Final Report Report: Satellite Ground Station & RF Frontend

Jumana Schmidt, Wiley Tong, Rishan Patel

Team 61

TA: Jason Jung

ECE 445

May 6th, 2026

1 Abstract

The purpose of the Satellite Ground Station project is to help democratize access to and educate on the information gathered and transmitted on public low earth orbit (LEO) and also geostationary (GEO) satellites. This report is to document the individual progress made in this regard. The component in which this progress has been made includes the motorized mount, power supply, RF reception, and antennas. These respective parts require system design, firmware, frontend software, power and motor electronics, CAD, and system verification, and they are detailed in the following. Thus far, this component's end-to-end testing has demonstrated success in accurately tracking an LEO satellite for an entire ten minute pass and receiving a well enough signal to return a decodable output.

Contents

2 Introduction

There are over 14,000 satellites currently orbiting the Earth. From real-time weather images, oscillating sea level data, and amateur (HAM) radio, to solar flare alerts from our Sun, audio transmissions from astronauts on their way to the Moon, and even leaked unencrypted military communications, each satellite is transmitting a plethora of readily available information. Most of this data is critical to our everyday infrastructure. For remote communities, natural disaster survivors, and those facing information censorship or internet shutdowns from their governments, this data can even be life saving. With such valuable and intriguing information available, it is no wonder why there has been a growing interest in satellite communications for so many different communities. However, accessing satellite data directly or indirectly typically requires internet based services, expensive tracking hardware, RF experience, and at the least, a lot of manual setup. Whether you are a student or hobbyist learning RF/satellite communications, an experienced researcher, or a regular person facing a dire situation outside of their control, this lack of repeatable and affordable equipment creates a large barrier.

Many relevant or interesting satellites, including those for weather, are low Earth orbiting (LEO). These satellites or stations generally require real-time tracking through the sky, either manually or through a motorized mount, but even if the satellites are transmitting overhead, it's difficult to reliably aim an antenna for 15 minutes at a time, track the signal by hand, and turn that RF into usable decoded output. Though the reception software is free, and a basic software defined radio (SDR) USB peripheral (like a RTL-SDR) is very cheap, the hardware required is often difficult to find and almost always customized. Generally, most hobbyists, and even researchers, cannibalize and hack old TV dishes and/or rotors, but these are all subject to second-hand markets and often require many weeks of tuning feed dimensions to a specific dish. If they do not want to track by hand, the only affordable option is to hack into the hardware and software of proprietary rotor control units, normally designed for geostationary (GEO) TV satellites. Even still they might require additional equipment or not even allow such customization. Most of the components required for this hacked setup are not even available outside of the U.S., and most TV dishes are significantly too heavy to build a rotorary setup or be even remotely portable. Most of these satellites also usually transmit in L-band and/or in S-band. So even if a user procures and tunes all of their customized equipment, some of these microwave band signals can still be hard or impossible to properly receive due to limited range. An MMDS or frequency downconverter is required for both a cheap, $30 option like an RTL-SDR and even a step up to an SDR like a $300 Hack RF One. Additionally, there are not many commercial and affordable downconverters available. As a result of both of these hardware obstacles, receiving any updated critical/useful data is often impractical, inconsistent, or too costly for most people to try.

As our society continues to become more dependent on digital material, our goal is to help democratize access to information and communications by making radio and satellite tracking/reception more accessible and, by consequence, more popular for educators, researchers, off-grid users, radio enthusiasts, and normal people alike. To accomplish part of this task, we seek to address two of the most inaccessible and unaffordable aspects while being portable, affordable, and repeatable: live tracking of moving LEO satellites and making their microwave transmissions receivable by cheaper SDR's. More specifically, we seek to create an affordable automatic, motorized satellite tracker/receiver and a custom S-band frequency downconverter.

2.1 Subsystem Overview

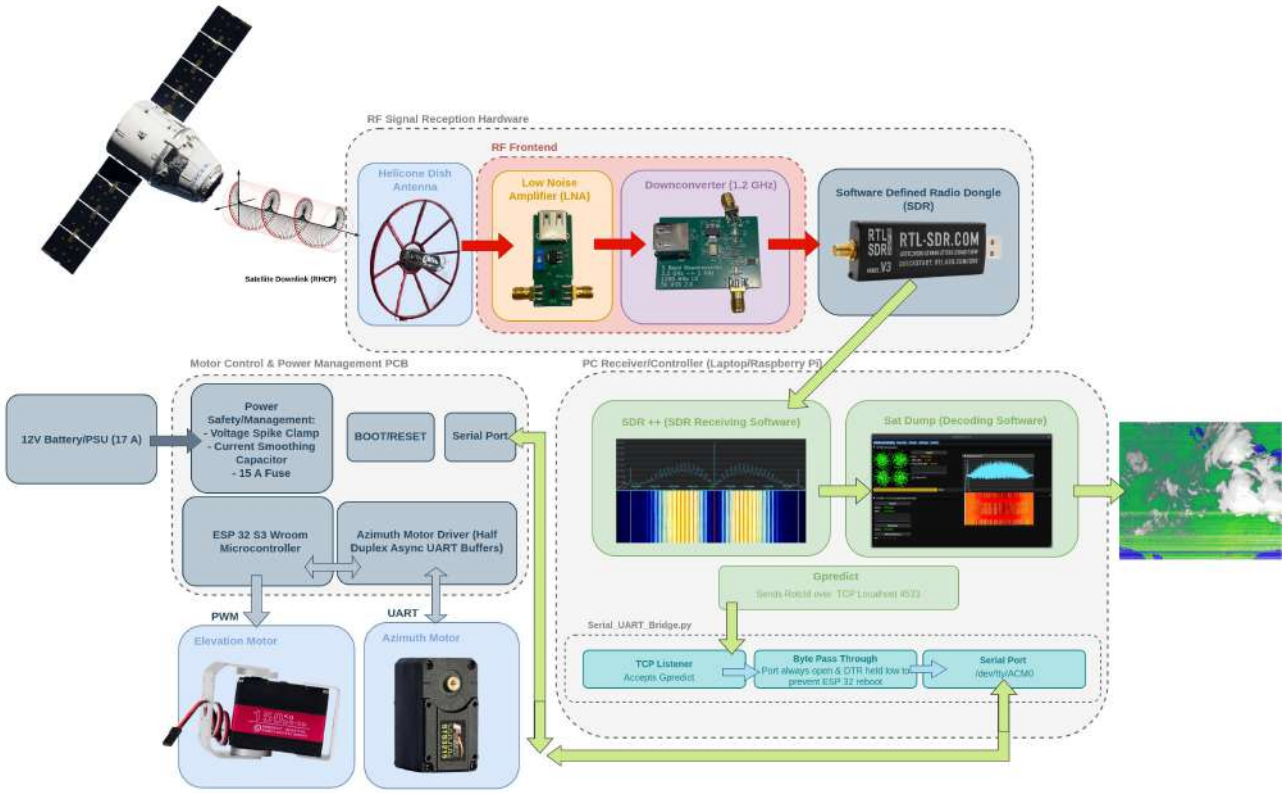

Figure 1: High level diagram of ground station

The high level overview of this project has three components: RF front end, motor controlled antenna, and a laptop. A helical antenna controlled by azimuth and elevation stepper motors tracks an LEO satellite through a microcontroller on the motor control and power management PCB. The Rf front is designed to amplify 1.7 or 2.2 GHz signal received by the antenna with the LNA board, downconvert the 2.2 GHz signal to 1.2 GHz with the downconverter board, and send that signal to the RTL-SDR. Using SDR++ we can view and save the digital data converted by the SDR as well as control its internal gain. Another program called Satdump is used to demodulate signals into message data (satellite weather images). And a third program, Gpredict, sends LEO telemetry data to the motor control board.

This report will go over in detail two subsections that we designed: the physical ground station with two antennas (S & L-Band) and the RF frontend boards.

Table 1: High Level Requirements

Requirements	Verification
• Any LEO satellite must be able to be tracked for 10-15 minutes.	• Any LEO satellite must be able to be tracked for 10-15 minutes.
• The low-noise amplifier increases the signal from the antenna by 20 dBm. The signal from the antenna will be at 2.25 gigahertz.	• By using a vector network analyzer, a test input signal is inserted into the low-noise amplifier. • Calculate the gain in dB by subtracting the power output of the LNA from the input power into the LNA • The test signal should be increased by 20 dBm at 2.25 gigahertz, as viewed on a spectrum analyzer.
• The downconverter is able to successfully shift the input frequency down by 1.2 gigahertz, at a power level (above a 15 dB net gain after losses) that can be received by the RTL-SDR module. .	• By using a vector network analyzer, a test input signal is inserted into the downconverter. • The test signal has a frequency at 2.25 gigahertz. The output signal of the downconverter should be viewed around 1.05 GHz on a spectrum analyzer.

