SSC21-XX-XX

A Versatile and Open-Source Radio Framework for the D3 CubeSat Mission

Carlos Carrasquillo
University of Florida Department of Mechanical & Aerospace Engineering
939 Center Dr, Gainesville, FL 32611
c.carrasquillo@ufl.edu

Faculty Advisor: Riccardo Bevilacqua
University of Florida Department of Mechanical & Aerospace Engineering

ABSTRACT

This paper details the design and implementation of the communications system for the Drag De-Orbit Device (D3) CubeSat mission. The D3 mission aims to validate the effectiveness of a novel approach to aerodynamically-based orbital maneuvering and controlled re-entry. Using four deployable tape-spring booms that can be retracted to any intermediate length, the D3 can accurately modulate the ballistic coefficient profile of the CubeSat in low Earth orbit (LEO). To compute the necessary boom deployment and reproduce a desired trajectory, the guidance profile of the CubeSat must be actively generated, which itself is a very computationally intensive process. The guidance will therefore be generated in a local computer, condensed into a transmittable file, and uplinked to the D3 CubeSat.

In order to enable the transmission of telecommands, the execution of telecommands, and the collection of the CubeSat's telemetry, robust radio libraries and protocols were developed to run on both the CubeSat's onboard computer and the ground station. The ground station program, written in Python, interfaces with a terminal node controller (TNC), which itself interfaces with the ICOM9100 transceiver. The onboard radio library runs was developed in C++ and built to be compatible with Linux. It interprets, executes, and responds to inbound telecommands when invoked using a single line of code. Both programs were developed using the same architecture and design principles, which can easily be ported over for other missions and applications. When fully tested, the software will become available for use by hobbyists, amateurs, and professionals for use in future CubeSat missions.

Introduction

One of the largest obstacles in the design of a CubeSat is the implementation and testing of the communications system. The communications system is responsible for transmitting commands to the CubeSat and relaying information about the CubeSat back to a ground station. Excluding fully-autonomous missions, a CubeSat mission is usually considered a failure if a secure connection with a ground station cannot be made.

One study that surveyed 178 deployed missions, 70 resulted in either partial or full mission failure. Out of these 70, 14% were identified to have failed due to problems with the communications system.[1] Another study found that 19 of 27 studied failed missions experienced anomalies that could have been avoided with more ground testing[2] (although it must be noted that these are all not necessarily attributed to communication failures). Because the flight software can not be restarted in the event of a communication failure, the radio must be rigorously developed and tested prior to launch.

Current radio applications are either closed source, mission-specific, or require a large learning curve to integrate into an existing software suite. One reason why it can be hard to integrate existing radio nodes into existing software is because the command and data handling node, considered a part of the radio node for the purposes of this paper, is often the backbone for the flight software. The command and data handling functionality is therefore inseparable from the flight software's main logic.

This paper outlines the development of a new set of software nodes set to run on a ground station and a CubeSat, respectively. The radio nodes were designed to be able to transmit large amounts of information over any number of packets. Robust packet structures and protocols were developed to identify packet losses and prevent illegal or disruptive telecommands. The architecture was also designed to be easily modified for any other CubeSat application with minimal adjustments to the communication protocol/interface. The CubeSat node can be

invoked by a single command from the main software loop. When invoked, the node interprets any incoming data, executes the inbound telecommand, and responds with with either an acknowledge message, error message, or the requested telemetry. Because all command and data handling occurs outside of the main software loop, developers can instead focus on the CubeSat's mission-specific functionality.

Mission Overview

The Drag De-Orbit Device (D3) is a <1U attachment to a CubeSat that can modulate its aerodynamic drag coefficient by deploying or retracting four independent tape-spring booms to any intermediate length. The variable aerodynamic drag coefficient can be used for orbital maneuvering, collision avoidance, and de-orbit location targeting.[3] The D3 was conceptualized and designed in support of the Space Situational Awareness (SSA) effort to keep track of objects in orbit and predict where they will be at any given time.

The D3 CubeSat mission is a testbed for this new technology, and aims to validate some of the results obtained from a MATLAB© simulation for the guidance algorithm. The satellite has a turnover date of August 2021 and a projected launch date of December 2021.

Avionics

All of the computing on the CubeSat is handled by a central BeagleBone Black developer board. For guidance generation, telemetry acquisition, and power, the BeagleBone Black relies on communication with the following off-the-shelf components:[4]

- **Battery and EPS**: a 20Whr DHV Technology battery and electronic power system (EPS) power all components, recharge the battery, and regulate the voltage and current on the power rails.
- **Solar Panels**: an array of five solar panels (supplied by DHV Technology) will be used to recharge the battery throughout the duration of the flight.
- **GPS**: a global positioning system will be used to report a position fix for critical steps in the guidance generation.
- **IMU**: a 9-axis inertial measurement unit used as the feedback mechanism for closed-loop control of the angular rates.
- **Magnetorquers**: using a magnetic dipole, the magnetorquers can provide the necessary action for attitude control, detumbling, and stabilization (as computed by the B-dot law, which relies on IMU data).
- **DC Motors**: these motors are equipped with encoders to perform closed-loop extension and retraction of the tape-spring booms.
- **UHF Transceiver**: a half-duplex UHF transceiver receives uplinked telecommands and downlinks telemetry from/to a ground station.
- **Antenna System**: four tape-spring antennas in a turnstile configuration are connected directly to the transceiver for UHF communication.

The two most relevant devices for the radio are the UHF transceiver and the antenna system, both of which will be explored thoroughly in later sections of this paper.

