Spring 2022

Autonomous Payload Design with Systems Engineering

Michael Downs
md204@uakron.edu

Christopher James Liebhart 2nd
cjl117@uakron.edu

Autonomous Payload Design with Systems Engineering

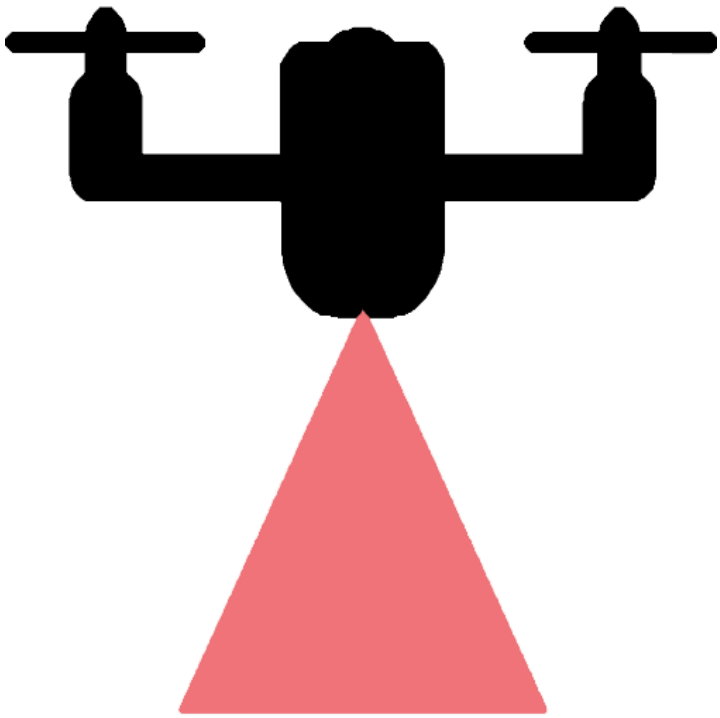
Michael Downs

Christopher Liebhart

Spring 2022

The University of Akron

Aerospace Systems Engineering & Honors Project



Contents

Abstract

As part of the Akronauts Spaceport Competition, the Autonomous Payload Design with Systems Engineering project encompasses three systems that together make up the 2022 payload. The three systems are a drone, payload bay, and ground control station. The mission for this payload is to deploy a drone from a grounded rocket, map the local terrain, and monitor the entire process through a ground station. The goal of this project is to accomplish the mission autonomously. Systems engineering, namely the many of its diagramming techniques, are used to document the design. Documentation is done in hopes of allowing future payloads to have a repository of knowledge or perhaps detailed failures to pull from when developing similar designs.

1. Introduction and Background

Previously, the Akronauts built a rover and flown an off-the-shelf drone for the payload system, so it seemed that a custom drone was the next logical choice if complexity and ambition were 'to be brought to the next level'. There are countless missions that a drone can perform, so giving it a purpose was relatively straightforward. Keeping with a space theme and operating under the notion of flying the drone in an unfamiliar, better yet, unexplored, location, it was decided that the drone would fly out of the grounded rocket and record data about the local terrain. The idea of a terrain map was thus conceptualized, and autonomous capabilities were added as a logical feature of the mission to facilitate such a goal.

The next problem to solve was the method of data acquisition. LiDAR was chosen as the sensor to meet this requirement due to its use in industry and its 'buzz-word' attraction.

Systems engineering diagramming techniques were used to document the project. See *Appendix Diagram* for the System Architecture diagram.

Images of the payload system are on the title page and in *Appendix Additional Images*. The three systems that make up the payload project are:

- The Drone
    - The drone, detailed below in the *Design* section, is a fully autonomous drone equipped with a LiDAR rangefinder that maps the ground it flies over. During flight, it records distance and attitude (angles of rotation) and sends them over to a ground station for processing.
- The Ground Control Station
    - The Ground Control Station is a touch screen user interface run on a Raspberry Pi Model 3 B+. It is developed with JavaScript and HTML. It provides monitoring and control of the drone. Its main function is to start the mission and graph the data the drone broadcasts during flight.
- The Payload Bay
    - The payload bay is the section of the rocket that houses the drone during flight and deploys it once landed. It consists of three main modules:
        - Rotational Module
            - The rotation module allows for the payload bay to rotate post flight to make sure the drone can be deployed upwards. It is made up of two identical bearing assemblies and a stepper motor.
        - Bay Door Module

- The bay door module provides support for holding the drone in place during flight and automatically opens to deploy the drone once the rocket has landed.

- Structural Reinforcement Module
  - Because of the cutout of the doors, the airframe lost structural integrity and upon testing failed at a low loading. The structural reinforcement module sought to fix this. It is made up of aluminum bulkheads and steel rods as a skeleton for the airframe.

2. Design

2.1 Drone

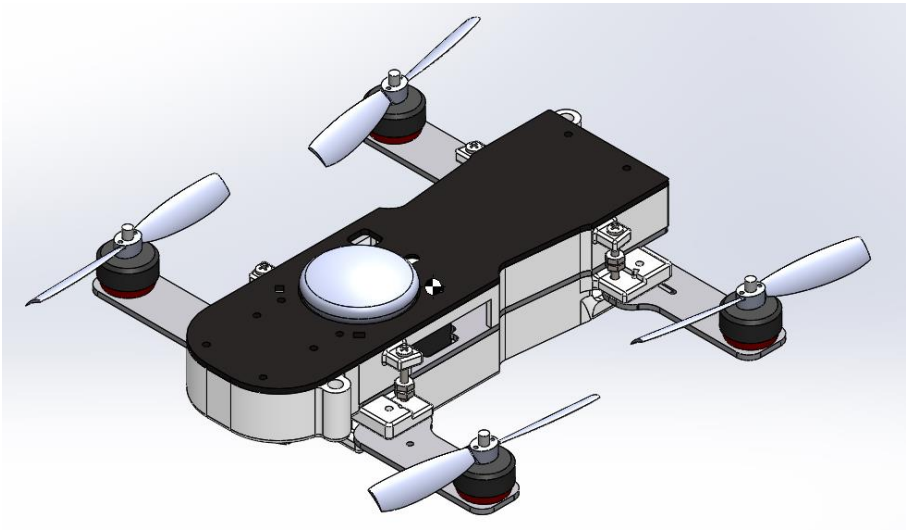


Figure 1 Drone Model

2.1.1 Components

The basic details of the drone are:

- Quadcopter with X-Frame orientation.
- 3D printed Frame from carbon fiber infused nylon [1].
- Pixhawk flight controller running Ardupilot [2].

- Controlled from the Ground Control Station (GCS) using dronekit python API [3].
- Scans terrain using attitude, GPS, and LiDAR readings.

For a full list of components see *Appendix: Drone List of Parts.*

2.1.2 Frame Design

The frame of the drone was constrained to the size of the rocket body tube (airframe). This required that the arms be foldable. The arms were spring loaded and would fold out once deployed.

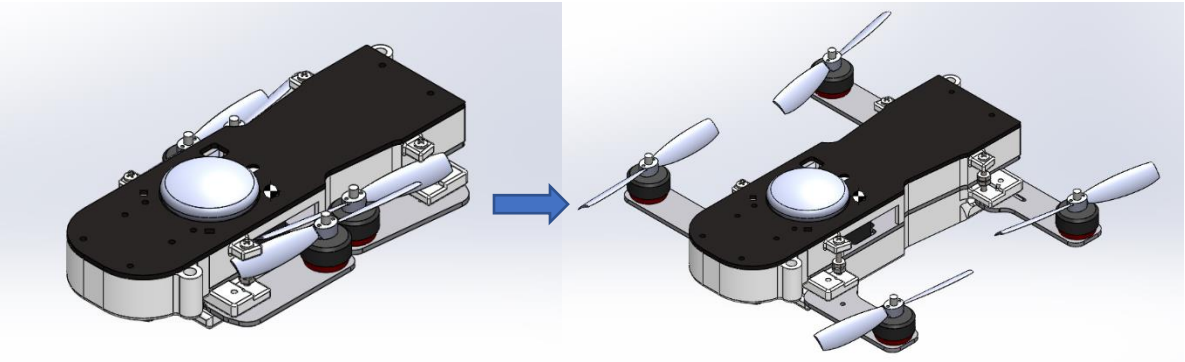


Figure 2 Arm Deployment

To 3D print a frame with that capability, the carbon fiber infused nylon was used for module arms. The components were kept on two layers, the battery, GPS, and receivers on top with the Pixhawk and ESC on the bottom. The LiDAR was placed at the front facing down.

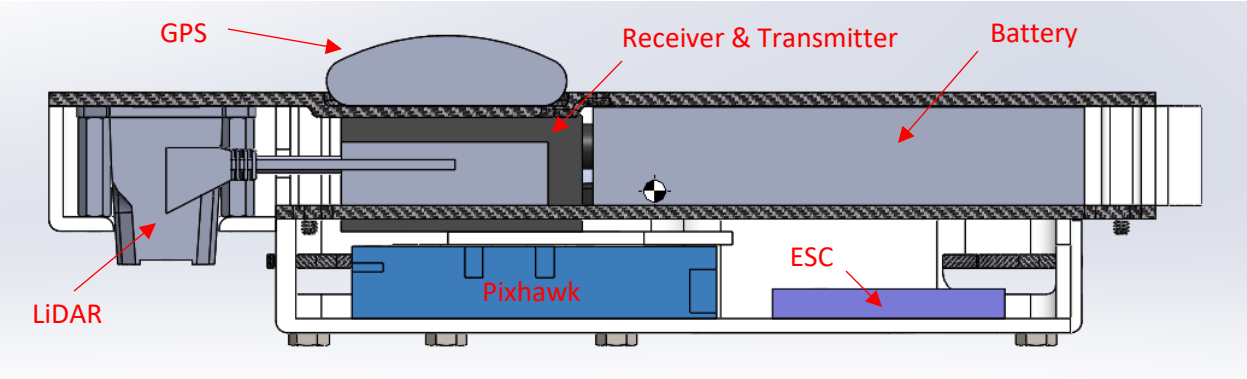


Figure 3 Drone Components

2.1.3 Mission Details

Once the payload bay rotates and opens with the drone upright, the Ground Control Station (GCS) will connect and upload the desired mission. From its retention in the payload bay, the drone will then fly to a specified height, which is around 10ft. above the rocket, in guided mode [4]. From there it will begin and complete the mission in auto mode [4], during which the terrain data will be streamed back to the Ground Control Station. After the mission completes the drone will switch to return to launch (RTL) flight mode [4] and land, where it will then disarm. The mission is considered successfully completed at this point.

