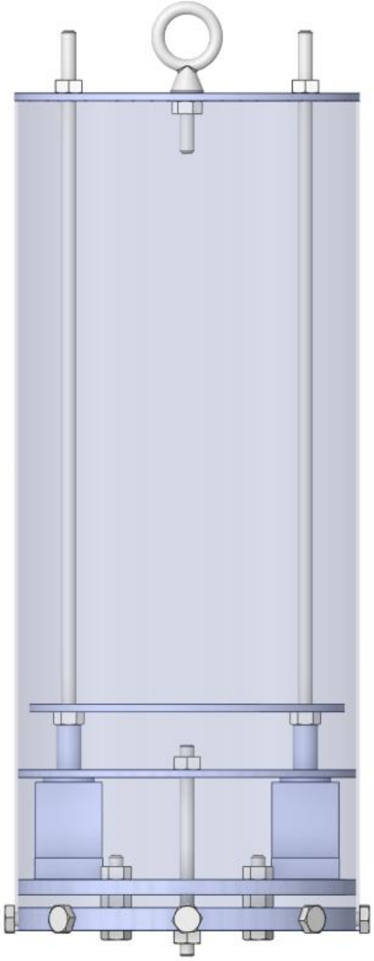
Aerojet FAR Robotic Payload: Blue Team RC Rover

Group 28



AEROJET ROCKETDYNE
Coleman Aerospace

Name	Contact	Major
Justice Cordova	justicecordo@knights.ucf.edu	EE
James Ellison	jhellison415@knights.ucf.edu	CPE
Wesley Fletcher	wkfletcher@knights.ucf.edu	CPE
Joshua Kissoon	kissoonjoshua@knights.ucf.edu	CPE

Contents

1.0 Executive Summary

One of the goals of space exploration is the investigation of terrestrial surfaces on celestial bodies, such as moons, planets, and asteroids. As these surfaces are hostile to human life, a safe and efficient method of exploration is needed. Using a remotely controlled vehicle, like the Curiosity Rover, presents an effective way to explore a different planet. The aim of this project is to make a lightweight, remotely controlled rover that can serve as a payload on a rocket.

The goal of this project is the design and development of a rocket payload system that deploys a small, lightweight, radio-controlled rover capable of video surveillance of the landing site. As this payload system will be housed in and released from a rocket, weight and durability concerns are paramount. The payload must be able to survive the G-forces and vibrational forces of launch, deployment, and eventual touchdown while remaining operational. Further, as we don’t have perfect control over the landing, we must ensure that our payload system can release the rover under a broad range of conditions, and that the rover can operate properly in those conditions. Finally, unknown factors and stochastic effects may damage our system in unpredictable ways before our rover gets a chance to work, so redundancy will be an important design principle. Ideally, our system should have sufficient redundancy to allow the bypass of any malfunctioning components while still completing the mission.

The rover will be housed in a payload canister. A release sensor will be triggered when the canister lands on the ground, triggering the canister to open. The rover must be successfully deployed from the canister and be able to drive freely in a desert environment. The rover will start transmitting a live video feed after deployment. Ground control will be able to send commands to the rover to remotely control it.

This is an interdisciplinary project that will involve collaboration with several teams of engineers. We must closely collaborate with the mechanical and aerospace teams building the rocket and ensure that our rover meets their payload specifications. As a result, our design may be subject to change to ensure a successful rocket launch and deployment.

The rover will need to be very compact and lightweight to meet payload requirements. This presents significant design challenges that would not be present in a larger rover. All design choices must be made with the size of the rover in mind.

2.0 Project Description

2.1 Project Motivation

In 1970, the Lunokhod 1 became the first remote-controlled rover sent into space. Designed by the Soviet Union, it explored the lunar surface for roughly 11 months, traveling more than 6 miles and sending valuable scientific data back to earth before its eventual failure in 1971.

At the height of the space race, rovers became a valuable scientific tool. The United States and Soviet Union sent dozens of rovers to the Lunar surface. Successful rover deployments allowed us to learn more about the moon without putting human lives at risk on a lunar expedition.

In recent decades, rovers have been successfully deployed on the surface of Mars, a breakthrough in space exploration. Rovers have made touchdowns on asteroids, collected rock samples, and taken photographs.

Rovers are a powerful symbol of the limitless potential of human ingenuity. Rover missions on Mars could potentially discover signs of life, a discovery that could fundamentally alter our understanding of the universe. Rovers can help us determine the feasibility of colonizing other planets and can identify valuable resources far away from earth.

Rovers are necessary tools in the exploration of space. They must be able to function under extreme conditions hostile to human life.

Six rovers have had successful Mars missions: Sojourner (1997), Opportunity (2004), Spirit (2004), Curiosity (2012), Perseverance (2021), and Zhurong (2021).[C17] The Curiosity Rover is still operational after 9 years, and its mission has been ext108ended indefinitely.

A noteworthy early attempt was the Prop-M Rover on the Soviet Mars 3 expedition in 1971. Communication failed and the signal was lost 14.5 seconds after landing, likely due to a Martian dust storm.[C18]

The failure of the Prop-M rover demonstrates the significant difficulties involved when a rover is deployed on an alien planet. The rover was unable to withstand a brutal dust storm. Conversely, the Curiosity rover shows how a successful deployment can exceed expectations. An intended two-year mission has lasted for more than 9 years.

The extremely high costs involved in sending a rover to Mars mean that there is enormous pressure to make sure that the rover has a successful mission and does not fail prematurely. Significant stress testing is required.

Future rover missions will require durable, low-cost designs that meet strict payload specifications.

As time goes on, curiosity of space exploration grows and rover designs need to evolve to investigate terrestrial surfaces on celestial bodies, such as moons, planets, and asteroids more efficiently. As these surfaces are hostile to human life, a safe and efficient method of exploration is needed. Using a remotely controlled vehicle, like the Curiosity Rover, presents an effective way to explore a different planet.

Being men of science, specifical engineers, we are motivated to design future rovers that may one day do things like Curiosity, but everyone has to start small first. That is why our goal is to design a rover intended to be a hardy, lightweight design that is weather resistant, can withstand significant mechanical stresses, and can be seamlessly deployed from the Arcturus rocket payload canister. This is our first step into the world of rover for scientific exploration.

2.2 Goals

Our goal is to design a rover that will serve as a payload on the Arcturus Rocket built for the FAR 1030 competition in summer 2022. On landing, the rover will deploy successfully from a payload canister, and will be able to move on command directed by the ground station. The rover will transmit live video and status updates to the ground station. The rover will be able to operate at a range of up to 600 meters. Due to its nature as a rocket payload, the rover is subject to strict size, weight, and durability constraints. Every design decision should be made with the rover's future status as a rocket payload in mind. Components will be selected based on their size, and the result will be a very modular design built around integrating many small components into various subsystems. A successful rover operation can be split up into the completion of several tasks:

The rover will meet size and weight constraints as the payload

Upon landing, the rover will successfully "wake up" and leave the canister

A live video feed will be transmitted to the ground station after deployment (range ~600m)

The rover will move in response to commands from the ground station (FAR 1030 competition requires 10 feet of movement, which the rover should accomplish easily)

The rover will be able to return a 360-degree scan of the environment on command from the ground station

2.2 Objectives

The core objective of our project is to prototype and develop a radio-controlled rover that will be capable of traveling a distance of at least 10 feet while simultaneously transmitting live video. For us to achieve this goal, we will need to also design a canister to safely contain the rover. Additionally, the canister must be restricted from movement within the rocket using a sled. For this project, we want to minimize and maximize certain parameters of our rover, sled, and canister. These characteristics are weight, size, power consumption, and durability. Durability will be especially crucial as all three components will be subject to harsh conditions. As for weight and size, they will need to be within the constraints of the FAR 1030 competition rules. Reducing power consumption goes together with the aforementioned ideas as we must consequently select smaller, more economical components. Once these goals are satisfied, we will aim to "min-max" the potential of our rover regarding its sensor functionality, operating range, and battery life.

Our primary objectives concerning the rover's functionality is that it must be able to provide video communication back to our ground station, which is situated near the launch base, and it also must travel 10 feet. We will achieve this by mounting a 5.8GHz FPV camera to the rover. Some sub objectives for the camera are that the video stream must be smooth, and the quality must be at minimum 480p. We will attain this goal using the Waveshare IMX219-83 Stereo Binocular Camera. To satisfy the distance requirement, we will need to select appropriate wheels, motors, and encoders, which will be discussed in later sections.

While not required, our team's secondary objective is to enhance the functionality of our rover through the employment of other technologies. One goal that we will strive to achieve is proper obstacle mapping using stereo cameras to assist with remote navigation. We will also integrate a GT-U7 GPS module to ensure that the rover remains in operational range. The rover will also be equipped with Inertial Measurement Units, particularly the SparkFun 9DoF, to help navigate the terrain and prevent accidents. Another idea that we aim to implement is autonomous retrieval in which the rover will rendezvous at the launch site when signaled.

As this is a project funded by Aerojet Rocketdyne Coleman Aerospace, we must meet strict size and weight requirements. Our goal for our payload, canister, and sled is to not exceed 4.31 kg and for the rover to weigh at least 1 kg. Our other assigned objectives are that the cylindrical payload sled should have a depth no greater than 1.27 cm and the canister and sled should fit in a 15.24 cm diameter by 40.64 cm length payload bay. These design constraints were held in mind when we started prototyping.

Our goal with the sled and canister is that they would be able to withstand the force involved with the vertical and lateral acceleration of the rocket. The force from vertical acceleration is approximately 8 G's, and lateral acceleration experienced by the canister is 7.69 m/s^2. Furthermore, the canister must survive the shock from the parachute, a force of 8 G's. In addition, the intense heat during payload ejection could cause irreversible damage to the canister or its components. To mitigate these issues, we will construct the sled and canister shell out of 6061-T6 aluminum. Since cost and weight are two factors we want to minimize, we decided to use this material instead of other alternatives, such as steel. 6061-T6 Aluminum disks will be used as a base for our components due to its ability to endure high temperatures without alteration. Since the safety of our components are of great concern, we will also fasten all our components securely with nylon ties and aerospace-grade RTV sealant. Two stainless steel lead screws will be coupled with two motors and threaded through the 6061-T6 Aluminum disks that holds the rover, and it will protrude out of the top of the canister. Threaded black oxide steel rods and nuts will be used for added stability. 3D-printed PLA will be used to mount components to the plates and help restrict the rover's movement within the canister. Finally, 303 Stainless Steel couplers will be used to bear the axial load produced by the drogue parachute.

Seamless rocket ejection is another objective of our team. A steel eyebolt will be fastened to the top of the canister as it is required for the deployment of the drogue parachute. Also, a magnetic proximity sensor will be employed to provide power to the microcontroller. The sensor from the Avionics team will switch on the canister's power system, which consists of three 18650 LiFePo4 batteries, and the Arduino Nano as soon as the canister exits the rocket at 1000 ft. During this event, the Freescale MPL3115A2 delivers pressure readings to the Nano to determine the altitude of the canister. Once the Nano is certain that the canister is grounded, another sensor will switch on the rover's power supply.

The canister must also be sophisticated enough to deploy a rover at touchdown. We will need to design a mechanism to release the rover from its canister. Our objective is to have a reliable way of deployment that doesn't compromise the safety of the rover and electrical components within the canister. From these premises, we will use Nema 11 motors, Allegro A4988 drivers, Acme lead screws, steel couplers, and AMT102-V encoders to slowly “unscrew” the rover out of this canister. The motors will need to have enough torque to raise the payload, and the couplers will transfer the torque to the lead screws.

After exiting the canister, the rover must also be durable. We will be ensuring that the rover will be able to traverse through the dangers of the desert. This means that the components used must be at least IP50 rated as dust is abundant in such a climate. Another hazard that we will secure our rover from is the desert’s irregular terrain. Our team will use wheels that will provide the necessary traction to cross this rough terrain. A mechanism will also be installed to the device to prevent falls that could be catastrophic rover’s operation. The rover will also be able to work normally in the event that it is flipped upside-down. A long-term objective that we have is to make the rover insusceptible to damage from other environments.

The rover will need some way to communicate with our ground station to send video and other telemetry and receive navigation information. To accomplish this, our team will be using a Reyax RYLR896 transceiver in the rover to send information back and forth to our ground station’s laptop. At the launch base, a LoStik LoRa USB will intercept this data, and send movement commands. These commands are then routed and interpreted by the Jetson Nano. As for relaying video, the Hyperion TS5823 AV Transmitter will be installed in the rover for this purpose. The ground station will receive this video using the Skydroid FPV Receiver. Radio interference will be mitigated as the Reyax RYLR896 module will use an operational frequency of 915 MHz, and the Hyperion TS5823 module will communicate using 5.8 GHz.

Upon deployment, the rover will be expected to transmit and receive data continuously for at least ten minutes. Components must be chosen to ensure an optimal battery life for the rover. The motors and the single board computer with peripherals are expected to be the most power-hungry modules. Selecting components that are optimized for low power applications and operating the single board computer and microcontrollers in low power mode, when possible, will give the rover a longer life. Although some submodules can be connected directly to the 3.3 V and 5 V pins on the single board computer, it is necessary to run certain modules directly from the battery power supply, necessitating power electronics to step-up and step down the voltage. Buck and boost converter topologies are selected that have a low footprint and high efficiency. Another consideration is limiting interference from the power supply by choosing switched-mode power supplies with significantly lower switching frequencies than the RF communications modules.

To ensure that the rover is durable and able to withstand the mechanical stresses of a rocket launch and canister landing, batteries must be chosen that are safe and stable. Initially LiFePO4 (Lithium Iron Phosphate) batteries were considered for their safety and stability, but due to supply issues and the issue of finding a reliable battery management system the much more commonplace Lithium polymer ion (LiPo) batteries were selected. Extra consideration has been taken to ensure that the batteries are secure to avoid the risk of a catastrophic thermal runaway.

The objective for the ground station is to allow control of the rover to users. To do this we will use python as our coding language and to do this we use a GUI creator such as a Qt designer to create the basic interface for the controls and place for displaying the coordinate. Once the GUI is made PyQt5 will be used to modify the code. At the moment a button input will be used, but if need be, we can try a physical controller of some kind. Once the interface and code are done the next step will be to make sure it is able to make a connection with the rover. If not, we edit the code to make sure it can. When a connection has been made the next step will be to make sure we can control it from the distance needed to operate it. Then the next step will be to make sure the GPS coordinates are right and if they are not making sure to test the code and hardware if necessary. Then the final step will be to make sure the GUI is able to log all the information correctly.

2.3 Requirements Specifications

The requirements in <REQUIREMENTS_TABLE> are a combination of those set out for us by the customer, extrapolated from competition details, or added due to personal interest in the implementation:

Table <REQUREMENTS_TABLE>: Design Requirements

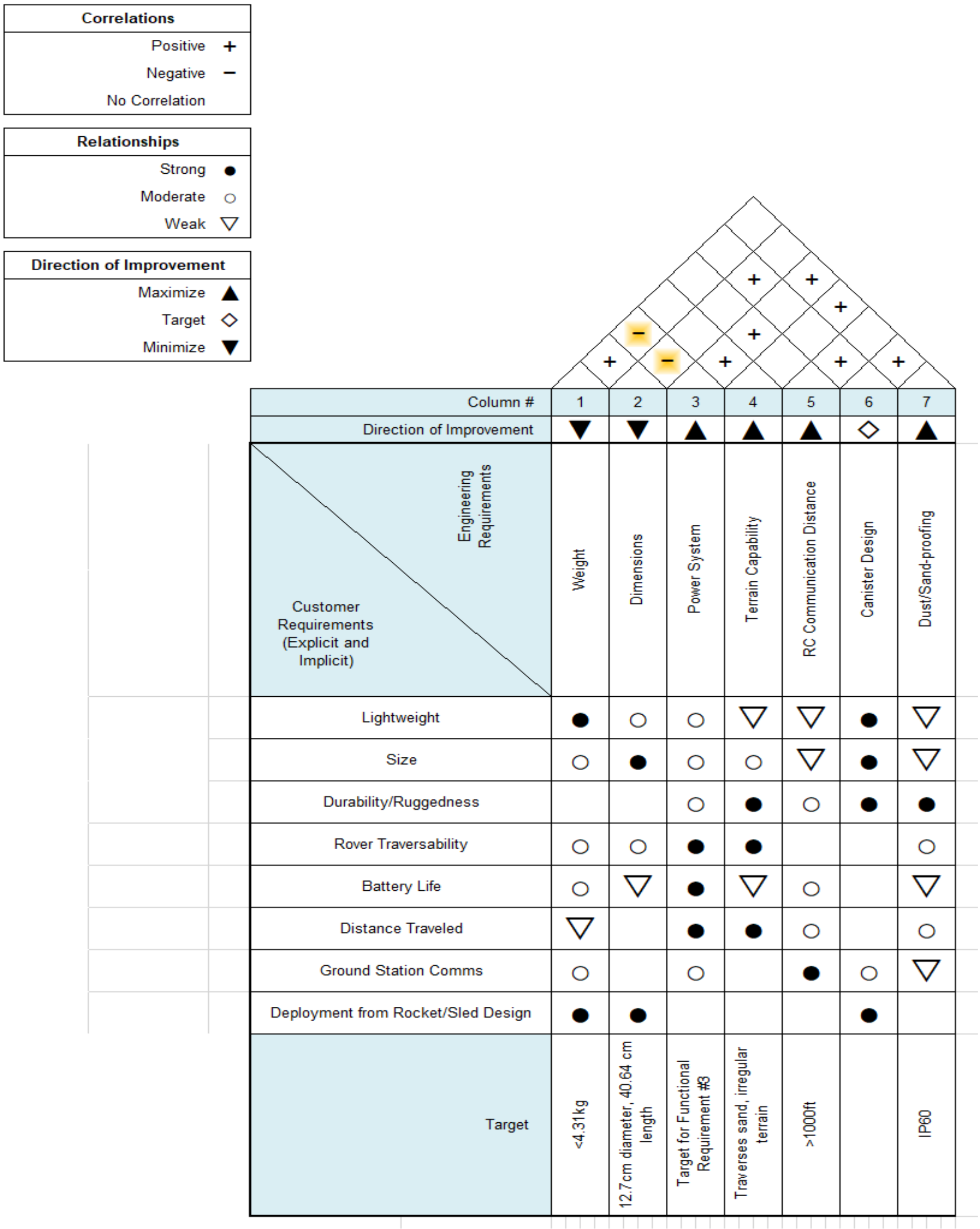
Trace #	Requirement	Value (if applicable)
R1.0	Mass and Dimensions	
R1.1	Payload ~~assembly~~ max weight ~~(sled, canister, and payload combined)~~	<4.31kg
R1.2	Payload (rover) minimum weight	1kg
R1.3	Payload ~~Canister~~ dimensions	<=12.7cm diameter <=40.64cm length
~~R1.4~~	~~Payload Sled Dimensions~~	~~Maximum 1.27cm depth on either side of payload~~
~~R2.0~~	~~Payload Sled~~	
~~R2.1~~	~~Payload Sled shall release Payload Canister at deployment phase via force generated by parachute.~~	~~-1.0 G~~