Flight Software

The D3 CubeSat mission also intends to validate the use of the Robot Operating System (ROS) framework in CubeSat missions. ROS is an open-source set of libraries used to facilitate the development and deployment of robotics systems. ROS uses a publisher-subscriber model, where software modules (called *nodes*) can communicate with each other over *topics*. A more detailed discussion of the ROS framework (and its specific application to the D3 CubeSat) is detailed in *A Novel Approach to CubeSat Flight Software Development Using Robot Operating System (ROS)*.[4]

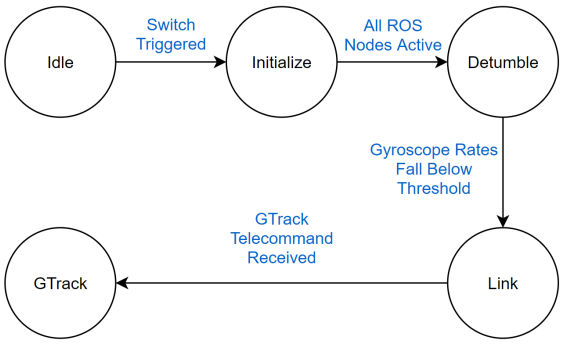

Figure 1: The finite state machine running aboard the D3 CubeSat.

	Flag	Destination Address	Source Address	Digipeter Addresses	Control Field	Protocol ID	Information Field	FCS	Flag
Bytes:	1	7	7	0-56	1	1	1-256	2	1

Figure 2: A diagram illustrating the AX.25 packet structure.

At its core, the flight software is a finite-state machine. Throughout its mission, the flight software will step through each of its five states, called *Idle*, *Initialize*, *Detumble*, *Link*, and *GTrack* (Fig. 1).

1. **Idle**: The CubeSat will remain in the idle stage from the time it is turned over to the launch provider to the time that it is ejected into orbit. In this state, all avionics are off and awaiting power from the EPS. When the switches on the 1U structure are triggered, all avionics are powered up and initialization of the flight software begins automatically.

2. **Initialize**: The flight software will be initialized in preparation for the detumbling phase. This stage is critical because improper initialization of the flight software will prevent the finite state machine from proceeding.

3. **Detumble**: Once all avionics are online, the CubeSat will automatically begin the detumbling phase. Detumbling aims to reach zero angular velocity about all axes (except the Zenith, which points along the length of the CubeSat). This is done by using the IMU and the magnetorquers to perform closed-loop control of the angular rates as governed by the B-dot law detumbling algorithm. The CubeSat moves on from this stage once the angular rates dip below a predetermined threshold.

4. **Link**: A two-way communications link will be established between the ground station and the CubeSat. The antennas will be sequentially deployed and the radio will begin scanning. All other software nodes will begin publishing to topics in preparation for a telemetry downlink. The CubeSat will remain in this stage until it receives an explicit telecommand to proceed with the guidance tracking.

5. **GTrack**: The telecommand to initiate guidance tracking (GTrack) is uplinked along with a guidance file.The CubeSat will use the DC motors to modulate the extension and retraction of the tape-spring booms as dictated by the guidance tracker control law.[?] The CubeSat will receive intermittent guidance file uplinks to account for error accumulation. This is the last state programmed into the finite-state machine.

Telemetry Specifications

All telemetry will be conducted over the 70 cm Amateur Radio bands. The Amateur Radio bands are free to use and are typically reserved for non-commercial endeavors, making them very popular amongst hobbyists and in education. The Amateur community also provides a lot of useful information about working with radios that utilize the Amateur bands, which is ideal for new missions like D3. The D3 mission has been designed to operate over UHF Amateur frequencies in half-duplex mode. Table 1 lists some notable system parameters for transmission.

Table 1: Transmission Specifications

Parameter	Value
Operating Frequency	437.08 MHz
Data Rate (Rb)	9600 bps
Modulation Type	GMSK
Protocol (AX.25) Overhead	7.25%
Desired Bit Error Rate	1E-5
Required Eb/N0	9.6 dB
eirp	33 dBm

AX.25 Protocol

The communication protocol to be used for transmission over the UHF bands is the AX.25 protocol. AX.25 is a data-link layer protocol frequently used in packet radio networks. AX.25 offers call setup and teardown, error detection and correction, and robustness for communicating over channels with long delays.[5] An AX.25 frame is shown in Figure 2. Since no digipeters will be used between the ground station and the CubeSat, only 20 bytes were considered for the protocol overhead calculation. For uplinks that utilize the maximum length

of the data field (256 bytes), the protocol overhead is 7.25%. For packets that consist of a single 8-bit telecommand, the protocol overhead can be as large as 95.24%. This was deemed acceptable because, using a baud rate of 9600 bps, the transmission lasts approximately 0.2 seconds, which is a small fraction (0.06%) of the time available during a single pass.

Satellite Passes

Planned parameters of the D3 CubeSat's orbit are listed in Table 2. Experimentation involving communication with other CubeSats in LEO at the Erich Farber ground station (the D3 mission's ground station) suggest that clear reception is received beginning at an elevation angles of 20 degrees. The estimated contact time during each ideal overhead pass, t_c, can be computed using simple trigonometry, where, $R =$ the radius of the Earth, $r =$ the perigee/apogee of the CubeSat, $\phi =$ the angle of elevation, and $T =$ the orbital period.1

Table 2: Planned Orbit Parameters

Parameter	Value
Apogee	6786 km (415 km altitude)
Perigee	6786 km (415 km altitude)
Inclination	51.6 deg
Period	1.55 h

$$t_c = 2\tan^{-1}\left(\frac{r-R}{R}\sec\left(\pi-\phi\right)\right)\times\frac{T}{2\pi} \quad (1)$$

The resulting duration of a directly overhead pass is 5.6 minutes. All uplinks and downlinks should ideally be transmitted within that 5.6 minute window with an additional factor of safety in the event of packet failures.

