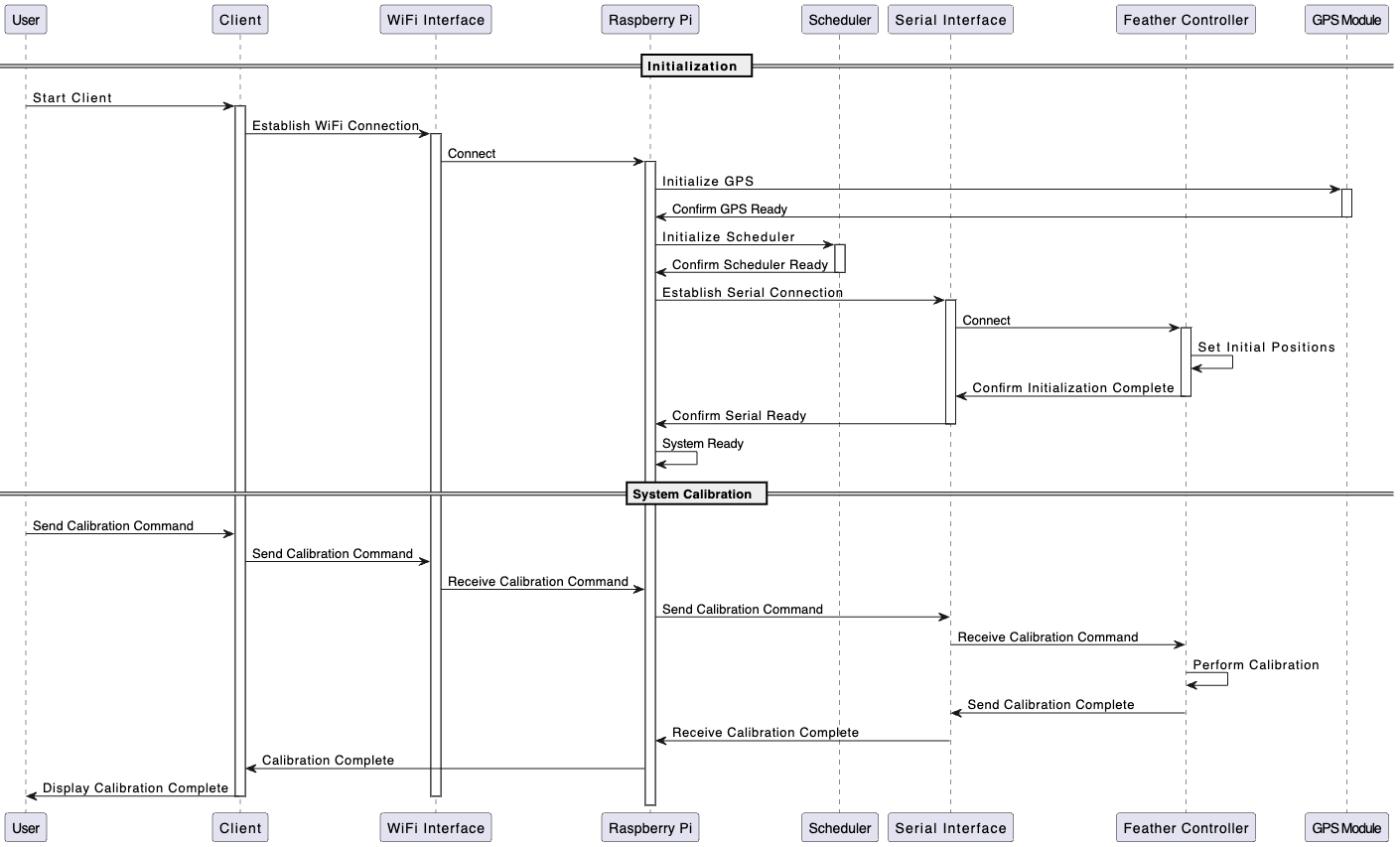
1. Initialize Wi-Fi server socket and await incoming connections.
2. Once connected, continuously listen for incoming data:
   1. "add\_to\_queue" commands are unpacked for TLE data and relayed to SatelliteTracker.
   2. Direct control commands like "move" are parsed and passed to the rotator controller for execution.
   3. Commands such as "calibrate", "start\_tracking", "stop\_tracking", and "setViewingWindow" are processed and executed accordingly.
   4. "get" commands are handled to retrieve or send files to the client.
3. Monitor the Wi-Fi connection:
   1. If the connection is lost or disconnected, display an error message and wait for a new connection.
4. Terminate the connection:
   1. When the system shuts down, close the Wi-Fi server socket, client socket, and release any resources.

# 2.2.4 SatelliteTracker

The SatelliteTracker is a comprehensive class within the operations software, central to the coordination of satellite tracking activities. It interfaces with hardware components and orchestrates the entire tracking process, from scheduling passes to actively positioning the antenna and recording data.

Initialization:

Upon initialization, the SatelliteTracker configures essential components such as GPS for geolocation and establishes a serial connection with the Adafruit Feather. The serial connection is used for commanding the Feather, which directly controls the antenna's motors based on the received instructions.



Core Responsibilities:

1. It computes the schedule for satellite passes and orchestrates the tracking sequence.
2. It transmits azimuth and elevation adjustments to the Feather, which maneuvers the antenna accordingly.
3. It conducts a calibration process to define the mechanical limits and ensure accurate antenna positioning.
4. It controls the recording of satellite signals, tagging the data with specific satellite information for retrieval and analysis.

The SatelliteTracker enables the seamless operation of satellite tracking and data acquisition within the system.

# 2.2.5 TLEParser

TLEParser parses the TLE data received from SatelliteTracker and returns the parsed data to SatelliteTracker. It is responsible for converting raw TLE data into a format that can be used by other components.

# 2.2.6 Scheduler

The Scheduler module in the L-Band Satellite Tracking and Characterization System is designed to arrange satellite tracking sessions by calculating when satellites will be visible from a specified observer's location. This is crucial for planning the recording of satellite signals.

Scheduling Algorithm

The scheduling algorithm consists of the following steps:

1. TLE Data Interpretation: It starts by interpreting TLE data to understand satellite orbits.
2. Satellite Position Calculation: Using the sgp4 algorithm, it computes the satellite's trajectory and position relative to the observer's location.
3. Time Window Establishment: It sets a time frame to monitor satellite passes.
4. Visibility Determination: The scheduler evaluates the satellite's position at regular intervals within the time frame to determine visibility above the horizon.
5. Pass Interval Documentation: It monitors the satellite's trajectory until it moves below the horizon, marking the end of a visible pass.
6. Schedule Formation: It compiles a list of visible passes, noting their start and end times for tracking.

Integration with the Data Management System

Within the data management system, upon receipt of TLE data, the Scheduler is invoked to calculate the visibility windows. This schedule is then utilized to instruct the hardware interfaces for recording and antenna positioning during the identified times.

Operational Functions

The Scheduler uses a series of functions to:

1. Calculate the local timezone based on geographical coordinates.
2. Determine all potential viewing windows for a given satellite within a specified time frame, applying the sgp4 model for accuracy.
3. Generate a non-repeating, non-overlapping schedule for multiple satellites to ensure efficient tracking without conflicts.
4. Create a sequential schedule that accommodates new satellite passes without disrupting the existing timeline.
5. Calculate accurate azimuth and elevation angles for the satellite tracking hardware to position the antenna correctly.
6. Translate raw TLE strings into satellite objects for further processing and tracking.