R3.0	Payload Canister	
R3.1	Payload Canister shall contain the Payload during launch, deployment, descent, and landing phases.	N/A
R3.2	Canister shall provide power-on signal to Payload during deployment phase.	N/A
R3.3	Canister shall act as RC signal relay for payload after landing.	N/A
R3.4	Canister shall open under its own power on command to release the rover.	N/A
R3.5	Canister shall be reusable for multiple potential launches.	N/A
R3.6	Canister shall be able to withstand force involved with the acceleration of the rocket.	8 G’s max axial acceleration 0.78 G’s max lateral acceleration
R3.7	Canister shall be able to withstand the force of the drogue parachute.	8 G’s max axial acceleration
R3.8	Canister shall be resistant to the heat generated by the ejection charges.	N/A
R3.9	Canister motors shall have enough torque to release the payload.	
R4.0	Payload (Rover)	
R4.1	Rover shall travel the specified distance from ~~the Payload Canister landing site~~ starting position under its own power.	10ft
R4.2	Rover shall be radio-controlled.	N/A

R4.3	Rover shall transmit a live video feed back to the Ground Station at given range.	<=600meters
R4.4	Rover shall be dust resistant.	IP50
R4.5	Rover shall be able to determine position and orientation relative to ~~Payload Canister~~ starting position.	N/A
R4.6	Rover shall be able to generate and store a rough map of immediate surrounds	N/A
R4.7	Rover shall create a 360-degree horizontal panorama on command.	N/A
R4.8	Rover shall log all relevant sensor and control data during operation for later retrieval.	N/A
R4.9	Rover shall transmit real-time telemetry data back to ground station at defined intervals.	N/A

2.4 House of Quality Analysis

Figure <HOQ> contains the House of Quality analysis done during initial project conception to determine how we would evaluate effectiveness of designs.



Correlations	
Positive	+
Negative	−
No Correlation	

Relationships	
Strong	●
Moderate	○
Weak	▽

Direction of Improvement	
Maximize	▲
Target	◇
Minimize	▼

Column #	1	2	3	4	5	6	7
Direction of Improvement	▼	▼	▲	▲	▲	◇	▲
Engineering Requirements / Customer Requirements (Explicit and Implicit)	Weight	Dimensions	Power System	Terrain Capability	RC Communication Distance	Canister Design	Dust/Sand-proofing
Lightweight	●	○	○	▽	▽	●	▽
Size	○	●	○	○	▽	●	▽
Durability/Ruggedness			○	●	○	●	●
Rover Traversability	○	○	●	●			○
Battery Life	○	▽	●	▽	○		▽
Distance Traveled	▽		●	●	○		○
Ground Station Comms	○		○		●	○	▽
Deployment from Rocket/Sled Design	●	●				●	
Target	<4.31kg	12.7cm diameter, 40.64 cm length	Target for Functional Requirement #3	Traverses sand, irregular terrain	>1000ft		IP60

Figure <HOQ>: House of Quality analysis for our rover

3.0 Research Related to Project Definition

3.1 Existing Similar Projects and Products

In the process of researching this project, we attempted to gather comparable robots with similar features and constraints. There weren't any *exact* matches to our proposed feature set or design constraint (with "hobbyist rocket payload" being a major exclusionary factor), so we gleaned what lessons we could from what was available. This required us to select some categories of comparables that don't necessarily match up with our proposed final product but share important functional attributes. Since small size is a major constraint for our rover, that is the axis the search began on.

3.1.1 Throwable Robots

A throwable robot is a small, ruggedized robot designed for use as a "scouting and surveillance tool" [C1]. They're literally "thrown into action" by first responders or military operators to gather information about dangerous spaces. Throwable robots are designed to be lightweight and shock-resistant; design principles shared with our own robot that must fit in a size- and weight-constrained space and survive launch/landing forces of a rocket. Just like our prospective rover, these robots place a premium on mobility, though they're frequently designed for urban/interior spaces, rather than the uncontrolled terrain of a desert environment. Further, they generally have our desired capabilities of video streaming, radio control, and reusability.

3.1.1.1 Teledyne FLIR FirstLook

Billed as a "throwable, rugged, and expandable robot that provides immediate situational awareness" [C2], the FirstLook is a tank-tread driven robot with multiple RGB cameras, a touchscreen interface, and extensible payload ports. The FirstLook does meet the weight requirements (R1.1 and R1.2) and the video-streaming requirement (R4.3). However, while exact dimensions are not readily available, it appears that the FirstLook is too large to meet our dimension requirements (R1.3). Notable in this robot is a "flipper" attached to the robot that augments its traversal capabilities, and allows the FirstLook to reorient as needed, a capability that may prove useful in the semi-controlled conditions of our landing canister.

3.1.1.2 Roboteam IRIS

A "hand-carried, ruggedized and throwable system fit for extreme conditions" [C3], the IRIS, much like the FirstLook, meets all design requirements minus the dimensions (R1.3) with a LxWxH of 24 x 20.5 x 11 cm. Further, it's IP66 and IP67 compliant, exceeding our requirement for dust-resistance (R4.4). Unlike the other comparables in this category, the IRIS is a 4-wheel drive robot, which theoretically makes it more capable in outdoors or rough terrain scenarios. This is a potentially important consideration for our own robot, which will be expected to perform in a sandy and uncontrolled environment.

3.1.1.3 ReconRobotics ThrowBot 2 Robot

Yet another “throwable micro-robot platform” [C4]. The ThrowBot is significant because it does meet our size requirements, with a major caveat. The ThrowBot makes use of a retractable “tail” that trails behind the robot and provides needed stability to the two-wheeled platform. This may end up being a useful design for us to explore, as it allows us to use just two drive wheels while minimizing the risk of tip-over events when traveling up and down grades. Otherwise, this robot is like our other comparables in terms of control and communications.

3.1.2 “Toy” Robots

In the process of searching for comparable systems, we evaluated “toy” robots, i.e. small-scale, low-power robots designed for educational use or entertainment. While less robust than the throwable robots in the previous section, they’re generally less expensive while meeting similar functional goals. Of those we found, the DDL Vector was the most technically impressive.

3.1.2.1 DDL (Formerly Anki) Vector

This toy robot was originally released in 2016, marketed as an electronic companion for children. Over the years, it has garnered a surprisingly large following of technically competent fans. While little official documentation is provided by DDL, there is an oddly comprehensive, unofficial technical manual [C5] containing details on electrical design, software implementations, and architecture diagrams. This treasure trove of information available on the Vector makes it a valuable comparable since many of the robots in the Throwables category are “military-grade” and appropriately tight-lipped about design.

The Vector’s sensor package has significant parity with the (projected) sensors onboard our rover: a RGB camera, an inertial measurement unit (IMU) to detect changes in orientation, a time-of-flight sensor for proximity awareness, and ground-sensing proximity sensors for negative obstacle detection. Further, the Vector makes use of a Bluetooth Low-Energy (BLE) sensor for communication. While we likely won’t need to send data back and forth between our rover and our landing canister, it might be useful for localizing between the two.

Despite the similarities between the Vector and our target design, there are some important areas where the Vector falls short: since the Vector is designed for indoor use, it doesn’t have the shock-resistance needed for our rover to survive landing, nor does it have the dust- and weatherproofing that our rover will require to operate in a dusty, windy, outdoor environment. Further, the Vector is not designed to traverse sand or other uncontrolled, not-quite-two-dimensional environments. It doesn’t have the mechanical capabilities, or the power capacity needed for long-term traversal of difficult environs, though how much of that is required for our target of 10ft is still up in the air.

3.1.3 Full-sized Mars Rovers

Clearly, we cannot reasonably design a true-to-scale Mars rover, such as Perseverance, but we can look to them as design inspirations.

3.1.3.1 NASA 2020 Perseverance Rover

NASA's Perseverance Rover was designed to explore harsh Martian terrain, beginning operations on the Red Planet in February 2021 [C6]. On board Perseverance is a comprehensive suite of sensors: stereo and RGB cameras, spectrometers, telescopes and microscopes, microphones, IMU, and more. Powered by a radioisotope generator and sporting two central computers for redundancy, Perseverance is built for the long-haul. A robotic arm mounted on the front of the rover allows it to drill for rock cores and otherwise manipulate the environment.

Technically impressive as it is, Perseverance doesn't meet our project requirements. First, being the size of a small car (2260 lbs., 10'x9'x7') excludes it from traveling onboard any hobbyist rocket. Further, we're unlikely to be able to commission a nuclear reactor for power onboard our own rover. So, we must take inspiration from its sensor capabilities and power management instead.

A major inspiration: the idea of making our radio-controlled rover capable of autonomous navigation ("blind driving") as well. Perseverance uses a stereo camera to provide 3D volumetric information about its surroundings for autonomous navigation. This is something we could explore in our own rover, as stereo cameras are widely available products. Combining the stereo camera with an IMU (also widely available) for positional awareness and wheel encoders, it's feasible for our small rover to implement this capability.

A second design inspiration is the use of redundant central computers. If one of the "brains" onboard Perseverance is damaged, it can simply switch to the other to continue operations. The possibility of a second computer on our own rover is attractive considering the shock and vibrational forces it will experience upon landing and the dusty environment it will operate in. Theoretically, adding an additional single-board-computer to the design is possible, though how much it will complicate the design (or stretch the budget) is yet to be seen.

Finally, Perseverance is designed to move very slowly, topping out at about 0.1mph [C7]. This speed is acceptable since Perseverance is more concerned with energy efficiency than speed. Like Perseverance, speed is not our rover's major concern. Ultimately, all we care about is meeting our distance requirement of 10ft from landing site [R4.1]. This means we can reap the energy efficiency benefits of traveling slowly to offset the power requirements of a more capable sensor suite.

3.1.4 Drone and RC hobbyist First Person Video (FPV) TX/RX

The rover will be expected to stream video, starting from the time of its initial deployment. Due to the size constraints on the rover, live video presents a significant technical challenge. A small, modular solution must be found.

Thankfully, the rapidly growing community of amateur drone pilots and RC car drivers has led to an explosion in resources for live video transmission in a compact package. A camera mounted on a drone or RC vehicle is connected to a transmitter module, which sends the signal to a receiver connected to a screen which displays the live camera feed.

Competitive events like drone racing, and the use of drones in commercial applications for onsite inspections of inaccessible or dangerous environments has led to the emergence of miniaturized, lightweight video transmitter module designs.

When the transmitted signal is received, the receiving screen displays a drone's-eye-view of the world. The received image has first-person quality. Hobbyists use "First Person View" (FPV) as a shorthand for the live video they receive when operating an RC vehicle or drone.

Dedicated racers will use a VR headset on the receiving end to create a more immersive, truly first-person experience. The growth of the drone community in the last decade has driven hobbyists to find effective DIY setups for FPV and has led to the proliferation of cheap and effective modular components that can be integrated into any enthusiastic hobbyist project.

As most drones are controlled in the 2.4 GHz band, this band is limited for video transmission. Most hobbyist products use 5.8 GHz for the video feed. This frequency band is ideal, as potential interference from the rocket telemetry will be limited.

An off-the-shelf 5.8 GHz transmitter module, like those used by FPV drone hobbyists, can be integrated into the rover electronics, and transmit live video. The hobbyist community also has in-the-field experience with various antenna configurations and their effectiveness at range, making the choice of antenna easier to make. 5.8 GHz antennas can be very small, as EM radiation at this frequency has very short wavelengths of roughly 5 cm.

Circularly polarized antennas are the standard for FPV setups, as they limit multipath interference (where the signal reflects off objects and hits the receiver at different phases, creating interference) and alignment problems are not as common as they would be with a linearly polarized antenna.

3.2 Relevant Technologies

3.2.1 Radiofrequency (RF) Technologies

3.2.1.1 LoRa 915 MHz Transceivers

The radio spectrum is tightly regulated by the FCC, and we are limited to using specific bands. Of particular interest are the Industrial, Scientific, and Medical (ISM) bands, which are used for many low power applications. Since the rocket's telemetry will be broadcasting in the 70-centimeter band, the frequency range 420-450 MHz should be avoided. A frequency band of interest is the 33-centimeter band (902-928 MHz), which is an ISM band commonly used in the United States for many applications. 33 centimeters is ideal because it is a commonly used band for small, low power devices. The 33 cm band does not require an amateur radio license and is free to use for low power applications, like remote control of a rover. Lower ISM frequencies will typically require larger antennas and will not meet our space requirements.

The LoRa networking protocol is a proprietary protocol developed in France, used for many low power applications. LoRa uses several sub-gigahertz frequencies (below 1 Gigahertz), including 915 MHz, the center frequency of the 33-centimeter band. There are many options available for LoRa modules, which can communicate with the microcontrollers using UART. A LoRa transceiver will transmit commands for the rover to the receiver, located on the rover, which will interface with the microcontroller that controls the motors.

3.2.1.2 FPV at 5.8 GHz

Live video transmission presents a significant technical challenge. Fortunately, the burgeoning community of drone hobbyists provides a significant number of resources on first person live video (FPV) transmission. Typically operating in the 5.8 GHz band, the range and penetration is limited compared to the 900 MHz signal. Circularly polarized antennas offer a high gain and limit interference from multipath propagation, where reflected waves reach the receiver out of phase and interfere with each other. A 5.8 GHz transmitter and antenna can be connected to the video subsystem on the rover and transmit live video to the ground station.

3.2.2 Power System

3.2.2.1 DC to DC converters

The single board computer and other modules operate at 3.3 V and 5 V. Certain modules, like video transmission and the motor controller, operate at voltages of 12 V, higher than the supply voltage from the battery pack.

The series-parallel configuration of the lithium polymer (LiPo) battery pack gives us an operating voltage of 6.6 to 8.4 V. It must be stepped up or stepped down to these voltages. Two options are available: linear voltage regulation or switched-mode regulation. Linear voltage regulation is "quiet", as it does not produce high frequency RF noise. However, linear voltage regulators are inefficient, and a lot of energy is dissipated as waste heat.

Switched-mode power supplies switch at a constant frequency and are highly efficient. This efficiency comes at a cost, as these power supplies are very “noisy”.

For the rover, the decision was made to use switched mode converters, as they offer a higher efficiency, and RF interference can be avoided at the operating frequencies of the rover and rocket.

A buck converter topology will be used to convert the DC voltage to these lower voltages with minimal losses. Switched mode power supplies produce noise, and consideration must be made to reduce possible electromagnetic interference (EMI).

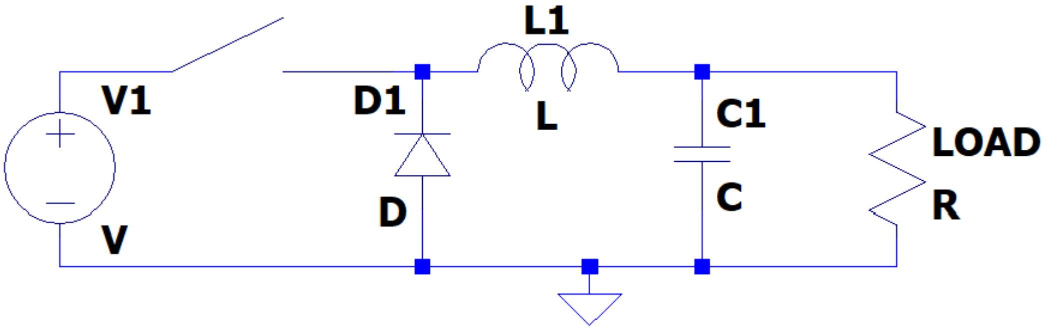


Figure : LTSpice schematic of a basic buck converter topology. The load voltage is dependent on the duty cycle of the switch.

One buck converter will be needed to step down the voltages for the single board computer, and one will be needed to step down voltages for the sensors and RF communications. A Boost converter will be needed for the motor controllers and video transmission. Texas Instruments’ Webench is an excellent resource for DC to DC converters, and the design of power electronics.

The global chip shortage has presented significant challenges to our designs for power electronics and has forced us to use suboptimal designs for our power supplies. These designs have been significantly more complicated and have made PCB assembly more difficult.

3.2.2.2 Battery management system

Because of the nature of their manufacture, lithium battery cells will have slightly different chemical properties and can become unbalanced. Unbalanced cells are hazardous and present a real risk of potential overcharge or over discharge, which can result in damage to the battery, or, in a worst-case scenario, a fiery explosion. A battery management system can prevent catastrophic failure by keeping the cells balanced and preventing overcharge and over discharge situations. Our design will be using LiPo batteries and battery management and battery safety is critical.