3 Ground Station With a Motorized Antenna Mount

3.1 Design

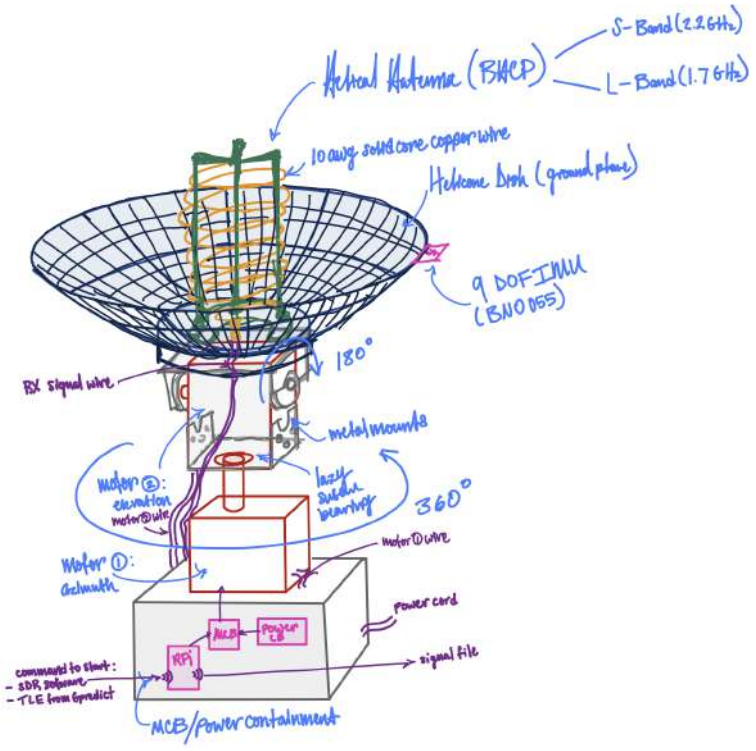

Figure 2: Ground Station Original Design

Figure 3: Ground Station Final Design

Excluding some challenges, most of the overall original design and procedure of the ground station remained the same throughout, except for the reliance on the IMU sensor for alignment calibration and the S-Band helicone antenna dimensions. The BNO055 magnetometer, accelerometer, and gyroscopic sensing was replaced by a simple, but manual, two step calibration upon each deployment. This decision proved successful because it made deployment of the unit much faster and more reliable than a sensor that was easily influenced by high-powered electronics and frequently required new internal calibration data after being physically moved. The two step manual calibration involved making sure the antenna was parallel with the horizon using a level and that the software knew the direction of North by aligning the antenna using a compass upon startup.

3.2 Satellite Antennas For L-Band & S-Band: Lightening the Load

Although the motorized antenna mount, and thus ground station, can be used for a wide range of frequencies, not just for our scope or even just satellites, L-Band (1700 MHz) and S-Band (2200 MHz) signals make up some of the most popular data to listen to and decode, and reception requires two separate antennas, or feeds, as well as dishes with separate geometries.

Figure 4: Repurposed TV Reflector Dishes: Prime v.s. Offset

All satellite signals, due to their weakening over such large distances, need a large dish to capture and focus the signal. Most users will use an old Ku-Band or C-Band TV dish to reflect the radio emissions to a custom-built helical feed. This setup usually will differ for all frequencies as either, an offset or prime focus dish, but these reflector dishes are often way too heavy for custom rotors and always need custom tuning of where to place the feed for each frequency. Thus, they are not automate-able, portable, repeatable, and thus, accessible to the average person. While generic repurposed dishes start at over fifteen pounds, require feed placement tuning, and regardless, building custom feeds, helicone dishes provide a much better alternative, weighing only 1-2 pounds for L-Band, require very easy to acquire materials, and work upon assembly with little to no tuning. This system's L-band antenna is based on the design by T0nito [1], which provides a lightweight, 3D-printable helicone optimized for 1.7 GHz HRPT reception. For S-Band, we seek to create a new design based off of the components in T0nito's, while maintaining their light, compact, and reproducible nature, so that a light, but effective, antenna can be used with this system's motorized mount.

A helicone dish antenna consists of a helical feed and a cone "reflector," which actually acts as a ground plane rather than simply a reflector. While the helical feed and ground plane connection can be tuned somewhat on a VNA, the active geometry of not just the feed but the reflector plane itself are extremely impactful when gathering such weak microwave signals. The number of turns, spacing between the wire, and direction of turning determine the frequency and handedness (left or right) of the circularly polarized light it is desired to receive. However the cone, or "mesh," dimensions also determine the amount of gain received. The current L-Band design prioritizes compact dimensions as well as a wider beamwidth, to allow for more tracking tolerance. Proven by the L-Band design's real world success, an effective S-Band design must also reflect these features. In a paper published by the University of Belgrade and Georgia Tech, the researchers demonstrated the optimal dimensions for a S-band helicone [2]. For operation at 2.26 GHz ($\lambda = 133$ mm), we would select a cone height and helix geometry by scaling with wavelength according to

$$\lambda = \frac{c}{f},$$

so that the S-Band antenna preserves the same electrical proportions while remaining compact. In practice, I chose a compact 3D-printed S-Band helicone scaled from the successful 1.7 GHz baseline so that it would remain lightweight, printable on a typical desktop 3D printer, and more likely to provide a reliable first result than a larger high-gain redesign. The resulting dimensions are a helix diameter of approximately 42 mm, 6.5 turns, and 18.8 mm turn spacing, with a cone base diameter of 101 mm, cone top diameter of 333 mm, and cone height of 66.5 mm. The helical pitch angle is determined by

$$\alpha = \tan^{-1}\left(\frac{S}{\pi D}\right),$$

where S is the turn spacing and D is the helix diameter, and the cone dimensions are set by the base diameter, top diameter, and height. This choice also remains consistent with the helicone literature, which shows that gain depends strongly on the coupled helix–cone geometry, while more compact designs can still provide strong performance without the increased size and tighter pointing tolerance of a more aggressively optimized standalone antenna [2, 3].