Incorporating the sgp4 algorithm ensures that the Scheduler can predict satellite positions with high precision, which is crucial for successful tracking and data acquisition during satellite passes.

# 2.2.7 RSPDuo Interface

The RSPDuo Interface is tasked with managing the interaction with the RSPDuo hardware for signal acquisition and recording. Its primary steps in the recording process are:

1. Device Setup: Initialize the RSPDuo device, configuring essential parameters like sample rate, center frequency, and gain.
2. Sample Acquisition: Begin streaming data and capture I/Q samples continuously for the duration of the satellite's visibility window.
3. Recording Management: On command from the SatelliteTracker’, commence recording by storing the I/Q samples to a file with a timestamp and satellite-specific information.
4. File Closure: Once the recording duration ends, close the device and save the data.

Interface Integration

This process is seamlessly integrated with the SatelliteTracker, which triggers the recording at the scheduled pass time of the satellite and manages the recording duration without the need for a stop command. The recording is based on a calculated interval derived from the satellite's rise and set times. The I/Q data samples are then stored in a uniquely named file that includes the satellite name, frequency, and a UTC timestamp, ensuring that each recording is easily identifiable.

# 2.2.8 Antenna Rotator Interface

The Antenna Rotator Interface is a critical component of the satellite tracking system, tasked with accurately orienting the antenna to maintain alignment with the satellites as they transit the sky. The interface operates by interfacing Python code running on a Raspberry Pi with an Arduino sketch uploaded to an Adafruit Feather microcontroller, which directly controls the antenna's motors.

1. Initialization: The process begins with initializing the Adafruit Feather, setting up serial communication from the Raspberry Pi. This setup includes configuring the microcontroller to respond to specific commands that will direct the antenna's movement.
2. Send command function: The system utilizes a Python function to send movement commands to the Feather. These commands are generated based on real-time calculations using the sgp4 algorithm, which takes into account the current time, the observer's geographical location, and the latest TLE data of the satellite. This function encodes the command as a string and sends it over the serial connection to the Feather.
3. Track satellite function: The Python code implements a tracking function that continuously calculates the current azimuth and elevation of the satellite using the sgp4 algorithm. The function then sends the calculated position to the Feather, which executes the movement. The Arduino sketch on the Feather reads the command and activates the motors, adjusting the antenna's position to match the calculated azimuth and elevation.
4. Arduino Implementation: On the Adafruit Feather, the Arduino sketch actively listens for movement instructions via the serial interface. Upon reception of a command, it deciphers the input and instigates a brief motor activation, moving the antenna incrementally towards the target azimuth and elevation. After each movement, the sketch reads the current azimuth and elevation from the connected potentiometers to ascertain the antenna's precise orientation. It then evaluates whether further adjustments are necessary to align with the commanded position.

The calibration routine incorporated into the sketch is designed to establish the full range of motion for the antenna. It achieves this by driving the antenna to its maximum azimuth and elevation extents. During this movement, the potentiometer values are monitored and recorded at the extremes. These values are then used to create a mapping that correlates the potentiometer readings to the corresponding azimuth and elevation angles, ensuring accurate real-time tracking and positioning of the antenna

1. Python and Arduino Integration: The integration of Python and Arduino code allows for a seamless operation where the complex computations and tracking logistics are handled by the Raspberry Pi's processing power, while the Adafruit Feather focuses on the real-time control of the hardware. The continuous loop in the Python code ensures that the antenna is consistently pointing at the satellite by frequently updating the position based on the sgp4 calculations.

Error handling and cleanup:

1. Implement error handling for exceptions that may occur during the execution of the tracking algorithm, such as serial communication errors or invalid TLE data.
2. In the case of an exception or interruption, exit the tracking loop and clean up any resources, such as closing the serial connection.

This system architecture allows for precise tracking capabilities. The antenna can constantly adjust its position to follow the satellite's movement across the sky, thanks to the accurate positional data derived from the sgp4 algorithm and the responsive motor control facilitated by the Adafruit Feather and the controller circuit.

# 2.2.9 Operations Software Implementation

The operations software is implemented using a combination of Python and Arduino programming languages. Python, with its comprehensive libraries for communication, data processing, and hardware interfacing, forms the core of the software, managing the system's operations and user interface. The GUI is developed using PyQt, a robust Python library that facilitates the creation of sophisticated desktop applications. Arduino is utilized for direct hardware interaction, particularly for controlling the movements of the antenna via the Adafruit Feather microcontroller

# 2.3.0 Required Libraries

* os (for operating system interaction, included in Python Standard Library)
* time (for handling time-related tasks, included in Python Standard Library)
* pickle (for serializing and deserializing Python object structures, included in Python Standard Library)
* datetime (for manipulating dates and times, included in Python Standard Library)
* socket (for network connections, included in Python Standard Library)
* subprocess (for running external processes, included in Python Standard Library)
* threading (for running tasks in different threads, included in Python Standard Library)
* queue (for queue implementation, included in Python Standard Library)
* pyserial (for serial communication, install via pip with pyserial)
* Adafruit-GPS (for GPS handling, install via pip with Adafruit-GPS)
* pytz (for timezone calculations, install via pip)
* timezonefinder (for finding timezones from coordinates, install via pip)
* skyfield (for TLE parsing and satellite position calculation, install via pip)
* SoapySDR (for interfacing with rsp duo devices, install via anaconda)

# 2.3.1 Executable Creation

Creating a standalone executable for the operations software allows for easy installation on Linux systems without requiring the client to have Python or other dependencies installed. To achieve this, the PyInstaller tool will be used to package the Python code and its dependencies into a single executable file.

The following steps outline the process of creating the executable:

1. Ensure that the operations software codebase is complete and functional.
2. Install PyInstaller if it is not already installed by running the following command:

pip install pyinstaller

1. Open a terminal or command prompt and navigate to the directory containing the operations software code.
2. Run the PyInstaller command with the appropriate options:

pyinstaller --onefile --windowed operations\_software.py

Replace `operations\_software.py` with the main Python script file name of the operations software.

* 1. - The `--onefile` option tells PyInstaller to create a single executable file instead of a collection of files.
  2. - The `--windowed` option specifies that the executable should run without showing a console window.

1. PyInstaller will analyze the code and its dependencies, and generate the executable in the `dist` directory within the operations software directory.
2. Test the generated executable on a Linux system to ensure its functionality and verify that all necessary dependencies are included.

Note that the size of the executable may be larger than the original Python script due to the inclusion of the Python interpreter and required libraries.

# 2.3.2 Raspberry Pi Linux Service for 5G Wi-Fi Hotspot and automatic python script initialization on boot

Ensure you use the provided Raspberry Pi image with SDRplay driver preloaded for the following steps.