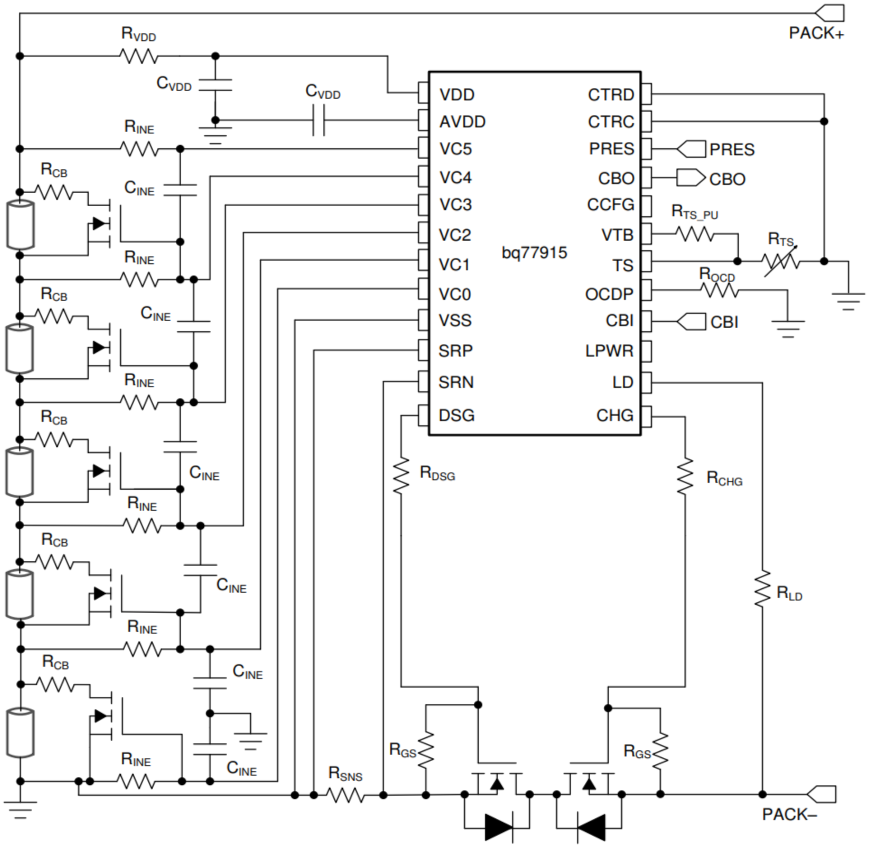


Figure <BMS_TI_SCHEMATIC>: Battery Management Schematic using the bq77915 IC, reprinted with permission requested from Texas Instruments

3.2.3 Sensors

3.2.3.1 Inertial Measurement Units (IMUs)

Inertial Motion Units (IMUs) are sensors that combine three gyroscopes and three accelerometers to report acceleration, orientation, and angular velocities [C8]. Each accelerometer or gyroscope produces information in one axis, so three of each are used to get information in all three relevant axes (roll, pitch, yaw). The information output by each individual sensor is then fused together by a sensor fusion algorithm, such as a Kalman or Extended Kalman Filter [C9].

3.2.3.1.1 Accelerometers

Accelerometers measure inertial acceleration. MEMS-technology based accelerometers determine these accelerations by measuring the deflection of a “proof mass” suspended by two small springs [C9]. The direction in which the proof mass moves is the sensitivity axis e.g., x-, y-, or z-axis. To measure all three relevant vehicle axes (roll, pitch, yaw), three accelerometers are used, one for each direction.

3.2.3.1.2 Gyroscopes

Gyroscopes measure angular rate with respect to an inertial reference frame [C10]. MEMS gyroscopes determine angular rate by measuring the Coriolis acceleration of a resonating mass suspended by springs inside a frame that is free to rotate. The mass moves along the sensitivity axis and when rotated, exerts a measurable force on its frame that provides acceleration information. Like accelerometers, a gyroscope only measures in one axis at a time, so three gyroscopes are needed to get information in all three axes.

3.2.3.1.3 Kalman and Extended Kalman Filters

A Kalman Filter is a state estimation algorithm that uses a recursive prediction-correction approach to measure the state of linear systems [C11]. By inputting sensor measurements and control signals at time t as well as the previous state belief at time t-1, we produce a state belief that represents our system [C12, p. 43]. Predictions of current state are updated by measurements, on and on for the duration of the system. The Extended Kalman Filter can be described as the implementation of the Kalman Filter for nonlinear systems [C12, p. 54]. In the context of IMUs, the Kalman and Extended Kalman Filters are used to fuse multiple sources of sensor data inputs into a single output stream that is more accurate than the sum of its parts.

3.2.3.3 Stereo Cameras

Stereo cameras make use of two horizontally displaced, rigidly affixed cameras to simulate human stereo vision. Seeing everything from two different angles allows a stereo vision system to determine depth, allowing us to track the distance from the camera to any object in view [C13]. Stereo vision systems are broadly used in the field of robotics by ground vehicles like our own as well as aerial drones, underwater vehicles, and industrial robotics.

Stereo cameras operate on the principle of disparity. The images produced by the left and right cameras will differ by some known horizontal distance i.e. the distance between the two cameras. Features in the images are correlated to each other, effectively matching points in space between the two. We can then measure the disparity between the two matching points, and use that to calculate the angle of disparity, which can then be used to calculate the depth of that point. This process is called triangulation [C15].

An important application of stereo cameras is in mapping the environment surrounding a mobile robot. By providing depth information, the stereo camera allows us to track obstacles in 3D and provide detailed spatial occupancy information, which we can then use to localize our robot in space and map a path through obstacles [C17]. This is part of the path-planning and navigation solution onboard Perseverance [C16], one of our comparable robots.

3.2.3.4 Wheel Encoders

Wheel encoders (sometimes called rotary encoders) are sensors designed to track how many times a wheel turns, which can then be used to determine distance traveled [C19]. There are many ways to implement a wheel encoder, but we'll focus on two common, low-cost options: optical and magnetic.

Optical wheel encoders track rotations by shining a photodiode through patterns of slits in a disk mounted perpendicular to the axis of rotation. The light received on the other side of the disk oscillates between on and off, matching the pattern of slits, generating a square wave that can be used to track rotations [C20]. Optical encoders are very accurate but have serious downsides in the context of this project. Their accuracy is seriously degraded by dust and dirt, of which there will be plenty, and the optical disk is vulnerable to shock and vibration damage [C20]. For these reasons, optical encoders are likely not the best fit for our needs.

Magnetic wheel encoders make use of a rotor and stator. The rotor rotates with the shaft and contains alternating magnetic poles around its circumference. The stator contains a conductive winding that can measure the change in voltage (Hall Effect) or the change in resistance (magneto-resistive) caused by the shifting magnetic field generated by rotating the shaft [C21]. Magnetic encoders are less accurate than optical encoders, but they mitigate the two greatest weaknesses of optical encoders: they're unaffected by dust or dirt, and they contain no moving parts to be damaged by shock or vibration. Even better, Hall Effect sensors are cheap and commonly available, making them a good choice for our project.

When used in combination with other data sources and fused by a Kalman Filter (see section 3.2.3.1.3), the data from the wheel encoders can be used for state estimation, providing information on the position of our robot relative to its starting point [C22].

3.2.3.5 GPS

3.2.3.6 Barometric Pressure Sensor

Modern barometric pressure sensors contain a flexible diaphragm that bends when the external pressure is larger than the reference pressure [C38]. MEMS like these come in sealed and open designs, with the sealed version being more common in consumer electronics. The displacement of the diaphragm can be measured with a capacitive or resistive circuit. The component that we will use measures the mechanical strain involved with bending the membrane (see figure <PRESSURE_SENSOR> below). Piezoresistors are connected in a Wheatstone bridge to measure this strain accurately. The resulting signal is amplified and converted into a digital signal. Since the sensor can provide accurate barometric pressure readings, we can use it as an altimeter for the canister.

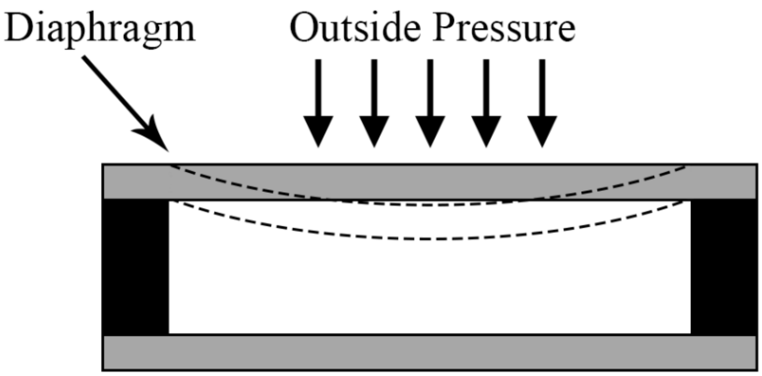


Figure <PRESSURE_SENSOR>: a pressure sensor

3.2.4 Software

3.2.4.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source set of software libraries and tools (an SDK) designed for robotics applications that has spawned an entire ecosystem of robotics developers and software modules. ROS and its community provide a wealth of resources that will allow us to rapidly expand our feature set and build a more robust and modular system.

At its core, ROS is a message-passing middleware that allows components of a system to communicate in a standardized format via a publish-subscribe model [C39]. Publish-subscribe is an asynchronous model that simplifies communication by allowing software components, called *nodes* in ROS, to “publish” messages without concern for who needs to receive it. Then, interested parties can “subscribe” to receive that published information. In this way, neither the publisher nor the subscriber needs to directly communicate their intentions, and software can be written in a manner that will “increase performance, reliability and scalability” [C40]

ROS provides more than just messaging middleware. Also included is a comprehensive suite of robotics software for debugging, introspection, and simulation. Programs like rviz and rqt allow us to visualize data in real-time and develop standard GUIs for our applications. The standard simulation package for ROS, Gazebo, allows us to quickly build high-fidelity physics-enabled simulations of our robot that can directly interface with ROS services, allowing us to develop code without a physical prototype. Our use of these packages is further discussed in sections 5.4.2 and 5.5. The SDK also provides important developer tools that enable launch, code introspection, data logging and playback, and others. By providing solutions to these common issues, ROS decreases our overall development overhead and frees us to focus on more impactful features.

The community surrounding ROS has developed a massive amount of software leveraging ROS capabilities for a multitude of tasks. Hardware drivers, state-of-the-art algorithms, and controllers, to full on SLAM implementations are all available for our use under open-source licenses.

For this project, ROS and its software ecosystem will be heavily leveraged to rapidly expand our feature set and facilitate the design of a more robust software system while decreasing our development overhead.

3.3 Strategic Components and Part Selections

3.3.1 Single-Board Computer (SBC)

The functional requirements of the rover will require the use of an onboard computer capable of handling, at the least: video streaming, sensor inputs, and control software. The size and space constraints of the rover make a single-board computer a reasonable choice. Important features were software compatibility, price, power consumption, and size. The following options were considered:

Name	Price (USD)	LxWxH (mm)	Weight (g)
Jetson Nano 4GB Dev. Kit	$99.00	100x80x29	240
RPi 4+ 8GB (FULL)	$75.00	85x56x11	46

The Jetson Nano and the Raspberry Pi 4+ (RPi) have some key similarities: the same number of GPIO pins, similar connectivity options (multiple USB, HDMI, MIPI CSI2 lanes, etc.), and similar Ubuntu Linux-based operating systems. However, the processor on the RPi utilizes the newer ARM-Cortex A72 core versus the ARM-Cortex A57 core in the Nano's Tegra module. The A72 boasts higher performance and energy efficiency [A29], making the RPi theoretically more powerful. But the opposite is true in terms of GPUs: the Nano makes use of the Maxwell GPU, which is significantly more powerful than the Broadcom VideoCore VI onboard the RPi. The GPU boosts 3D data manipulation and data science computations, meaning the Nano is more powerful in contexts where machine learning, 3D data representations, or other simple matrix operations are important [C30]. This makes the Nano theoretically more suitable for complex robotics tasks, like those required for this project.

Software Compatibility: Both the Jetson Nano and the RPi provide Linux operating systems, based on the popular Ubuntu distro, distributed by Canonical. The Jetson Nano uses a custom image of Ubuntu 18.04 called Linux4Tegra (L4T), designed to work specifically with the Tegra processors onboard the Jetson line of SBCs [C28]. Ubuntu is the operating system targeted by the Robot Operating System, ensuring that ROS software modules will run "out of the box." Unfortunately, the use of a custom image in L4T puts the burden of updating the OS on Nvidia, the company that develops the Jetson line of SBCs. As such, the version of Ubuntu that acts as the baseline for L4T is 18.04, released in 2018, and as such is out-of-date. The version of ROS that is available for 18.04 is similarly out-of-date. No such issue exists for the RPi since the corresponding image of Ubuntu is published by Canonical itself and is up to date with the newest releases.

Price: The Jetson Nano and RPi are in comparable price ranges, with the Jetson Nano being slightly more expensive. This price differential feels appropriate considering the slightly higher capabilities of the Nano in comparison to the RPi.

Power Consumption: The Jetson Nano module is software-constrained to a maximum power consumption of either 10W or 5W. Combined with the maximum 1.25W power consumption of the carrier board (minus peripherals), the total power consumption is a max 11.25W [C26]. The RPi requires 5V at 3A, meaning maximum power consumption is 15W, minus peripherals [C27].

Size: The Jetson Nano has a larger footprint than the RPi, takes up more vertical space, and is significantly (5x) heavier. Much of this weight is contained in a passive heatsink atop the Nano’s Tegra module that can be removed if another cooling solution is found, though that would increase design complexity. This significant weight difference is a serious detractor for the Nano since we’re operating under severe weight and size constraints.

The Jetson Nano Development Kit is a combination of a peripheral carrier board and an attached Jetson Nano compute module. The compute module can be detached and re-used with a custom carrier board. This would allow us to design our own carrier board PCB with only the peripherals needed for our application, potentially decreasing total weight and footprint size. Further, it allows for a custom active cooling solution to be implemented to reduce vertical height and offer better heat exchange than the passive heatsink bundled with the module. The downside is, of course, that designing this PCB would be a non-trivial endeavor and may well be beyond what is reasonably achievable in the scope of this project.

The results of this comparison are basically a wash. In terms of power consumption and software compatibility, they’re roughly equal. And while the Nano is more expensive than the RPi, it’s also more performant. The greatest detractor to the Nano is the footprint of its packaging and its weight, but these can both be potentially ameliorated by designing a custom carrier board. Frankly, both options would likely perform equally as well as the SBC onboard the rover, despite their differences. Ultimately, the Jetson Nano was selected due to availability and price: our team already had a Jetson Nano available to us, and so it was free.

3.3.2 Drivetrain

These are the parts that our rover will use for movement, including motors, motor controller, and wheels.

3.3.2.1 Motor Controller

Movement of the rover will require providing power and control inputs to multiple brushed DC motors. To handle this, the following brushed DC motor controllers were considered:

Name	Price (USD)	Interface	LxWxH (mm)	Mass (g)
Sabertooth dual 5A motor driver	$59.99	Analog, R/C, Serial	45x30x14	19g
Cytron 10A Bi-Directional DC Motor Driver	$16.99	PWM	85x62	<30g

The Sabertooth driver offers a slew of advanced features like regenerative braking, packetized serial commands, hardware DIP switch configuration, and thermal protection. Further, it's total footprint is roughly half that of the Cytron driver. However, its 3.5x more expensive, and many of those advanced features can safely be foregone for a more affordable option. The Cytron driver is less advanced, not offering regenerative braking or DIP switches to configure it but is significantly less expensive. Also, it can be interfaced with using simple PWM signals, meaning we can use any of our numerous GPIO pins to interface with it. While the footprint is 2x the that of the Sabertooth, the price differential makes it worth it. Additionally, the Cytron driver is capable of twice the sustained current and although our current motors don't go anywhere near that 10A continuous limit, if we must make a change to higher power motors it could be a major benefit to have that extra capacity.

Ultimately, the Cytron motor driver was selected. It's low price, high current capacity, and simple PWM interface more than make up for its larger footprint.

3.3.3 Sensors

In this section, we will cover our part selections for the sensors on the rover. These sensors will provide the sensor streams outlined in section 5.4.2.

3.3.3.1 Camera

Transmitting live video from the rover to a ground station is a requirement for this design [R4.4], so a camera to sit onboard the rover was necessary. While many factors were considered for this selection, the most important factors were capabilities, price, compatibility with selected SBC, and the size of the package. The following options were considered:

Name	Part #	Price	Module	LxWxH (mm)
Raspberry Pi Camera Module V2	913-2664	$25.00	IMX219	150x25x15 (L includes cable)
Leopard Imaging Camera - 136-degree FOV	LI-IMX219-MIPI-FF-NANO-H136	$29.00	IMX219	25x23x9
Waveshare IMX219-83 Stereo Binocular Camera	IMX219-83	$44.99	IMX219	85x24x15

Capabilities: In terms of actual video quality and resolution, all of these choices make use of a Sony IMX219 8MP module and should be effectively indistinguishable. The RPi Camera and the Leopard Imaging Camera are both standard machine vision RGB cameras, providing a video feed and nothing else. The Waveshare binocular camera is a stereo camera, providing stereo vision via two identical, fixed-interval camera modules, opening the door for 3D volumetric information along with the video feed (see section 3.2.3.3). Further, the Waveshare camera comes with a built-in IMU, the ICM20948 [C31], which could replace or augment the stand-alone IMU(s) intended for state estimation. However, the use of a stereo camera will likely require significant effort in software integration, as no off-the-shelf libraries seem readily available.

Price: The RPi and Leopard Imaging cameras are similar in price ($25 vs. $29, respectively), while the Waveshare camera is almost double the price of the other options. This price delta seems reasonable, since the Waveshare offering combines two modules in one package. Further, with much of our costs being subsidized by using already-acquired parts, the inflated price is less of an issue than it otherwise might be.