Communications System Hardware Design

The communications system consists of both the ground station radio and the onboard radio. The ground station is considered the master device while the onboard radio is the slave device. The implications of this design choice are discussed in the *Communications System Software Design* section.

Ground Station Radio

The D3 mission will make use of the Erich Farber ground station based at the University of Florida's Energy Research and Education Park. The Erich Farber ground station has been used (and is currently in use) for a number of CubeSat missions, including the University of Florida's CHOMPTT and SwampSat II missions. Table **??** shows the unique configuration of the ground station hardware as required for the D3 mission.

The station was designed to operate as a *packet radio* station, where packets are composed (on uplink) and interpreted (on downlink) purely in software. Having a software solution for data management can cut down on both operator time and experience while simultaneously reducing the chances of user error. The terminal node controller (TNC) is the primary controller for the packet radio scheme. It interfaces with a computer over a serial port on one end and with the transceiver over the other.

The output of the transceiver is then fed into a linear amplifier to increase the overall power of the signal up to 550 Watts prior to transmission.

The output of the linear amplifier is then amplified again by the preamp, which can be adjusted has an adjustable gain of 10-20 dB. The gain will be tuned until the power received by the CubeSat's transceiver is no more than -70 and no less than -115 dm, as per the onboard transceiver's absolute ratings.[6] The output of the preamp is input to the 70 cm Yagi antenna.

The antenna is a directional antenna, so it relies on rotor to track the satellite throughout the pass. The rotor is cable of setting the azimuth and elevation angles of the antenna, These angles are updated by software running on the main computer, which are fed into the Rotor Computer Interface via a RS-232 (COM) connection.

Figure 3 shows a high-level schematic of all the aforementioned equipment. Table 3 describes the precise equipment models for the D3 CubeSat mission.

Onboard Radio

Perhaps the most trivial yet meaningful difference between the ground station and the onboard radio is that the onboard radio is immutable. Once the CubeSat is in orbit, very little can be done to diagnose hardware problems from the ground. For this reason, tested OTS components were selected whenever possible.

The Clyde Space URTX Transceiver is a commercial half-duplex UHF transceiver with the ability to communicate with the onboard computer using the I2C bus. It features built-in AX.25 support, transmit and receive buffers, two redundant microcontrollers, an inactivity beacon, and health

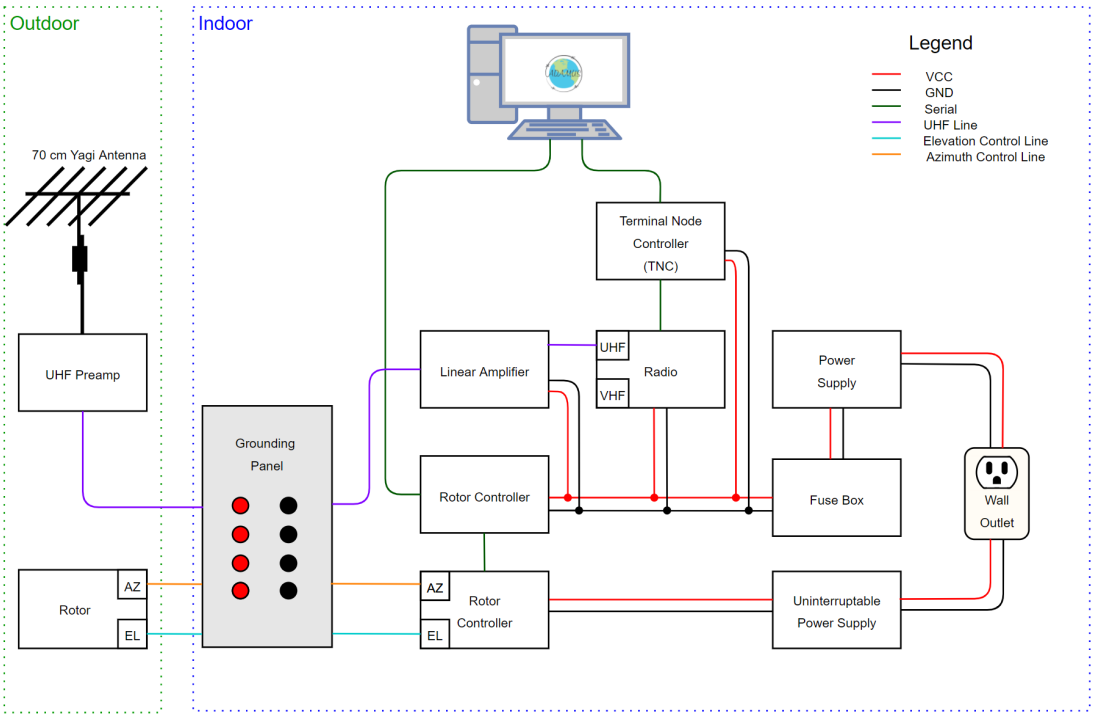

Figure 3: High-level ground station hardware schematic.

telemetry. The output power of the transceiver is also adjustable through software, but will remain at 2W throughout the duration of the mission. The transceiver utilizes an SMA connector to send and receive modulated signals though the antenna system.