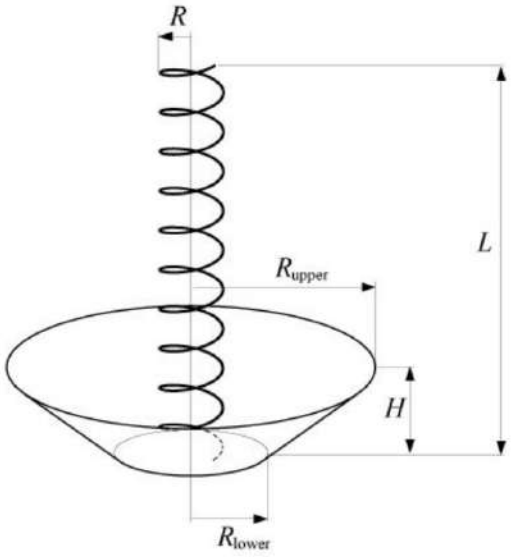

Figure 5: Helicone Dish Antenna Dimensions

3.3 Motor Control & Power Management PCB

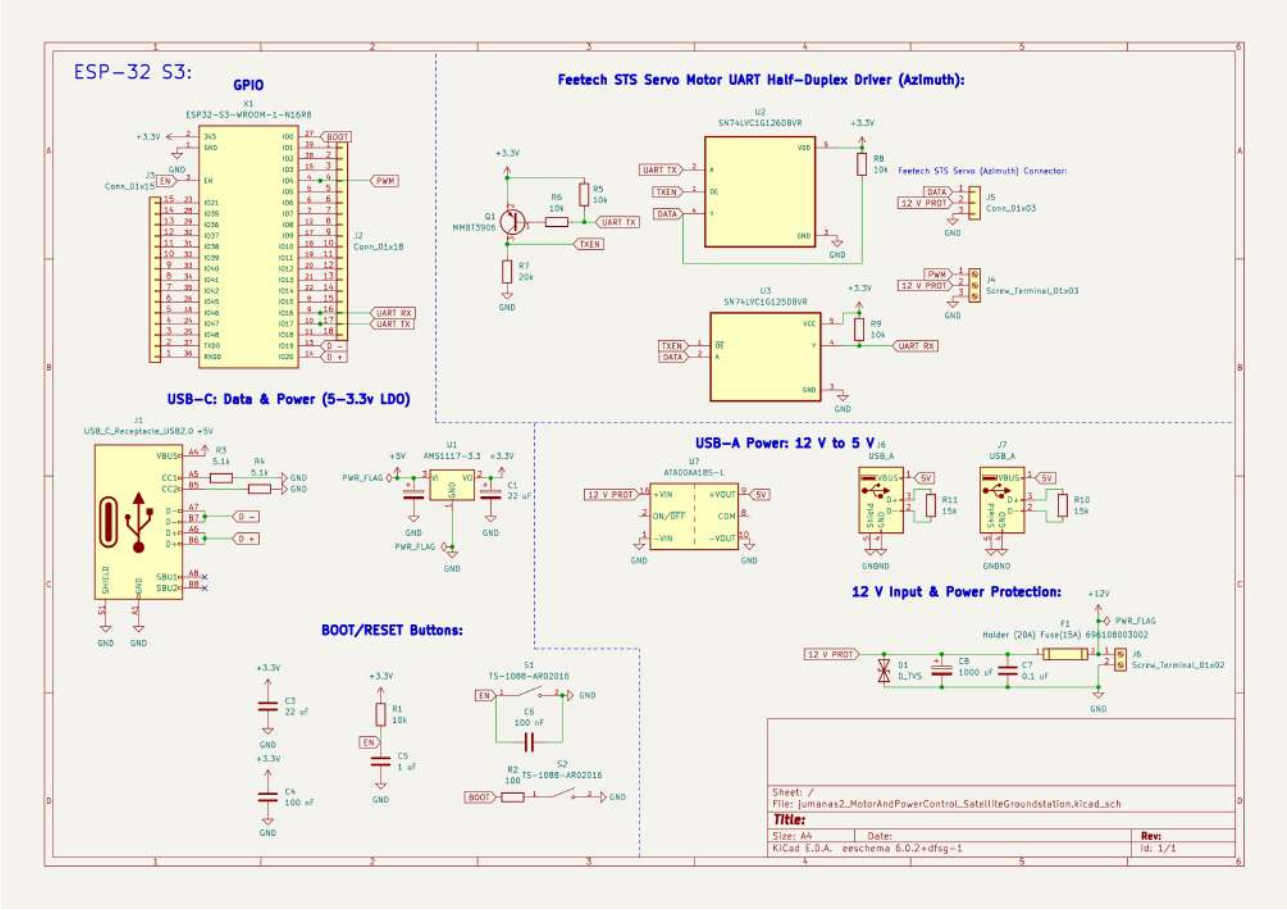

Figure 6: Motor Control & Power Management PCB Schematic

There are three distinct sections of the PCB for motor control and managing/protecting power supply for different parts. One section is entirely for the microcontroller, the ESP-32 S3 Wroom-1. This MCU control the communications for the entire system, specifically the UART and PWM servo buses. The other components in the section are to do with powering, booting and resetting, and flashing the ESP-32. The USB-C port provides data to the MCU, allowing ease of flashing, and more importantly, providing a serial bus for the Hamlib azimuth and elevation commands to come in from Gpredict on a PC. The BOOT and EN/reset button circuits allow for flashing and restarting. Then, the AMS 1117 regulator (LDO) converts the 5V from the USB-C into a usable 3.3V for the MCU. The next section is essentially a communication driver for the Feetech, azimuth, motor. The servo uses a half-duplex asynchronous UART protocol to communicate commands, feedback, and register changes on its EEPROM. This driver is arguably one of the most important parts. Specifically, it allows for position feedback from the servo, closing the motor control loop, and it prevents UART bus physical overload and contention. The main components are the two buffers. The SN74LVC1G126 buffer is used as the transmit side line driver. This chip takes the ESP-32 TX data and, when enabled, passes it onto the shared DATA line that goes to the servo. When it is disabled, it stops driving the line, which allows the servo to talk back. This is one of the most important parts of the interface, because it prevents constant contention on the shared data wire. The second, SN74LVC1G125 buffer, is being used on the receive side. Its job is to take the signal from the shared DATA line and present it cleanly to the ESP-32 as RX. That lets the ESP32 read status packets, position feedback, ping responses, and other data coming back from the servo. Lastly, The PNP transistor, and its surrounding resistors, function as the TX enable control logic and determines when the TX buffer is able to send data. Finally, the 12 V Input & Power Protection section is the main power entry point for the board. The screw terminal is where the external 12 V supply connects. The fuse provides overcurrent protection at 15 A, the TVS diode helps clamp voltage spikes, the 1000 µF capacitor helps smooth larger disturbances on the rail, and the 0.1 µF capacitor filters higher-frequency noise. Together, these parts create the protected 12 V rail that powers the motors and the 12 V-to-5 V converter for two external USB-A receptacles.

3.4 Firmware & Software

Regarding the software side of the motorized antenna mount, at a high level, the two motors that control azimuth and elevation are controlled directly by an ESP32-S3 running the firmware, while a laptop running GPredict provides the desired azimuth and elevation angles. GPredict is a free, open-source satellite tracking program that downloads two line element (TLE) data, which tells the user where selected satellites will be in the sky at any given time (see. This location data is essentially the azimuth, relative to North at zero degrees, and elevation, relative to the horizon/ground at zero degrees. GPredict uses a Hamlib network protocol called Rotctld to control antenna rotators, sending simple ASCII commands over a TCP connection.

Since GPredict expects a Rotctld-compatible TCP socket, but the ESP32-S3 communicates over USB serial, a Python serial UART bridge is used between the two. The bridge listens on TCP port 4533 and forwards all incoming GPredict traffic to the ESP32-S3 serial port, while also forwarding any serial responses from the ESP32-S3 back to GPredict. This allows GPredict to operate as if it were connected to a normal network rotator controller, while the ESP32-S3 firmware only needs to parse commands arriving through its serial interface. The bridge was also necessary because the ESP32-S3 uses native USB CDC rather than a dedicated USB-to-UART chip, so opening and closing the serial port can assert DTR and reset the board. To avoid this, the Python bridge opens the serial port with DTR held low before the port is opened and keeps the port open continuously, allowing GPredict to disconnect and reconnect without resetting the ESP32-S3. The firmware then parses incoming commands for GPredict tracking, manual control, and azimuth calibration.

3.4.1 Firmware

This azimuth motor is a Feetech STS 3215 servo, and it was chosen specifically for its affordability, high torque at 30 kg per cm, and ability to turn more than 360 degrees, while maintaining position control relative to a reference, which for this purpose would be North as zero degrees. In this multi-turn/stepper mode, this servo is able to take values in steps up to seven rotations in either direction, and the reference angle is lost after power loss. This artifact actually serves as a feature because with each deployment of the system, North will always need to be aligned before functioning. Additionally, the purpose of having the motor be able to make multiple turns is to account for a Meridian Flip, where values can jump from 0 to 359 and vice versa. Thus, the main function of this azimuth component of the firmware is to maintain continuous rotation, even when there is a Meridian Flip, so it finds the shortest distance to the new input angle. For setup, the motor has four different modes, so this continuous stepper mode is activated by writing the register 0x21 with mode value 3. The minimum and maximum angle limits are also set to zero, to allow for indefinite stepping. It communicates

these changes to the EEPROM and angle commands through half-duplex asynchronous UART to a driver board that splits the received packets from sent into separate TX & RX lines, allowing for a readable format.

The azimuth servo's default PID loop settings for the internal closed loop motor control (P being 3 and D being 32) were not optimized for operation with load, including the two pound L-Band antenna. With these default settings and load, the increased rotational inertia causes the control loop to overshoot and oscillate slightly in an attempt to over-correct, especially in smaller angle increments. By reducing the proportional gain (P) and derivative gain (D) to 16, the controller responds less aggressively to position errors, eliminating the jitter. Adding a 4-count dead band (for about 0.35 degrees) also helps the servo to ignore tiny errors. These settings are set in the firmware setup section by writing the values in registers at addresses 0x15 and 0x16. This PD tuning prevents signal loss and increased wear from servo hunting. After these setup procedures, the main part of the program then requires you to calibrate the reference angle to North, and setZero() defines the antenna's current physical direction as azimuth 0 by resetting the tracking variables isZeroSet, lastInputAz, continuousAz, and currentCmdAz. Each new azimuth input goes through getShortestDistAzimuth(inputAz), which compares the new wrapped azimuth input (such as values jumping between 359 and 0 degrees) to lastInputAz, corrects any wraparound by adding or subtracting 360 degrees if the jump is bigger than 180, and builds a smooth unwrapped heading in the global variable, continuousAz. In setAzimuth(), that unwrapped target is compared to currentCmdAz to find the actual move needed (deltaDeg), then converted to servo steps with degToSteps(), and finally sent with st.WritePosEx().

The elevation motor is the RDS1150 servo, and it uses simple pulse width modulation (PWM) to receive absolute angle positions. This servo has a range of 0 to 270 degrees, but it's absolute internal angle for 0 is below the mechanical clearance allowable for the mount bracket. So, there is an added offset of ten degrees called EL_OFFSET. The firmware's main function here is to convert the inputted elevation into a PWM signal in microseconds (us). Since it has 270 degrees of range, the servo has pulses from 500 to 2500 us in width for that range. It converts the inputted angles to a pulse width by linearly mapping the desired input angle, plus mechanical offset, to a fraction of the servo's entire 270 degree or 500-2500 us range.

4 RF Subsystem

The RF subsystem contains two boards, an LNA and a downconverting mixer board. Both were designed on two layer FR4. They use USB-A 2.0 jacks for power and SMA coax female ports for RF input and output.

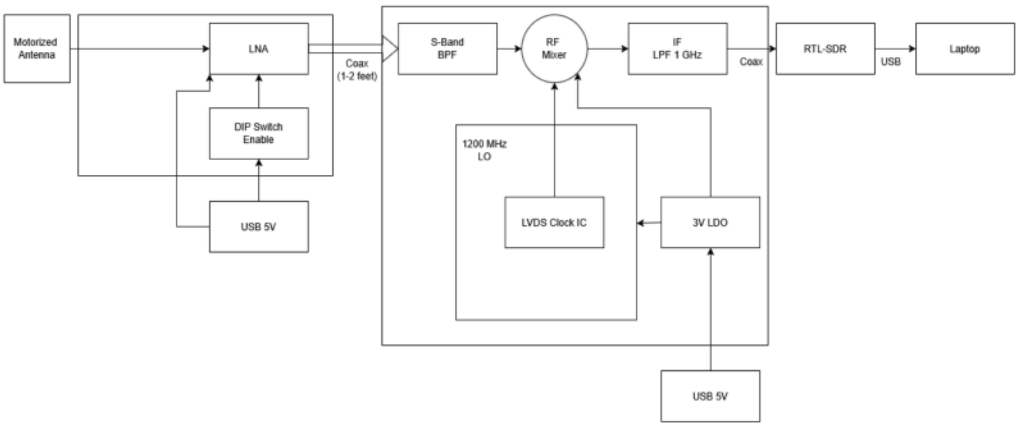

Figure 7: RF Subsystem

4.1 Design

The LNA board uses the TSS-14252LN+ LNA which has a typical gain of 35 dB in the frequency range 1427 to 2570 MHz. The only other active component is an LD1117 LDO to drop 5V input to 3.3V. The TSS-14252LN+ is low enabled which is controlled via a dip switch. SMA coax has a characteristic impedance of 50 ohms and designing an RF board requires impedance to be matched and consistent between components. Using the transmission line impedance calculator from JLCPCB we found that a coplanar waveguide with ground (CPWG) transmission line needs to have a width of 1.27mm and a 0.3mm distance from coplanar ground given

the perimeters of 2.25 GHz, 2 layer 1.6mm, FR4. When designing this amplifier board it was determined that transmission line effects could be ignored because of the small distance between the sma port to the LNA IC. The wavelength of an electromagnetic wave can be calculated with the equation $\lambda_g = \frac{\lambda}{\sqrt{\epsilon_r}}$ where ϵ_r is around 4.5 in FR4. The quarter wavelength of 2.25 GHz is 15.7mm. Total width of the LNA board and RF signal path length. Because the trace would be short and the pins of the IC were too small for a proper 50 ohm trace on 2 layer FR4 we decided to ignore trace width and instead tapered the trace from SMA signal pin to the LNA RF input and output.