Compatibility: All of the options that were considered for this project claim off-the-shelf compatibility with our SBC selection, the Jetson Nano, through the MIPI-CSI interface. This is valuable as it doesn’t require use of USB ports, saving them for other functions as needed. An important note: the Waveshare camera uses both available MIP-CSI ports on the Jetson Nano, whereas the other options use one port each.

Size: The RPi Camera is the smallest of the components considered (excluding the length of the cable), with the Leopard Imaging camera taking up slightly more space with a large, fixed lens attached. The Waveshare is undoubtedly the largest of the options, with two camera modules and supporting electronics on the package. However, the size difference is mostly in the length of the module which will run parallel to the longest axis of our rover, which should mitigate some of its effects.

Ultimately, the Waveshare Stereo Camera was selected despite its greater price due to providing significantly more capability and data sources via the same interface, with the only practical trade-off being a (relatively) small size delta.

3.3.3.2 Inertial Measurement Unit (IMU)

A prerequisite for more advanced state and environmental modelling is a data stream that provides robot pose and orientation information. An inertial measurement unit (IMU) can provide this data stream. The following options were considered:

Name	Part #	Price (USD)	Interface	Module(s)
BerryIMUv3-10DOF	BIMULSM9DS0	$27.95	SPI, I2C	LSM6DSL, LIS3MDL, BM388
SparkFun 9DoF IMU Breakout	SEN-13284	$15.95	SPI, I2C	LSM9DS1
SparkFun 6 Degrees of Freedom Breakout	SEN-18020	$11.95	SPI, I2C	LSM6DSO

Ultimately, the SparkFun 9DoF IMU was selected. Many of these modules are near-identical and effectively provide the same information (+/- some accuracy) across the same interface, so our selection was based on the immediate availability of the SparkFun 9DOF module, which we already had on hand.

3.3.3.3 GPS Module

A GPS module is needed both for retrieval of the Payload Canister and for providing an absolute, world-fixed position to augment our rover state estimation. The following options were considered:

Name	Price (USD)	Interface	Module	LxWxH (mm)
GPS Module GPS NEO-6M	$11.59	UART, USB, SPI, DDC (I2C)	GT-U7 module (Clone of NEO-6M)	27x28
SparkFun GPS Breakout	$39.95	UART, DDC (I2C)	SAM-M8Q	42x42 (approx.)

The GT-U7 module in the first option (referred to as GT-U7 from here on) appears to be a clone of the u-blox NEO-6M module. Included on the GT-U7 chip is an IPEX antenna connector, allowing an (included) external antenna to potentially be mounted to the outside of the rover. The chip provides a micro-USB interface, as well as a set of pins for data and power. The presence of pins means that we can keep USB ports on the SBC free for other purposes. The data sheet provided by the manufacturer of the GT-U7 is made up primarily of screenshots from the NEO-6M datasheet. The NEO-6M datasheet claims a horizontal position accuracy of 2.5m and a heading accuracy of 0.5 degrees

[C32]. This GT-U7 chip can be interfaced using DDC, which appears to be the u-blox implementation of the I2C synchronous serial bus.

The SparkFun GPS Breakout makes use of a SAM-M8Q GPS module with an embedded (directly on the module) patch antenna. Like the NEO-6M, the SAM-M8Q datasheet claims a 2.5m horizontal position accuracy [C33]. Like the GT-U7 module that was also considered, the SAM-M8Q breakout provides access to pins for data using the u-blox I2C implementation

The SAM-M8Q breakout has a larger footprint than the GT-U7 option but doesn't require the use of an external antenna. In practice, it may be that the use of an external antenna isn't necessary to get a proper GPS signal in our outdoors environment. In that case, the SAM-M8Q breakout would likely be a better choice.

As both the SAM-M8Q and NEO-6M (GT-U7) are u-blox modules, both can be accessed and configured via the proprietary u-blox u-center software. While the GT-U7 provides a micro-USB port for this purpose, the SAM-M8Q does not. However, as the GT-U7 is a clone of the NEO-6M, there is no guarantee that all (or any) of the features of u-center center will be supported for it.

Both options have identical stated position accuracy, and the importance of an external antenna hasn't been rigorously evaluated and therefore cannot be used as a discriminating criterion. Additionally, they use the same interfaces and can (theoretically) be configured via the same u-center software. Ultimately, the GT-U7 option was selected due to its lower price and smaller footprint.

3.3.4 RF Communications

3.3.4.1 LoRa TX/RX

A small, modular solution is needed for the remote control and communication with the rover. Initially, the ESP32 module was considered, as it offered the options of using the 802.11 wi-fi protocol or Bluetooth for communications. It is common in hobbyist projects and interfaces easily with microcontrollers and single board computers like the Raspberry Pi or Jetson Nano. However, 802.11 Wi-Fi operates only at a very short range, completely ruling it out as the rover is expected to operate at distances up to 1 km.

As Wi-Fi could not meet our requirements, a long range, low power solution was needed. A LoRa module, which uses the LoRa (Long Range) communications protocol, seemed like an optimal choice. LoRa can operate on several bands in the US, without requiring any radio licenses or permission from the FCC. UHF (ultra-high frequency) bands seemed the most practical. The 70-centimeter band (440 MHz) seemed like an appealing initial choice, due to its common use in various remote-control applications. However, the rocket will be using the 70 cm band for transmitting telemetry data, eliminating it as an option. The Arcturus Rocket teams also specified that frequencies in the 1-2 GHz range should be avoided to limit interference. The decision was made to use 915 MHz, an ISM (industrial, scientific, and medical) band, which offers several advantages over lower frequencies, as there are off-the-shelf options available which meet size constraints. A choice was made to use the RYLR896 LoRa Transceiver. These small, six-pin modules offer two UART pins, for input and output, which makes connection with other rover hardware very simple.

Name	Price (USD)	Interface	Module	LxWxH (mm)
Reyax RYLR896 LoRa RC Transceivers (2)	$48.93	UART	Reyax RYLR896 Module Sx1276	40x15x5mm
LILYGO TTGO dev module WiFi+Bluetooth (2)	$26.56	I2C	LILYGO TTGO dev module	51.52x25.04x8.54mm

The RYLR896 module is based around the LoRa(Long Range) modulation technique. It uses a type of proprietary chirp spread spectrum modulation that allows for low power, long range communication, extending up to several miles in ideal conditions. It has an operating voltage of 2.2-3.6 V.

Trade Study

	Price	Range	Size	Overall
Weighting	0.1	0.7	0.2	
Reyax RYLR896 LoRa RC Transceiver	4	5	5	4.9
LILYGO TTGO dev module WiFi+Bluetooth	5	1	4	2

3.3.4.1 Video TX

For live video transmission, higher frequencies must be used. 5.8 GHz was chosen, as it is a common frequency used in drone video transmission. The TS5823 Module has a range of more than 1000m in an open space, meeting the design requirements. It will require its own small heatsink to prevent overheating. It has an operating voltage range of 7-24 V.

Name	Price (USD)	Module	LxWxH (mm)
Hyperion TS5823 AV Transmitter Module	$8.99	Hyperion TS5823 AV Transmitter Module 600 mW	40x23x8mm
Heat Sink	$3.00	N/A	40x20x10 mm

3.3.5 Canister Components

3.3.5.1 Microcontroller Board

A microcontroller board is needed within the payload canister in order for the rover to be deployed safely after the canister lands. The microcontroller board will be in charge of many different tasks, such as reading barometric pressure, reading from the motor encoder, and interfacing with the motor driver. The most important factors in choosing a microcontroller board is its price, communication interfaces, power consumption, size, and mounting features. The following microcontrollers boards were considered:

Name	Price (USD)	MCU	Input Voltage (V)	Logic Voltage (V)	LxW (mm)	Weight (g)
Arduino Uno	$23.00	ATmega328P	7 - 12	5	68.6 x 53.4	25
Arduino Nano	$20.70	ATmega328	5 - 12	5	45 x 18	7
Teensy 4.0	$19.95	ARM Cortex-M7	5	3.3	35.56 x 17.78	2.8

MCU: When comparing microcontrollers, the Arduino Uno and Arduino Nano are nearly identical. The only major difference is that the Arduino Uno's ATmega328P has a "Picopower" feature that allows it to run with low power consumption. On the other hand, the Teensy 4.0 has an ARM Cortex-M7 has a clock speed of 600 MHz compared to the Arduino's meager clock speeds of 16MHz. This performance advantage would be noteworthy if it wasn't for the fact that it is entirely unnecessary. The faster clock would let us generate faster pulses leading to a faster motor, but high RPMs aren't a requirement for this project. The clock rates of the Arduino MCUs are capable enough of driving the stepper motors. There is no winner as all MCUs are sufficient.

Price: Factoring in the prices of the boards, the Teensy 4.0 seems to be the best value. However, since these boards are all nearly identical in price, there shouldn't be a winner in this category.

Size: After sizing up the boards, the Teensy 4.0 wins in the size department with a size of 35.56 x 17.78 mm and weight of 2.8g. The Arduino Nano comes in at a close second with dimensions of 45 x 18 mm and a weight of 7g. The Arduino Uno is the clear loser with a size of 68.6 x 53.4 mm and weight of 25g. The Teensy 4.0 and Arduino Nano are currently tied with the Arduino Uno in last place.

Input / Output: When it comes to I/O, the boards must be capable of I2C or SPI as we will need to connect our barometric pressure sensor. As it stands, all boards support these interfaces, so there is no victor here.

Power consumption: Out of the box, the Arduino Uno consumes 50 mA. The Arduino Nano consumes 20 mA. The Teensy 4.0 uses 100 mA when running at 600 MHz, but a downclock can make it as efficient as the Nano. These results aren't very important as all three boards have power-saving modes that allow them to last much longer.

Mounting: Mounting features are the most important metric when comparing these boards. The Arduino boards have M1.6 and M3 mounting holes located at the corners of the boards. Unfortunately, the Teensy 4.0 does not have any holes for easy canister integration. To mount the Teensy 4.0, a special mounting structure would have to be designed and 3D-printed. The ease and security with mounting the Arduinos gives them the edge in this section.

Conclusion: In the end, our choice was between the Arduino Nano and the Teensy 4.0. The Arduino Uno was ruled out because it's form factor, power consumption, and weight are unnecessarily large. Additionally, the Uno has too many features that are simply not needed. Deciding between the Arduino Nano and Teensy 4.0 was a little harder, but ultimately we chose the Nano as the Teensy 4.0 can't be secured to our electronics plate with bolts.

Trade Study

	Price	Clock Speed	Functionality	Power Consumption	Size / Weight	Canister Integration	Overall
Weighting	0.2	0.1	0.2	0.1	0.2	0.2	
Arduino Uno	4	2	5	2	1	5	3.4
Arduino Nano	4	2	5	5	4	5	4.3
Teensy 4.0	4	5	5	4	5	1	3.9

3.3.5.2 Barometric Pressure Sensor

A barometric pressure sensor is vital for payload deployment due to its ability to act as an altimeter. The most important factors in choosing a barometric pressure sensor is its price, communication interfaces, and accuracy. The following barometric pressure sensors were considered:

Name	Price (USD)	Interface	Logic Voltage (V)	Accuracy (m)	Module
Bosch BMP-388	$9.95	I2C, SPI	1.2 - 3.6	± 0.5	BMP-388
Freescale MPL3115A2	$9.95	I2C	1.6 - 3.6	± 0.3	MPL3115A2

Price: The price of these two sensors are both the same at $9.95, so next we look at I/O.

Input / Output: Both pressure sensors are capable of communicating with our Arduino using I2C. If we were to use the BMP-388, however, we could also hook it up with SPI or I2C.

Accuracy: When it comes to accuracy, the BMP-388 has a resolution of 8 Pascals, or ± 0.5 meter accuracy. The MPL3115A2 has a resolution of 1.5 Pascals, or ± 0.3 meter accuracy. Both are precise enough for our needs.

Conclusion: For our project, we must be able to determine if our canister has been grounded. We can’t execute canister deployment until we do that. Both sensors have proved to be accurate and fairly inexpensive. Our team has decided to go with the Freescale MPL3115A2 as it is slightly more accurate than the Bosch BMP-388 for the same price and that would be beneficial to the reliability of payload deployment.

Trade Study

	Price	Accuracy	Power Consumption	Size / Weight	Overall
Weighting	0.3	0.3	0.2	0.2	
Bosch BMP-388	4	4	5	5	4.4
Freescale MPL3115A2	4	5	5	5	4.7

3.3.5.3 Stepper Motor

Our rover deployment design revolves around using two bipolar stepper motors to “unscrew” the rover out of the canister. The most important factors in choosing a motor are its price, torque, power, and size. The following barometric pressure sensors were considered:

Name	Price (USD)	Holding Torque (oz-in)	Voltage (V)	Current (A)	LxWxH (mm)	Weight (g)	Model
NEMA 17	$9.52	31	2.8	1.33	42 x 42 x 34	230	17HS13-1334D
NEMA 11	$14.27	9.91	3.75	0.67	28 x 28 x 31.5	110	11HS20-0674D

Note: While there are a plethora of motors that can be called “NEMA 11” or “NEMA 17”, when we use this abbreviation, we are referring to specific motors that we chose with their specifications shown in the table above.

Price: Looking at prices, the NEMA 17, although being beefier, is actually cheaper than the NEMA 11. It is noteworthy to mention that the NEMA 17 is more popular than the NEMA 11, which is probably the explanation as to why that form factor is much cheaper.

Torque: Comparing torques, the NEMA 17 also surpasses the NEMA 11 in this metric. Although the NEMA 17 is three times more powerful than the NEMA 11, we only need to consider the required torque for our application. In our case, we need two motors capable of supplying 3.2 oz-in. Both of these motors are clearly sufficient in this category.

Power: Since we will use two motors, we must make sure that they can be driven by the proposed 12V 2A battery. The NEMA 11 and NEMA 17 have low voltages, but the phase current of the NEMA 17 would require a larger battery to be bought. The total power that the two NEMA 17 motors would use is (2.8V)(1.33A per phase)(2 phases)(2 motors) = 14.896W. Using a similar calculation the NEMA 11 motors would use 10.05W. Since the NEMA 11 would use less power and a cheaper battery, it beats the NEMA 17 here.

Size: The NEMA 17 has over twice the volume and weight of the NEMA 11. Keeping in mind that we need two motors, the NEMA 17 would have a mass of 0.46kg, and the NEMA 11 would be 0.22kg. The NEMA 11 wins in the size division as the NEMA 17 would take up over a tenth of our maximum allotted mass.

Conclusion: After looking at each metric, the NEMA 11 appears to be the best choice. We went with this motor because, although it is a bit more expensive, it’s low-speed torque will satisfy our use case while being lighter, smaller, and more power efficient.

Trade Study

	Price	Torque	Power Consumption	Size / Weight	Overall
Weighting	0.2	0.1	0.3	0.4	
NEMA 17	5	5	2	2	2.9
NEMA 11	4	2	4	5	4.2

3.3.5.4 Motor Encoder

The function of our system's motor encoder is to ensure that the motors move at a speed with sufficient torque to release our rover, and also to check if the motor has rotated enough to fully displace the rover. Motor encoders, also called rotary encoders, can be implemented with different technologies. Our choice will be between two CUI Devices incremental capacitive encoders. CUI Devices quadrature encoders send information to microcontrollers with the use of channels. In our case we will need to use all three channels. The most important factors in choosing a motor encoder is its price, PPR, and current. The following motor encoders were considered:

Name	Price (USD)	PPR	Current (mA)	Input Voltage (V)	Model
CUI AMT102-V	$23.00	48 - 2048	6	3.6 - 5.5	ENC-AMT102-V
CUI AMT112S	$31.00	48 - 4096	16	4.5 - 5.5	ENC-AMT112S-V

Price: In terms of price, the AMT102-V is cheaper than the AMT112S, and would save us $16.

PPR: Pulses per revolution is essentially the encoder's resolution. The AMT102-V has a max PPR of 2048, and the AMT112S has a max PPR of 4096. For our use case, the motor encoder will be attached to a low RPM motor so the pulses won't be very fast. Additionally, the slow 16 MHz clock of the Arduino Nano will not be able to receive all the pulses that the encoder transmits. This means that both encoders are equal in this metric.

Current: In the case of the AMT102-V, the power consumption is 6mA. The AMT112S uses over twice this amount, coming in at 16mA. The edge in this comparison goes to the AMT102-V.

Conclusion: Our team decided to go with the AMT102-V as the less accurate encoder meets all of our requirements while also being cheaper than the AMT112S.

Trade Study

	Price	Resolution	Power Consumption	Size / Weight	Overall
Weighting	0.4	0.2	0.2	0.2	
CUI AMT102-V	4	3	5	4	4.0
CUI AMT112S	3	5	3	4	3.6

3.3.5.5 Motor Driver

The need for a motor driver is self-explanatory - it's used to drive motors. Since we are using bipolar stepper motors, we will need a driver that can drive both coils. The most important factors in choosing a motor driver are its price, logic voltage, and current per phase. The following motor drivers were considered:

Name	Price (USD)	Current per phase (A)	Input Voltage (V)	Logic Voltage (V)	LxW (mm)	Module
TI DRV8825	$11.95	1.5	8.2 - 45	2.2 - 5.25	20.32 x 15.24	DRV8825
Allegro A4988	$5.95	1	8 - 35	3 - 5.5	20.32 x 15.24	A4988

Note: The Allegro A4988 requires an electrolytic capacitor to prevent LC voltage spikes.

Price: Comparing prices, the DRV8825 driver is twice as expensive as the A4988 driver. The Allegro A4988 is the clear winner in this regard.

Voltage: Both the TI DRV8825 and Allegro A4988 can operate with voltages greater than 8V. The DRV8825 has a higher limit or 45V compared to the A4988's 35V, but any driver that can operate with 9V is sufficient for our needs. Both drivers can also use the 3.3V or 5V logic of our Arduino Nano.