Preparing the Raspberry Pi Environment

1. Clone the project repository.

cd Desktop && git clone https://github.com/samueltv250/MDE.git

2. Install required Python packages.

pip install skyfield && pip install timezonefinder && pip install serial && pip install adafruit-circuitpython-gps

Configuring Raspberry Pi as a Hotspot with Static IP Address

Update and Upgrade DietPi

sudo apt-get update && sudo apt-get upgrade -y

Install Hostapd and Dnsmasq

sudo apt-get install hostapd dnsmasq -y

Stop Services to Configure Them

sudo systemctl stop hostapd && sudo systemctl stop dnsmasq

Configure Hostapd

1. Edit the configuration file.

sudo nano /etc/hostapd/hostapd.conf

Add the following lines to the file:

interface=wlan0

driver=nl80211

ssid=SatelliteTrackingSystem

hw\_mode=g

channel=7

wmm\_enabled=0

auth\_algs=1

wpa=2

wpa\_passphrase=12345678

wpa\_key\_mgmt=WPA-PSK

wpa\_pairwise=TKIP

rsn\_pairwise=CCMP

2. Point Hostapd to the Configuration File.

sudo nano /etc/default/hostapd

Replace the #DAEMON\_CONF line with:

DAEMON\_CONF=\"/etc/hostapd/hostapd.conf\"

Configure Dnsmasq

1. Rename the original configuration file.

sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig

2. Create a new configuration file.

sudo nano /etc/dnsmasq.conf

Add the following lines to the file:

interface=wlan0

dhcp-range=192.168.220.10,192.168.220.50,255.255.255.0,24h

Set Static IP Address

1. Create a systemd service file.

sudo nano /etc/systemd/system/static-ip-wlan0.service

Add the following lines to the file:

[Unit]

Description=Set static IP address for wlan0

Wants=network.target

Before=network.target

[Service]

Type=oneshot

ExecStart=/sbin/ip addr add 192.168.220.1/24 dev wlan0

ExecStart=/sbin/ip link set wlan0 up

RemainAfterExit=yes

[Install]

WantedBy=multi-user.target

2. Enable and start the service.

sudo systemctl daemon-reload && sudo systemctl enable static-ip-wlan0 && sudo systemctl start static-ip-wlan0

Start Hostapd and Dnsmasq

sudo systemctl unmask hostapd && sudo systemctl enable hostapd && sudo systemctl start hostapd && sudo systemctl enable dnsmasq && sudo systemctl start dnsmasq

Reboot the Raspberry Pi

sudo reboot

Improve Wireless Configuration for faster offload speeds

1. Call dietpi-config.

2. Enable 802.11n/ac/ax.

3. Change Frequency to 5 GHz.

Enabling slave.py at Boot

1. Create a New Service File.

sudo nano /etc/systemd/system/slave.service

Add the following lines to the file:

[Unit]

Description=Python Script Service

After=network.target

[Service]

Type=simple

User=dietpi

WorkingDirectory=/home/dietpi/Desktop/MDE

ExecStart=/usr/bin/python3 /home/dietpi/Desktop/MDE/slave.py

Restart=on-failure

[Install]

WantedBy=multi-user.target

2. Reload the System Daemon.

sudo systemctl daemon-reload

3. Enable the Service.

sudo systemctl enable slave.service

4. Start the Service.

sudo systemctl start slave.service

5. Check the Service Status.

sudo systemctl status slave.service

# 2.3.3 Mounting a usb drive to use as a database for the backend software

1. Find the UUID of the USB Drive:

- Run the `blkid` command to find the UUID of the USB drive:

sudo blkid

- Look for the line that corresponds to your USB drive (`/dev/sda1` in this case) and note the `UUID`.

2. Create a Mount Point:

- Choose a location for the mount point. this should be set to `/mnt/usbdrive`, in order to properly work with the backend software:

sudo mkdir /mnt/usbdrive

3. Edit the `/etc/fstab` File:

- Open the `/etc/fstab` file in a text editor with root privileges:

sudo nano /etc/fstab

- Add the following line at the end of the file:

UUID=your-uuid /mnt/usbdrive exfat uid=1000,gid=1000 0 2

Replace `your-uuid` with the UUID you noted earlier.

If the USB drive is not `exfat`, replace `exfat` with the actual filesystem typ.

4. Mount the Drive:

- Now lets mount all drives with:

sudo mount -a

- This will apply all the new configurations set in the `/etc/fstab` file.

# 3.1 Mount Subsystem

The mount subsystem encompasses the majority of physical hardware of the system, and is primarily tasked with mounting the antenna and receiver systems, stability of the system, azimuth and elevation rotation and environmental shielding. There are four separate subcomponents of the mount, those being the rotator system for azimuth and elevation control, the tripod for stabilizing and mounting purposes, the mount control box used to interface with the computer system, and the enclosure.

# 3.2.1 Mount System Requirements

The components chosen for the mount subsystem were picked primarily in accordance with the needs of the other subsystems, as the majority of the requirements for the mount were determined by the constraints of the other subsystems. The initial requirements for the mount subsystem are shown below:

* The system shall track a satellite until the signal from it disappears.
* The system shall observe a point of space given an azimuth and elevation measurement.

This is the primary subsystem requirement, as this is the main requirement for the rotator component, which is what the rest of the subsystem is designed around.

* The system shall operate within an ambient air temperature band of 14°F to 100°F (-10°C to 40°C).
* The system shall operate within the regular weather climate of Blacksburg (ex: light rain, snow).
* The system shall withstand lightning level current.

These requirements are the environmental concerns associated with the mount, specifically the tripod and controller portions of the mount.

* The system shall be transportable a minimum of 100 yards by two average adults.
* The system shall be shut down, disassembled, and packed up in less than ten minutes.
* The system shall be assemblable in less than five minutes.
* The system shall fit into two medium sized hard shelled containers

These requirements are the size concerns associated with the mount, to ensure that the mount design is sized appropriately to the desired task.

# 3.2.2 Mount System Derived Requirements

Using the above design considerations as a basis, the derived requirements for this project are as follows:

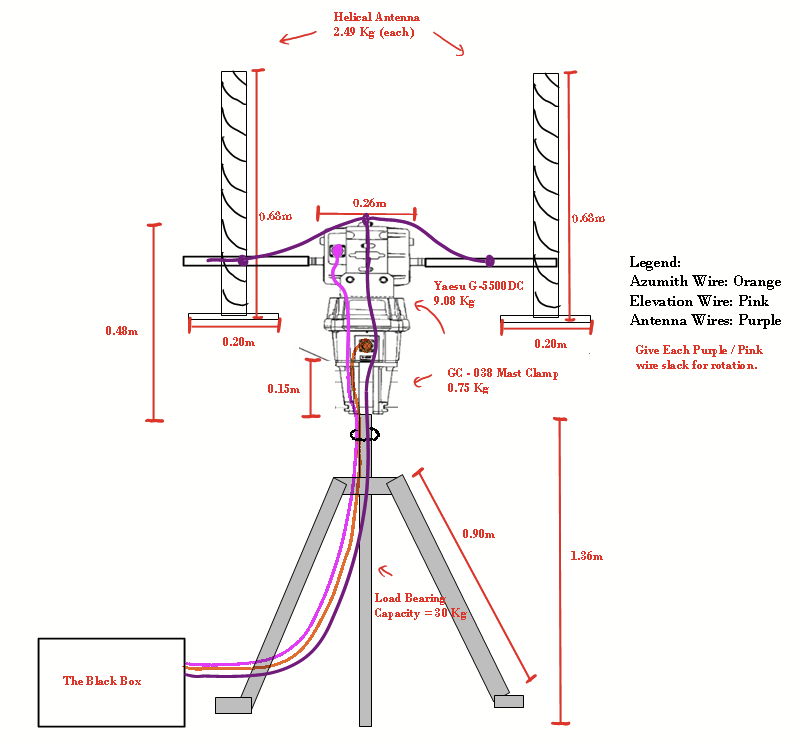
The mount shall:

* Meet the rotational torque requirement to support our antenna at an Azimuth angle of 0° - 360° and Elevation angle of 0° - 90°.
* Be weather resistant to snow and light rain, as well as being capable of functioning in these environments for the allotted time.
* Have the rotational speed and continuous operational time necessary to track satellites during their orbit.