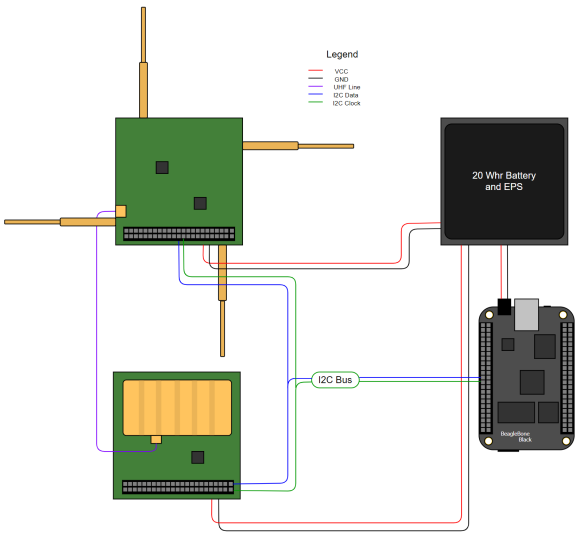

Figure 4: High-level onboard radio hardware schematic.

The ISIS Turnstile Antenna System consists of four deployable antennas whose operation is also controlled over the I2C bus. When the antennas receive a command from the central computer, a spark is ignited and spring tensioned wire begins to burn. The wire burns from anywhere between <1 s to <20 s, depending on the temperature (and by extension, the CubeSat's exposure to sunlight). When the wire fails, the corresponding antenna is deployed. All antennas can be deployed manually or using an automatic burn sequence. A schematic of the transceiver and antenna circuit is shown in Figure 4.

Table 4: Onboard Radio Link Budget

Device	Idle (W)	Peak (W)
URTX Transceiver	<0.240	5.1
Antenna System	<0.040	2

Both the transceiver and the antenna system are powered by the EPS and 20 Whr battery. Table 4 shows a simplified link budget of the onboard radio. It must be noted that the ISIS Antenna will only require the peak power for less than one minute throughout the duration of the mission (during the *Link* state). The URTX Transceiver will only require peak power draw while telemetry is being downlinked.

Table 3: Ground Station Equipment List

Device	Model	Purpose
70 cm Yagi Antenna	M2 436CP30	UHF (432-438 MHz) antenna
Preamp	SP-7000	Increase the gain of the UHF antenna
Rotor and Rotor Controller	Yaesu G-5500	Azimuth and elevation controller for the antenna
Rotor Computer Interface	Yaesu GS-232B	Enables rotor control through software
Radio	iCOM IC-9100	Uplink/downlink frequency control
Terminal Node Controller	KAM-XL	Packet radio, modulation/demodulation
Fuse Box	MFJ-1129	Overcurrent protection
UHF Linear Amplifier	Beko HLV-550	UHF gain amplifier
Power Supply	Astron RS-20A	Dedicated power supply for the tranceiver
Uninterruptible Power Supply	APC LS-700	Emergency/back-up power

Communications System Software Design

The radio will operate in a master-slave configuration, where the ground station is the master device and the onboard radio is the slave. The onboard radio will not transmit unwanted telemetry unless it is prompted to do so by a telecommand from the ground station. The only exception to this rule is the inactivity beacon, which downlinks health telemetry automatically after a period of inactivity.The D3 mission will use the inactivity beacon to transmit the CubeSat's health data, including the battery's charge, IMU data, temperature data, voltages on the power rail among other information. The inactivity beacon serves two very important purposes. The first is that, in the event of a ground station failure, Amateurs will be able to publish the CubeSat's health data on the mission's website. This will allow project overseers to know the status of the satellite until the ground station is fixed. The second purpose is to provide validation that the CubeSat is still online. This is especially useful during stretches where contact is unsuccessful.

Data Flow Overview

All telemetry exchanges will commence with the ground station. The ground station software will package all of the data into a KISS frame, which will be transmitted to the TNC. The TNC will use the KISS packet to generate the AX.25 packet, which it modulates into an audio signal. The audio signal will then be transmitted over UHF by the radio. The UHF signal will propagate until it reaches the D3 CubeSat, where the transceiver will demodulate the data and convert the digital data from an AX.25 packet to a Simple Protocol packet.

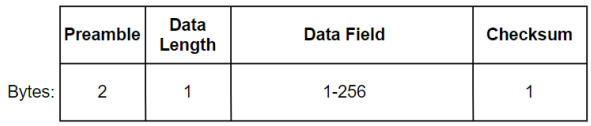

Figure 5: A diagram illustrating the Simple Protocol packet structure.

The Simple Protocol is a special packet structure implemented by the URTX UHF Transceiver (Fig. 5). It acts as an extra layer of security for inbound and outbound transmissions. If the incoming or outbound packet does not abide by the Simple Protocol, it is discarded by the transceiver. If it passes, the packet is then interpreted by the radio software to extract the data field. The reverse process is used to downlink telemetry. Both the uplink and the downlink process are shown in Figure 6.

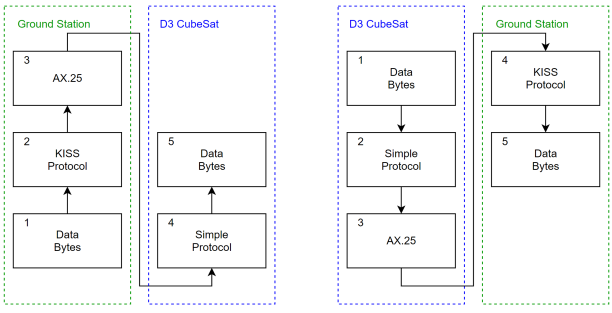

Figure 6: A flow chart showing how data is packaged and transmitted during uplink (left) and downlink (right).