A mission is a set of destinations called waypoints grouped in a particular pattern. The waypoints represent the latitude, longitude, and altitude that the drone must fly to. There are several missions that the Ground Control Station supports. The following are missions used for testing purposes, as they are short compared to the final mission that the drone will fly:

- Takeoff and Land
    - This mission simply has the drone takeoff, hover for a few seconds, then land.
- Forward and Backward
    - This mission simply has the drone take off, fly a specified distance forward, then that same distance back to the point of launch where it then lands.
- Square Mission
    - This mission has the drone fly to four waypoints representing a square centered at the point of launch.
- Simple Go To
    - This mission has the drone fly short distances to random waypoints around the point of launch.

The final mission at the time of this report has the drone fly a grid back and forth across a rectangle of specified length and width centered at the point of launch. This mission is called the Rectangle Mission.

2.1.4 Data Acquisition and Streaming

The drone uses four main sources of data to map the terrain:

- The LiDAR sensor. The LiDAR sensor is used to measure depth.
- The IMU internal to the Pixhawk. The IMU is used to transform the depth reading into plottable coordinates.
- The Barometer internal to the Pixhawk. The Barometer is used as a reference for the depth reading (i.e., the depth reading is relative to the height of the drone).
- The GPS. The GPS is used to record the location relative from the start point in terms of distance traveled North and East, which are later just considered the X and Y global positions.

The transmission of data between the Pixhawk and Ground Control Station is done through two transceivers operating at 433 Mhz. The Pixhawk and Ground Control Station communicate through those transceivers with a protocol called mavlink [5].

2.1.5 Data Processing

Figure 4 Data Acquisition and Axis shown below is a graphical representation of the data acquisition and axis system. Not shown are the angles of rotation and Distance North (into the page):

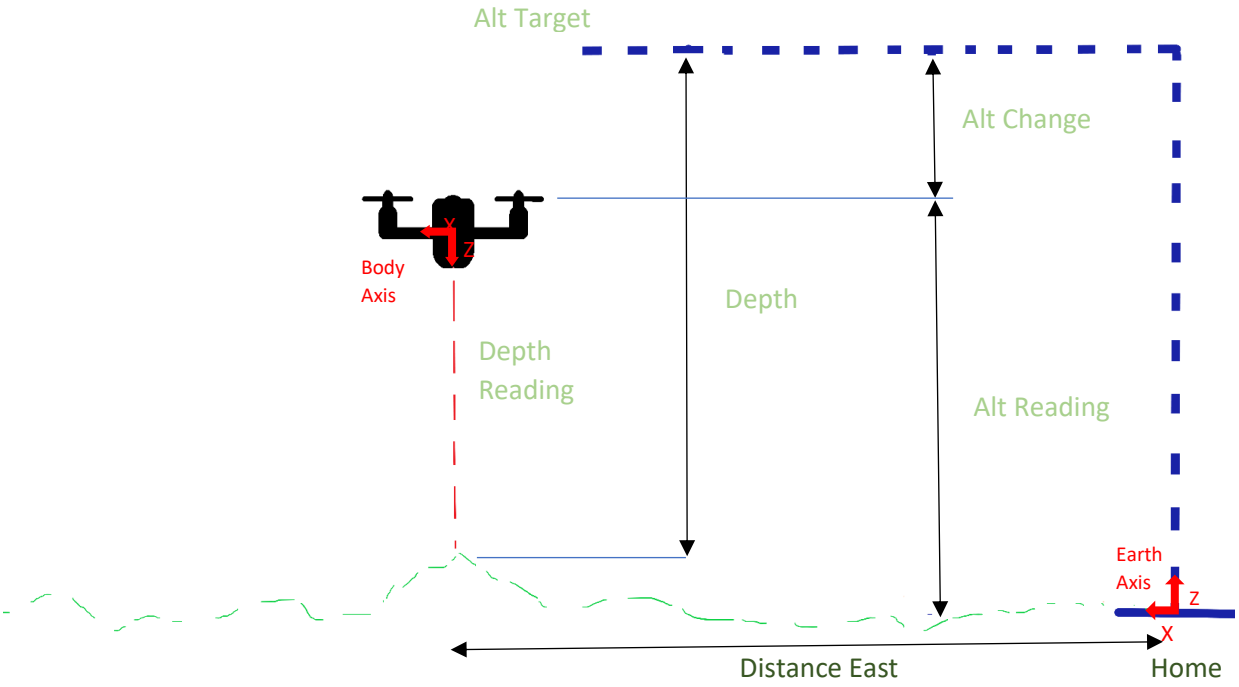


Figure 4 Data Acquisition and Axis

The sequence of the data processing begins with the LiDAR point on the body axis:

$$P_{\text{Body}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Body}} = \begin{bmatrix} 0 \\ 0 \\ \mathbb{R} \end{bmatrix}$$

Where $\mathbb{R}$ is the LiDAR distance reading. To transfer the data to the earth axis so it can be plotted, it must undergo transformations through the roll (Φ), pitch (Θ), and yaw (Ψ) axis. This is done by using the following transformation matrices R1 (Φ), R2 (Θ), and R3 (Ψ) [6]:

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{bmatrix}$$
$$R_2 = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{bmatrix}$$
$$R_3 = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

These matrices describe the rotation of a body through the three angles from the earth axis to the body axis:

$$P_{\text{Body}} = R_1 R_2 R_3 P_{\text{Earth}}$$

However, since the body axis point is known, the transformation must go back through those angles to determine the earth axis point. This is done by transposing the transformation matrices:

$$P_{\text{Earth}} = {R^T}_3 \, {R^T}_2 \, {R^T}_1 \, P_{\text{Body}}$$

After the transformation, the data point from the LiDAR is now relative to the earth. However, it is still offset from the home location by the distance the drone has travelled North and East, and also not relative to any specific altitude. The GPS data gives distance traveled North and East relative to the home location, so this can be directly added to the point to correctly calculate the global X and Y positions. For adding a reference for the depth reading, consider the situation where the drone reads a distance with the LiDAR sensor at a specific altitude. Then the drone decreases its altitude an arbitrary amount and reads the same data point. Even though the same spot in space is recorded by the drone, the distances will be different. To account for this, the drone must consider its altitude. A target altitude that the drone will attempt to stay at is set at the beginning of the mission and used as the reference for

the LiDAR data. It can find the change in altitude by subtracting the current altitude reading from the target attitude. It can then add the change in altitude to the LiDAR point to obtain the true Z or true depth. The full equation that converts the LiDAR data into a global 3D point is thus:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Earth}} = \begin{bmatrix} D_N \\ D_E \\ \text{Alt}_C \end{bmatrix} + R^T{}_3 \, R^T{}_2 \, R^T{}_1 \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Body}}$$

where

$D_N = \text{Distance North}$

$D_E = \text{Distance Earth}$

$\text{Alt}_C = \text{Altitdue Change} = \text{Alt}_T - \text{Alt}_R$

$\text{Alt}_T = \text{Altidue Target}$

$\text{Alt}_R = \text{Altitude Reading}$

After converting to global coordinates, the data point can be plotted.

2.2 Ground Control Station

The Ground Control Station (GCS) is a touch screen user interface run on a Raspberry Pi computer. It is built on Node.js using JavaScript and HTML to control and style the front end while using python as the backend interface to the drone. See *Appendix Diagrams* for Functional and Package software diagrams describing the interfaces and data flow.

2.2.1 User Interface

The frontend of the GCS provides three pages, each controlling a different part of the payload. The three pages are the Simulation page, GCS page, and the Deployment page. The controls are sized to reduce the amount of information that the user sees during operation. This is to reduce the possibility of erroneous commands, such as selecting the wrong control or reacting too slow. The color coding used for the controls is done to emphasize the consequence of using them and the situations where they should be used. For instance, start and stop controls throughout the program are green and red to give the user an implicit understanding of their purpose. This becomes crucial during the situations where immediate action is required by the user to stop the drone. Panic is often the response of inexperienced pilots and following ‘gut reactions’, such as clicking the ‘big red button’ to stop the drone, is ideal to reinforce.