In addition, the mount should:

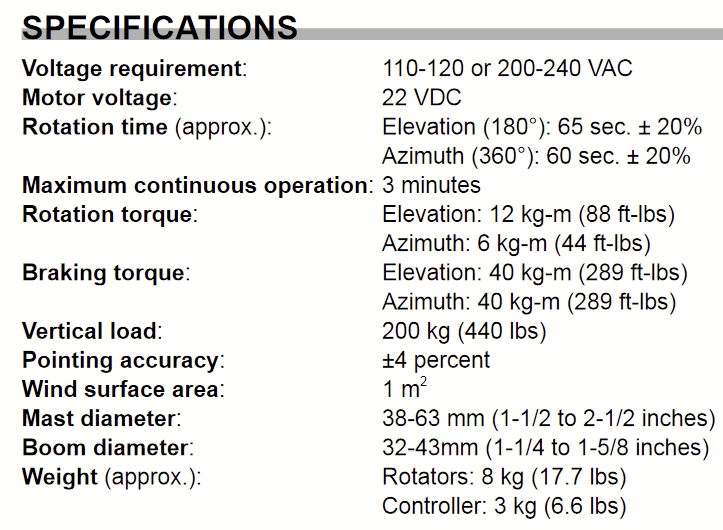
* Be USB capable, and able to communicate with the onboard computer over a USB connection.
* Be assemblable in less than three minutes.
* Remain under a total weight of 30 Kg and cost less than $1100 for the rotator and all required additional systems.

# 3.3 Mount Schematic



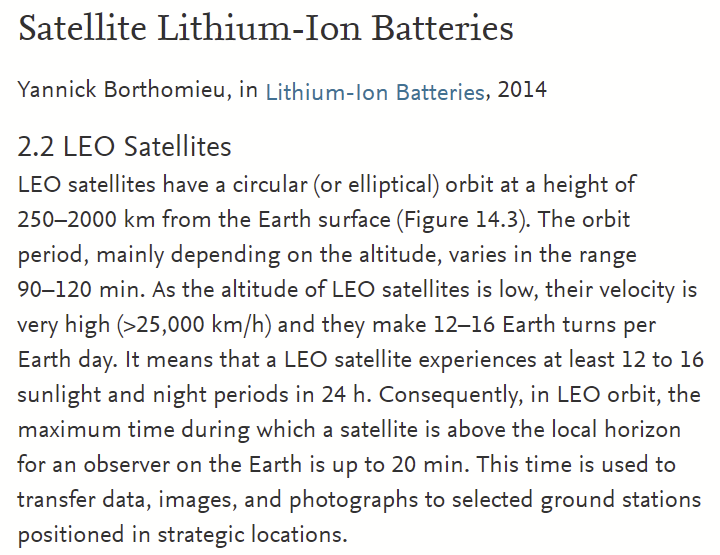
# 3.4 Rotator Design

The rotator system is the main component of the mount, and is tasked with installing the antenna system and pointing it towards incoming satellites. This task requires that the mount is capable of supporting the chosen antenna at all required angles, as well as having the slew rate necessary to adjust the azimuth and elevation of the antenna at a high enough adjustment speed to track satellites. The rotator chosen for this is the Yaesu G-5500 DC, a dual azimuth / elevation rotator designed to support portable satellite communications installations and currently being sold for $759.99 USD. The device specifications for the Yaesu G-5500 DC are shown below:



The primary concern for the rotator is whether or not it is able to be mounted by the antenna for our project while still being able to track a satellite, which was the main driving force behind the choice of mount. The antenna system, due to it mainly being constructed out of PVC and copper wire, is estimated to weigh around 5.5 kg, which is able to be handled by our rotator system as the maximum rotation torque is 6 kg-m. The rotator is also capable of moving in a full azimuth angle of 360 degrees and an elevation angle of 180 degrees, giving the rotator the full capability of tracking satellites in their movement throughout the horizon. The two rotator components are also encased inside of a melamine coated aluminum casing to keep them weather resistant, although the controller components of the rotator are not weather resistant. As such, when designing the enclosure for the system, the components of the rotator that are not weather resistant (those being the rotator controller and the computer interface) are stored inside of a weatherproof container.

The full rotation time (for both azimuth and elevation) is measured to be 75 seconds for both planes, which meets our requirements as the requirement for this system is to be able to have the slew rate necessary to track the fastest data satellites (LEO satellites). As stated below, the visible time for an LEO satellite is 20 minutes for data transfer purposes, meaning that we should have more than enough time with the estimated 60s rotation time to make the necessary adjustments to track the satellite.



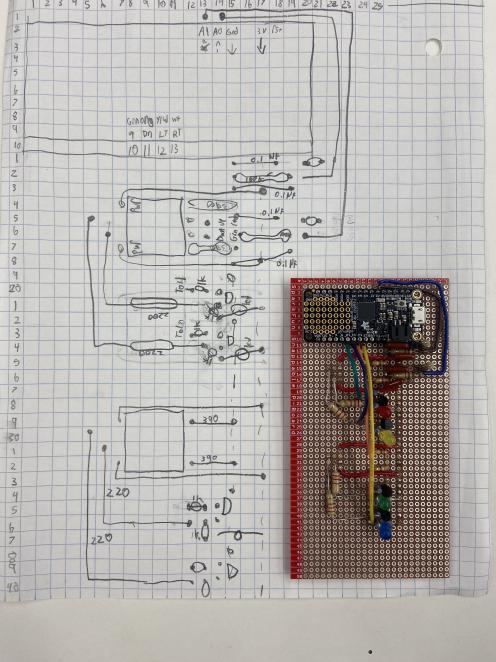
<https://www.sciencedirect.com/topics/engineering/low-earth-orbit>

The Yaesu G-5500 DC is delivered including the required mount controller unit and mast clamp, meaning that the necessary components for initial setup of the mount are provided within the mount package. The issues associated with this mount choice is that the provided mount controller component does not initially have USB capability, and in addition the rotator system needs a platform upon which to mount itself and additional wires to connect to the battery and computer. The additional components of the mount are aimed to solve these issues.