The radio aboard the CubeSat was implemented as a software library responsible for both I2C communication with the avionics and command data handling. The library was written entirely in C++

for speed and cross-platform operating system compatibility. The functionality of the radio is invoked using a single function call, *Radio.scan()*. When the *Radio.scan()* function is called, a telecommand is read from the receive buffer (if one is available), executed, and the corresponding acknowledge signal/telemetry is transmitted.

The simplest form of the radio library consists of five different classes as shown in Figure 6. When an instance of the *Radio* class is created, it creates instances of the *UHF_Transceiver*, *Interpreter*, *Handler*, and *Packager*. The functionality of these classes is described below.

- **I2C_Functions**: This library was developed to facilitate the development of all avionics libraries. One instance is created for each device to store the active I2C bus and the device address. Data can then be written or read to/from the device by using only the register address.

- **UHF_Transceiver Library**: This library is responsible for getting and/or setting data to and from registers in the URTX transceiver. There is one getter function for each readable register in the transceiver. Each getter function requires no parameters and returns an 8-bit unsigned integer array containing anywhere from 1 to N bytes of data (i.e., one byte per register, up to N registers). Similarly, there is one setter function for each writable register in the transceiver. Setter functions have no return value but require an 8-bit unsigned integer array of 1 to N bytes of data as the only parameter (one byte per register, up to N registers). This library also has additional support functions, including a *sendString* function for sending strings of data instead of 8-bit unsigned integers. The only dependency of the UHF Transceiver library is the *I2C_Functions* library.

- **Interpreter**: The interpreter class is responsible for accepting an incoming package and extracting the telecommand and parameter fields. When the interpreter is invoked by the *Radio*, it uses the *UHF_Transceiver* library to read the incoming bytes from the receive buffer. The incoming bytes are organized in an unsigned 8-bit integer array according the Simple Protocol packet structure (Fig. 5). The packet is separated into the corresponding fields and the data field is extracted. The first byte of the data field is extracted to form the telecommand and the remaining bytes are assigned to the parameters (one long string). The telecommand and parameters are then returned as a structure.

- **Handler**: The Command Data Handler, or *Handler*, is responsible for executing some action and/or collecting the data to be downlinked. It is organized as a large switch-case statement (or optionally a look-up table), where the compared expression (or index in a look-up table) is the telecommand. If necessary, the handler spawns a new thread to execute the task while the parent thread sets the downlinked signal's value. Examples of executed tasks can include spawning a ROS node, initializing a class and calling a function, or running a shell script. The downlinked signal's value can either be an acknowledge or error signal, bytes of telemetry, or a filename. The *Handler* then passes this value to the *Packager* before it is transmitted. The *Handler* also keeps a time-stamped log of all of the telecommands executed by the CubeSat, which can be downlinked on request.

- **Packager**: The *Packager* is responsible for assembling the data according to the simple Protocol packet structure. If the data is telemetry or an acknowledge/error signal, the length and checksum are computed and the string is converted into an 8-bit unsigned integer array. array. If the data is a filename, the file is opened, converted into a byte array, converted into packets, and transmitted. The *UHF_Transceiver's sendString* function is then invoked and the data is transmitted.

- **Radio**: The radio (software) is responsible for configuring the URTX UHF_Transceiver upon startup and facilitating the transmission of data between objects. When the *Radio.scan* function is invoked, the radio calls on the Interpreter to fetch the inbound command. If the command exists, a command structure is passed to the Handler. The radio class also has functions that routinely check the configuration of the URTX UHF_Transceiver to ensure that they have not changed (as recommended by the datasheet).

A special case occurs when the telecommand *Upload File* is received. The handler identifies this command and extracts the packet number from the first byte of data. If the packet number is 1, the num-

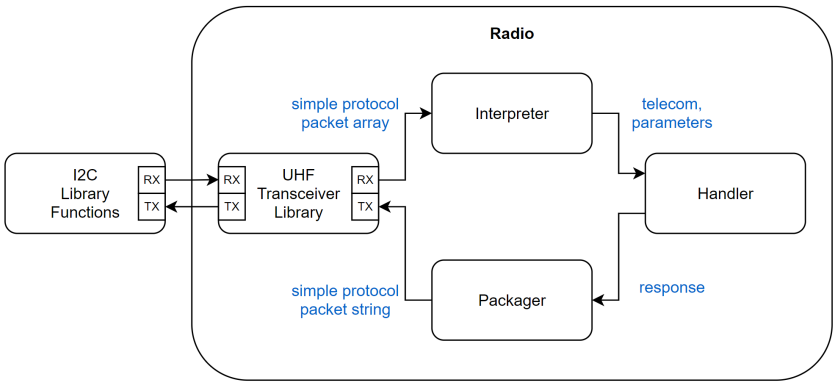

Figure 7: The basic architecture of the onboard radio software. All nodes are implemented as classes. Data flow between nodes is controlled by the Radio class.

ber of packets to be received is read from the second byte. The length of the destination field is read from the third byte, followed by the destination field (which includes the filename). A Start of File (SOF) ASCII character is then expected, followed by the contents of the file. Each inbound packet is scanned for the packet counter bit, which is expected to increment for each packet received. If the packet counter bit jumps by more than 1, a packet was lost and an error message is downlinked. The file is closed when the End of File (EoF) character is received.