Current: The maximum amount of current per phase is an important metric as it needs to be greater than the current per phase of our motor. The Allegro A4988's current per phase stands at 1A, which is much greater than the NEMA 11's 0.67A per phase. The TI DRV8825 provides a larger margin of safety as it is capable of 1.5A per phase. Both motor drivers are viable for our project, and neither driver would overheat hooked up to our low current motors.

Conclusion: Based on their specs, both drivers seem to be almost identical. However, due to its significantly cheaper price tag, our team will be using the Allegro A4988 with our motors.

Trade Study

	Price	Compatibility	Max Current Per Phase	Size / Weight	Overall
Weighting	0.3	0.3	0.2	0.2	
TI DRV8825	2	5	5	4	3.9
Allegro A4988	4	5	3	4	4.1

3.3.5.6 Power Supply

When the ejection charges ignite, the canister's power supply will be switched on. For successful deployment, we will need to power the microcontroller and motors with enough power to last until the rover is operational. Our original design involved using two separate power systems for the MCU and motor drivers. After several considerations, we decided to power both systems using one central power source. The most important factors in choosing a power supply is its price, compatibility, capacity, and weight. The following power supplies were considered:

Original Design

Microcontroller Power:

Name	Price (USD)	Voltage (V)	Capacity (mAh)	Weight (g)
Energizer Max 9V battery (2-pack)	$6.50	9	600	45
Rayovac Rechargeable 4x AA batteries	$8.60	4.8	1350	84

Price: The price of buying a 9V battery versus four rechargeable AA batteries is so small that it can be considered negligible. Actually, the 9V comes in packs of two, so there would be an extra battery that could be used for pre-flight tests.

Compatibility: The Arduino Nano's internal voltage regulator must be supplied with 7-12V. The 9V battery fits this requirement. The Nano can also be powered through the 5V pin. In this case, the four AA batteries in series would sum to 4.8V which is enough to provide power to our components.

Capacity: When it comes to capacity, the Rayovac batteries would last twice as long compared to the 9V. However, we do not need to power the microcontroller for very long. Given that the components and microcontroller will consume less than 50mA, the Energizer 9V wouldn't deplete until after several hours.

Weight: The four rechargeable batteries are double the weight of the 9V. They will also need a larger case which would increase that figure. The 9V is the victor in this section.

Conclusion: With these comparisons in mind, we initially decided to use the 9V battery for our microcontroller power supply. Although more short-lived, the 9V would have allowed our canister deployment system to function reliably at a slightly lower cost and at half the weight of the rechargeable batteries.

Motor Power:

Name	Price (USD)	Voltage (V)	Capacity (mAh)	Weight (g)	Model
Bioenno Power LiFePO4 Battery	$34.99	9V	3000	300	BLF-0903W

Price: Although a little steep, the Bioenno battery at $34.99 is a complete package with several useful features such as internal cell balancing, and will not create EMI/RMI. These are both due to its protection circuit module.

Compatibility: This battery will be adequate with both motor drivers as it meets the 8V+ requirement, and the current discharge rate is a safe 6A.

Capacity: The maximum current that the motors can draw is (0.67A per phase)(2 phases)(2 motors) = 2.68A. This battery would last about an hour before needing a recharge.

Weight: The battery is also less than a tenth of the canister's total weight which is beneficial for us.

Conclusion: The Bioenno battery has a lot of nice features, but has a large price tag. Before our revision, this battery seemed to be the best for driving our motors.

Revised Design

Name	Price (USD)	Voltage (V)	Capacity (mAh)	Weight (g)	Model
LiFePO4 18650 Rechargeable Cell (3)	$4	3.2	1500	41	IFR18650EC-1.5Ah

Price: For three cells and a battery holder (~$2), the total cost would be ~$14. This is much cheaper compared to our previous design.

Compatibility: In series, three cells would be capable of outputting approximately 9.6V. Considering that 8V is the minimum for our motor drivers and these batteries have a 5A discharge rate, these LiFePO4 batteries will be sufficient for the task.

Capacity: If we sum the maximum current draw from the MCU, the MCU's sensors, and both NEMA motors we get (~50mA from MCU and components) + (2.68A from motors) ≅ 2.73A. With 1500 mAh, we should expect to achieve at least a half hour battery life. After realizing that the canister wouldn't need to operate for very long, switching to a power supply with less capacity seemed logical.

Weight: Three batteries, each 41g, leads to a total weight of 123g. When we compare this to the 345g of the original design, we see a weight reduction of over 60%.

Conclusion: Looking back at both designs, it is clear that the new design beats the old one in nearly every category. Although it lacks the protection module circuit from the initial design, the revised design is smaller, inexpensive, and more suited to the task. Since the LiFePO4 18650 cells were more economical and satisfied our requirements, we decided to use them.

Trade Study

	Price	Compatibility	Battery Life	Weight	Overall
Weighting	0.4	0.2	0.1	0.3	
Initial Design	2	5	5	2	2.9
Final Design	4	5	3	4	4.1

3.3.6 Rover Power Subsystem

3.3.6.1 Lithium Polymer Batteries

The choice of batteries for the rover is significant, as the rover will be subjected to extreme stresses during launch and deployment. Lithium Polymer batteries are small and energy dense, but run a very significant risk of thermal runaway leading to violent explosion. Battery cells rupturing during landing is a very real possibility, leading to internal shorts, thermal runaway, and a fiery death for the rover. A safer, yet less energy dense solution is lithium iron phosphate (LiFePo) batteries.

LiFePo batteries were initially considered, but limited availability and difficulties with selecting the proper BMS meant the standard lithium polymer cells were a better option. For this design, LiPo batteries were ultimately chosen for a simpler design.

The rover will be subjected to significant stresses during the rocket launch and deployment, making thermal runaway a very real possibility. Initially, 14430 Cells were considered, but they have a much lower capacity (roughly 650 mAh per cell) than 18650 cells for roughly the same size. The final decision was made to select LiPo 18650 cells.

Name	Price (USD)	Voltage range	Weight	dimensions (mm)
LiFePO4 Rechargeable 18650 Cell (4)	$10.50	3-3.6 V per cell	80 grams	18.3 x 65 mm per cell
LiFePO4 Rechargeable 14430 Cell (4)	$7.70	3-3.6 V per cell	50 grams	14 x 43.40 mm per cell
LiPo 2-Cell (2)	$15.98	14-15.2 V	70 grams	20 x 45 mm

Trade Study

	Price	Energy Density	Safety	Availability	Overall
Weighting	0.1	0.3	0.2	0.4	
LiFePO4 Rechargeable 18650 Cells	5	3	5	2	3.2
LiFePO4 Rechargeable 14430 Cells	5	2	5	2	2.9

Lithium Polymer cells	4	5	2	4	3.9

3.3.6.2 Battery Management System

The choice of battery management system is critical for proper operation of the rover. The cells must be balanced, and protected from overcharge and undercharge conditions. Texas Instruments (TI) offers a wide variety of integrated circuits for battery protection. Of initial interest is the BQ77216 series, which offer protection for systems of 3 up to 16 series cells. This module can be integrated with an external thermistor, to allow for temperature protection. However, the BQ77216 does not offer cell balancing.

The BQ77915 offers protection for 3 to 5 series cells, adequate for our rover design. Notably, the BQ77915 also offers cell balancing making it a more appealing option than the BQ77216.

However, due to parts shortages, we are unable to procure chips for assembling our own BMS and opt to use an off-the-shelf model instead.

Name	Price (USD)	Module	LxWxH (mm)
BQ77216	$2.35	BQ7721600PWR	34 x 7.7 x4.4
BQ77915	$1.30	BQ7791500PWR	34 x 7.7 x4.4
Anmbest 7.4 V BMS	$10.49	N/A	127x70x3.81

Trade Study

	Price	Efficiency	Features	Size	Overall
Weighting	0.1	0.2	0.6	0.1	
BQ77216	5	5	3	2	3.5
BQ77915	5	4	4	2	3.9
Anmbest 7.4 V BMS	4	4	4	5	4.1

3.3.6.3 DC to DC Conversion

Texas Instruments WeBench is an incredibly useful resource for designs on DC-DC converters. Our initial plans for using TPS series chips had to be scrapped due to the global chip shortage, and we simply used the designs where parts were available.

Our design will require several DC-to-DC converters, both buck and boost converters, to step up and step down the voltages for different rover modules.

The most important criteria in determining which design to use is minimizing space and preventing interference with the RF communications. These designs have negligible interference in the frequency bands of importance, and they all have a small footprint. Care must be taken in designing the PCB for the rover and minimizing space between the different components.

Name	Price (USD)	Module	Area
TPS62901(buck converter) Old design	$1.94 (Total BOM)	TPS62901RPJR	49mm^2
LMR62014(boost converter) Old Design	$1.36 (total BOM)	LMR62014XMF	53 mm^2
LMZM23600 (buck converter) Old Design	$2.23 (total BOM	LMZM23600V3SILR	41 mm^2
LM2700LD (Boost Converter) New design	$1.65	LM2700MTX-ADJ/NOPB	439 mm^2
LM21305(Buck Converter) New design	$2.42	LM21305SQXNOPB	252 mm^2
LM51420 (Buck Converter, New Design)	$0.60	LM51420YDDCR	131 mm^2

3.4 Possible Architectures and Related Diagrams

A possible design for this system is composed of four subsystems: RF communications, a payload canister to contain the rover during deployment and descent, the rover itself, and a ground station that communicates with the rover. This initial conception of the system is summarized in figure <ARCH_REV_1>.

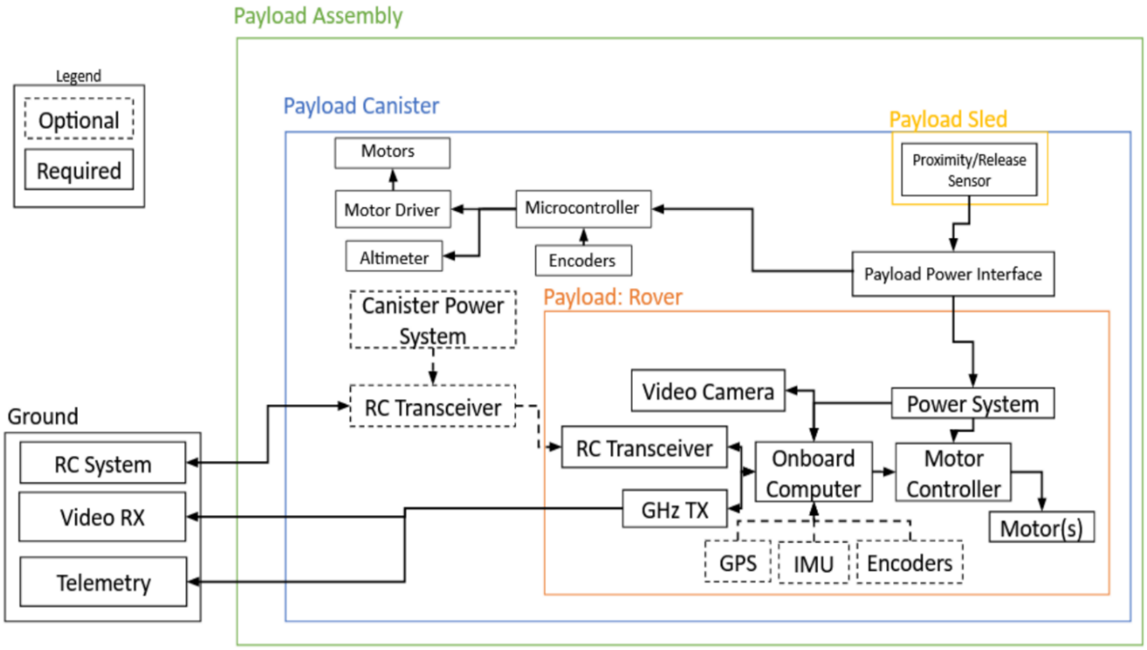


Figure <ARCH_REV_1>: initial architecture of system

3.5 Parts Selection Summary

What follows is a table that summarizes all the parts selections made in section 3.3:

Table <PARTS_SELECTION>: Summary of selected parts

Name	Price	Dimensions (mm)	Weight(g)	Relevant Specs
Jetson Nano 4GB Dev. Kit	$99.00	100x80x29	240	
Waveshare IMX219-83 Stereo Binocular Camera	$44.99	85x24x15		
~~SparkFun 9DoF IMU Breakout~~	~~$15.95~~			~~SPI, I2C~~
GPS Module GPS NEO-6M	$11.59	27x28		UART, USB, SPI, DDC (I2C)
Reyax RYLR896 LoRa RC Transceivers (2)	$48.93	40x15x5		UART
Hyperion TS5823 AV Transmitter Module	$8.99	40x23x8		
Arduino Nano	$20.70	45x18x18.6	7	
Freescale MPL3115A2 (Barometric Pressure Sensor)	$9.95	18x19x2		I2C, 1.6 - 3.6 V logic voltage, ± 0.3 m accuracy
LiFePO4 Rechargeable 14430 Cell (3)	$5.80	14 x 43.40 (per cell)	50	9-10.8 V supply
BQ77915 Battery Management IC	$1.30	34 x 7.7 x4.4		
Allegro A4988 (Motor Driver)	2x $5.95	20.32 x 15.24		1A current per phase, 3 - 5.5 V logic voltage
NEMA 11 Motor	2x $14.27	28 x 28 x 31.5	110	9.91 oz-in, 3.75V, 0.67A per phase
CUI AMT102-V	2x $23.00	28.77 x 43.38	20.5	48 - 2048 PPR, 3.6 - 5.5 logic voltage
Zeee 4600 Mah battery	$27.89	50 x 65.0x100	41	1500 mAh, 3.2 V
Sabertooth Dual 5A driver	$59.99	45 x 40 x 13 mm	50	5 A 6-18 V

4.0 Related Standards and Realistic Design Constraints

Standards are established guidelines to follow in design or implementation. Standards are important in engineering to ensure that a safe, reliable product is designed. A universal standard means that a product can be adaptable and universal and can be integrated into larger systems without difficulty. Standards ensure that our rover design will function reliably and safely, based on tried and tested industry best practices.

4.1 Standards

4.1.1 IEEE 830

This project will require the development of software to perform complex tasks in control and sensor integration. Following IEEE 830, the Recommended Practice for Software Requirements Specifications (SRS), our rover's software will follow industry best practices for software development.

Section 5.2.1.1 System Interfaces

This section states how we should list each system interface and identify the functionality of the software used to accomplish the requirements for the system. For our rover designs we will be using a Linux station to make the programs necessary to control the rover. The Linux system will be Ubuntu 18.04.

Section 5.2.1.2 User Interfaces

This section outlines how we should specify the logical characteristic of each interface between software and its users such as window layouts, screen formats, content of reports, and availability of programmable function keys. As well as a list of does and don'ts of how the system will appear to the user. For now, the system interface will consist of a custom GUI that will display video output, GPS data, and text log of inputs and outputs that have occurred, and either a button input or a physical controller input.

Section 5.2.1.3 Hardware Interface

The section explains how we should specify the logical characteristics of each interface between the software product and the hardware components of the system such as number of ports, instruction sets, etc.

Section 5.2.1.4 Software Interfaces

This section outlines how we should specify the use of required software products such as an operating system, data management systems, or mathematical packages. More specifically we need to give a name, mnemonic, specification number, version number, and source. One of the tools used to make the GUI is Qt Designer

Section 5.2.1.5 Communication Interfaces

The section is about the various types of interfaces to communication such as local network protocols. Now our design will consist of a duplex 915Mhz LoRa Rx and Tx for communication with the rover to send and receive data such as control input. And our rover will consist of a 5.8GHz signal using a Hyperion TS5823 Module to send video data from the rover to the Skydroid FPV Receiver and this signal for the video will be simplex

Section 5.2.1.6 Memory Constraints

This section is about the limits we will have on memory. For the rover we will have 4GB of memory with the Jetson Nano and it is very unlikely that we'll exceed the limits on memory.

Section 5.3.5 Design Constraints

This section is about if we have any design constraints that could be imposed by other standards, hardware limitations, etc. For the rover we are limited by power considerations, and we are limited to using a small single board computer that has limited functionality compared to a conventional personal computer. Also, the rovers size limits us to the various components such as the motors for the wheels, and the type of camera we can use.

4.1.2 IEEE 1625

The rover will be powered by Lithium Polymer batteries. Improperly charged Lithium-ion batteries of any type are a fire hazard. IEEE 1625, The IEEE Standard for Rechargeable Batteries used in Portable Computing, offers a good starting point for battery safety. Many of the guidelines in the first 5 sections refer to the manufacture and testing of battery cells, irrelevant to this project as we will be purchasing standardized battery cells and the manufacture will not be under our control. Section 6 offers good general advice for battery safety in electronic design. The relevant sections are outlined below.

Section 6.3 External Short Circuit Precautions

This section outlines precautions that can prevent a short circuit condition in the battery cells. Precaution 6.3.1 indicates that the battery pack must be able to limit current in the event of an external short circuit in the external device electronics. Precaution 6.3.2 recommends protective redundancy, with at least two separate methods for current limiting. Precaution 6.3.3 encourages the use of protective circuits for current limiting. This will be one of the functions handled by the battery management system.

Section 6.4 Overheating Precautions

This section outlines precautions made to prevent overheating of the battery pack. Precaution 6.4.1 recommends use of the temperature ranges provided by the battery supplier. 6.4.2 recommends ambient temperature consideration. Our design will use the temperature ranges provided by the manufacturer to prevent over-temperature conditions.

Section 6.5 Overcharge Precautions

This section outlines precautions made to prevent overcharge of the battery pack. Precaution 6.5.1 recommends using the maximum voltage and current limits given by the battery manufacturer. Precaution 6.5.3 recommends at least three independent overcharge protection functions. Precaution 6.5.5 recommends overvoltage protection for each cell. These functionalities will be handled by the battery management system (BMS).