- Simulation page

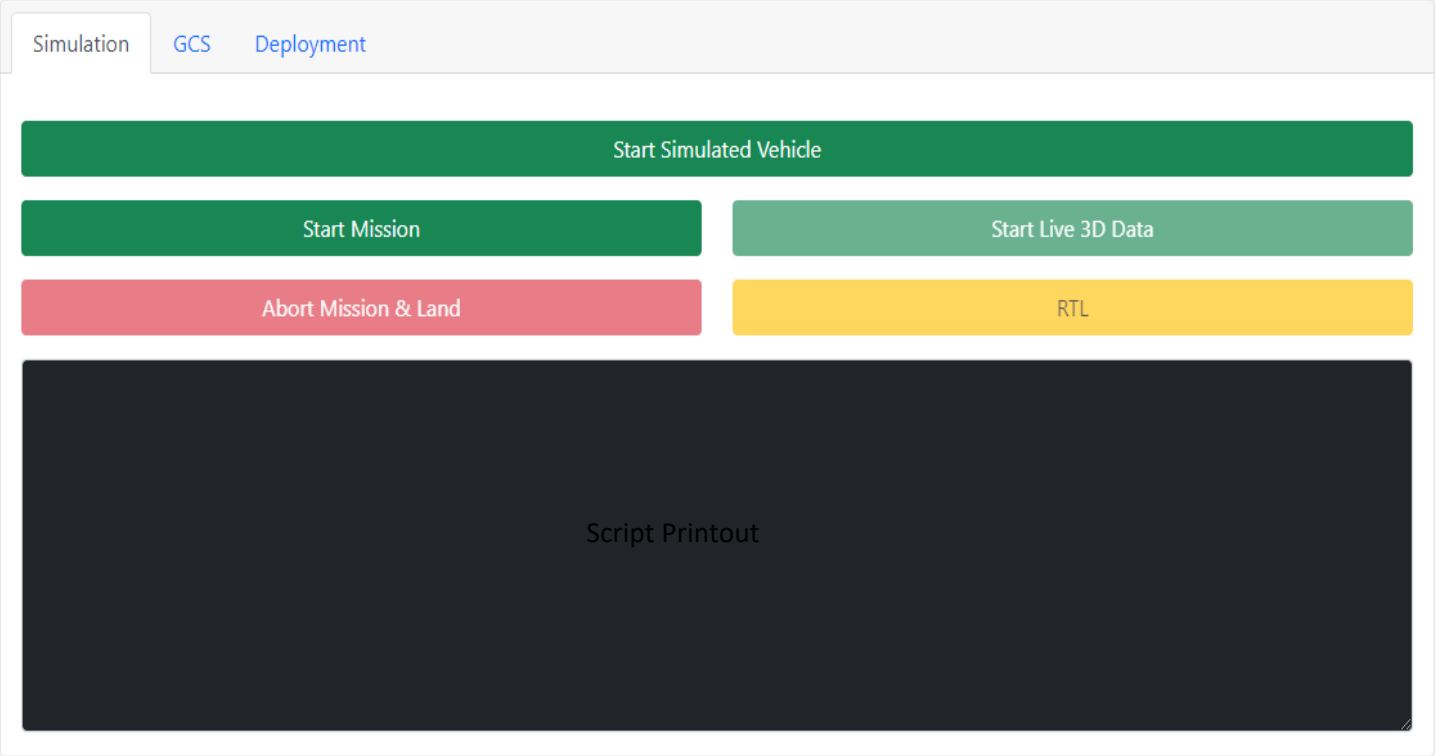


Figure 5 Simulation Page

- The simulation page allows the developer to test the mission code and data streaming before flying it on the drone. It is essentially a duplicate of the GCS page (where the actual drone is controlled). It provides four controls common to the GCS page and a fifth which starts the simulated vehicle. The common four controls are *Start Mission*, *Abort Mission & Land*, *RTL* (return to launch), and *Start Live 3D Data*.
- Along with the controls, the simulation page provides python script printouts and a live data chart that is populated with data points during a mission. The python script printout is basically a replication of what the script output would be if the script were run from a command line console. It displays messages and errors from the python script by logging the standard output and standard error buffers.

- Unfortunately, the LiDAR rangefinder was not supported with the simulation program at the time of the project so the data sent to the graph was not exactly the same as what would go through the GCS page, but it was sufficient for testing the graphing capabilities and data transfer. Below is a sample of the live graph during a simulated mission:

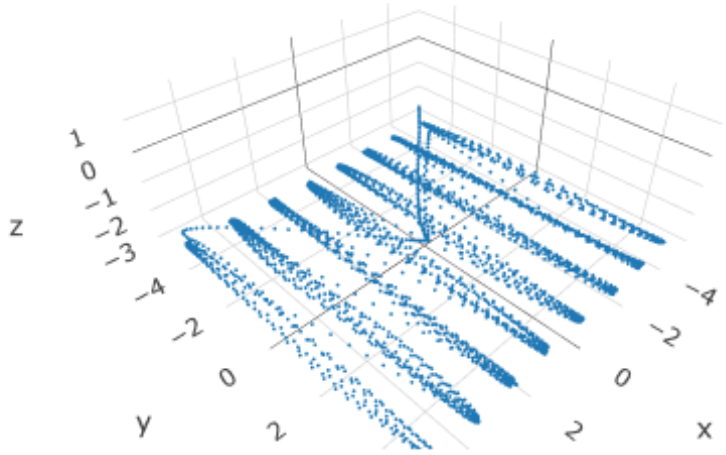


Figure 6 Example Simulation Live Graph

- GCS page

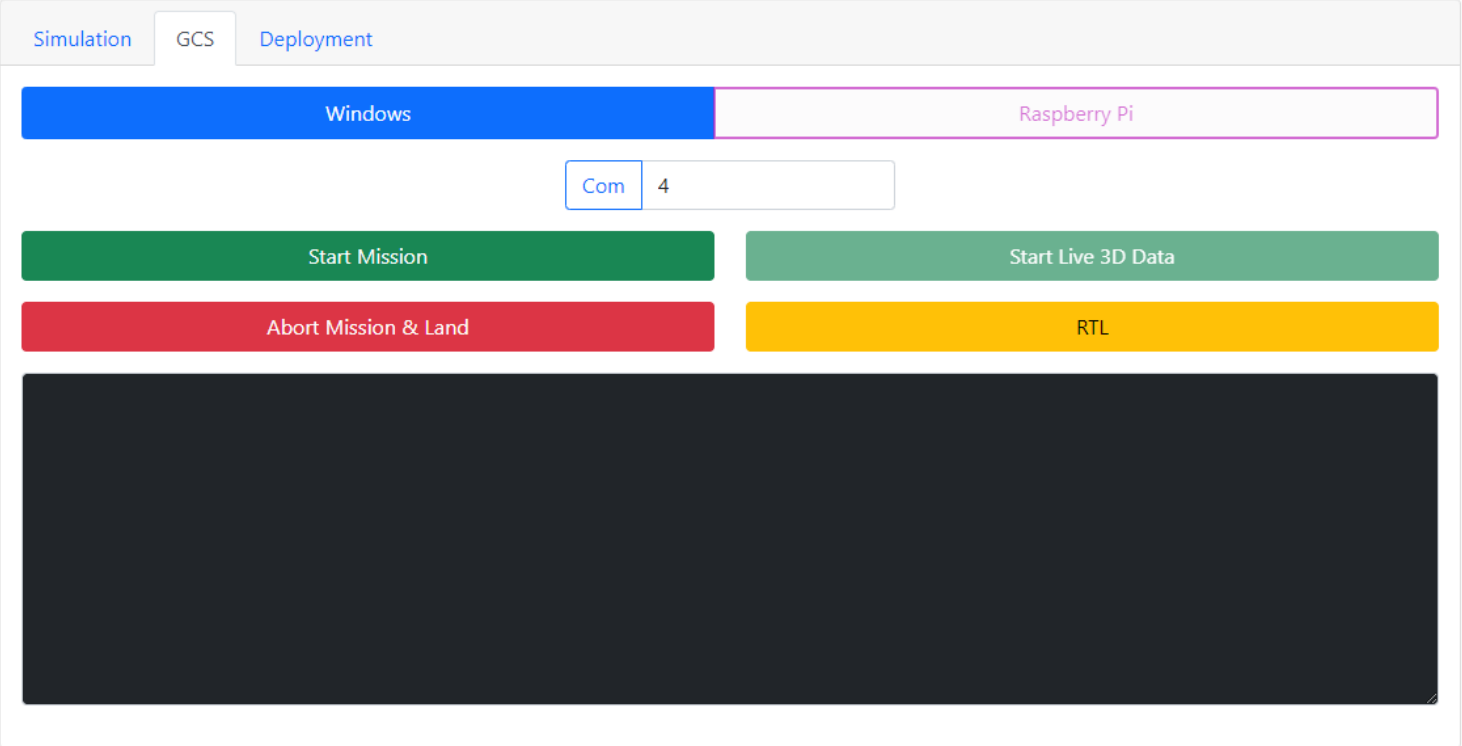


Figure 7 GCS Page

- GCS page controls and monitors the actual drone; thus, it was given the same abbreviated name as the entire system (Ground Control Station). Almost identical to the simulation page it provides the same four common controls; start, abort and land, return to launch, and live stream the data. It also provides the live data graph below the python script printout.
- It additionally allows the user to select between operating systems due to the different naming conventions for USB ports ('com' vs 'usb').
- At the writing of this report, no data graphs have been successfully recorded.

- Deployment Page

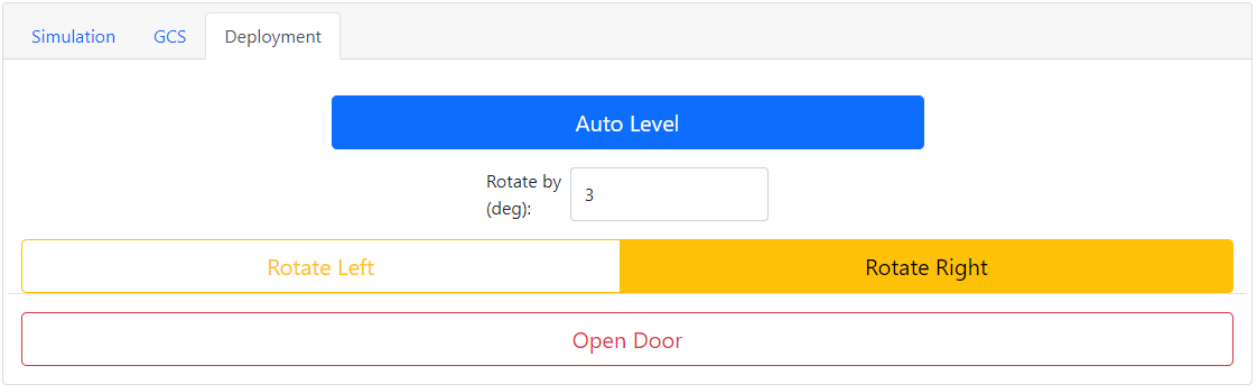


Figure 8 Deployment Page

- The Deployment Page allows for the manual and autonomous control of the payload bay by the user. This is done after the rocket has landed. Provided controls are the *Auto Level*, *Rotate Left*, *Rotate Right*, and *Open Door*. The *Auto Level* will autonomously level the payload bay so that the drone is facing upwards. The *Rotate Left* and *Rotate Right* will allow the manual rotation of the bay by the user. Clicking each of those buttons will rotate the airframe in that direction by the number of degrees specified in the *Rotate by (deg)* input box. The *Open Door* control will simply open the door of the bay. The drone is deployed after the door opens and takes off.

Additionally, the user interface provides a mission select menu. This is so the user can select what mission is to be run. Each mission is represented by its respective python file. This is located above the tabs to allow for global use in the program.

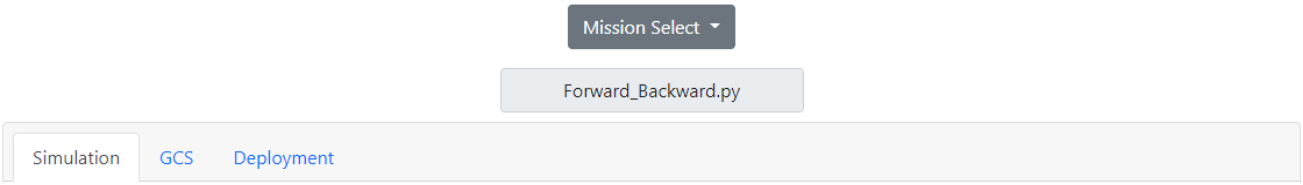


Figure 9 Mission Menu

2.2.2 Frontend Coding

As stated, HTML and JavaScript run on Node.js to style and control the frontend user interface of the Ground Control Station. There were many options for developing a user interface at the start of the project, such as Tkinter and Qt to name a few, but the experience developing single page web applications using JavaScript was already present. It also provides the shortest learning curve compared to other user interface options which is imperative to keep the interest of new team members and allow for continued development after the project has been completed.