File uploads are a time-consuming process, so file backups were implemented into the software to undo an accidental file upload. If the uplinked file already exists, a backup of the last file is created. A user can then send an *Undo File Upload* telecommand to delete the latest version and restore the previous version.

To downlink a file, a similar procedure is repeated. The packet number is set as the first byte, and, exclusively for the first packet, the expected number of packets is transmitted. Both the destination and length of the destination field are omitted from the downlinked packet, and the decision over the destination address is surrendered to the user. The number of packets is therefore immediately followed by the SOF character and the data itself. All subsequent packets behave as described in the uplink procedures.

The radio was designed to abide by the principle of encapsulation. All radio exchanges are handled by the this library, which greatly reduces the complexity of the remainder of the flight software. This structure also has the added benefit of preventing unintentional downlinks. Other than beacon downlinks, no telemetry is transmitted without the radio first receiving an explicit telecommand from the ground station.

Perhaps most importantly, the radio is versatile. The with minor adjustments to the communication protocols and hardware/avionics libraries, the library can be ported to any other operating system or computer. Future iterations of the software will support more communication protocols as well as automatically configuring unique communication protocols based on the contents of a configuration text file. Upon initialization, the software will read the contents of the text file and define both inbound and outbound packet structures.

Ground Station Software

There two programs that must be actively running for telemetry to be exchanged- SatPC32 and the D3 Ground Station program. SatPC32 is a program developed by Erich Eichmann (callsign DK1TB) to compute the orbits of satellites. The program has a graphic user interface (GUI) that continually updates the map of the satellite being tracked as it passes over the station. SatPC32 has the added functionality of interfacing with the Yaesu GS-232B rotor computer interface, which allows for satellites to be automatically tracked throughout a pass. SatPC32 also has automatic Doppler correction. SatPC32 runs on Windows.

The D3 Ground Station program is responsible for transmitting telecommands, awaiting on the satellite's response, and presenting the information to the user in a digestible manner. The D3 Ground Station software transmits and receives data through the TNC using the KISS packet structure over a COM port. Python was selected as the primary pro-

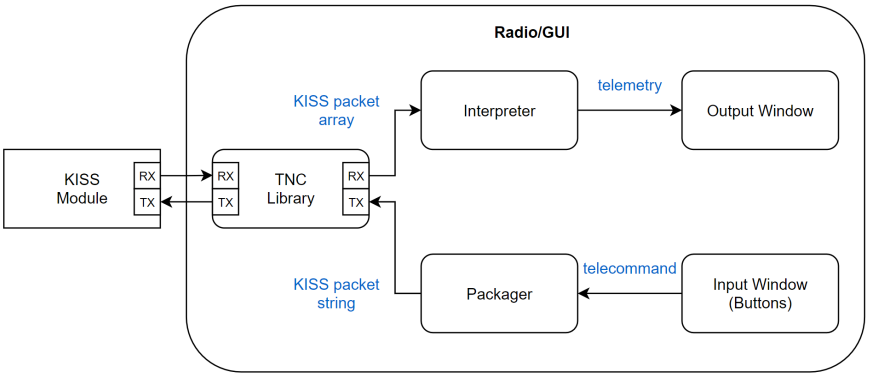

Figure 8: The basic architecture of the D3 Ground Station program. The main GUI is always running and all other processes are handled by threads.

gramming language for the ground station based on the availability of GUI libraries. ‘

The ground station software was specifically designed to mirror the Onboard Radio software. It consists of a *TNC* library, an *Interpreter*, *Packager*, and *GUI* nodes. Of these nodes, both the *Interpreter* and the *Packager* imitate those found in the onboard radio (requiring only lexical changes from C++ to Python and protocol changes from Simple Protocol to KISS). The *TNC* library also mirrors the functionality of the onboard radio's *UHF_Transceiver* library.

The biggest difference between the onboard radio and the ground station radio is the command data handling process. Unlike the CubeSat's radio, which is a fully autonomous software loop, the ground station radio was designed to require a user to both select the outbound telecommand and interpret the incoming telemetry (i.e., user-in-the-loop). The user-in-the-loop requirement is responsible for the majority of the differences between the two programs.

There are two layers of security between a user and the CubeSat. The first is the Windows login, and the second is the D3 Ground Station software authentication window. Once a user has gained access to the main GUI, they will be presented with a list of buttons, each of which corresponds to a telecommand. When one is selected, the user is prompter to confirm their input (Fig. 9). The software then switches to a listening state, where the interpreter scans for an input from the TNC. When the interpreter receives an input, all of the data is presented in an output window. If no input is received for a user-determined number of seconds, the program times out and attempts to re-transmit. If the re-transmit is unsuccessful, the program escapes from the transmit/listen sequence and reports an error message.

If a file is to be transmitted, the user is provided with a file browser to find the file. The user is then prompted to find the destination of the file in a simulated directory of the onboard computer. The file is then transmitted packet by packet. If an acknowledge signal is not received after a packet, the software retries sending the packet until it either receives an acknowledge signal or a timeout exception is thrown. The file downlink procedure is identical from the perspective of the CubeSat. All downlinked data is also checked to ensure that it complies with the *Downlink File* packet structure.