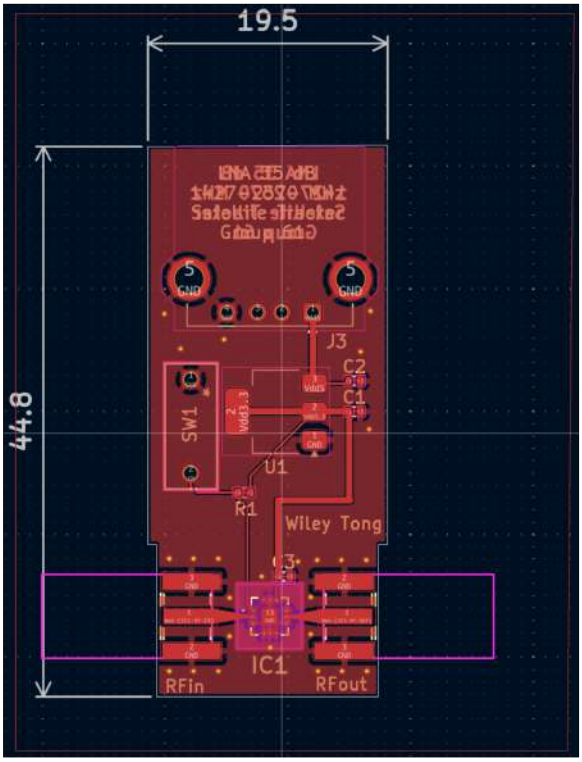

Figure 8: PCB view of the LNA board in KiCad.

The downconverter board uses the LT5560 active mixer and a 1200 MHz clock oscillator (XLL538C00.000000X) as LO. The LT5560 datasheet lists passive components for differential impedance matching at some frequencies. Above 2 GHz at the input a 50:100 balun and a 1pF capacitor across the differential lines is enough to match the internal differential inductors and resistive load inside the input port. This can be solved through L matching the input and out put impedance.

$$Q = \sqrt{\frac{R_s}{R_L} - 1}$$

$$X_c = R_s/Q$$

$$C = \frac{1}{\omega * Xc}$$

$$X_L = R_L * Q$$

These are the equations used to solve for the 1 pF capacitor across the differential lines into the input. At 2.25 GHz inductors are not needed because the internal and reactance is just greater than total reactance of the L-match. The LO input impedance match was taken directly from the application information for a 1200 MHz LO. The same cannot be said about the output matching.

There is not a given output impedance for the mixer at 1 GHz. Instead, values are given at 900 and 1500 MHz and a value was interpolated with the rest of the table data. The output resistance is modeled as a 1.2k resistor in parallel with a 0.7 pF capacitor. At 1 GHz the impedance was estimated as 910||-j260. Using the same L network method. This results in a 22.5 nH inductors on the differential lines and capacitance handled internally.

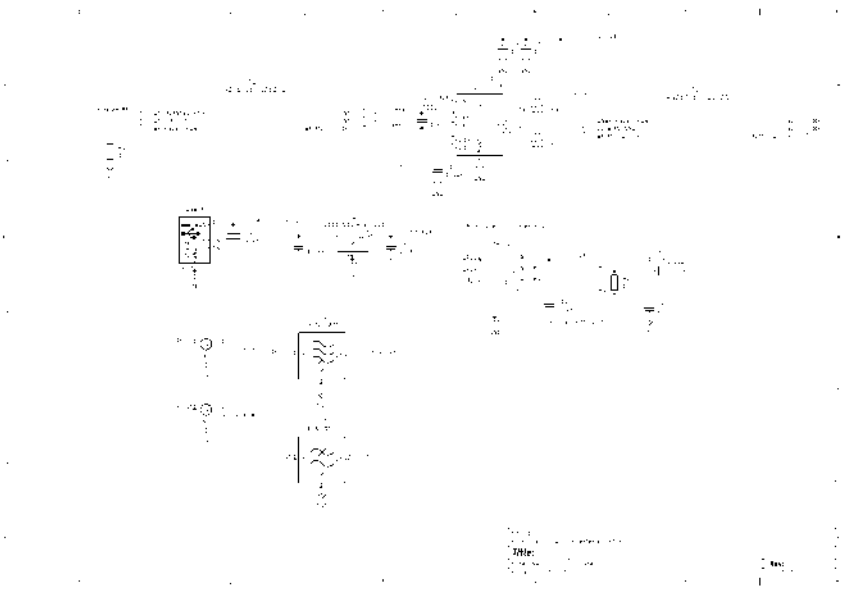

Figure 9: The downconverter board schematic.

5 Requirements and Verification

5.1 Integrated Verification Testing Procedure & Results

In one testing instance of a normal testing procedure, the full mount system was set up in a field to track a public weather LEO satellite called Meteor-M2 4 and receive High Resolution Picture Transmission (HRPT) images. HRPT are a great litmus test for how well the system can track a satellite because the nature of the signal and how it's moving is very unforgiving. Unlike APT or LRPT, HRPT requires a moving mount with 10 to 15 minutes of uninterrupted tracking. The 1.7GHz signal also has a narrow beamwidth, demanding continuous precision. It's QSK demodulation will not decode anything under a signal to noise ratio (SNR) of 15 dB. Despite all of the components of that are needed to receive a strong signal. The motorized mount and both L and S-Band antennas were able to accurately track and decode several passes. Even passes with significant cloud cover, some moderate 5 mph wind, and only 15 degrees in maximum elevation, the ground station system was able to return usable weather images, and in multiple spectrum channels. To further study the effectiveness of the tracking itself, we only recorded a pass (using the consistently successful L-Band antenna), miles away from noise and, most importantly, when the satellite was not eclipsed by the tree/obstacle-line. In an analysis of the decoded frames or channel access unit (CADU), we can see 100% framelock, meaning the antenna was locked onto the signal 100% of the time, and there are only about 15 to 40 sync errors, which is fantastic for a high resolution HRPT image signal. This analysis suggests that tree-line obstacles are the main sources of interference, besides noise. This hypothesis is supported by the consistent locations of these interference lines going across the outputted images. These lines indicate some drops in the reception, and the signal is received line by line (generally in portrait framing). Thus our tracking and antenna geometry is highly tuned to receive and track LEO satellites. The S-Band antenna also was able to receive signals from both DMSP F-17 and F-18, notably old satellites with weaker transmitters. However, these satellites only encrypt their signal over the very Northern quarter of the U.S., specifically after the 40 degrees attitude, so, in Champaign-Urbana, this antenna is only able to receive the last 2 to 5 minutes of a pass. Despite this impediment and knowing the interference is mainly caused by treeline, the fact that it provides a decidable HRPT signal is proof of its ability. In the future some modifications could be further improved upon to compensate with larger gain. Although it wasn't a strict high-level requirement, unfortunately, due to the insertion loss and reflections within the RF frontend, full integration with groundstation, antennas, and RF filtering was not successful. These HRPT signals were received easily through the LNA, with some distinct data lines present, but significantly distorted. After the downconverter, the 28 dBm loss was too significant to receive, and debugging with personal SDR's ended up disabling two of them.

5.2 Quantitative Results

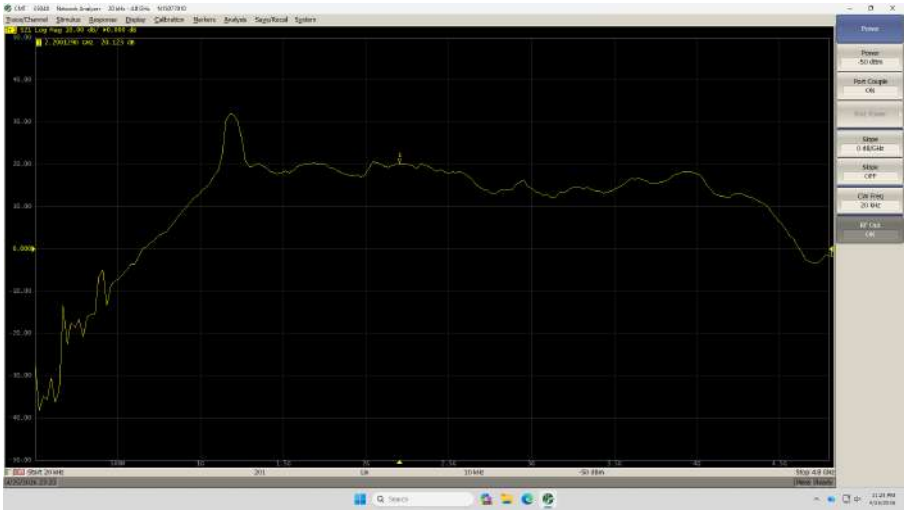

Figure 10: S21 (gain) of the LNA board.