To aid in the styling and functionality of the design, a styling package for web-like applications called Bootstrap [7] is used. This allows for quick, simple, and self-explanatory controls to be built. Buttons are the main style of control and the simple color scheme that Bootstrap supplies by default is sufficient to obtain the desired functionality and conformity to the user bias mentioned above (red means ‘danger’, green means ‘success, etc.).

For the 3D plotting functionality, a JavaScript package called Plotly.js is used. This provided the basic 3D point plotting capabilities sufficient to show a live data graph. Currently it is only a point plot displayed on the GCS and Simulation tabs, but further development is planned for surface and mesh plots.

2.2.3 Backend Coding

Node.js [8] is the backend engine that compiles and manages the entire program. Without getting into irrelevant details, Node.js is a platform that runs JavaScript outside a browser. To make the Ground Control Station a desktop application, a Node.js package called Electron [9] is used. Electron allows for native (meaning applications run on the current operating system) application development and cross-platform functionality needed for working with both Windows and Raspbian (Raspberry Pi OS).

Python is used as the interface to the drone. It uses a package called dronekit [3] that abstracts away most of the more difficult inner workings of the communication protocol package pymavlink [10]. NumPy [11] is used to process the data from the sensors and convert into plottable points. Another Node.js package is used for handling python scripts, called python-shell [12]. It handles running the scripts and any data transfer between the frontend and python.

Python also handles the simulation program. It creates what is referred to as a simulated vehicle that is hosted on one of the computer's internal network ports to which the code can connect in the same manner it can with the actual drone. The package that provides this functionality is part of the dronekit Simulation in the Loop program [13]. It provides several different vehicles such as a rover and airplane, but for this project the copter simulation is used.

Several python mission scripts exist to test the drone functionality. The main files are the rectangle mission, takeoff and land, and the forward and backward mission.

Essentially the python mission scripts follow the same steps:

- Connect to a vehicle
- Upload mission commands (or sometimes just called waypoints)
- Switch flight modes to accomplish arming, taking off to a specific altitude, and starting the autonomous mission.
- Process data and print 3D points to the standard output to be received by the JavaScript frontend for plotting.
- Monitor any data coming from the frontend, such as the *Abort Mission and Land* and *RTL* commands. The mission code then forwards the command to the drone, monitors the drone status, and disconnects once the command is finished.

2.3 Payload Bay

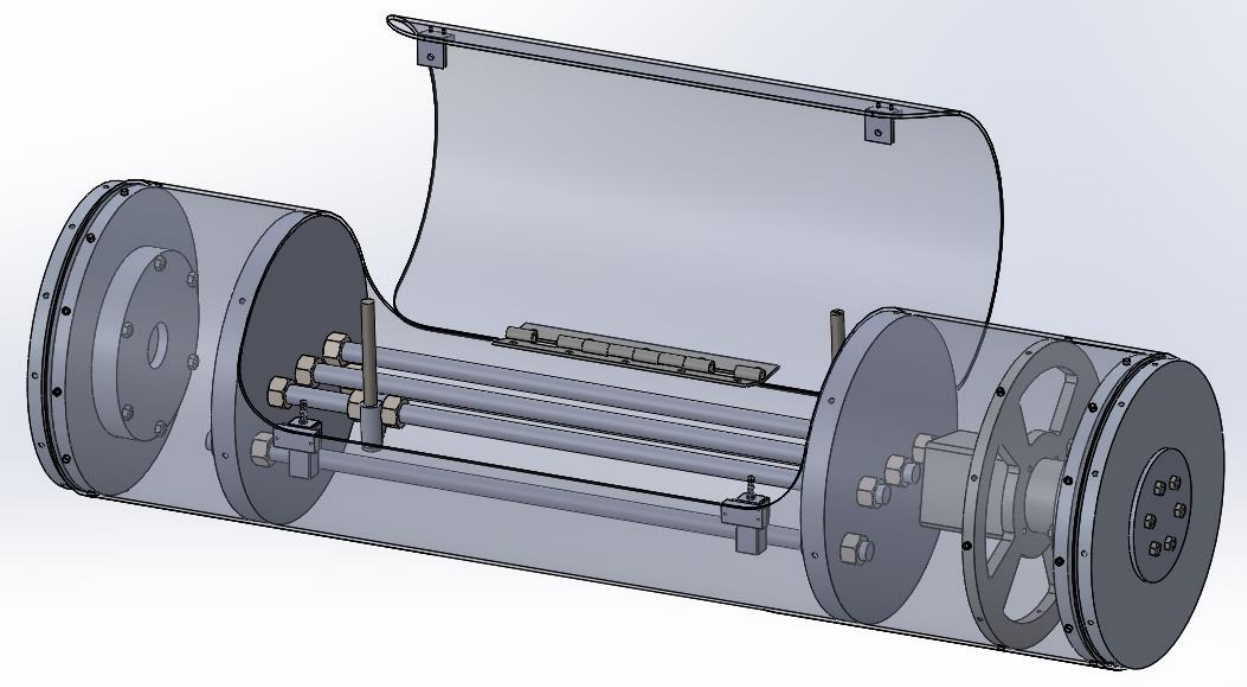


Figure 10 Payload Bay

The design of the payload bay must satisfy three criteria. First, the bay must house and autonomously release the payload; the payload being the previously mentioned drone. The system responsible is the Bay Door Module. Second, the bay must be self-leveling so that when the rocket lands the payload is able to successfully release in the upright position. The system responsible is the Rotational Module. Finally, the design must withstand the forces present during the rocket's launch, descent, and landing. The system responsible is the Structural Reinforcement Module.

2.3.1 Drone Housing and Release (Bay Door Module)

To satisfy the first criteria it was decided that a door with spring loaded hinges would be the best approach to the problem of releasing the drone. The original door design involved a twin door

system that would mate the doors with a pin. When necessary, the connection would disengage, freeing both doors to swing open.

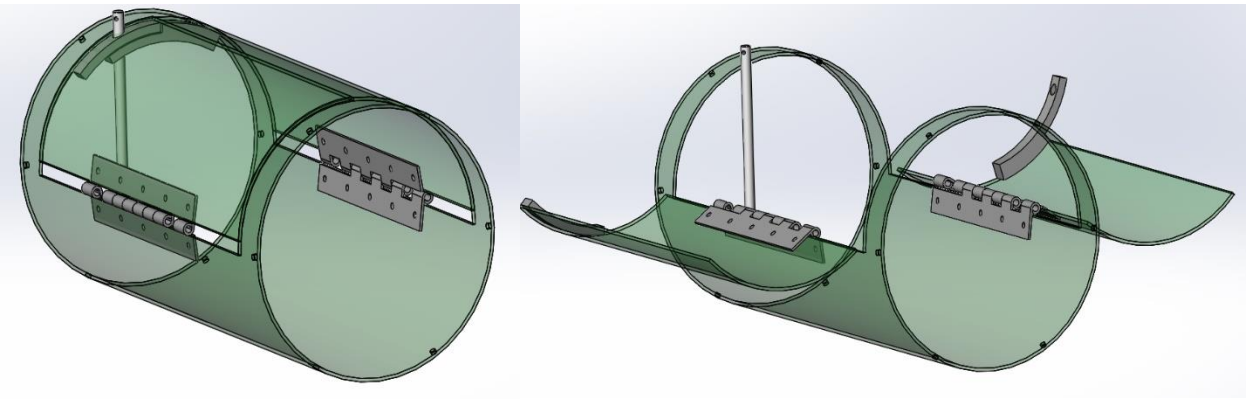


Figure 11 Original Door Design

It was decided that this design had too many moving parts, leaving too much room for error. A similar yet simpler design was then implemented with a single door. The single door would close and lock with the same pin system as the twin door design. The pin would connect the door to the inside wall of the rocket. When the payload bay needed to be opened the pin would rotate and disengage the lock to allow the door to swing freely, opening the bay for the drone to fly out.

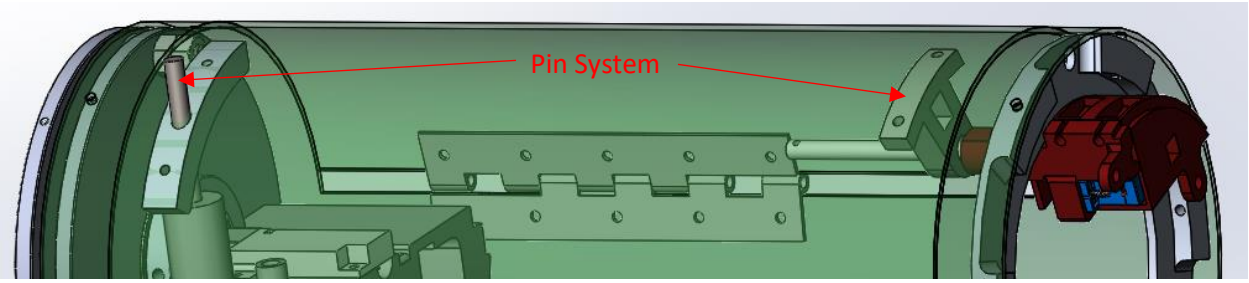


Figure 12 Pin System

Machined and 3D printed metal parts were used for the single door and pin system. However, the tolerances were such that it was nearly impossible to provide a tight fit while at the same time low enough friction for the pins to move in and out of the locking components. Thus, it was decided to take a third approach shown below in *Figure 13 Bay Door Module*.