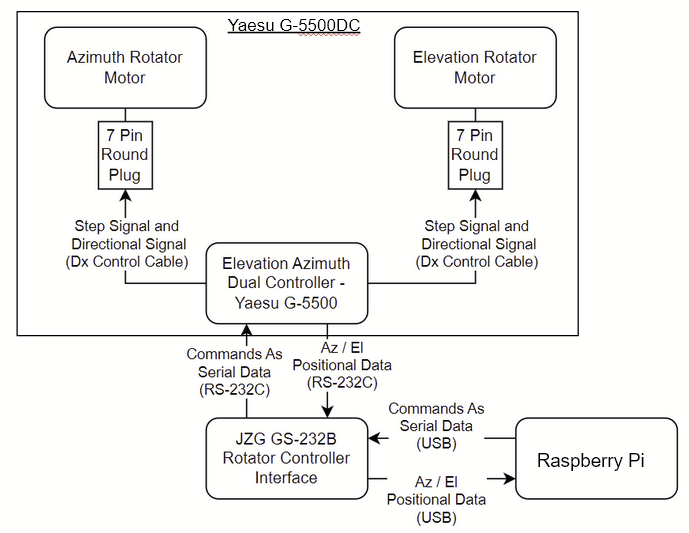
# 3.5 Additional Mount Components

3.5.1 The Computer Interface

The component that allows the rotator control box to interface with the onboard computer is a custom-designed interface. The original design for this component was to use a component called the Yaesu GS-232B interface, but due to cost requirements this component was unviable. Another potential component to use was determined to be the JZG GS-232B interface that they had developed for Yaesu rotators, but this component was prone to issues and thus the idea was chosen to design a custom interface. The circuit developed for this project is made with a set of common-emitter subcircuits designed with a set of ILD2 optically coupled phototransistors, to provide optical isolation. The interface works by having a series of azimuth and elevation commands sent by the onboard computer to an Adafruit Feather M0 microcontroller, which then transmits these commands in the form of active-low voltages. These voltages are then transmitted across pins 10-13 of the Feather M0 to the rotator control box, which then uses these values to determine which direction to move. The diodes are also introduced in the form of LEDs as a checksum to ensure that the circuit is working as intended, and were used throughout the development of this board as a hardware debugger. A picture of this circuit is shown below:



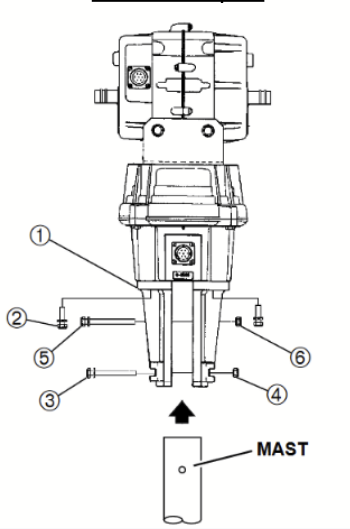
The software aspects of the interface between the mount and the computer are shown above, and the above picture depicts the wiring diagram and rough structure of the setup of the mount.



3.5.2 The Tripod

The last necessary component of the mount subsystem is the tripod component, which is used to stabilize the system and provide a base on which the rotator and antenna can be mounted. The chosen tripod for this project is the WEVZENEY Antenna Tripod, currently being sold for $59.99. The load-bearing capacity of the antenna is stated to be 30 kg, and as can be seen from the diagram above the antenna components in addition to the rotator will weigh roughly 15.5 - 16 kg. The torque requirement for the tripod, based on the torque requirement of the mount subsystem and the weight of the antenna, is roughly 25 kg, which approaches the maximum load of the tripod. As such, the tripod will have to be staked into the ground, which it is initially designed to do because of the added stakes in the tripod. Doing this will increase the load bearing capacity of the tripod, and return the tripod to normal operation levels.

The tripod is stated to be weather resistant and rust-proof due to the iron varnish on the device, and the reviews from previous users support these claims. The tripod mounting diagram is shown below:



# 3.6 Enclosure

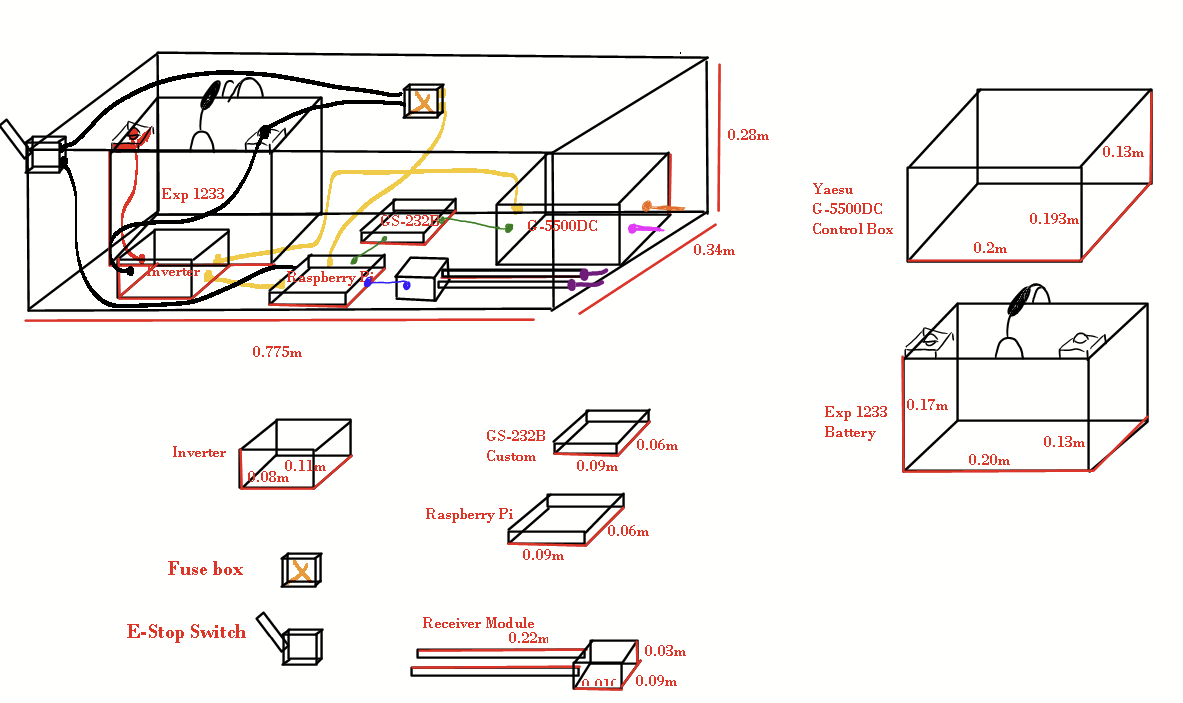
The main enclosure for the system will house the computer, battery, filters, receiver, and the mount controller. This is to ensure that all of these subsystems remain in a dry environment. To this end, the main enclosure will have to be fairly large. To ensure waterproofing of the design, the holes with wires coming out of them are covered in a waterproof silicon.

The magnetometer will also require a separate enclosure though much smaller, as it has to be mounted to the top of the Yaesu G-5500 DC, to properly calculate the heading of the system, ensuring that we are pointing north during setup.

# 3.6.1 Enclosure Size

The size of the main enclosure will have to house the battery, which is 7.72”x 5.16”x 6.4”, all of the filters in a solid line, which totals 9” when connected to the LNA, which also has to connect to the receiver. Together, this totals a length of nearly a foot. The main enclosure will also have to house the controller for the mount, which is 7.87”x7.48”x5.11”. This, together with the converters and the GPS module that goes with the computer will take up most of the ground room in the storage container. The GPS module will have to be placed on top of the main enclosure, but due to the module and the associated wires having an IPX4 weather resistance rating in terms of snow and rain, this component can be exposed to the environment without concerns of weather-associated damages.

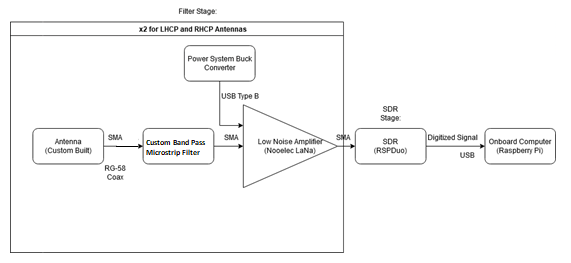
3.6.2 Main Enclosure Layout



The general layout of the main enclosure is shown above. The only cable exit will be the cables going to the mount, and the two antenna cables. The emergency switch will be built into one of the sides. any holes that are drilled into the main enclosure will be sealed with silicone to keep the box watertight. The fuse box will be centrally located in order to reduce hassle when replacing fuses.