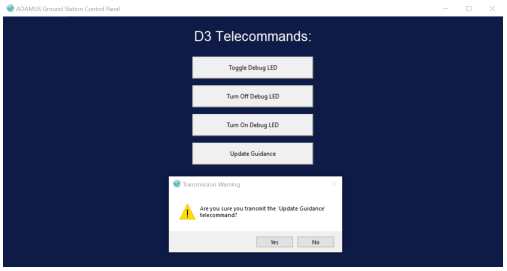

Figure 9: The D3 Ground Station Software in use. A button was clicked, and the program responded with a confirmation prompt.

Testing

The field tests documented below have been conducted with the D3 mission's hardware.

Ground Station

To test the transmission of a telecommand from the ground station, a spectrum analyzer was con-

nected to the output of the iCOM-910 transceiver. The D3 Ground Station software was launched and telecommands were sent through the TNC and to the iCOM-9100 transceiver. The transmit test was successful, and a power spike at the center frequency of 437.08 MHz was observed (Fig. 10).

Figure 10: Testing the ground station transceiver at a secondary ground station. The ground station used for testing utilizes an iCOM-910 (the previous generation).

The ground station software also logs all of the outbound telecommands, which were used to test the onboard radio.

Onboard Radio

To test interpreting and transmitting data from the onboard radio, the output of the URTX Transceiver was connected to a spectrum analyzer. A 25 dB attenuator was connected in series between the transceiver and the spectrum analyzer to reduce the power of the signal. The telecommands from the ground station output log were fed into the *Interpreter* class. Both file uploads and standard telecommands commands were tested. The onboard radio was able to accurately perform all of the required functionality and transmit the response (as evident by the power spike at the center frequency of 437.08 MHz).

Future steps for testing include testing the complete radio loop with the ground station. Testing the URTX Transceiver's ability to communicate with the ground station can be accomplished by (1) connecting the turnstile antenna system to the transceiver and deploying the antennas, (2) transmitting a telecommand from the ground station computer, and (3) interpreting the onboard computer's response on the ground station computer.

Conclusion

By segmenting the radio into *Interpreter*, *Handler*, and *Packager* classes, the onboard radio library can be easily ported to other CubeSat missions with minor architectural and simple protocol changes. This architecture allows the functionality of the radio to be invoked using a single command and operate independently from all other satellite functions. Implementing the radio as a separate software module also has the added benefit of reducing the development time of, testing time of, and risk to CubeSats that utilize it. The deliberate encapsulation of the radio framework also pervents unintended downlinks. If successful, this radio open-source framework has the potential to make CubeSats more accessible to a larger audience.

Acknowledgements

The author acknowledges the contributions of all members of ADAMUS Laboratory towards the D3 CubeSat mission. The contributions of Riccardo Bevilacqua, Ph.D., and Camilo Riano-Rios, Ph.D. in particular have been essential to the development of the radio.

References

[1] Martin Langer and Jasper Bouwmeester. Reliability of cubesats-statistical data, developers' beliefs and the way forward. 2016.

[2] Catherine Venturini, Barbara Braun, David Hinkley, and Greg Berg. Improving mission success of cubesats. 2018.

[3] David Guglielmo, Sanny Omar, Riccardo Bevilacqua, Laurence Fineberg, Justin Treptow, Bradley Poffenberger, and Yusef Johnson. Drag deorbit device: a new standard reentry actuator for cubesats. *Journal of Spacecraft and Rockets*, 56(1):129–145, 2019.

[4] Samuel Buckner, Carlos Carrasquillo, Marcus Elosegui, and Riccardo Bevilacqua. A novel approach to cubesat flight software development using robot operating system (ros). 2020.

[5] Alexander Kleinschrodt, Andreas Freimann, Steffen Christall, Maximilian Lankl, and Klaus Schilling. Advances in modulation and communication protocols for small satellite ground stations. 09 2017.

[6] Cape Peninsula University of Technology, F'SATI. *User Manual*, 2020. Rev. C.