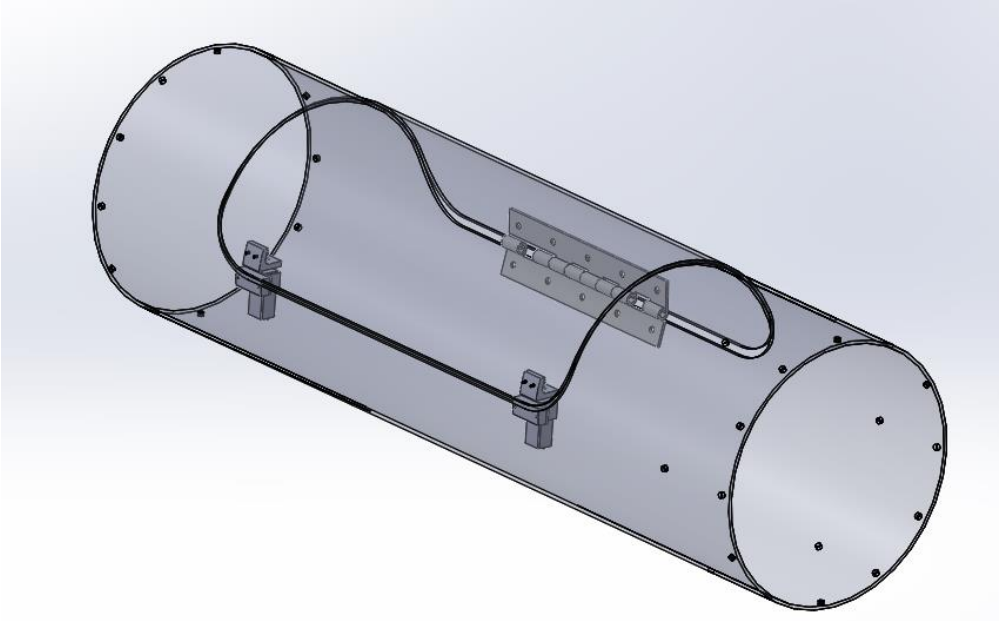


Figure 13 Bay Door Module

Instead of servo actuated pins, this design used DC motors to screw lead rods into two metal brackets, one on the door and one on the airframe as shown in *Figure 14 Door Retention*. The rods and brackets were able to be compressed and actuated by the motors, providing the tight fit and mobility that the pin system could not.

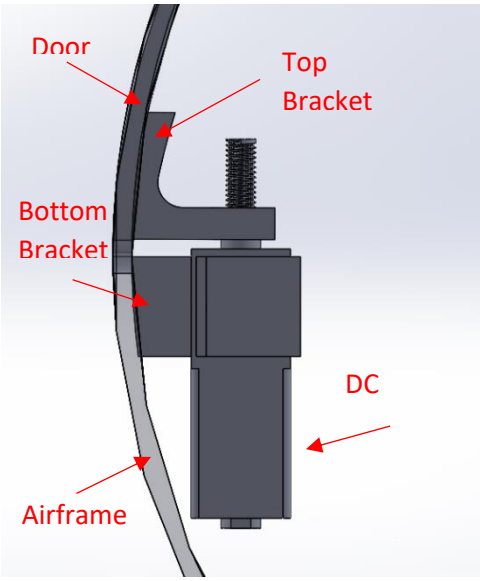


Figure 14 Door Retention

2.3.2 Self-Leveling (Rotation Module)

Once the rocket lands, the system uses an Arduino Nano IoT to self-level based on readings from an IMU. A stepper motor and a custom bearing assembly are used to provide the desired functionality. The motor and two of the bearing assemblies on either side of the payload bay make up the rotational module.

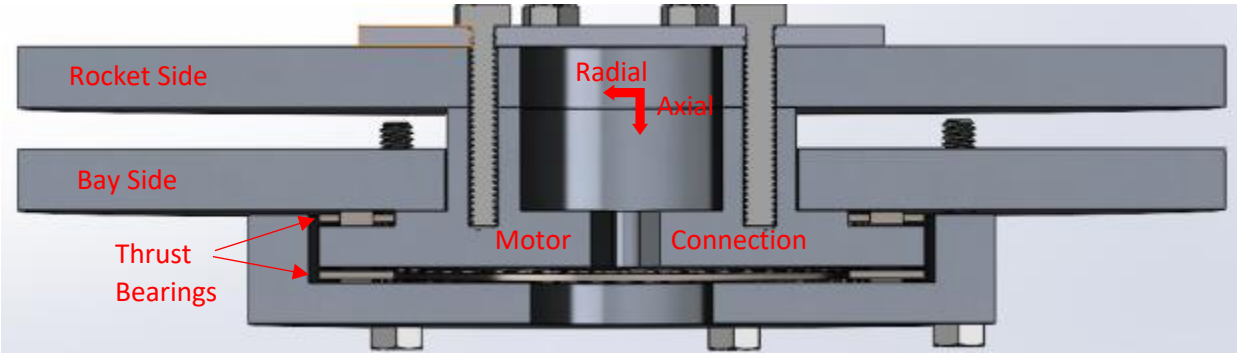


Figure 15 Bearing Assembly

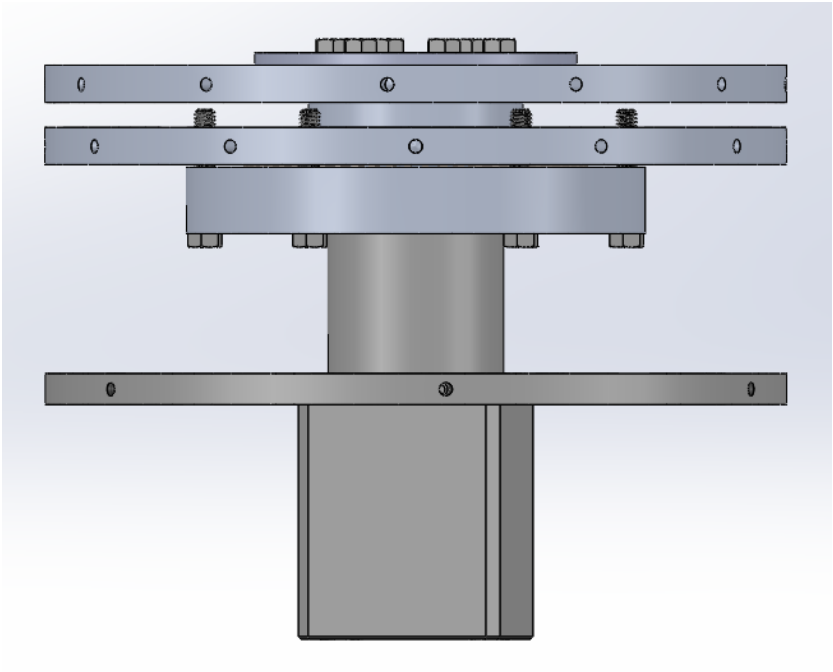


Figure 16 Rotational Module

The rocket side is the part of the module that connects to the airframe of the rest of the rocket. The bay side is the part of the module that connects to the payload bay. The idea behind the design is

that there can be no translational movement radially or axially. The only movement can be rotationally around the axial direction. To accomplish this, the bolts compress the cap of the bay side onto the joint of the rocket side with two thrust bearings between the surfaces. This is to make sure no translational movement is possible while still providing a low friction surface.

As stated, two identical bearing assemblies are used on either side of the bay. The stepper motor is fixed to only one of the bearing assemblies. It attaches at the motor connection in the bearing assembly and to a stationary bulkhead within the bay.

2.3.3 Structural Reinforcement Module

The entire bay structure had to be capable of withstanding the forces experienced during flight while also allowing the payload bay to perform the rest of its functions, such as providing a door. The bay was originally designed without any reinforcement and a simple tensile test showed it was structurally unsound. To reinforce the airframe, four ¼" steel rods were added. The rods were fixed to either side of the bay on the aluminum ¼" bulkheads.

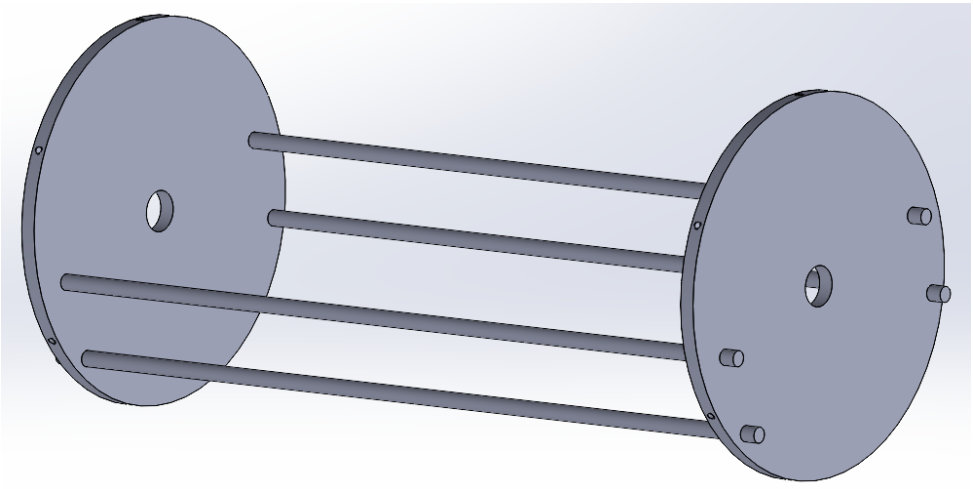


Figure 17 Old Structural Reinforcement

However, because of how the rotational module connected to those bulkheads, the forces applied during flight would be concentrated on a very small area at the center of the bulkheads. The area of applied force was approximated to be only .36” based on the rotational module geometry. Using the FEA (Finite Element Analysis) program Abaqus it was determined that the stress would exceed the yield strength of the aluminum bulkheads, and that the deformation was large enough to destroy the bearing assembly (see *Appendix Abaqus Tensile Test Images*). Upon a physical tensile test, the structure did deform, so much so that the bulkheads reached plastic deformation (see *Appendix Tensile Test Images*). It was concluded that the design was unsuitable for flight.

To improve the design, thicker rods (3/8”) and thicker plates (1/2”) were used. Additionally, a new pattern was followed for the placement of the rods. This second design showed significant improvement in similar Abaqus trials. Once the design produced acceptable values in FEA the structure was built and then physically tested in a tensile test. The physical test produced similar results as the FEA analysis and was determined suitable for flight (see *Appendix Tensile Test Images* and *Appendix Abaqus Tensile Test Images*).