# 4.1 Receiver Subsystem

The receiver subsystem consists of two stages, the filtering stage, and the SDR stage.



| Frequency (MHz) | RspDuo Noise Figure (dB) | External LNA Noise Figure (dB) | External LNA Gain (dB) | Cable Loss ~10ft (dB) | Reference Noise Temperature (K) | Antenna Noise Temperature (Estimate) (K) | System Noise Temperature (K) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1400 | 4.66 | 1.2 | 26 | 2.06 | 300 | 35 | 129.989 |
| 1500 | 4.95 | 1.24 | 26 | 2.159 | 300 | 35 | 133.847 |
| 1600 | 5.27 | 1.3 | 26 | 2.256 | 300 | 35 | 139.592 |
| 1700 | 7.47 | 1.4 | 26 | 2.352 | 300 | 35 | 151.575 |
| Average | 5.5875 | 1.285 | 26 | 2.20675 | 300 | 35 | 138.75075 |

The receiver subsystem design is constrained by five system requirements:

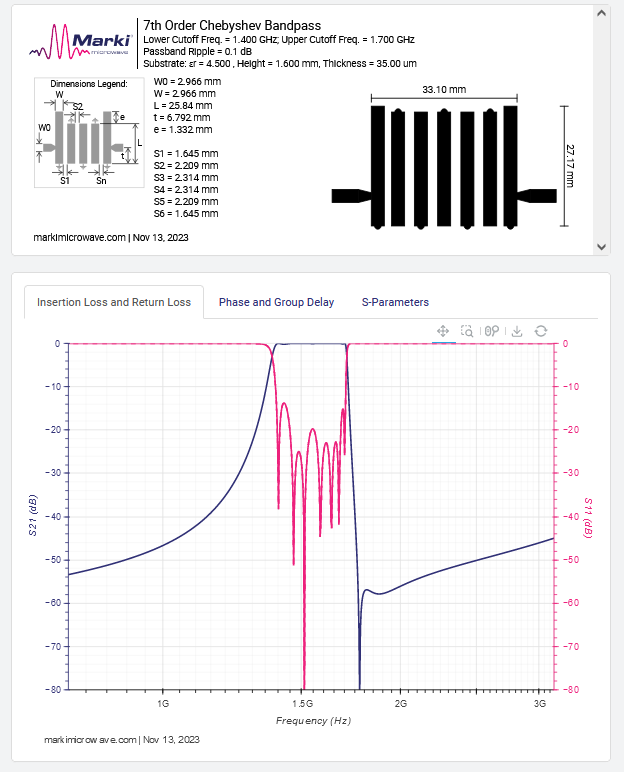
* FUN-2: The system shall have a receiver noise temperature less than 200K.
* FUN-4: The system shall have a receiver whose bandwidth is variable up to 2MHz minimum, with up to 10MHz desired.
* FUN-5: The system shall have >40 dB rejection of 0.5-1350 MHz and 1800-2700 MHz signals.
* PER-2: The system's receiver shall have a tunable range between 1550-1700MHz. 1400-1700MHz desired.
* COST-1: The system shall be replicable for < $2000 USD

FUN-2 is relevant to both stages of the receiver, while FUN-4 and PER-2 are relevant to the SDR stage, and FUN-5 is relevant to the filtering stage.

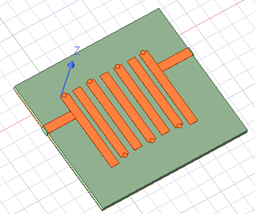
# 4.1.1 Filtering Stage

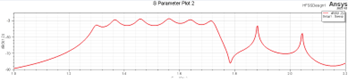
The preselection filtering is taken care of by a custom built interdigital microstrip filter.

Calculated Filter Parameters:

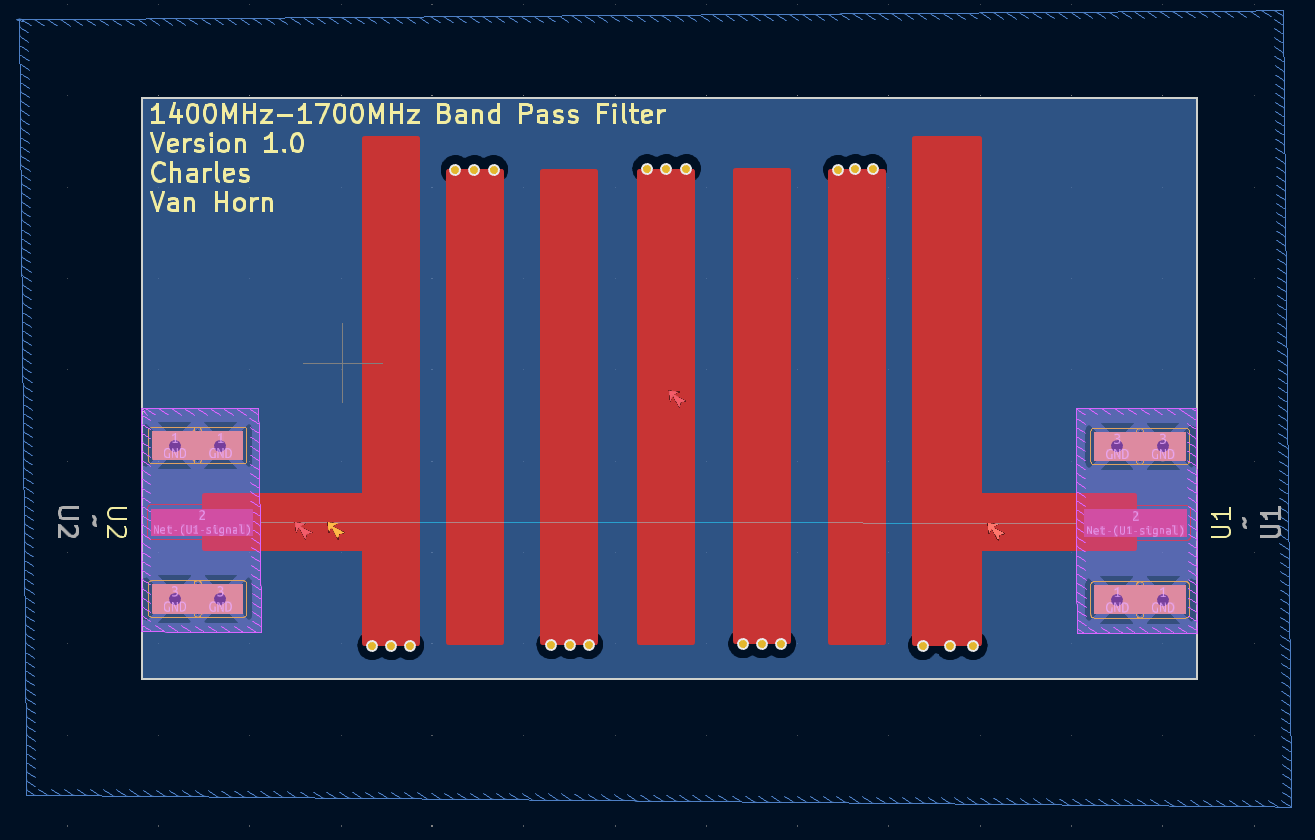


Filter built in HFSS:





Filter layout in Kicad:



The filter’s dimensions were calculated using the MarkiMicrowave microstrip filter design tool. Then simulated using HFSS to confirm its effectiveness. Layout was done using KiCad and sent to be fabricated by JLCPCB.