Section 6.6 Over-discharge Precautions

This section outlines precautions made to prevent over-discharge of the battery pack. Precaution 6.6.1 recommends following the battery manufacturer guidelines for minimum voltage. Precaution 6.6.2 suggests that there be at least one undervoltage protection circuit. The battery management system will work to prevent over-discharge situations.

Section 6.7 Overcurrent Precautions

This section outlines precautions made to prevent an overcurrent condition in the battery pack. 6.7.1 recommends following manufacturer guidelines on maximum current. 6.7.2 recommends adding overcurrent protection circuitry. 6.7.4 recommends having some redundancy, with two overcurrent protection functions.

Section 6.8 Mechanical Stress Precautions

This section outlines precautions that should be taken to avoid mechanical stress. As size is a major constraint, consideration must be made that the battery cells still maintain adequate spacing between cells, outlined in 6.8.1. Movement and vibration should be minimized, as outlined in 6.8.2. Proper insulation should be made between points where a short is possible, as outlined in 6.8.4.

Section 6.12 Precautions for cells connected in series and/or parallel to form a battery pack

Since this design will have three batteries in series, it is prudent to follow these precautions. Section 6.12.1 stresses that the cells should be matched. 6.12.3 outlines the importance of keeping fresh cells together and making sure the cells are made by the same manufacturer. Our batteries will be sourced from the same manufacturer as we will be following the standard procurement process.

4.2 Realistic Design Constraints

4.2.1 Aerojet Constraints

The following constraints are from the Arcturus rocket team and are subject to change:

- Payload canister must fit in the rocket body tube, which has an inner diameter of 15.24 cm.

- Rover will need to fit in the payload canister which, due to the wall, has a maximum inner diameter less than 15.24 cm, and a maximum length of 40.64 cm.
- Rover and payload canister must not exceed a total mass of 4.31 kg.
- Rover must meet a minimum required mass of 1 kg.
- Payload canister must have an eye bolt affixed at its top.
- The payload canister must have an internal power supply that will arm the rover during deployment.
- RF communication in the 420 MHz – 450 MHz and 1 GHz – 2 GHz ranges are forbidden.

4.2.2 Economic and Time Constraints

Our team will be working under the assumption that the maximum allotted budget from Aerojet Rocketdyne Coleman Aerospace is $500. Many components will have to be selected based on its price rather than functionality as this is a major constraint.

Our team will have until June 2022, the date of the FAR 1030 competition, to design and manufacture a working product. Since we don’t have an exuberant amount of time to prototype and test our designs, some additional features might have to be discarded. Gantt charts must be created to assist the team in scheduling deadlines.

4.2.3 Environmental, Social, and Political Constraints

- Landing site will be in the California Mojave Desert, which means sand, dust, and windy conditions. As such, the rover will require environment-proofing.
- Sandy, irregular terrain will require greater motor torque, clearance, and other “all-terrain” considerations.
- Rover heat generation because of the compact placement of the rover’s electronics and summer desert heat will need to be reduced as much as possible.
- Forces due to the rocket’s acceleration and parachute shock will have to be considered when designing the structure of the payload canister and rover. Strains due to force in the axial direction will need to be minimized through proper material selection. Electrical components will have to be sufficiently harnessed as lateral forces can lead to malfunction.
- The ejection charges will subject the payload canister to hot gases that could potentially create major deformations in the payload canister’s structure.
- The exact longitudinal distance travelled by the Arcturus rocket from the launch pad cannot not be determined until after launch. However, it is estimated that the rover will have to be able to communicate 1000 - 2000 feet from the launch pad.

4.2.4 Ethical, Health, and Safety Constraints

This project will involve the handling and operation of Lithium Polymer (LiPo) batteries. When a lithium battery is damaged, a short circuit can occur between the anode and cathode, causing a dramatic temperature spike, which triggers an exothermic reaction in a positive-feedback effect known as “Thermal runaway”. Hot gases produced in the rapid chemical reaction will cause the battery to explode, risking severe burns and shrapnel injuries.

4.2.5 Manufacturability and Sustainability Constraints

When designing the rover and payload canister, ease of manufacturing is critical. The parts selected need to be easily accessible. The team will need to be able to adjust the design of the rover and payload canister in the case of product unavailability.

Various materials will have to be milled or 3D-printed to our design specifications in order to construct the payload canister and rover. The use of the University of Central Florida’s manufacturing labs and Robotics Lab will provide the means to accomplish this.

Payload canister and electronics will be subjected to rocket launch, deployment, descent, and landing conditions: will need to be capable of withstanding these and remaining operational.

Metals, plastics, or other materials should never be subjugated to a stress that exceeds its elastic range during testing. If that is the case, the material should not be selected for use and an alternative should be considered.

Landing orientation will be difficult to control due to many unknown environmental factors such as wind. The payload canister’s deployment mechanism needs to be able to eject the rover despite these complications.

The rover electronics are very tightly packed, so undissipated heat can cause irreversible damage to our components, whether we know it or not, and lead to abnormal behavior. We must make sure that the rover’s internal ambient temperature doesn’t exceed the upper limit of the electronics commercial temperature rating, which is typically 70°C.

5.0 Project Hardware and Software Design Details

This section contains the design and implementation details of our overall system, as well as details of each of four subsystems: RF communications and power, canister, rover, and ground station.

5.1 Initial Design Architectures and Related Diagrams

The final architecture of our system is summarized in figure <ARCH_REV_FINAL>.

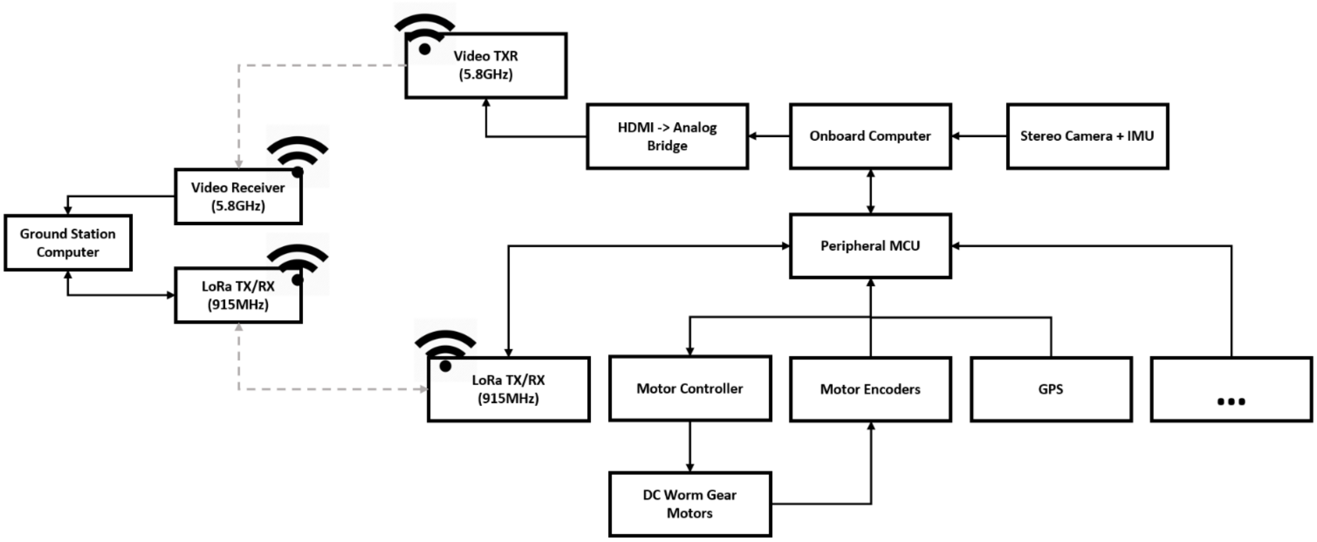


Figure <ARCH_REV_FINAL>: the final block diagram summarizing our system design

The breakdown of team responsibilities is summarized in figure <TEAM_RESPONSIBILITIES>.

Task	Justice Cordova	James Ellison	Wesley Fletcher	Joshua Kissoon
System Design	S	S	P	S
Power Systems	P			S
RF Communications	P	S		
PCB Design	P			S
Embedded Software			P	P
Ground Station (GUI)		S		P
Rover Electronics			P	
Rover Software			P	
Prototype Construction			P	

Figure <TEAM_RESPONSIBILITIES>: Table of team responsibilites

5.2 Subsystem - RF Communications and Power

This section contains design and implementation details surrounding our RF communications modules, i.e., LoRa transceiver and GHz video transmitter, and our power electronics.

5.2.1 RF Comms. Overview

The rover will be communicating with the ground station on two frequencies: 915 MHz and 5.8 GHz. The 915 MHZ frequency will be used to receive remote control inputs from the ground station and send back telemetry data on the rover's status to be displayed at the ground station. The setup will be duplex. The rover will send relevant data to the ground station every 30 seconds in a short transmission from the LoRa transceiver. When the ground station sends commands to the rover, the LoRa module will relay those commands to the single board computer, which will respond accordingly by turning the wheels. Video transmission will be simplex and will go from the rover to the ground station. The LoRa RYLR896 uses a system of AT commands to configure the transceivers and to send information. The receiver and transmitter will each be assigned addresses allowing them to communicate with each other. The ground system software will translate

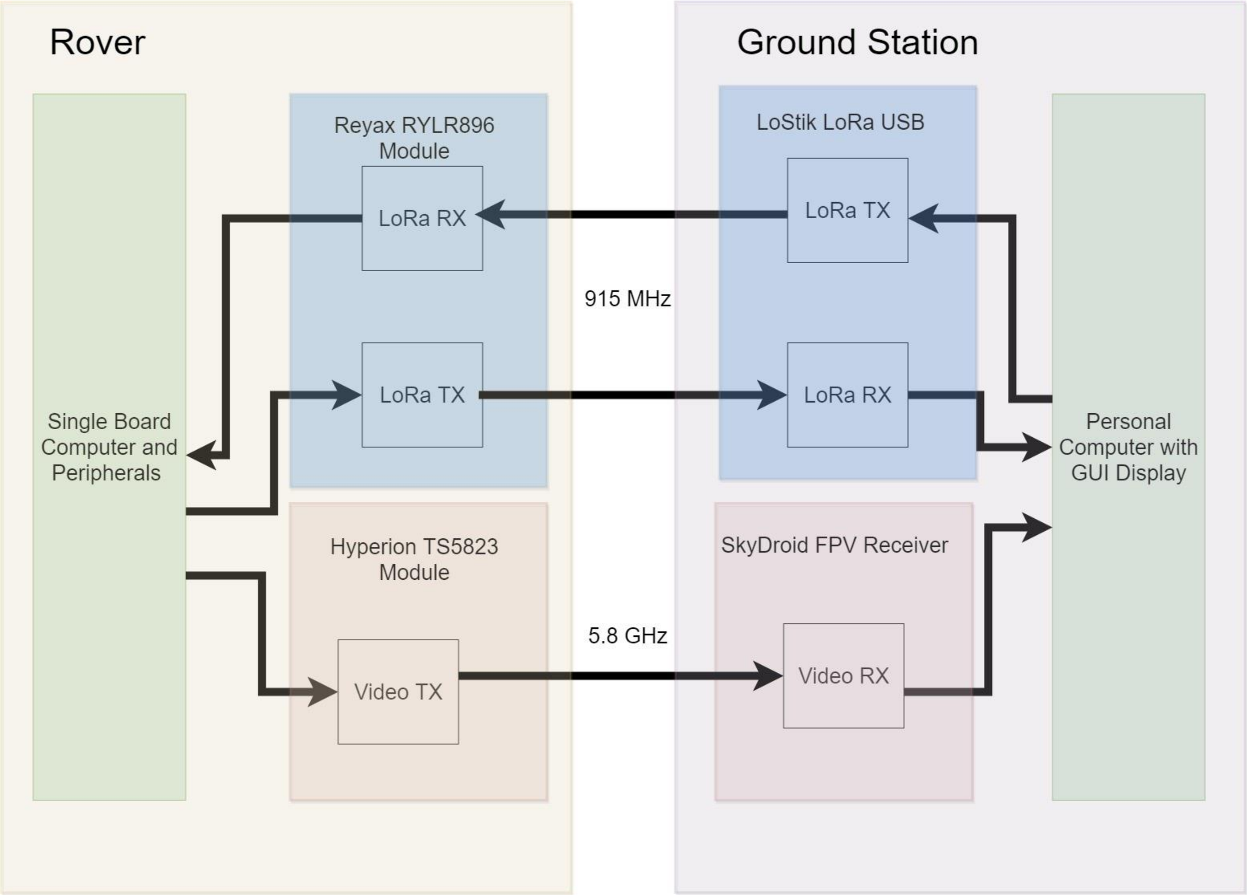


Figure <RF_BLOCK_DIAGRAM>: shows the RF communications setup of the rover keyboard inputs into AT commands, which will be transmitted to the receiver on the rover’s end and translated into directional inputs.

The LoRa protocol and the RYLR896 Module offer a great deal of flexibility when it comes to the overall rate of data transmission. LoRa is designed around *chirp spread spectrum* modulation. “Chirp” means that a linear increase or decrease in frequency is used to modulate the signal.

There are three parameters we can change to achieve a desired data rate. The bandwidth describes the difference between the maximum and minimum frequency. The spreading factor controls the duration of the chirp, and a higher spreading factor will mean more symbols are used in each transmission. The coding rate is a ratio between the actual data and the inserted bits for error correction. These three parameters can be used to calculate the data rate in bps according to the following equation:

$$R_b = \frac{SF\left[\frac{4}{4+CR}\right]}{\frac{2^{SF}}{BW}}$$

SF represents the spreading factor, CR is the coding rate, and BW is the bandwidth. The data rate can be used to calculate the duration of data transmission from the rover to the ground station from the ground station to the rover.

5.2.2 Power

Since the rover is designed to function as a rocket payload, there are significant size and weight constraints limiting the total battery capacity of the rover. We determined that a LiPo battery pack using a 2s2p configuration would be ideal for rover operations. The voltage range for this battery pack is 6.6-8.4 V. We designed our buck and boost converters with this input voltage in mind.

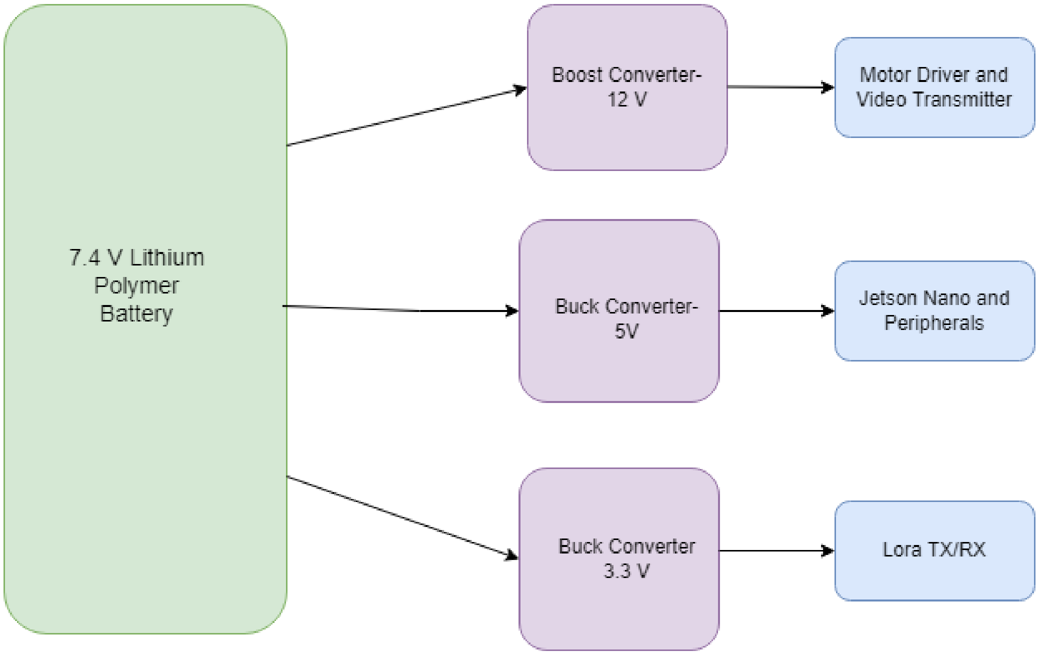

7.4 V Lithium Polymer Battery
Boost Converter- 12 V
Motor Driver and Video Transmitter
Buck Converter- 5V
Jetson Nano and Peripherals
Buck Converter 3.3 V
Lora TX/RX

Figure <BATTERY_CONFIGURATION>: shows the series-parallel battery configuration

This gives the rover a 7.4 V, 4600mAh power supply. The cells will be configured with a battery management system to balance the load and prevent an over discharge from happening.

The rover subsystems do not operate at 7.4 V, so DC to DC conversion is necessary. Power electronics are necessary to provide a constant, regulated DC power supply to the various rover subsystems. A major design decision is the choice to use switched mode or linear voltage regulators for the rover subsystems. Linear voltage regulation from a DC supply can be very compact and use a very small number of discrete components. However, this comes at a cost, as a lot of power is dissipated as waste heat, making linear regulators very inefficient. In a switched mode power supply, little power is dissipated as waste heat, giving the rover a longer battery life. If potential electromagnetic interference (EMI) is accounted for, switched mode supplies offer significant advantages over linear regulators. Most designs use a small number of integrated circuit components and a handful of discrete passive elements, making the size tradeoff between linear and switched negligible.