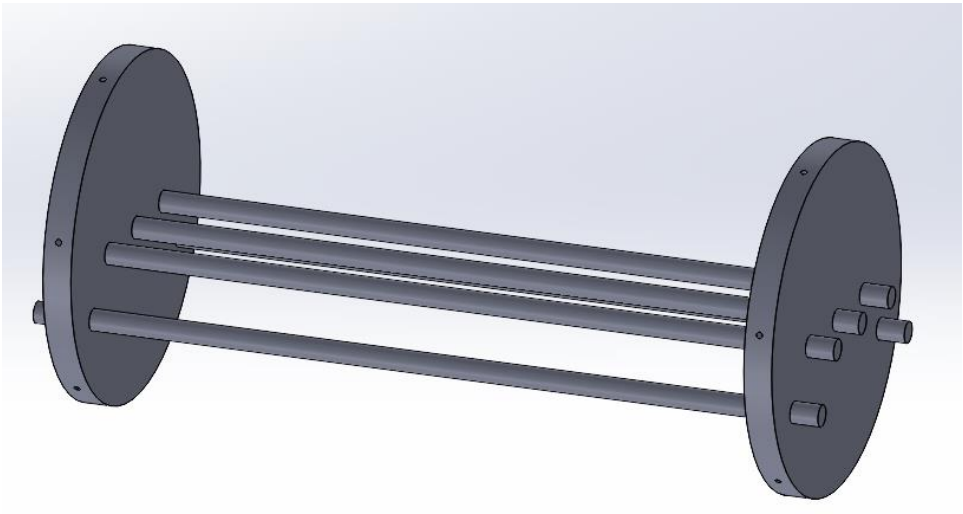


Figure 18 Structural Reinforcement

2.3.4 Bay Electronics and Communication to Ground Control Station

For a full list of electronic components in the payload bay see *Appendix: Bay Electronics List of Parts*, but the most notable parts are the Arduino Nano IoT, HC-12 transmitter, stepper motor, and batteries. Except for the batteries and stepper motor, all the electronics are kept on a custom PCB (Printed Circuit Board).

The Arduino Nano IoT communicates with Ground Control Station through a HC-12 wireless serial transmitter. Rotation and deployment commands are sent through the JavaScript Serial Module to the Arduino running its version of C++.

For self-leveling, the Nano IoT uses an internal IMU and a custom sensor fusion algorithm that combines the readings from the internal gyroscope and accelerometer into a single value. The value determines the distance from level in terms of angle of rotation and is used to control the motor.

The PCB design and sensor fusion algorithm are not within the scope of this project yet are worth mentioning for completeness.

3. Conclusion

3.1 Future Development

The following additions will continue to be developed until the end of the project's timeframe in June 2022:

- Compressive and bending testing for the structural reinforcement.
    - The design did meet the criteria for tensile loading; however, it will need to be tested in compressive loading before being considered completely flight ready.
- Optimized mission code and complete Ground Control Station functionality.
    - There are several bugs in the mission code and Ground Control Station program, and they have not fully implemented all the features described in previous sections.
    - Live data streams have yet to be performed during an actual mission. Currently only the simulated vehicle can provide live graphs.
- Complete assembly of the final payload bay design.
    - Manual control and self-leveling capabilities will be show-cased, and the drone will start a mission from the bay in a demonstration program.

Basic autonomous functionality, data processing, and the user interface design will be developed to a state that they could be reused by future projects. This project and report will provide enough documentation to make re-use and development of similar, but better, systems possible.

3.2 Final Remarks

The project was started with a romantic idea of generating a live point cloud and surface map, but several caveats became apparent as development progressed. Designing the drone to come directly out of the airframe caused structural instability that proved difficult to remedy. The majority of the project's timeframe was spent solving this issue. This entailed many long hours of CAD modeling, FEA

simulation, manufacturing, and testing, none of which were envisioned. One of the main factors in attempting the autonomous, terrain mapping, and LiDAR heavy project was out of the interest of working with and developing the related software. Working in the more mechanically inclined structural discipline put strain on the desire to even finish the project when its purpose was to be heavily focused in the software realm. Consequently, the LiDAR and autonomous development were only started near the end of the project timeframe, severely limiting its potential.

The entire payload design suffered from many issues start to finish. Having the drone fly out of a door in the airframe was an inherently bad idea due to the structural issues but also because there was no testing or any sort of analysis done to ensure the door was adequately secured to the airframe. Wind tunnel testing may have provided some indication but even that would not come close to the speeds the rocket will experience during ascent. Other solutions to deploying the drone from the rocket should have been researched, or at the very least, the door design should have been conceptualized in more detail before moving on to the embodiment phase.

Besides the door design, the most unfortunate mistake was not researching how LiDAR is commonly used to map terrain before purchasing the sensor and designing the drone. The sensor chosen was 1D, stationary, and commonly used as a rangefinder for altitude and object avoidance purposes, not mass data collection.

There were several more issues with the project but the structural integrity, lack of testing the door retention, and the added weight of the reinforcement resulted in the decision not to fly the payload system in the rocket for the 2022 Spaceport competition. The payload bay will continue to be developed, but only for demonstration and showcase purposes.

4. Appendix

4.1 Drone List of Parts

- Pixhawk 1
- ESC - XRotor Micro 60A 4in1 BLHeli32 Dshot1200 6S
- 4 X 5045 5” Propellors
- GPS – Radiolink M8N SE100
- 4 X 4300 KV Racerstar Brushless Motors
- LIDAR-lite v3 Garmin
- Holybro Transceiver 433 MHz
- FrSky X8R Receiver 2.4 GHz
- 3D Printed Frame – Onyx Carbon Fiber infused Nylon
- 3 Cell 4300 mAh 11.1 V Battery

4.2 Bay Electronics List of Parts

- Arduino Nano 33 IoT
- HC-12 Wireless Serial Transmitter
- Barometer BME680
- IMU MP6500
- 100 µF Capacitor
- 220 Ohm Resistor
- 1N4148 Diode
- P2N2222A Transistor
- 2 X 6V 30RPM DC Motor with M3*55MM Threaded Screw
- NEMA 17 Stepper