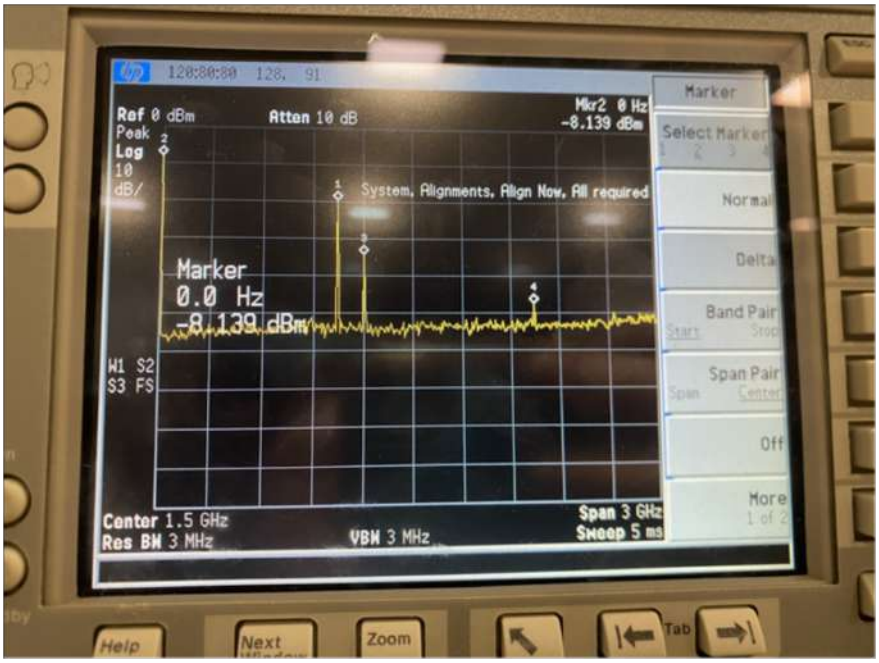

Figure 11: Downconverter output

Figure 10 is the S21 of the LNA board on a Vector Network Analyzer. A VNA sweeps Rf frequencies across each port and measures them at each port to calculate the scattering parameters of a two port system. S21 is the ratio of power exiting port 2 over port 1 at a 50 ohm match, this is essentially gain of the system. The marker shows 20 dB gain at 2.25 GHz and 20 dB gain can be seen at 1.7 GHz as well. This is lower than the datasheet's 31 dB gain and is likely due to an impedance mismatch related to the implementation of the SMA edge launch connectors. From figure 8 it can be seen that copper ground pour on the first layer seeps in-between the ground and signal pads of the SMA footprint. We suspect this lead to a non 50 ohm connection.

The mixer has an insertion loss of -20 to -28 dB. Figure 11 shows three peaks. The lowest frequency peak is 1 GHz, middle is 1.2 GHz, and highest is 2.4 GHz. 1 GHz is the downconverted signal, 1.2 is due to LO leakage entering the output port and 2.4 GHz is the second harmonic. Note that figure 11 seems to show the system working. However this result is only achievable with high RF input power (¿-20 dB). The downconverter does

not work for antenna signals even with our LNA. This result does show that the mixer is working but something in our circuit is producing significant insertion loss.

6 Cost and Schedule

6.1 Cost Analysis

Item	Part / Notes
RF LNA	TSS-14252LN+ $7.14
S-band bandpass filter	BFCN-2360+ $14.95
Active RF mixer	LT5560EDD#PBF $5.51
1200 MHz Oscillator	XLL538C00.000000X$13.67
IF filter	LFCN-900 $12.84
Baluns	X4BD10L1-50100G $2.36
Misc passives	Misc $12.73
Subtotal RF:	$69.20
Connectors (SMA)	Edge Launch SMA (E-Shop)
Connectors (USB-A)	U231-091N-3GRC10-5 (CSV)
Connectors (PWM header)	B3B-XH-A(LF)(SN) (CSV)
Connectors (UART header)	B4B-XH-A(LF)(SN) (CSV)
Connectors (prog header)	640456-6 (CSV)
Connectors (power terminal)	31001102 (CSV)
RC (E-Shop)	SMD resistors/caps
Resistors (CSV)	Generic, qty 5 (assign values/MPNs)
Capacitors (CSV)	Generic, qty 8 (assign values/MPNs)
Inductor (CSV)	Generic, qty 1 (assign value/MPN)
Power (high-side)	BTS50080-1TEA (CSV)
Power (rev protect PMOS)	DMP3013SFV (CSV)
Power (buck regulator)	MP1584 (CSV)
Power (3.3V LDO)	AP2112K-3.3TRG1 (CSV)
Fuses (PTC)	1812L400/16GR ×2 (CSV)
ESP32 module (option)	ESP32-WROOM-32 (CSV)
RTL-SDR	$30
Antenna & dish parts	Usually negligible (could be free depending on sources)
Azimuth motor	$28; Product link
Elevation motor	$37; Product link
PCB	$30
Lazy Susan bearing	$15
MCB & power management + parts	$8 + negligible extras
ESP32 (bare)	$8
Mount brackets	Machine shop

Week	Task	Person
2/23	**Design Doc due (11:59p)**	
	Design review sign-up due (11:59p)	
	JLCPCB order pass audit (4:45p)	
	Submit PCB orders for downconverter and power/motor Control board	all
	Tune L-Band Antenna	Jumana
	Meet w/ Machine Shop about ground station mount design	Jumana
3/2	**Design Review (8a–6p)**	
	Design LNA board	Wiley, Rishan
	Order parts for RF subsystem	Wiley, Rishan
	JLCPCB order pass audit (4:45p)	
	Finalize ground station motor mounts w/ machine shop	Jumana
	Complete & debug ground station motor code	Jumana
	Test L-Band Antenna	Jumana
	Assemble new S-Band Antenna	Jumana
	Start Designing Downconverter board	Wiley, Rishan
3/9	**Breadboard Demo**	
	Teamwork Eval I due (11:59p)	
	Continue on downconverter design	Wiley, Rishan
	JLCPCB order pass audit (4:45p)	
	Last day for machine shop revisions (ECEB 1048)	
	Test S-Band Antenna	Jumana
	Design Power Management PCB	Jumana
	Order inductors for downconverter	Wiley, Rishan
	Assemble and test LNA board if arrived	Wiley, Rishan
3/16	Spring Break	
	Continue on downconverter design	Wiley, Rishan
3/23	JLCPCB order pass audit (4:45p)	
	Bugfix Ground station motor control board	Jumana
	(Bugfix LNA board)	Wiley, Rishan
3/30	Individual progress reports due (11:59p)	
	Assemble and test downconverter	Wiley, Rishan
	Test RF frontend with dummy signal	Wiley, Rishan
4/6	**Progress Demo**	
	Team contract assessment due (11:59p)	
	(Test complete system)	all
4/13	(add task)	
	(Bug fix complete system)	all
	(Decode signal with Satdump, configure RTL-SDR)	Rishan
4/20	**Mock demo (weekly TA meeting)**	
	Mock Presentation	
	Video assignment due (11:59p)	
	Receive as man signals as from S-band LEO for final paper	all
	Work on final paper	all
4/27	**Final Demo (8a–6p)**	
	Final Presentation (8a–6p)	
	Finish up final paper	all
5/4	Best Projects judging/awards (2:00p, TBD)	
	Pitched Project Reviews (2070)	
	Final papers due (11:59p)	
	Lab checkout (3:00–4:30p)	
	Lab notebook due (11:59p)	

According to FCC regulations, radio compliance for our system means operating within the authorized frequency bands, power limits, and equipment authorization requirements that apply to the specific RF hardware and use case, and these laws are generally accepted or equivalent even outside the U.S. [4]. However, we will also have documentation making users aware of the legal limitations or protections with radio respective to their region. Although the motorized mount is primarily a mechanical/electrical subsystem, it interfaces with RF equipment that must meet these rules. Even for a receive only device, our end-to-end design follows good engineering practice and safe wiring to minimize radiated and conducted noise that could interfere with nearby electronics. For electrical and mechanical safety, our PCBs and connections follow hazard-based design practices reflected in modern equipment safety standards, such as insulation, enclosure integrity, and protection against overcurrent/overtemperature and fire risk. [5]. Although the device could be repurposed for some nefarious uses, such as tracking unauthorized targets, we reduce the incentive or ability to misuse the ground station by having firmware specific for tracking publicly available satellites/targets. Additionally, it is framed as an educational, practical, and/or research application. Arguably, it is also part of our goal to encourage more awareness and safety surrounding digital communications, especially from satellites, by having such an accessible device.

7 Conclusion

We succeeded in building a base station for L band RHCP LEO signals. This includes the antenna, the motorized mount, and the LNA. However, we failed to receive a decodeable S band signal. This is due to high insertion loss in the downconverter circuit making it unusable for mixing low power signals even after amplification.

One of the design decisions for the RF boards was ignoring transmission line effects because of the short length of traces. Furthermore these boards were not built on impedance controlled substrate, we used two layer FR4 which has variant relative permittivity. A four layer board with a controlled permittivity substrate would allow thin transmission line traces at 50 ohms which can easily connect to the small footprints of the LNA and downconverter ICs. We believe that this method of impedance control and electrically long transmission lines would result in better impedance matching and thus less insertion loss across the downconverter board.