DC to DC Buck Converter for the single board computer (SBC)

The range for battery voltage should be between 6.6 and 8.4 V during safe operating conditions. To power the single board computer, a stable 5 V DC is required. A switched-mode buck converter will be more efficient than a linear power supply at stepping down the voltage to the required 5 V. Since the standard TPS series chips were not available, we had to use a more complicated design centered around the LM21305 chip. The switching frequency is 888.4 kHz, well out of the range of interference for our RC transceiver and video transmitter.

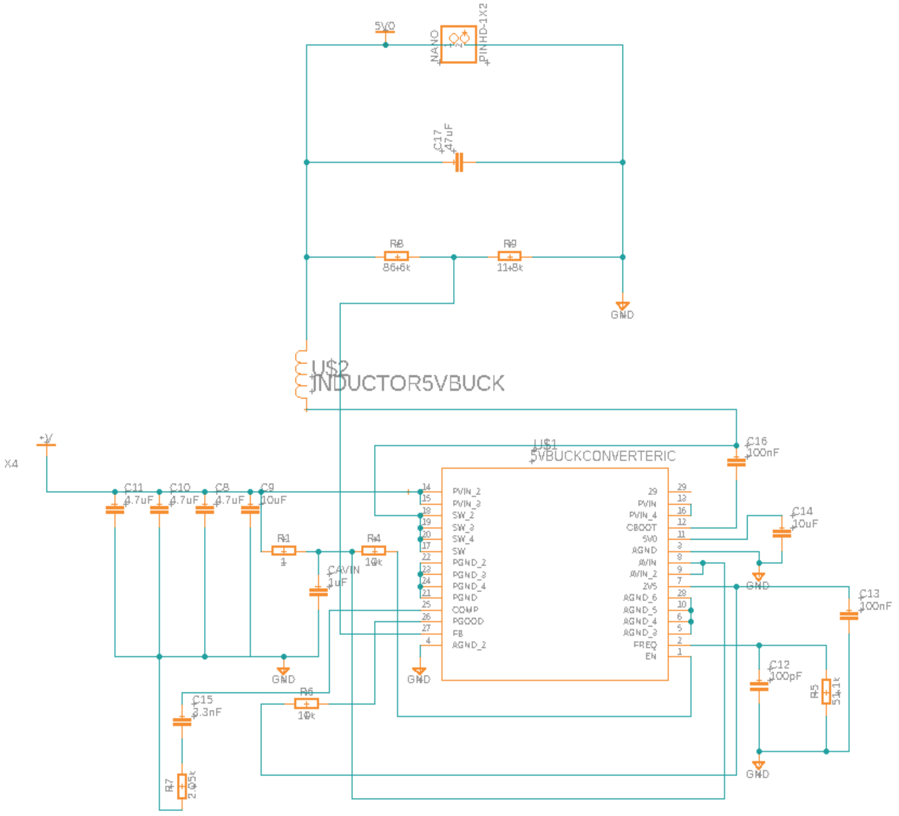


Figure <BUCK_CONVERTER_SBC>: DC to DC converter circuit that steps down the voltage to 5 V for the single board computer

DC to DC Buck Converter for RYLR896

A similar procedure is followed for the 3.3 V RYLR896 transceiver. The current draw for this device does not exceed 50 mA. The LiPo batteries give a supply voltage in the range of 6.6-8.4 V. These parameters are entered into WeBench to get a design for a DC-to-DC converter. A switching frequency of 278 kHz and any harmonics should have a negligible effect on the rover and rocket communication subsystems. This design was readily available.

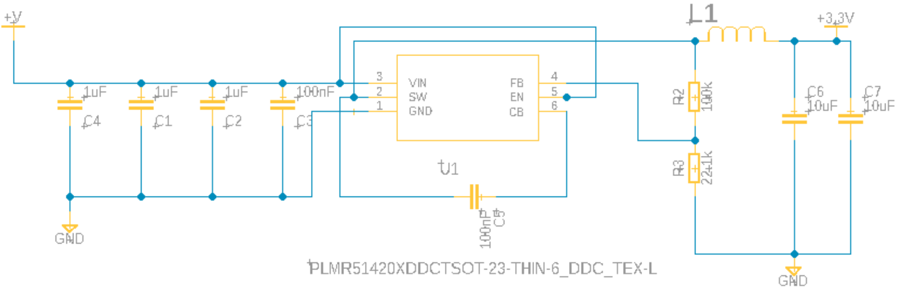


Figure <BUCK_CONVERTER-3.3V>: 3.3 V buck converter for RYLR896 Module

DC to DC Boost Converter for Hyperion TS5823 and motors

The Hyperion TS5823 video transmitter and the motors are some of the more power-hungry components in our design. They require a 12 V supply voltage and an output current of roughly 1 amp. Since the max output from the battery is 8.4 V, a boost converter is needed to step up the voltage. This design requires a substantial number of discrete components, including a Schottky Diode. With a switching frequency of 1.6 MHz, EMI will be limited. This design has a 53 sq mm footprint.

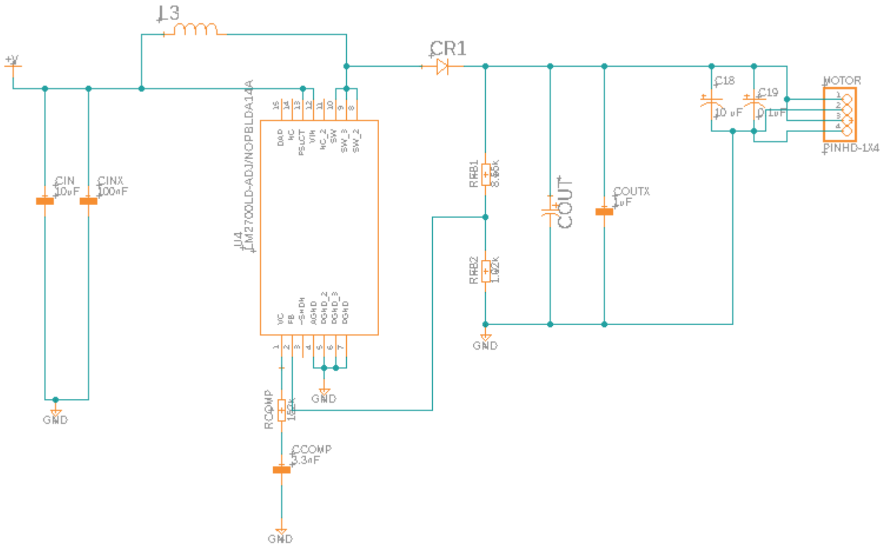


Figure <BOOST_CONVERTER_AV>: shows the Eagle schematic for the boost DC to DC converter for the Hyperion AV module and motor

Power Calculations and Estimated Battery Life

To make a rough estimate of the total battery life of the rover, the power consumption of the various modules is tabulated and added together. The total capacity of the batteries is estimated to be:

$$7.4\ V\ x\ 4600\ mAh\ x\ 4\ =\ 34.04\ Wh$$

The power calculations will initially assume the worst-case scenario: every module is operating at its maximum power consumption. The Jetson Nano will be using multiple peripherals, the motors will be operating continuously at their highest sustained current draw, and the transmitters will be constantly transmitting. This estimation will be very unrealistic for actual rover operation but will serve to estimate the lowest bound for battery life. Prototyping of the design will allow for a much more accurate estimate.

Module	Current Draw (mA)	Voltage (V)	Estimated power

Hyperion TS5823 AV transmitter	190	12	2.28 W
Jetson Nano carrier board	variable	5	0.5 W-1.25 W
Jetson Nano	2A	5	20 W (assuming worst case w/ peripherals)mo
Reyax RYLR896 LoRa RC Transceiver	Variable (0.5 uA sleep, 16.5 mA receive, 43 mA transmit)	3.3	0.142 W (worst case, constant TX)
SGMADA 12V MR-36127000-40KY	0.56 A (sustained stall current)	12	6.72 W (worst case, assuming sustained stall current)
Total			30.4 W

This crude worst case power estimate gives an estimated battery life of:

$$Battery\ Life(Hours)\ =\ 34.04\ Wh/30.4\ W = 1.12\ Hours\ =\ 67\ minutes$$

This is an absolute worst-case scenario, calculated to give an estimate of the absolute lowest limit of the battery life. Even in a worst-case situation, the rover will be able to meet the payload requirements for the FAR competition and should not have a problem traveling 10 feet from the landing site.

As for the payload canister, three LiFePo4 cells will be used to power its electronics. This gives the following maximum watt-hours:

$$3.2\ V\ x\ 1500\ mAh\ 3\ =\ 14.4\ Wh$$

Like the rover, we will assume the worst possible case where the canister is operating at full capacity. The power consumptions are roughly tabulated here:

Module	**Current Draw (mA)**	**Voltage (V)**	**Estimated power**
Arduino Nano	20 (max idle)	9.6	0.192 W

2x Allegro A4988	8 (f_{pwm} < 50 kHz)	5	0.08 W
Freescale MPL3115A2	2 (max current during data acquisition)	3.3	0.0066 W
2x NEMA 11 Motor	1.34A (assuming both phases are used)	3.75 (from driver)	10.05 W (constant motion)
2x CUI AMT102-V Encoder	6	5	0.06 W
Total			10.39 W

A similar calculation is done to find the worse-case battery life:

$$Battery\ Life(Hours)\ =\ 14.4\ Wh/10.39\ W = 1.38\ Hours$$

Based on these results, we should expect our canister’s power supply to last well after the rover is operational.

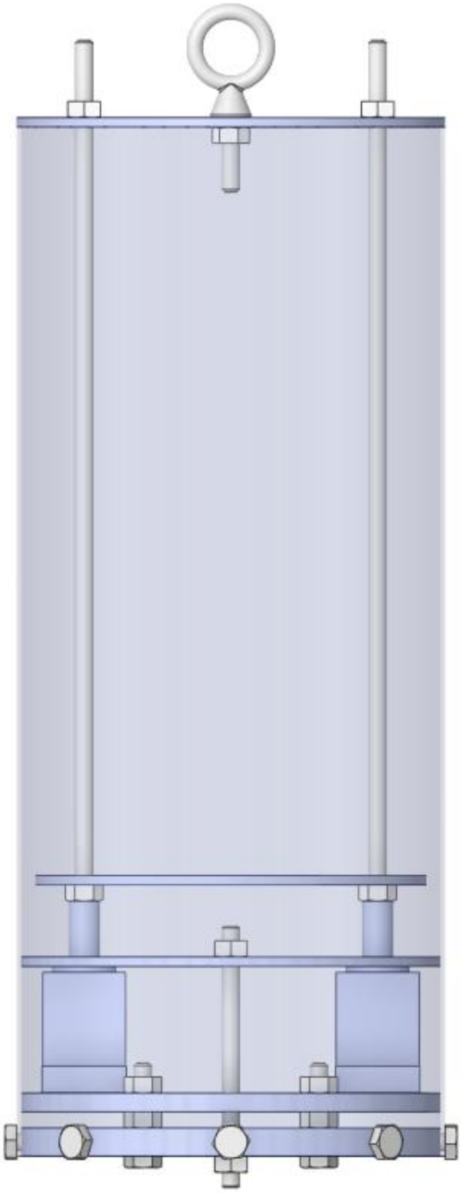
A schematic for the power electronics hierarchy is shown below. Many of the rover subsystems will be connected to the Jetson Nano board as peripherals, simplifying the overall design and saving space. The 5.8 GHz video transmitter and the motor driver both require the same voltage, a so one 12 V converter is used. Two extra decoupling capacitors are added at the output end to keep the output voltage stable.

Thermal Considerations

Although switching DC-DC converters are more efficient than linear regulators, they still produce waste heat that can damage the printed circuit board and surface mount components. A relatively straightforward solution to heat dissipation is the use of heat sinks. Small heatsinks are readily available and can be mounted to the buck and boost converter ICs with thermal glue. The boost and buck converters will be spaced far apart, allowing for more surface area for the heat to dissipate.

5.3 Subsystem - Canister

5.3.1 Overview



The payload canister is used to safely encapsulate the rover until it's deployed. It is how our project fails or succeeds. A canister that is not up to par with the harsh conditions of payload deployment would cost the team hundreds of dollars in damages. To avoid this outcome, our team will follow strict procedures to rigorously design and test the canisters.

To design the canister, we must first select materials that are sturdy enough to withstand 8Gs of axial acceleration and 7.69 m/s^2 of lateral acceleration. During this phase, we must keep in mind the goals, objectives, and constraints that were previously mentioned. The material that we choose to construct the payload canister would have to be light enough so that we don't breach our 4.31 kg weight limit, yet inexpensive to manufacture. We should also note that other factors are at play during the rocket's ascent, notably thermal fluctuations, and vibrational forces. With that said, the material we finally decide on must be thermally resistive to the drogue parachute's ejection charges, and that the force imposed should not impede on the canister's structural integrity, which could be detrimental to the rover deployment system. A strong, 3D-printable, material will also need to be selected for harnessing the canister's electrical components as well as the rover itself. After much thought, the team decided to use 6061-T6 Aluminum for the canister's structure, 316 Stainless Steel for our various fasteners, and PLA for the component/rover mounts.

Furthermore, we will need to purchase suitable components that will supply the canister with the necessary information to deploy the rover successfully. The required data that will be needed are the canister's current altitude and the stepper motors' rotational displacement. We will accomplish this using a barometric pressure sensor and an incremental capacitive encoder. The Freescale MPL3115A2 will feed information to the Arduino Nano regarding the canister's current vertical displacement. This will allow the Nano to gauge whether the canister should engage rover deployment. The CUI Devices AMT102-V is a precise stepper motor encoder that will be used to fine-tune the required current needed to drive the motors at its highest torque. Two NEMA 11 motors, model 11HS12-0674D, are employed due to their inexpensiveness and small form factor. Since we need two motors, two Allegro A4988 motor drivers will be used to relay the optimal amount of current required to drive the motors and release the rover.

An Arduino Nano will be used to integrate these components and coordinate payload deployment. The Nano was chosen as it is very cheap, lightweight, small, and has enough pins for our electronics. The Arduino IDE will be used to develop the software to be run on the Nano. The IDE will allow us to flash our program to the Arduino Nano and contains other useful libraries and features specific to Arduino devices. Figure <CANISTER_ELECTRICAL_SCHEMATIC> shows how the electrical components will be connected to the microcontroller. The software will be debugged until we are certain that the program will release the rover without fail.

Power will have to be provided internally as the Arcturus rocket team has declared that they will not supply external power. As the canister will be disconnected due to payload ejection, we decided to use our own power supply. After consideration, we ended up using 3 LiFePo4 18650 cells rated for 3.2V and 1500mAh. In series, these cells would provide enough voltage to power both motor drivers. As for longevity, using an earlier rough calculation, we estimated that in a worst-case scenario the canister would be able to last for 1.38 hours. Since the flight and deployment are approximated to take less than ten minutes, we are confident that the batteries we have selected are suitable for the task presented.

Afterwards, extensive testing must be done for the canister. FEA and CFD analysis must be executed to prototype an inexpensive design, and to verify that the design doesn't have errors before we purchase parts. Any analysis that will be performed on the canister will be done using the Ansys Mechanical APDL and Ansys FLUENT solvers. Ansys Mechanical is a powerful tool capable of simulating difficult mechanical problems. In our case, we will be doing static structural analysis with the FEA solver. This is to ensure that the canister is not being subjected to abnormal amounts of stress or affected by excessive strains. The APDL solver will also be used for modal analysis to measure the canister's vibration and ensure that its natural frequency doesn't couple with the one that is generated by the rocket. Once that is confirmed, we will be confident that the canister's electronics will function reliably. The Ansys FLUENT solver will also be used for thermal analysis once the Arcturus team communicates the estimated ejection charge temperatures. Since the top-half of the canister that houses the rover is near the ejection charges, we must be careful to design the canister so that the rover doesn't get damaged.

Finally, we must take care of any loose ends such as the securing of wires with nylon cable ties. An RTV sealant will be used at the places where it is applicable.

5.3.2 Mechanical Theory

$$F = ma$$

$$T_r = \frac{Fd_m}{2}\left(\frac{l + \pi f d_m sec\alpha}{\pi d_m - flsec\alpha}\right) \text{ [41]}$$

5.3.3 Materials / Components

- 6061-T6 Aluminum (body tube and disks)
- Steel (fasteners)
- Arduino Nano
- Freescale MPL3115A2 Barometric Pressure Sensor
- 2x NEMA 11 Bipolar Dual-Shaft Stepper Motor (11HS12-0674D)
- 2x Allegro A4988 Motor Driver
- 2x CUI AMT102-V Incremental Capacitive Encoder
- 303 Stainless Steel Set Screw Coupler (5mm to ¼ in)
- 3x LiFePO4 18650 Rechargeable Cell
- 3-Cell LiFePO4 18650 Battery Holder
- Magnetic Proximity Switch
- Epoxy Resin
- Nylon ties

The reasons for selecting electrical components are outlined earlier in 3.3.5 Canister Components.