- 4 X 3.7 V Lipo Batteries
- 5 V Regulator
- Assorted Terminal Blocks

4.3 Tensile Test Images

Figure 19 Tensile Test

Figure 20 Tensile Test Old Design

4.4 Abaqus Tensile Test Images

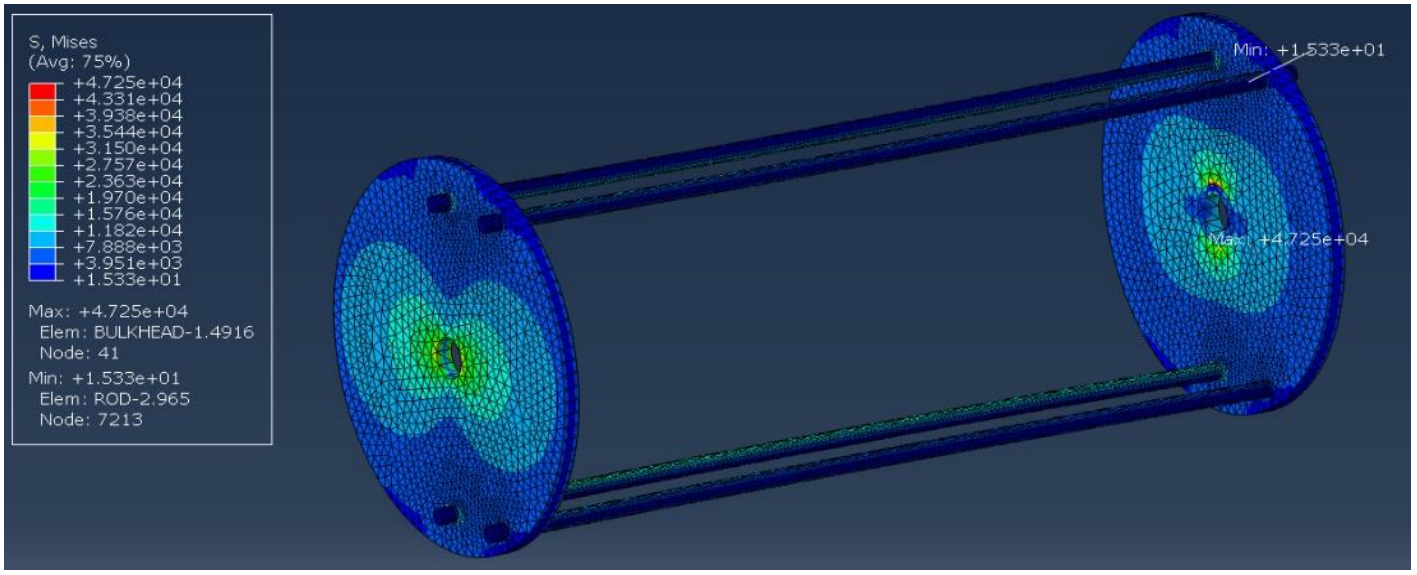


Figure 21 Stress Old Design

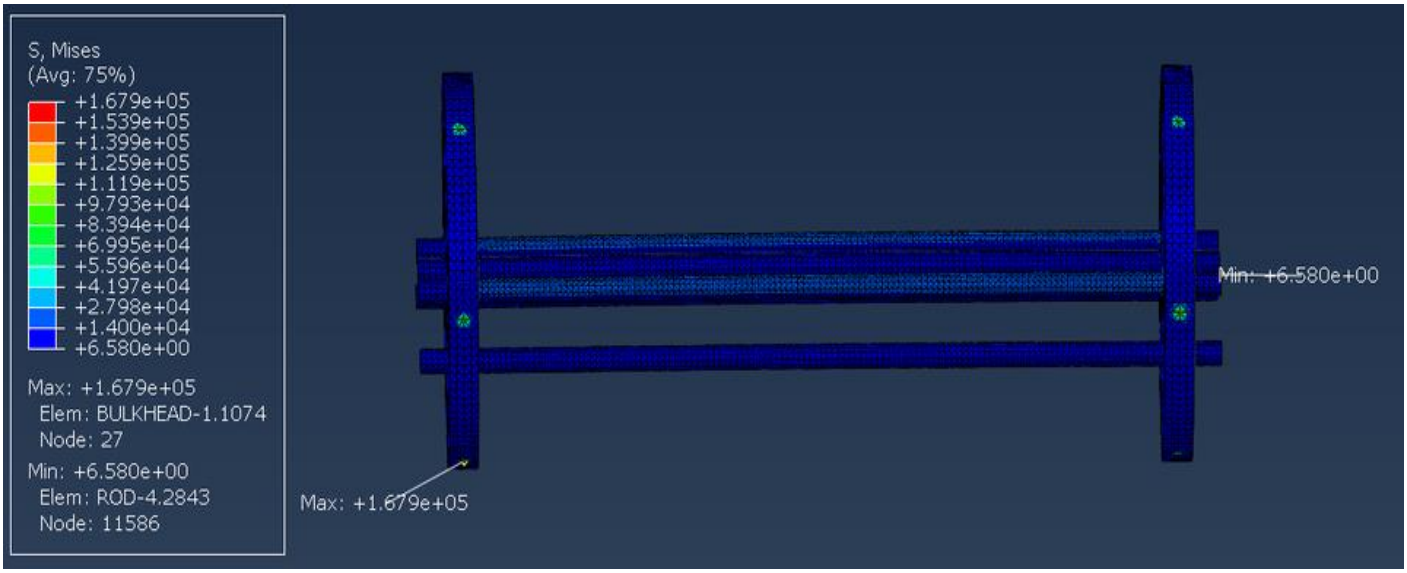


Figure 22 Stress Structural Reinforcement

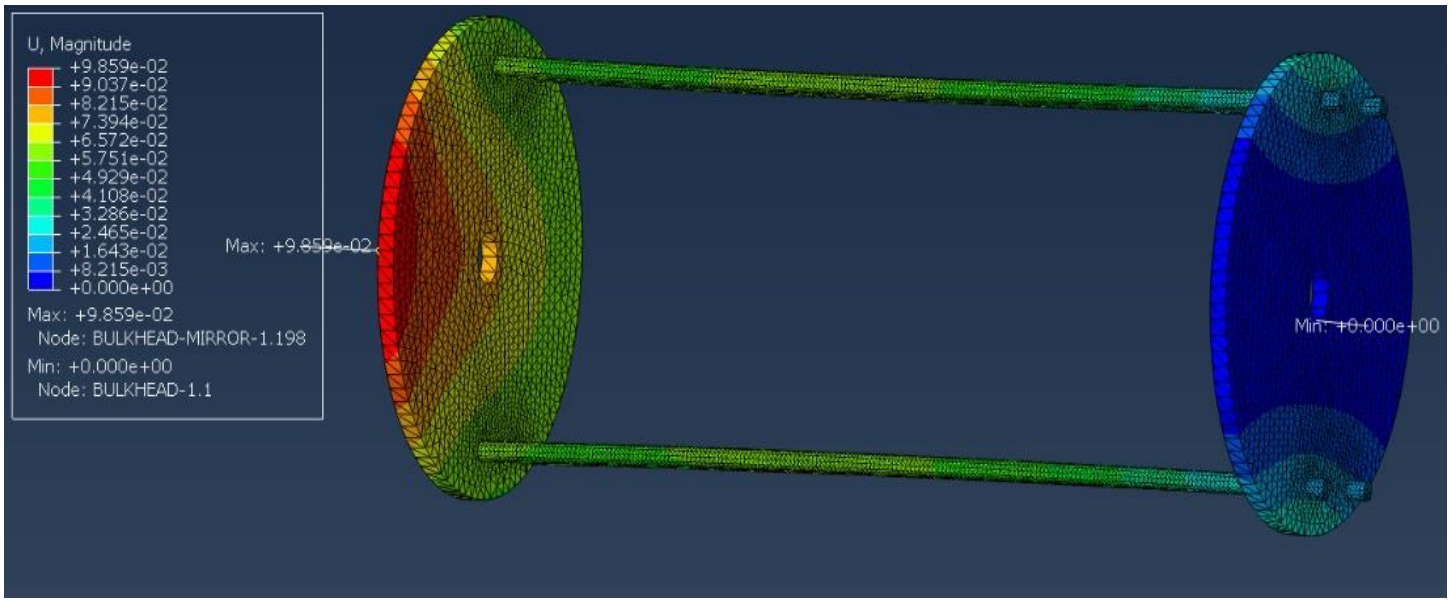


Figure 23 Deformation Old Design

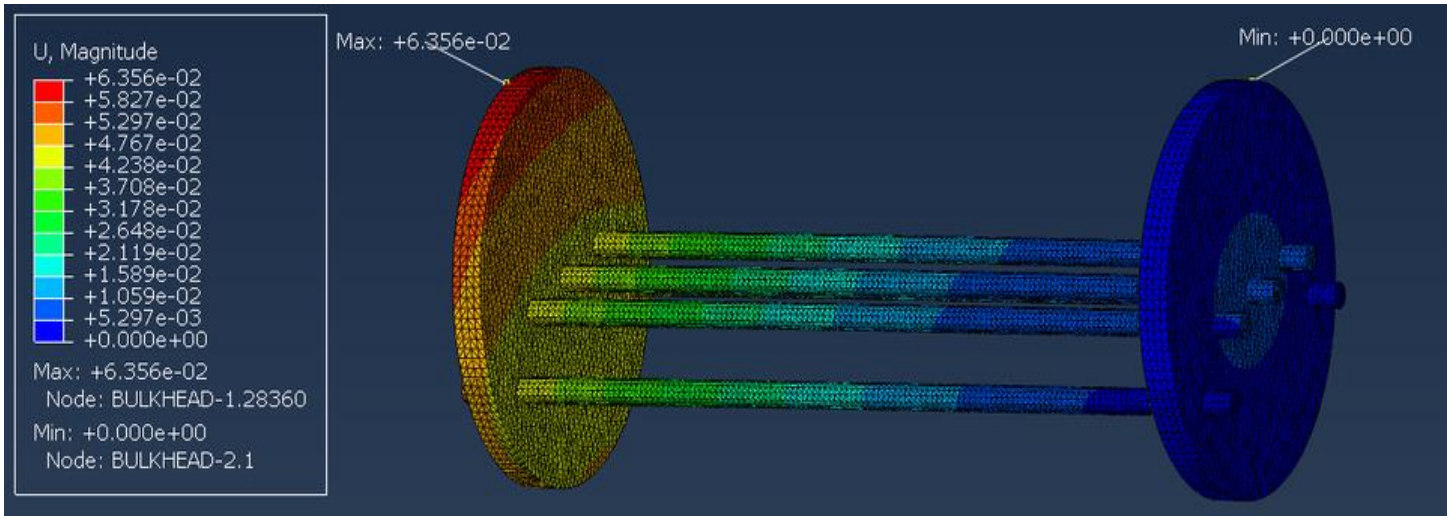


Figure 24 Deformation Structural Reinforcement

4.5 Additional Images

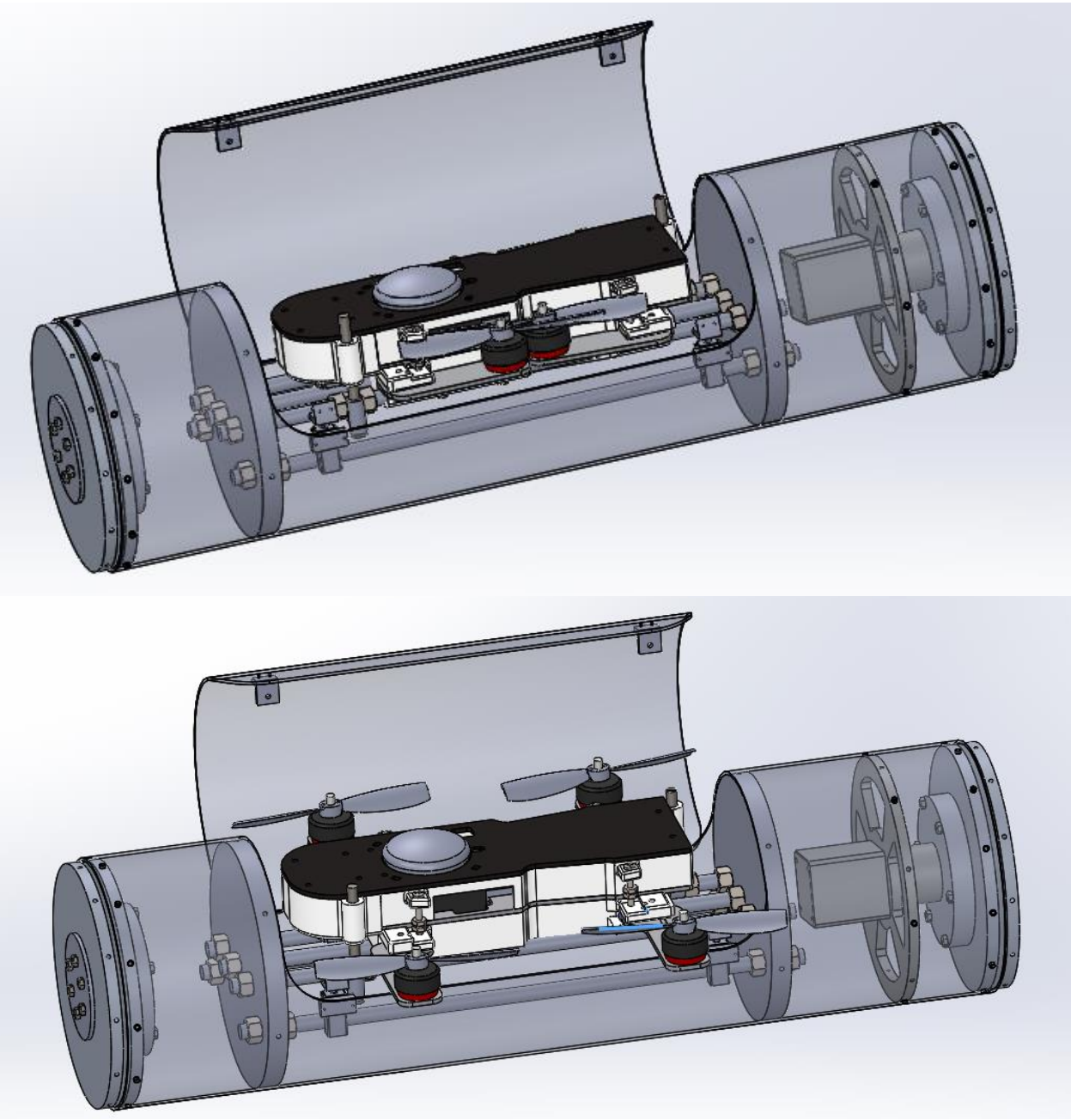


Figure 25 Payload

4.6 Diagrams

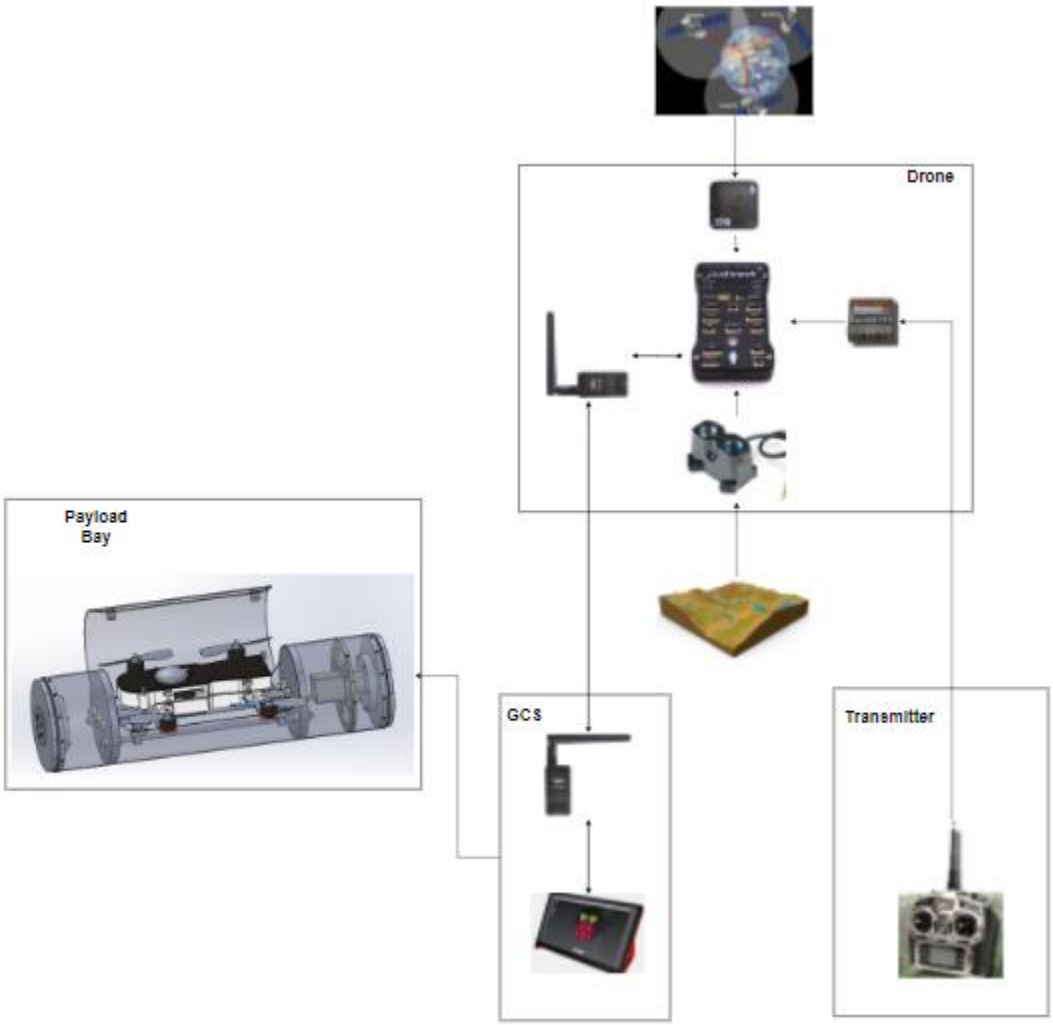


Figure 26 System Architecture

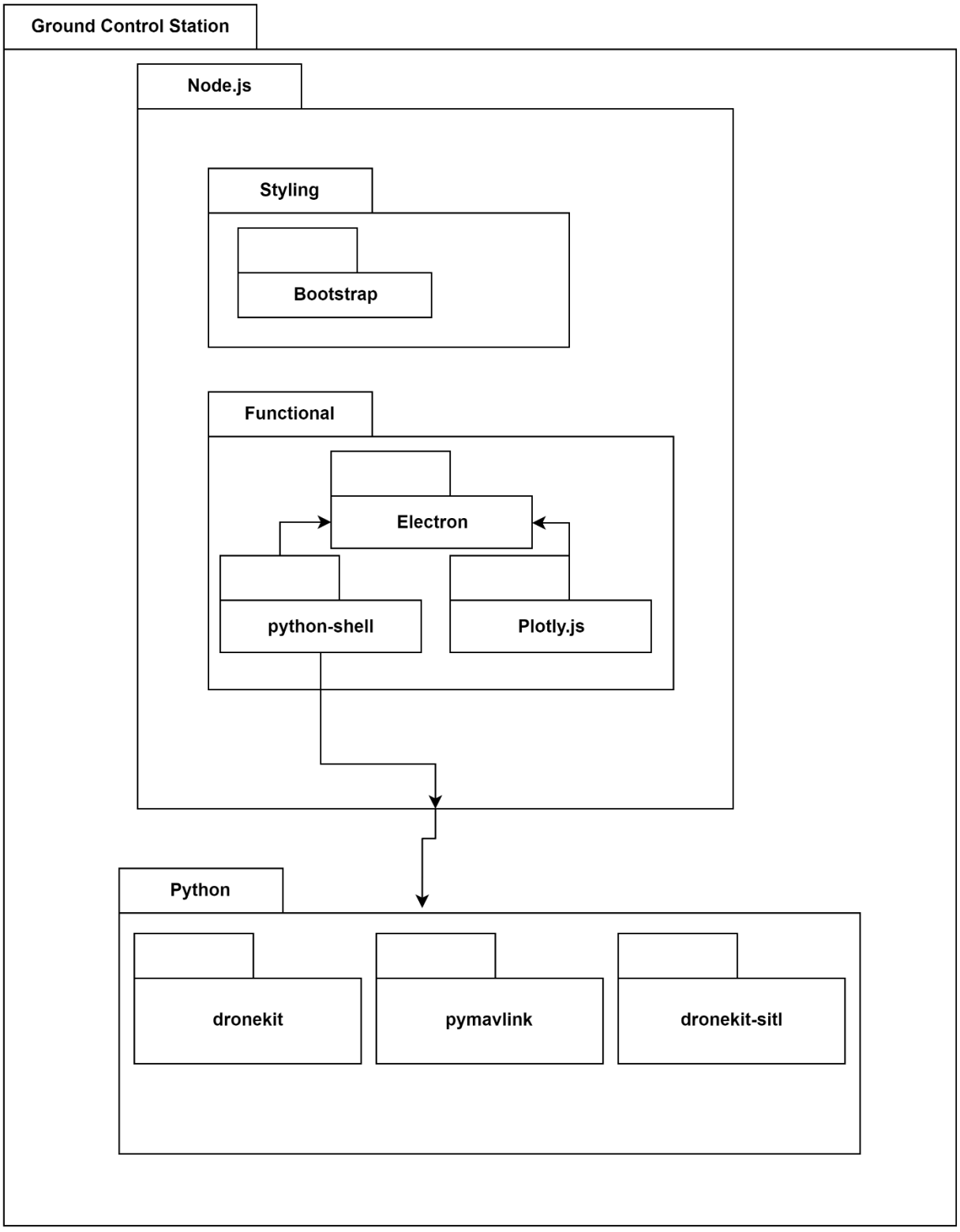


Figure 27 Package Diagram

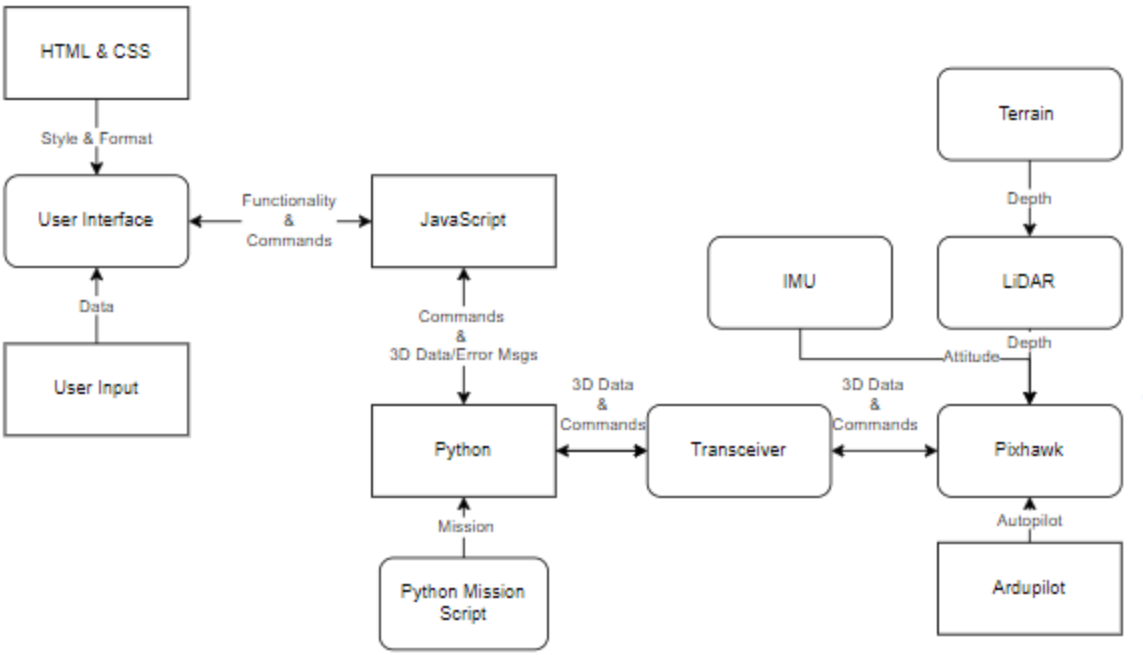


Figure 28 Functional Diagram

5. References

[1] Markforged, "Onyx," [Online]. Available: https://markforged.com/materials/plastics/onyx/ .

[2] "Ardupilot," [Online]. Available: https://ardupilot.org/.

[3] "dronekit," [Online]. Available: https://dronekit-python.readthedocs.io/en/latest/.

[4] "Ardupilot Flight Modes," [Online]. Available: https://ardupilot.org/copter/docs/flight-modes.html.

[5] "mavlink," [Online]. Available: https://mavlink.io/en/.

[6] T. R. Yechout, Introdcution To Aircraft Flight Mechanics, AIAA Education Series.

[7] "Bootstrap," [Online]. Available: https://getbootstrap.com/.

[8] "Node.js," [Online]. Available: https://en.wikipedia.org/wiki/Node.js.

[9] "Electron," [Online]. Available: https://www.electronjs.org/ .

[10] "pymavlink," [Online]. Available: https://pypi.org/project/pymavlink/ .

[11] "NumPy," [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.array.html.

[12] "python-shell," [Online]. Available: https://www.npmjs.com/package/python-shell.

[13] "dronekit-sitl," [Online]. Available: https://dronekit-python.readthedocs.io/en/latest/develop/sitl_setup.html.

[14] "Plotly.js," [Online]. Available: https://plotly.com/javascript/ .