7.1 Ethics

The motorized dish antenna mount and RF filters do not directly violate any ethical or legal practices, but they still pose some electrical and mechanical hazards to users. The mount uses two 12V high-torque motors capable of producing a pinch or crush hazards. Though, no motion will be sudden nor is the voltage very high, there is some risk of unintended motion if given wrong or late degree inputs and shock/fire hazards if the electrical components are left unprotected from weather/foreign elements or surges/incorrect connections. Thus, we will integrate a visible way to stop rotation, software prevention against sudden motion, mechanical guarding, some housing for components, fuses, and over-current protection [5].

References

[1] T0nito, "1.7 GHz (HRPT) helicone antenna," Thingiverse, 2024, accessed: 2025-04-09. [Online]. Available: https://www.thingiverse.com/thing:6436342

[2] D. I. Olćan, A. R. Zajić, M. M. Ilić, and A. R. Djordjević, "On the optimal dimensions of helical antenna with truncated-cone reflector," in *Proceedings of the 2nd European Conference on Antennas and Propagation (EuCAP 2007)*, Edinburgh, UK, November 2007, pp. 1–6.

[3] F. Sadeghikia and A. K. Horestani, "Design guidelines for helicone antennas with improved gain," *Microwave and Optical Technology Letters*, vol. 61, no. 4, pp. 1016–1021, 2019.

[4] Federal communications commission (fcc) rules and regulations (47 cfr). https://www.ecfr.gov/current/title-47. Federal Communications Commission. Accessed 2026-02-13.

[5] (2018) Iec 62368-1: Audio/video, information and communication technology equipment – safety requirements. International Electrotechnical Commission.