At the end of the filtering stage is a low noise amplifier (LNA). The LNA chosen is the NooElec LANA. This LNA enables us to increase the gain of our desired signals within the passband, as well as helps us to decrease our overall noise temperature of the system, allowing us to meet FUN-2.

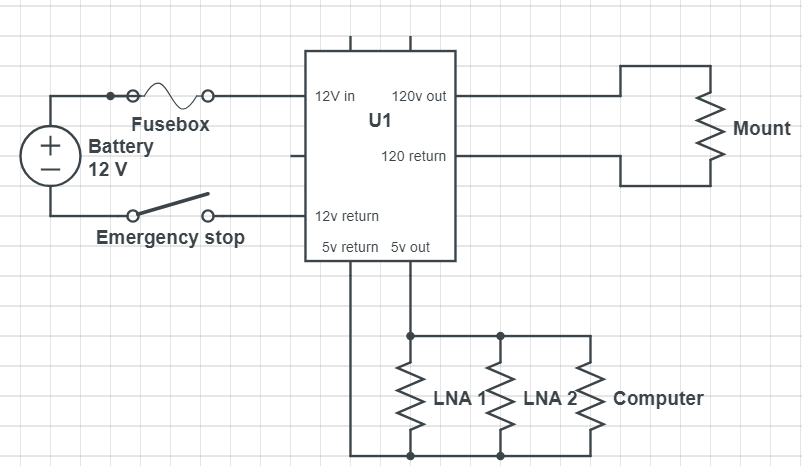
All of these connections between filters, the LNA, and the SDR are made using SMA-SMA connections.

# 4.1.2 SDR Stage

The SDR used is the SDRPlay RSP-Duo. The RSP-Duo is tunable to up to 2GHz, and has a tunable bandwidth of up to 2MHz when both inputs are used, and up to 10MHz when one input is enabled at a time.

# 5.1 Power System Introduction

The power system is responsible for safely supplying a steady supply of power to all components of the system for a continuous period of 18 hours, without significant voltage drop, while ensuring that no components are harmed by current overdraw. The generalized circuit for the system is shown below. This includes all fuses and switches.



# 5.2 Power system specifications

To facilitate the power requirements of the systems the battery will have to power, a power budget was calculated using the current and voltage requirements of each system. The time used for each system varies, and each is shown below.

* The mount will have to be powered for a total of 1.5 hours.
* The computer will have to be powered for 16 hours at low power/idling.
* The computer will have to be powered at maximum current draw for 2 hours to facilitate the recording of signals and the processing of data.

The calculated power budget is shown below.

| System | Voltage required | Amperage total | Time (hours) | Watt Hours | Amp Hours |
| --- | --- | --- | --- | --- | --- |
| Computer | 5V | 3A | 2 | 30Wh | 2.5Ah |
| Computer(low power) | 5V | 0.514A | 16 | 41.12Wh | 3.51Ah |
| Mount | 24V | 4.5 | 1.5 | 162Wh | 13.5 |
| Total | N/A | N/A | N/A | 233.12Wh | 19.51 |

# 5.2.1 Battery Voltage

The chosen battery is a deep cycle lead acid battery, which has a capacity of 33 AH at 12V. A deep cycle battery was chosen over a standard car battery because of the superior discharge over time characteristics. Whereas a car battery would lose voltage very quickly over a smaller period of time, a deep cycle battery is more suited for steady current draw over a larger time window.

# 

# 5.2.2 DC-DC Converters for Voltage Regulation

Since the various components of the other systems do not require 12 volts, this voltage will have to be stepped up or down in order to facilitate their voltage requirements, This will be done using an off the shelf inverter. This is a pure sin wave inverter that can turn a 12v DC input into a 120v AC output as a pure sin wave. This was important since we are working with sensitive electronics, and a non pure sinusoidal waveform could damage these or at the very least cause them to malfunction.

# 5.2.2.1 Computer power

The computer’s power will be supplied by the battery through the same inverter, as it also contains a 5v DC port, as a usb output. This will be used to power both the main raspberry pi, as well as the two low noise amplifiers. All of these devices will be fed by the two usb output ports, using a splitter in the case of the LNAs.

# 5.2.2.2 Mount Power

The mount will require a 120v AC power supply, similar to a wall plug. This is provided by one of the two wall outlets provided by the inverter. This is to simplify the production of future systems, as a simple off the shelf component is much easier to work with than several homemade components wired together.

# 5.3 Safety

There will be two main safety features on the power system.

* An emergency stop switch
* A fuse box

These two safety features will keep the system from damage in case of current overdraw, and will provide a quick and easy way to switch off the power in case of an emergency should the need arise. The emergency stop switch will also provide a way to cut the system off completely.

# 5.3.1 Fuses

There will be two fuses in the system. One will be placed directly before the computer, but after the converter. This is because the inverter can supply more current than the computer can take, and so that should an overcurrent occur, the computer will not be damaged before the fuse breaks the circuit. The other fuse will be in a similar location, before the direct path to the mount but after the DC to DC converter. This should keep the mount safe from a dangerous overcurrent.

# 5.3.2 Emergency Cutoff Switch

The emergency cutoff switch is a manually operated single pole single throw switch connected to the negative terminal of the battery. It is intended to provide a manual way of safely cutting off all power to the system. It does this by eliminating the path to ground from the entire circuit.

# 5.4 Wire Gauge

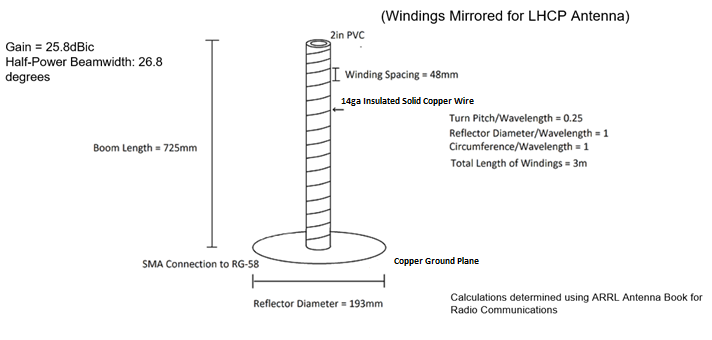
The wire gauge required for the various parts varies. To go from the positive terminal of the battery to the to DC/DC converters, #10 AWG is required. This is due to the current limitations imposed on wire gauges.For this project, a single core #10 AWG is used, which is rated for 52 amps. Another reason for choosing this wire is because the thicker the wire, the more temperature resistant it is. Since this system could easily be operating in cold weather as well as hot, the wire must be temperature resistant.

# 6.1 Antenna Subsystem

Relevant Requirements:

* FUN-3: The system shall have an antenna that is RHCP polarized. LHCP Polarization is desired but not required.
* PER-2: The system's receiver shall have a tunable range between 1550-1700MHz. 1400-1700MHz desired.