Appendix

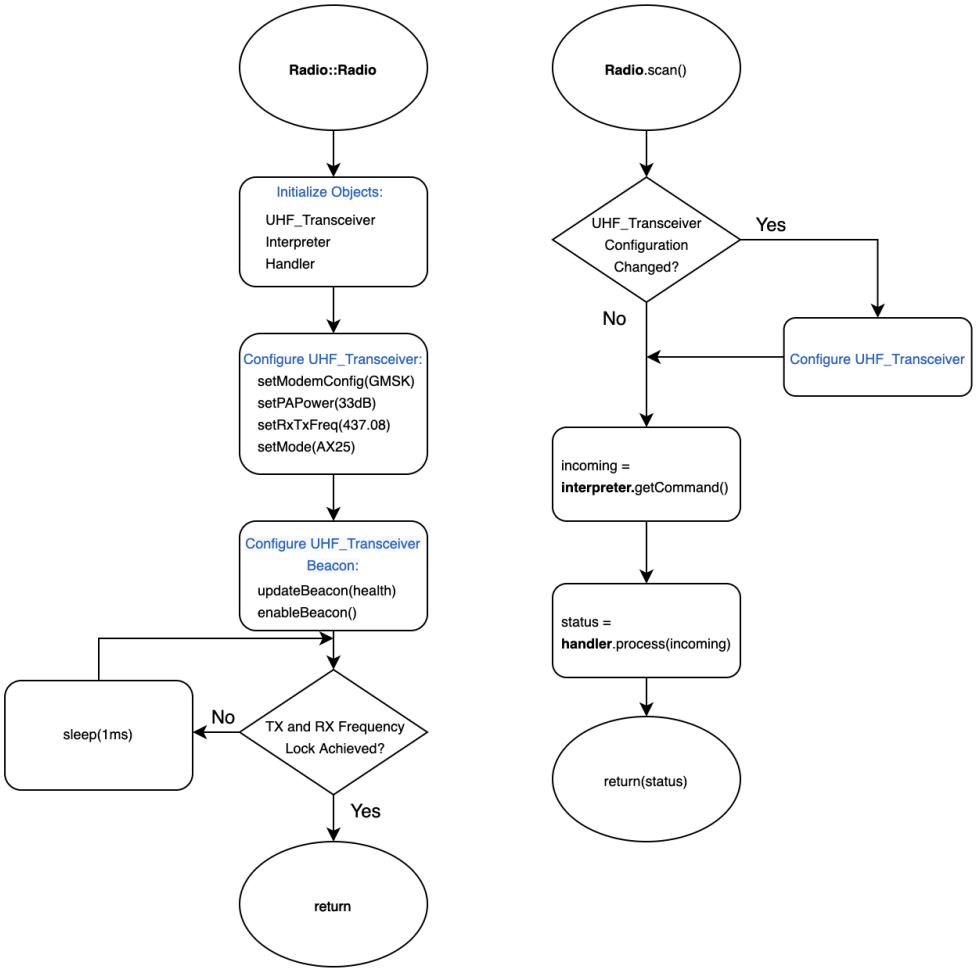

Figure 11: A simplified flowchart of the Onboard Radio's *Radio* class.

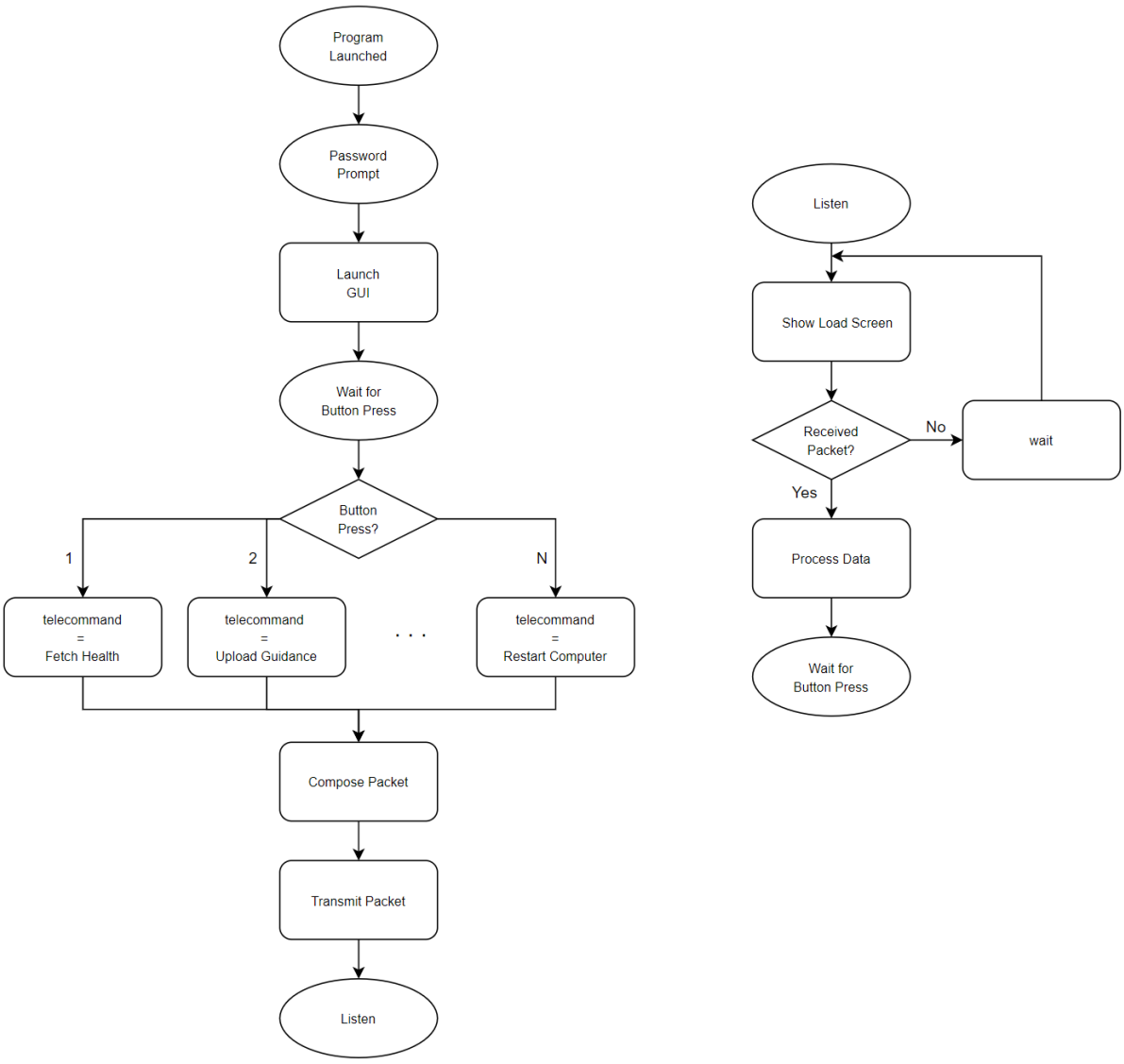

Figure 12: A simplified flowchart for the D3 Ground Station software.