8 Appendix A: Additional L-Band Tracking Results

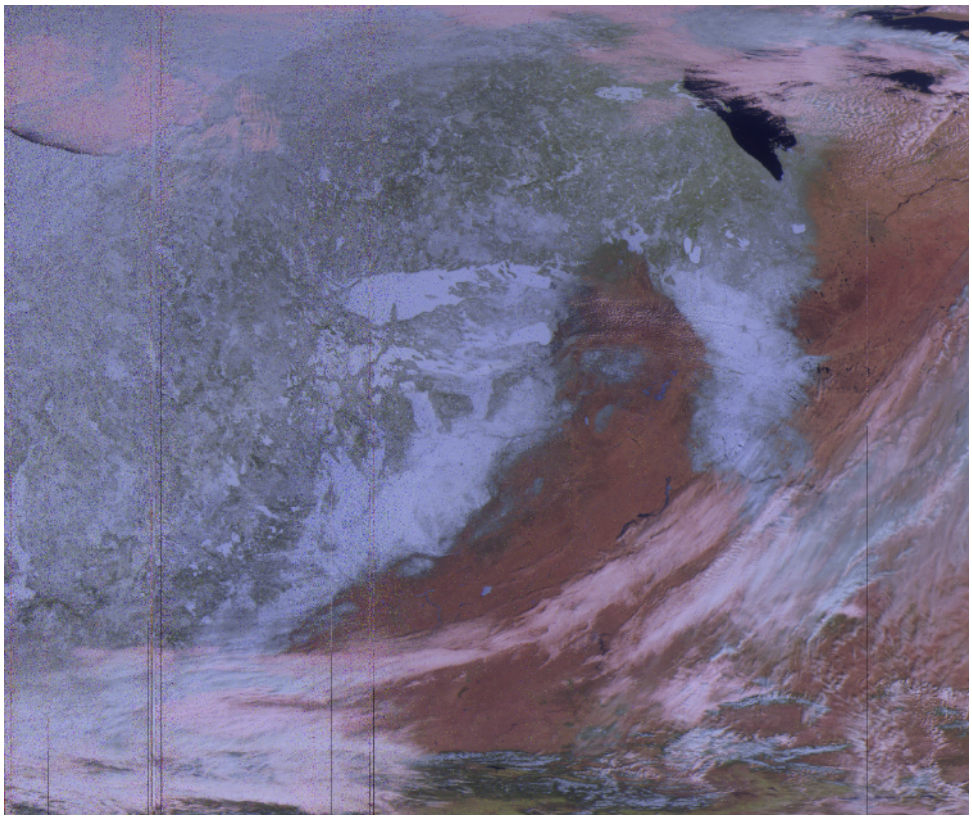

Figure 12: Meteor-M2 4 Real Time Photo of Earth: Natural Color/Visible Light

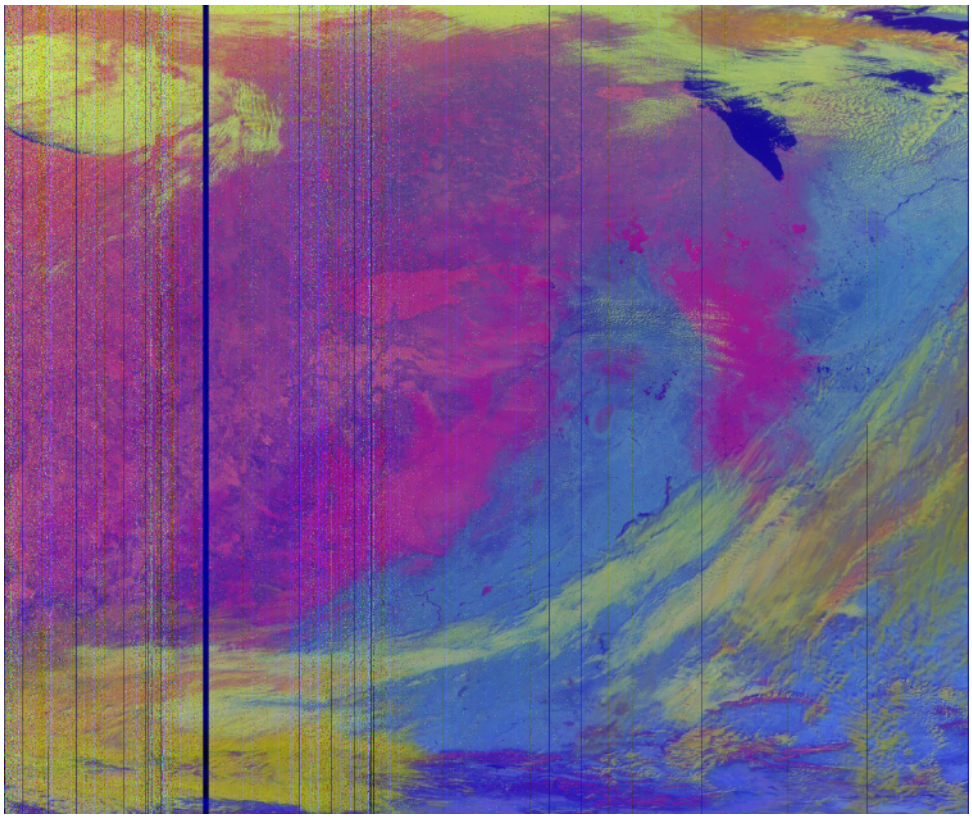

Figure 13: Meteor-M2 4 Real Time Photo of Earth: Day Microphysics

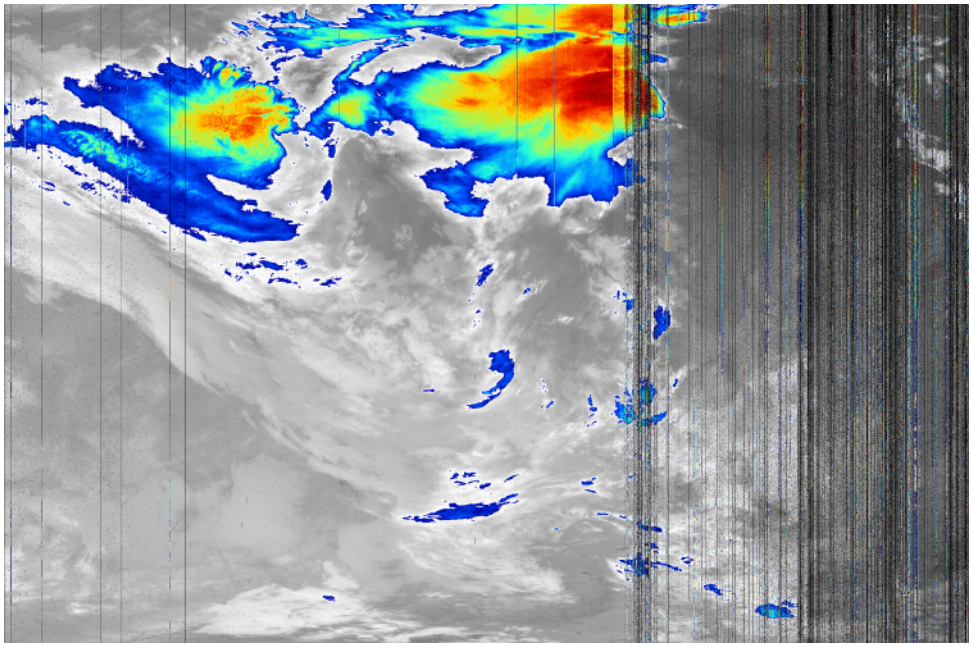

Figure 14: Meteor-M2 4 Real Time Photo of Earth: Cloud Top IR

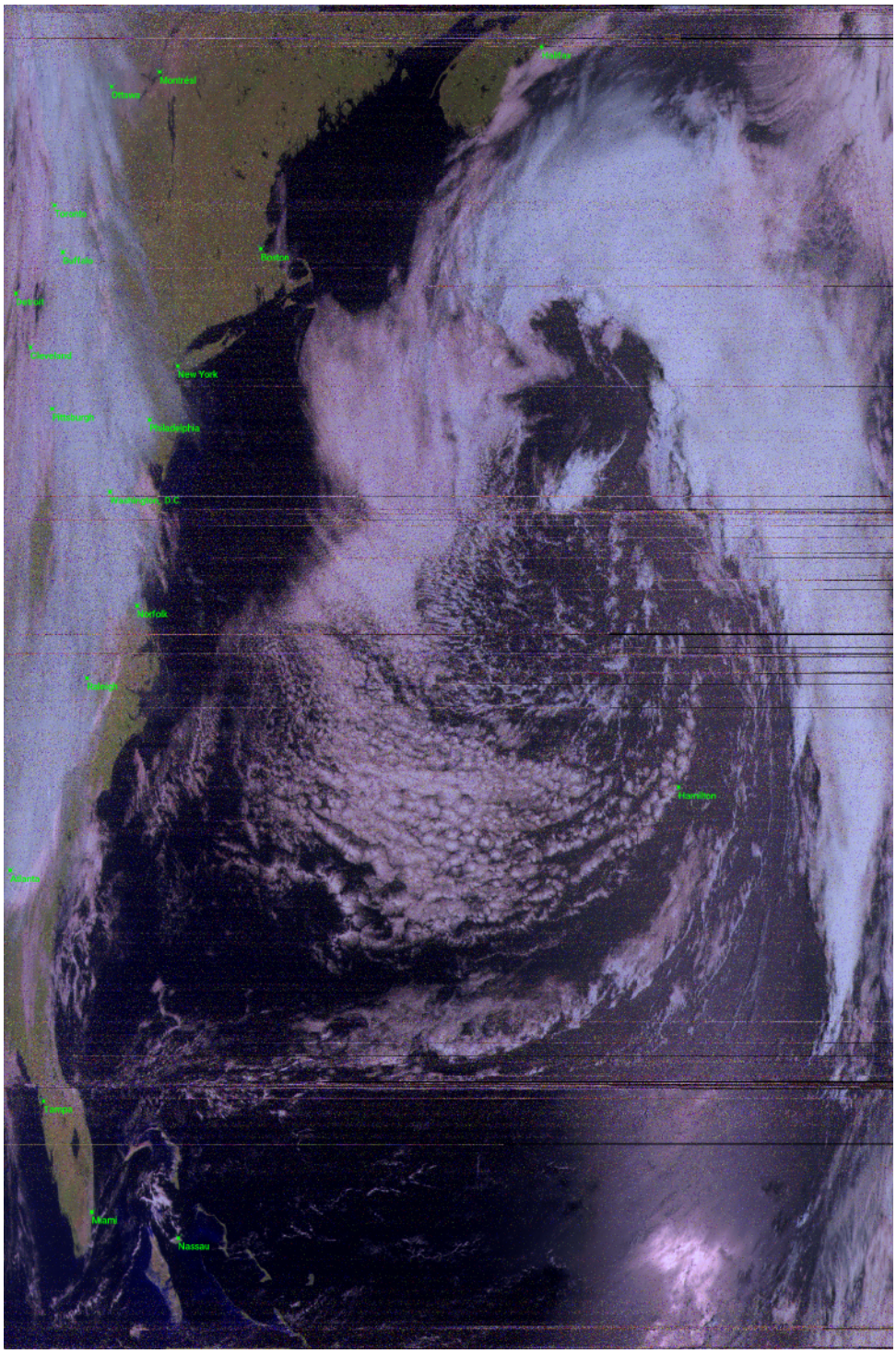

Figure 15: Meteor-M2 4 Real Time Photo of Earth: Natural Color/Visible Light with Cities Overlay

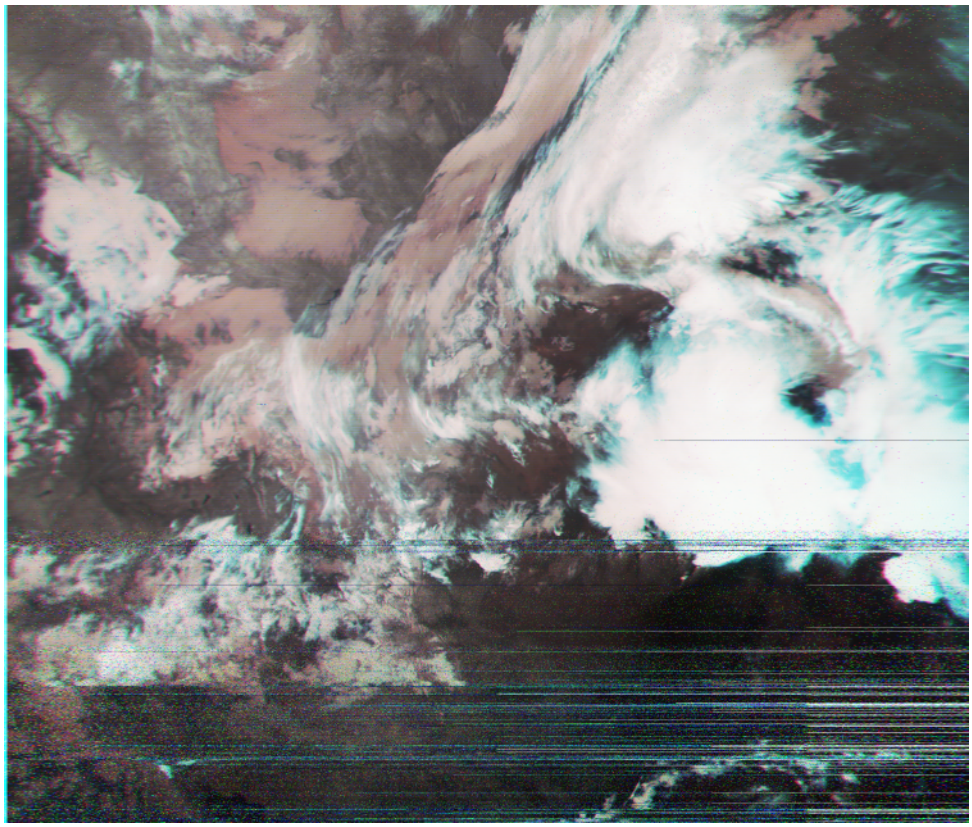

Figure 16: Metop-B 4 Real Time Photo of Earth: False Color IR

9 Appendix B: Serial UART Bridge

```
import socket
import serial
import threading
import sys
import time

SERIAL_PORT = sys.argv[1] if len(sys.argv) > 1 else "/dev/ttyACM0"
SERIAL_BAUD = int(sys.argv[2]) if len(sys.argv) > 2 else 115200
TCP_HOST = "0.0.0.0"
TCP_PORT = 4533

def open_serial():
    while True:
        try:
            # ESP32-S3 was resetting whenever the port opened,
            # so DTR is held low before opening the connection.
            ser = serial.Serial()
            ser.port = SERIAL_PORT
            ser.baudrate = SERIAL_BAUD
            ser.bytesize = serial.EIGHTBITS
            ser.parity = serial.PARITY_NONE
            ser.stopbits = serial.STOPBITS_ONE
            ser.timeout = 0
            ser.dsrdtr = False
            ser.rtscts = False
            ser.dtr = False
            ser.rts = False
```

```
            ser.open()

            print(f"[bridge] Opened {SERIAL_PORT} at {SERIAL_BAUD} baud")
            return ser

        except Exception as e:
            print(f"[bridge] Waiting for {SERIAL_PORT}: {e}")
            time.sleep(2)

def handle_client(conn, addr, ser):
    print(f"[bridge] GPredict connected from {addr}")
    conn.settimeout(0.05)

    try:
        while True:
            # Forward GPredict TCP data to the ESP32-S3 serial port.
            try:
                data = conn.recv(256)
                if not data:
                    break
                ser.write(data)

            except socket.timeout:
                pass

            except Exception as e:
                print(f"[bridge] TCP read error: {e}")
                break

            # Forward ESP32-S3 serial responses back to GPredict.
            try:
                waiting = ser.in_waiting
                if waiting:
                    data = ser.read(waiting)
                    conn.sendall(data)

            except Exception as e:
                print(f"[bridge] Serial read error: {e}")
                break

    finally:
        conn.close()
        print(f"[bridge] GPredict disconnected from {addr}")

def main():
    ser = open_serial()

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind((TCP_HOST, TCP_PORT))
    server.listen(1)

    print(f"[bridge] Listening on TCP:{TCP_PORT} -> {SERIAL_PORT}")
    print(f"[bridge] Point GPredict at localhost:{TCP_PORT}")
    print("[bridge] Press Ctrl+C to stop\n")

    try:
        while True:
```

```
            conn, addr = server.accept()

            thread = threading.Thread(
                target=handle_client,
                args=(conn, addr, ser),
                daemon=True
            )
            thread.start()

    except KeyboardInterrupt:
        print("\n[bridge] Shutting down")

    finally:
        server.close()
        ser.close()

if __name__ == "__main__":
    main()
```