The design of the antenna is relatively simple. Calculations are made based on the ARRL Antenna Book for Radio Communications. The dimensions are calculated with a center frequency of 1550MHz. Axial mode helical antennas have a fractional bandwidth of 56% of the center frequency. Therefore the antenna will have an operable bandwidth of 1,116MHz to 1,984MHz. This meets the PER-2 requirement. There will be two antennas. One wound in the right hand direction in one wound in the left, which achieves RHCP and LHCP polarization.



# 

# 7 Acceptance Test Plan

Material to be delivered to customer NLT 2 weeks prior to date of in-person demo:

[ ] Electronic copy of the user manual (deliverable) explaining how to set up the user device, including software access if needed.

Material to be delivered to customer NLT 48 hours prior to the date of in-person demo:

[ ] Video (deliverable) documenting a successful acquisition and tracking of an Iridium satellite.

[ ] Electronic copy of the construction manual (deliverable) detailing assembly instructions, component list, and necessary tools.

[ ] Report documenting measurement of antenna beamwidth, ensuring it meets the requirement of having a bandwidth variable up to 2MHz minimum, with up to 10MHz desired (FUN-4).

[ ] Report documenting measurement of receiver frequency response, demonstrating preselection requirement of having >40 dB rejection of 0.5-1350 MHz and 1800-2700 MHz signals (FUN-5) is met.

[ ] Report documenting measurement of receiver temperature, demonstrating that the requirement of having a receiver noise temperature less than 200K (FUN-2) is met.

[ ] Complete, documented, and user-accessible GitHub repo (deliverable) containing all source code, schematics, and documentation.

In-person demo: Conducted indoors, location Whittemore landing/balcony. Customer brings a laptop running Linux Ubuntu 22.04, configured according to the user manual provided.

Demo script:

[ ] Demonstrate that pre-deployment equipment meets size (fits into two medium-sized hard-shelled containers - ME-4) and weight (transportable a minimum of 100 yards by two average adults - ME-1) requirements.

[ ] Demonstrate setup within 5 minutes (ME-3).

[ ] Demonstrate use of user device to command pointing to specified azimuth and elevation (I/O-4).

[ ] Demonstrate accessible azimuth and elevation ranges, confirming they meet the requirement of physically pointing the antenna at any satellite traveling across the sky from horizon to horizon (FUN-1).

[ ] Demonstrate slew rate, confirming required slew rate of motor subsystem by measuring the azimuth and elevation turning rates, and confirming that these values are adequate to keep up with LEO satellites

[ ] Set up a weak tone transmitter in the room. Transmitter turns on and off with 1 second period. Using a "fixed" mode acquisition task, starting with antenna pointing away from transmitter, demonstrate acquisition of this signal, including viewing of spectrum on the user device.

[ ] Demonstrate tuning range requirement of having a tunable range between 1550-1700MHz, with 1400-1700MHz desired (PER-2) is met.

[ ] Demonstrate ability to select a bandwidth between 2MHz and 10MHz (FUN-4). Confirm by observing the spectrum of the noise floor on the user device.

[ ] Self-interference test: Replace antenna with termination. Demonstrate absence of self-interference (designed such that self-generated radio interference is minimized and undetectable by the system - PER-4).

[ ] Demonstrate ability of user device to disconnect and reconnect without disrupting ongoing observing tasks (accessible by the wireless user at any time during operation - I/O-7).

[ ] Demonstrate teardown within 10 minutes (ME-2).

[ ] Demonstrate ability to offload acquisition data after teardown, including logs, averaged spectrum vs. time, signal strength vs. time, and relevant metadata (store data in non-volatile storage - I/O-3)

Customer will take possession of the instrument at the conclusion of the in-person demo.

Within 1 week of the conclusion of the in-person demo:

[ ] Report documenting acceptance test and results (deliverable), including verification of:

# 8 Actions Table

| Action # | Open/ Closed | Existing documentation | Plan to fix | Fixed |
| --- | --- | --- | --- | --- |
| Make a physical drawing for this system so that we can flesh out the system. | Closed | [**3.3 Mount Schematic**](#_heading=h.wnfgmjmrt641) | Created a detailed design of the total system, including all measurements of components and wiring diagrams for subsystems. | Yes |
| Account for GPS antenna position so that it does not degrade the GPS signal strength | Closed | [**3.3 Mount Schematic**](#_heading=h.wnfgmjmrt641) | Included the position of the antenna on the exterior of the black box in the system design diagram, accounted for weather conditions. | Yes |
| Figure out how to tell which direction is North | Closed | [**1.3 Component Connections**](#_heading=h.8n42ayi75136) | Magnetometer module: Connected to the Raspberry Pi via I2C (SDA and SCL pins), providing magnetic field measurements to determine the heading and orientation of the rotor. | Yes |
| Solve issue with device used for execution of software (windows, linux or mac) | Closed | [**2.3.2 Executable Creation**](#_heading=h.67r2kxezkigg) | To address the issue with the device used for the execution of the software, we will use Linux Ubuntu as the operating system. Ubuntu is free and open-source, which meets the requirement that all software used must be free and open-source. | Yes |
| Solve uml sequence on slide 16 of CDR | Closed | [**2.2.2 Client-Server Communication (CSComm)**](#_heading=h.965ts4or3zcn) | Created a much more detailed sequence diagram. | Yes |
| Clear up the mount/computer controller on slide 17 of CDR | Closed | [**2.2.9 Yaesu Rotator Interface**](#_heading=h.i53meb305ztd) | Created a detailed sequence diagram of the algorithm. | Yes |
| Make mount / computer sequence diagram ( create deeper level of UML) | Closed | [**2.2.9 Yaesu Rotator Interface**](#_heading=h.i53meb305ztd) | Created a detailed sequence diagram of the algorithm. | Yes |
| Clear up the receiver interface on slide 18 of CDR | Closed | [**2.2.8 RSPDuo Interface**](#_heading=h.6q9wv2da1mx) | Created a detailed sequence diagram of the algorithm/ | Yes |
| Is the updating of the mount going to be continuous? Go more in depth with this. All of the commands for the slewing of the mount should be much more defined. Determine the accuracy of the slew rate of the mount. | Closed | [**2.2.9 Yaesu Rotator Interface**](#_heading=h.i53meb305ztd) | The continuous updating process of the mount is managed in the track\_satellite function. It calculates the time needed to move the mount based on its max\_elevation\_speed and max\_azimuth\_speed, using this as a sleep time between commands. This ensures efficient position updates without overloading the mount with commands. | Yes |
| Make a diagram of what the system will look like once it is set up. | Closed | [**3.3 Mount Schematic**](#_heading=h.wnfgmjmrt641) | Created a detailed diagram with measurements of the system. | Yes |
| Figure out the torque that the tripod can support. (Physical testing) | Closed | [**3.5.2 The Tripod**](#_heading=h.wnfgmjmrt641) | Did research into issues with the mount subsystem tripod, developed solution to potential torque issue. | Yes |
| Make a list of tests to test the entire system(acceptance test plan). The flow diagram should reference that assignment. Put in a slide for each subsystem. | Open | [**7 Acceptance Test Plan**](#_heading=h.4roj4hhmf3bt) | Made recommended changes after the first revision by Dr. Ellingson. | Yes |
| Ask Dr. Ellingson about power control from the computer. | Closed |  |  |  |
| Talk to Mrs. Stover about the schedule. | Open |  |  |  |