10 Appendix C: Azimuth & Elevation Motor Firmware

```
#include <ESP32Servo.h>
#include <SCServo.h>

Servo pwmELServo;
SMS_STS st;

const int EL_SERVO_PIN = 6;

const int EL_MIN_US = 500;
const int EL_MAX_US = 2500;

const float SERVO_TRAVEL_DEG = 270.0;
const float EL_OFFSET = 10.0;

#define S_RXD 4
#define S_TXD 5

const uint8_t SERVO_ID = 1;
const float COUNTS_PER_DEG = 4096.0 / 360.0;

const uint8_t REG_MIN_ANGLE_LIMIT = 0x09;
const uint8_t REG_MAX_ANGLE_LIMIT = 0x0B;
const uint8_t REG_OPERATION_MODE  = 0x21;

const float STEPS_PER_REV = 4096.0f;
const float STEPS_PER_DEG = STEPS_PER_REV / 360.0f;

const uint16_t MOVE_SPEED = 1500;
const uint8_t  MOVE_ACCEL = 50;

// Tracking state
bool isZeroSet = false;
float lastInputAz = NAN;
float continuousAz = 0.0f;
float currentCmdAz = 0.0f;
float currentEl = 0.0f;

// Operating mode
enum OpMode {
  MODE_MANUAL,
```

```
41    MODE_TRACK
42  };
43
44  OpMode currentMode = MODE_MANUAL;
45
46  enum AzOpMode {
47    POS_SERVO_MODE = 0,
48    CONST_SPEED_MODE = 1,
49    PWM_SERVO_MODE = 2,
50    MULTI_TURN_STEP_MODE = 3,
51  };
52
53  void setAZOperationMode(AzOpMode mode) {
54    st.writeByte(SERVO_ID, REG_OPERATION_MODE, mode);
55  }
56
57  long getCurrAngle() {
58    return st.ReadPos(SERVO_ID);
59  }
60
61  void setZero() {
62    isZeroSet = true;
63    lastInputAz = NAN;
64    continuousAz = 0.0f;
65    currentCmdAz = 0.0f;
66
67    Serial.println("Zero set here.");
68    Serial.println("Current physical direction is now azimuth 0.");
69  }
70
71  float getShortestDistAzimuth(float newInputAz) {
72    if (isnan(lastInputAz)) {
73      lastInputAz = newInputAz;
74      continuousAz = newInputAz;
75      return continuousAz;
76    }
77
78    float delta = newInputAz - lastInputAz;
79
80    if (delta > 180.0f)  delta -= 360.0f;
81    if (delta < -180.0f) delta += 360.0f;
82
83    continuousAz += delta;
84    lastInputAz = newInputAz;
85    return continuousAz;
86  }
87
88  int degToSteps(float deg) {
89    return (int)lroundf(deg * COUNTS_PER_DEG);
90  }
91
92  void setAzimuth(float inputAz) {
93    if (!isZeroSet) {
94      Serial.println("Zero not set. Calibrate first (send 'Z' or 'MANUAL').");
95      return;
96    }
97
98    float targetAz = getShortestDistAzimuth(inputAz);
99    float deltaDeg = targetAz - currentCmdAz;
100   int deltaSteps = degToSteps(deltaDeg);
101
102   if (deltaSteps != 0) {
103     st.WritePosEx(SERVO_ID, deltaSteps, MOVE_SPEED, MOVE_ACCEL);
```

```
104       currentCmdAz = targetAz;
105     }
106   }
107
108   void moveEL(float inputAngle) {
109     inputAngle += EL_OFFSET;
110     inputAngle = constrain(inputAngle, 0 + EL_OFFSET, 90 + EL_OFFSET);
111     currentEl = inputAngle - EL_OFFSET;
112
113     int pulseUs = EL_MIN_US + (inputAngle / SERVO_TRAVEL_DEG) * (EL_MAX_US - EL_MIN_US);
114     pwmELServo.writeMicroseconds(pulseUs);
115   }
116
117   void sendRotctldResponse(float az, float el) {
118     // GPredict expects: "AZ EL\n"
119     Serial.print(az, 1);
120     Serial.print(" ");
121     Serial.println(el, 1);
122   }
123
124   void handleGPredictCommand(String cmd) {
125     cmd.trim();
126     /*
127       GPredict rotctld protocol commands:
128       p - get position
129       P <az> <el> - set position
130       S - stop
131       q - quit
132
133     */
134     if (cmd == "p") {
135       // Return current position
136       float az = currentCmdAz;
137       while (az < 0) az += 360;
138       while (az >= 360) az -= 360;
139       sendRotctldResponse(az, currentEl);
140
141     } else if (cmd.startsWith("P ")) {
142       // Set position: P <az> <el>
143       int firstSpace = cmd.indexOf(' ');
144       int secondSpace = cmd.indexOf(' ', firstSpace + 1);
145
146       if (secondSpace > 0) {
147         float az = cmd.substring(firstSpace + 1, secondSpace).toFloat();
148         float el = cmd.substring(secondSpace + 1).toFloat();
149
150         if (currentMode == MODE_TRACK) {
151           setAzimuth(az);
152           moveEL(el);
153         }
154
155         // Always respond with RPRT 0 (success)
156         Serial.println("RPRT 0");
157       }
158
159     } else if (cmd == "S") {
160       // Stop command - just acknowledge
161       Serial.println("RPRT 0");
162
163     } else if (cmd == "q") {
164       // Quit - ignore or acknowledge
165       Serial.println("RPRT 0");
166
```

```
  } else {
    // Unknown command
    Serial.println("RPRT -1");
  }
}

void setup() {
  Serial.begin(115200);
  while (!Serial) {}

  pwmELServo.setPeriodHertz(50);
  pwmELServo.attach(EL_SERVO_PIN, EL_MIN_US, EL_MAX_US);

  Serial1.begin(1000000, SERIAL_8N1, S_RXD, S_TXD);
  st.pSerial = &Serial1;
  delay(1000);

  int servo_id = st.Ping(SERVO_ID);
  if (servo_id == -1) {
    Serial.println("Ping error: Azimuth motor not found");
    return;
  }

  Serial.println("Azimuth motor found");
  st.unLockEprom(SERVO_ID);

  Serial.println("Configuring PID for stability...");

  int p_gain = st.readByte(SERVO_ID, 21);
  int d_gain = st.readByte(SERVO_ID, 22);
  int i_gain = st.readByte(SERVO_ID, 23);

  st.writeByte(SERVO_ID, 21, 16);
  st.writeByte(SERVO_ID, 22, 16);
  st.writeByte(SERVO_ID, 26, 4);

  Serial.println("New PID: P=16, D=16, DeadBand=4");

  setAZOperationMode(MULTI_TURN_STEP_MODE);

  st.writeWord(SERVO_ID, REG_MIN_ANGLE_LIMIT, 0);
  st.writeWord(SERVO_ID, REG_MAX_ANGLE_LIMIT, 0);

  st.LockEprom(SERVO_ID);
  delay(100);

  Serial.println("\nGround Station Controller Commands ");
  Serial.println("In MANUAL mode (use TRACK to enable GPredict)");
  Serial.println("\nCommands:");
  Serial.println("  MANUAL     -> Manual control mode");
  Serial.println("  TRACK      -> Tracking mode (GPredict)");
  Serial.println("  Z          -> Set current pointing as azimuth 0");
  Serial.println("  AZ <deg>   -> Move to azimuth (e.g., AZ 180)");
  Serial.println("  EL <deg>   -> Move to elevation (e.g., EL 45)");
  Serial.println("  STATUS     -> Show current position and mode");
  Serial.println("\nGPredict: Use rotctld protocol on this serial port");
  Serial.println("Baud: 115200, Mode: rotctld\n");
}

void loop() {
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();
```

```
230
231       // Convert to uppercase for command checking
232       String cmd = input;
233       cmd.toUpperCase();
234
235       // Swihtc modes:
236       if (cmd == "MANUAL") {
237         currentMode = MODE_MANUAL;
238         Serial.println("Switched to MANUAL mode");
239         Serial.println("Use AZ/EL commands to control");
240
241       } else if (cmd == "TRACK") {
242         currentMode = MODE_TRACK;
243         if (!isZeroSet) {
244           Serial.println("Warning: Zero not set!");
245           Serial.println("Current position will be treated as 0°");
246           setZero();
247         }
248         Serial.println("Switched to TRACK mode");
249         Serial.println("Ready for GPredict commands");
250
251       } else if (cmd == "Z") {
252         setZero();
253
254       } else if (cmd == "STATUS") {
255         Serial.print("Mode: ");
256         Serial.println(currentMode == MODE_MANUAL ? "MANUAL" : "TRACK");
257         Serial.print("Zero set: ");
258         Serial.println(isZeroSet ? "YES" : "NO");
259         Serial.print("Current Az: ");
260         Serial.print(currentCmdAz);
261         Serial.print("Current El: ");
262         Serial.print(currentEl);
263
264       } else if (input.startsWith("AZ ") || input.startsWith("az ")) {
265         float az = input.substring(3).toFloat();
266         setAzimuth(az);
267         Serial.print("Moving to Az: ");
268         Serial.println(az);
269
270       } else if (input.startsWith("EL ") || input.startsWith("el ")) {
271         float el = input.substring(3).toFloat();
272         moveEL(el);
273         Serial.print("Moving to El: ");
274         Serial.println(el);
275
276       } else if (currentMode == MODE_TRACK) {
277         // In tracking mode, treat unknown commands as GPredict rotctld
278         handleGPredictCommand(input);
279
280       } else {
281         Serial.println("Unknown command. Available:");
282         Serial.println("MANUAL, TRACK, Z, AZ <deg>, EL <deg>, STATUS");
283       }
284     }
285   }
286   }
```