6061-T6 Aluminum Properties

Property	Metric	Units
Density	2.70	g/cc
Ultimate Tensile Strength	310	MPa
Tensile Yield Strength	276	MPa
Poisson’s Ratio	0.33	-
Shear Modulus	26.0	GPa
Shear Strength	207	MPa
CTE	23.6	µm/m*°C
Melting Point	582 - 651.7	°C

Low Carbon Steel Properties

Property	Metric	Units
Grade/Class	ASTM A108 Grade 2	-
Hardness	Rockwell B61	-
Tensile Strength	53,000	psi

Medium-Strength Steel Properties

Property	Metric	Units
Grade/Class	ASTM A193 Grade B7	-
Hardness	Rockwell C35	-

Tensile Strength	125,000	psi

Black-Oxide Steel Properties

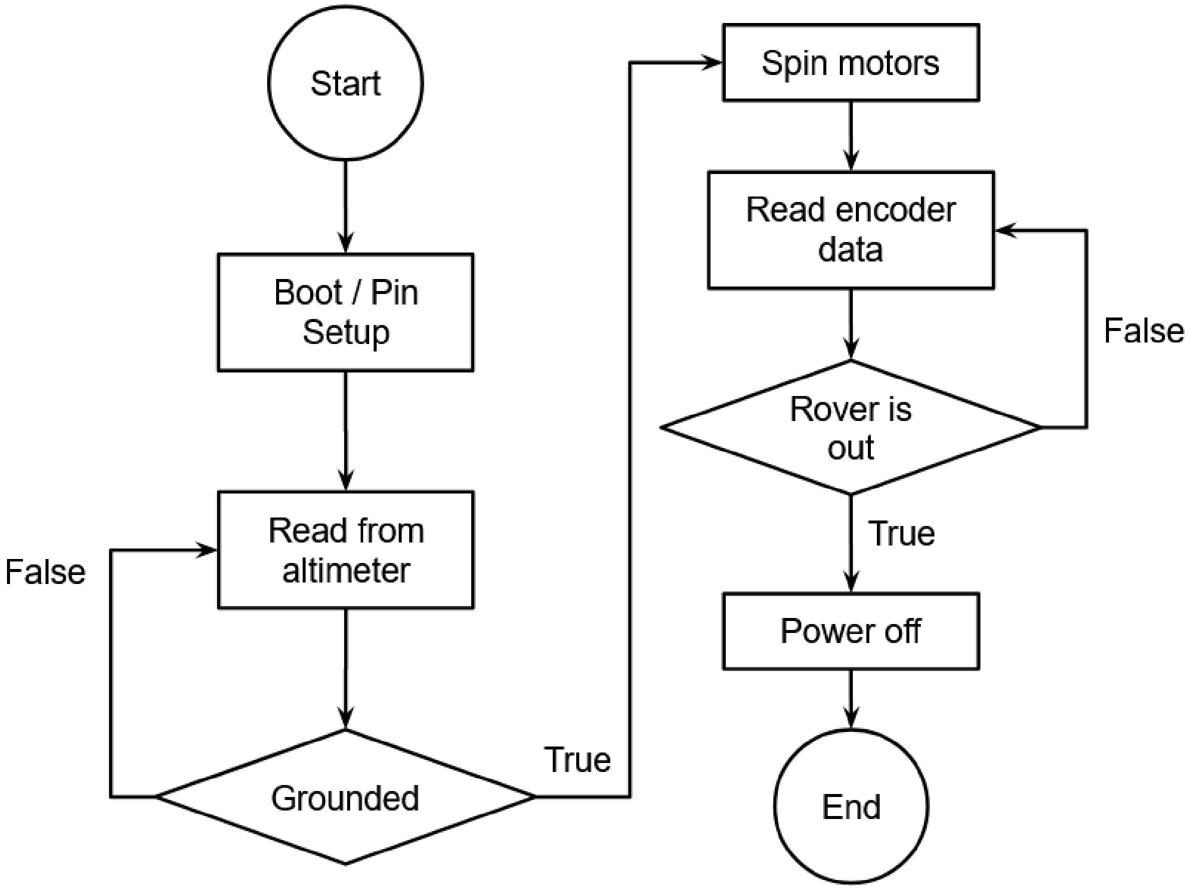
Property	Metric	Units
Grade/Class	ASME B18.2.1, SAE J429 Grade 8	-
Hardness	Rockwell C33	-
Tensile Strength	150,000	psi

303 Stainless Steel Properties

Property	Metric	Units
Density	8.00	g/cc
Ultimate Tensile Strength	620	MPa
Tensile Yield Strength	240	MPa
Poisson’s Ratio	0.25	-
Shear Modulus	77.2	GPa

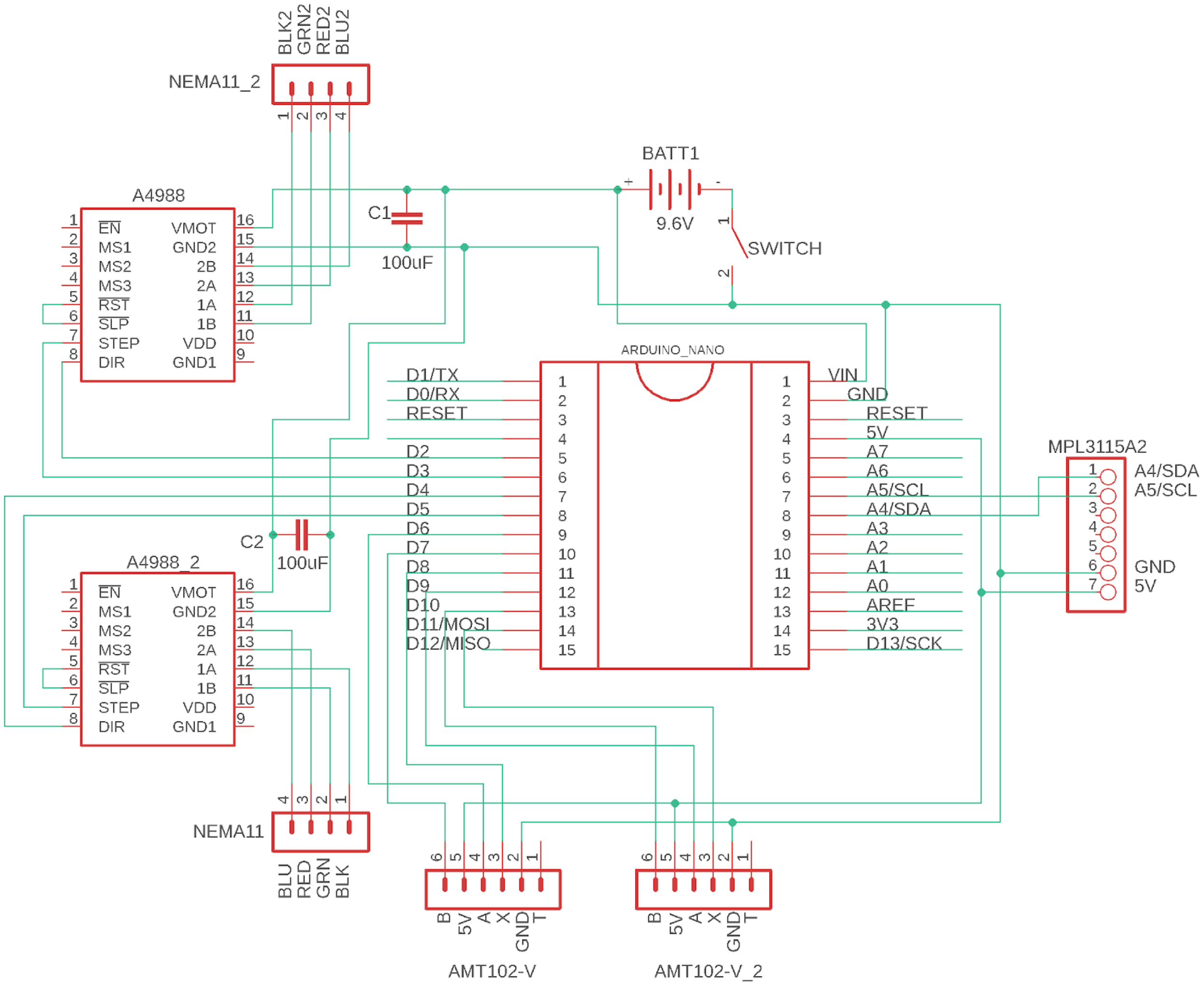
5.3.4 Canister Design

5.3.4.1 Software Design



To design the payload canister’s software, we must be meticulous and thorough as one unnoticed bug could have severe consequences. Above, Figure <CANISTER_SOFTWARE_DESIGN> shows the logic that we will employ on the canister’s Arduino Nano. When the Nano is first booted, pins are setup as inputs and outputs. In the case of the motor drivers, they will have a pin for stepping and direction for our bipolar motors. The encoder channels will have their pins initialized to receive as input into the MCU. The pressure sensor, which communicates with I2C, will be interfaced with a 9600 baud rate serial connection. Once setup concludes, the microcontroller will start reading altitude data from the pressure sensor using functions from an open-source library. Checks will be done to see if the rocket has started ascending and if it has reached apogee. The final check of the altitude is to determine if the canister is grounded. When that is true, we will engage rover deployment by stepping the motors. The microcontroller will start reading from the encoders and spin the motors until they have been stepped a certain number of revolutions. When that number has been reached, we have successfully deployed the rover and the payload canister will safely power off. When we develop the software, we will use many of the Arduino’s power-saving techniques to achieve an optimal battery life. The software will be developed using the Arduino IDE and incorporate relevant libraries to greatly simplify the program.

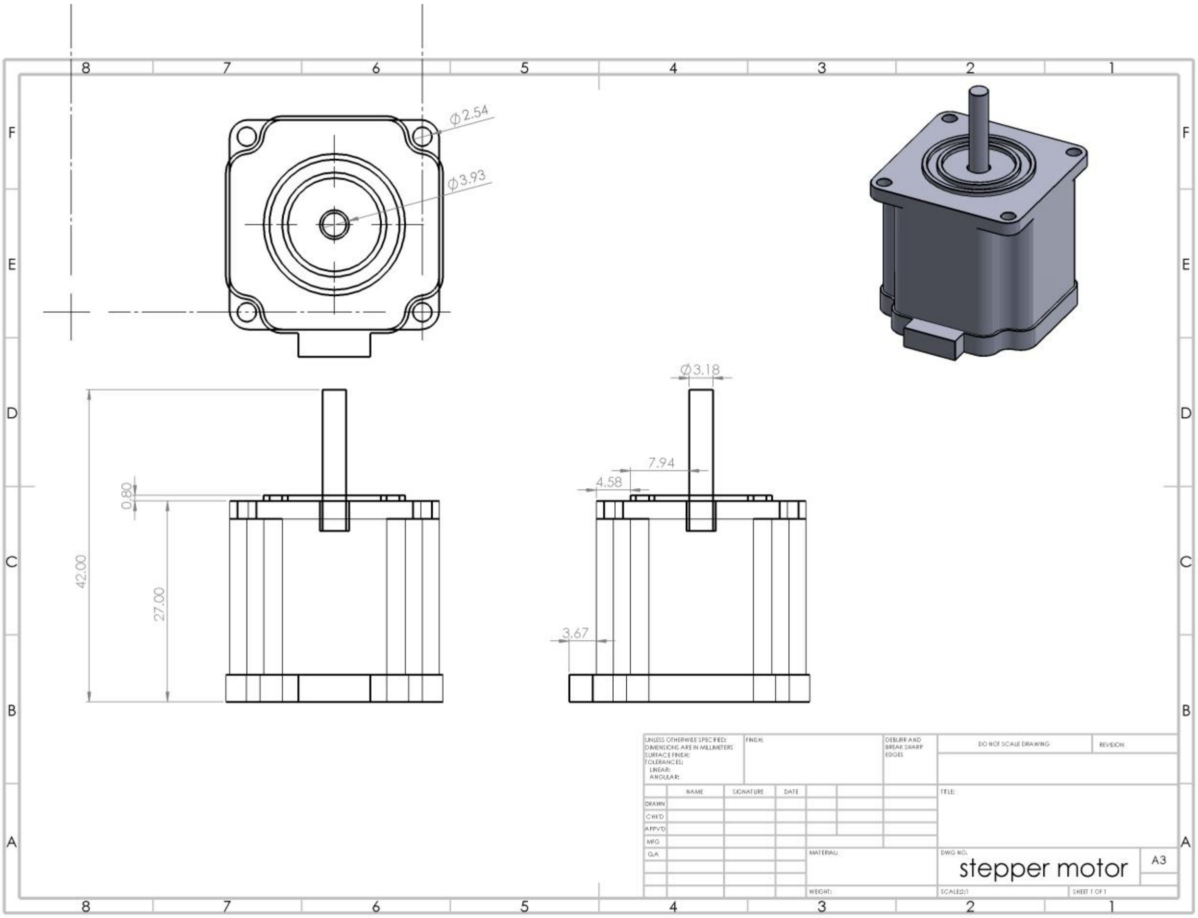
5.3.4.2 Electrical Design



After our electronics have been received, we must wire them up. Figure <CANISTER_ELECTRICAL_SCHEMATIC> shows how we plan to connect the components. On one side, the Arduino Nano has fourteen 40 mA digital pins that have 5V logic. We will use these to attach the motor drivers and encoders. The other side has 8 analog pins that can be used for components with voltages between ground and 5V. The barometric pressure sensor will use these two of these pins for I2C communication with microcontroller. To prevent LC voltage spikes, the Allegro A4988 motor drivers are accompanied with 100uF electrolytic capacitors. The NEMA 11 motors will be wired to the motor controllers based on the color-coded lead setup supplied by the manufacturer’s datasheet. A switch will be used to activate the payload canister before it is inserted into the rocket.

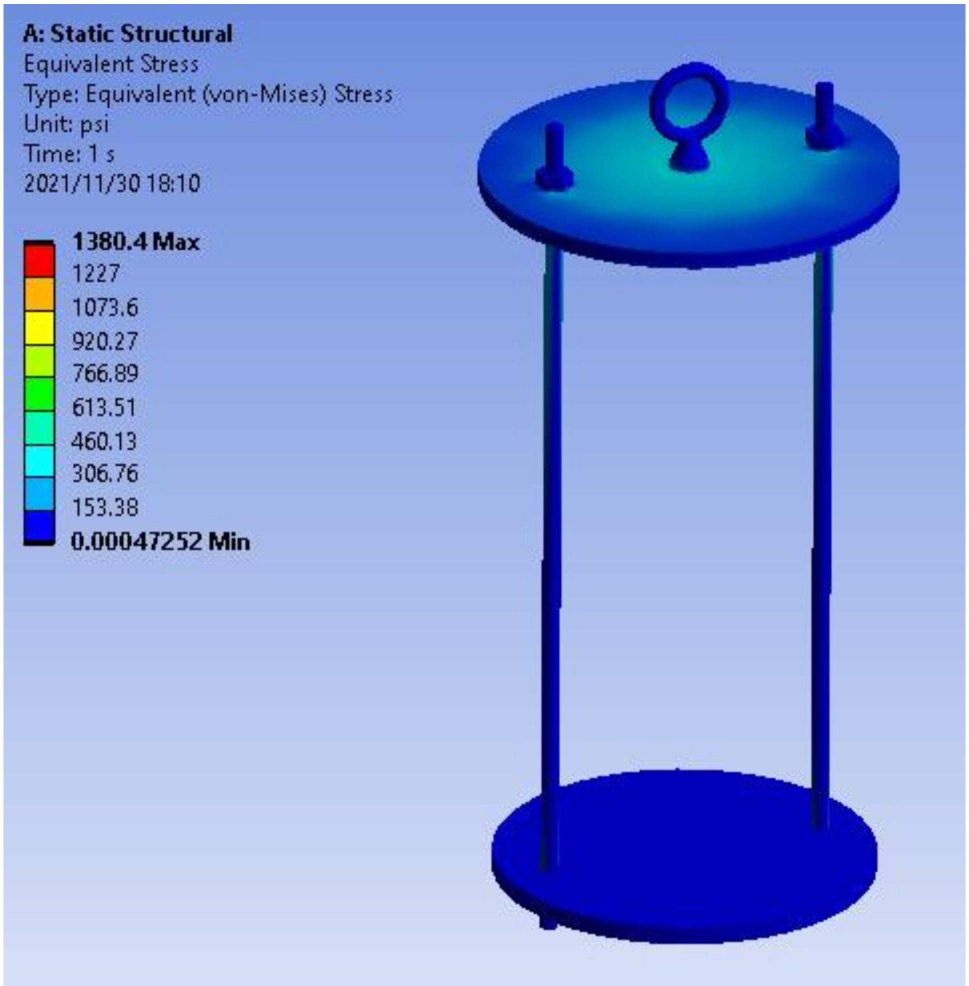
5.3.4.3 Rover Release Mechanism

The motors will have to be able to overcome thread friction and lift the payload. To find the required torque, we must use this formula: $T_r = \frac{Fd_m}{2}(\frac{l+\pi f d_m sec\alpha}{\pi d_m - flsec\alpha})$. T_r is the required torque to lift the rover and its encompassing plates out of the canister, F is the axial load, l is the lead, f is the coefficient of friction, d_m is the mean diameter of the lead screw, and is half the thread angle. We will be using the value of the entire canister to ensure that the motors have enough torque to release the rover in the case that the canister does not land parallel to the ground. We will also assume the maximum load possible given the 4.31 kg constraint. The reasoning for this is that the torque will be sufficient for our design if it can support the maximum load. The resulting force that the motors need to lift is approximately 9.5 lbf. Since we will be using two stepper motors, the axial load becomes divided in half, which leaves about 4.751 lbf for each lead screw. Using this axial load, the specifications for a ¼” - 16 Acme thread, and the above formula, the required torque ends up being 3.2 oz-in.



5.3.5 Structural Analysis

Stresses



This is a preliminary FEA of the canister.

Vibration

We cannot proceed with vibration testing until the natural frequencies of the rocket are known.

Thermal

Until the Arcturus team relays the thermal characteristics of their selected ejection charges, we cannot proceed with the CFD thermal analysis of the canister's northern boundary.

5.3.6 Bill of Materials (BOM)

The complete bill of materials for the proposed design of the Payload Canister is as follows:

Component	Type	Price (USD)	Quantity	Total

Arduino Nano	MCU	$20.70	1	$20.70
Freescale MPL3115A2	Sensor	$9.95	1	$9.95
Allegro A4988	Driver	$5.95	2	$11.90
NEMA 11 Motor	Motor	$14.27	2	$28.54
CUI Devices AMT102-V	Encoder	$23.00	2	$46.00
0.250" to 5mm 303 Stainless Steel Set Screw Shaft Coupler	Coupler	$4.99	2	$9.98
LiFePO4 18650 Rechargeable Cell	Power	$4.00	3	$12.00
6" OD x 0.125" Wall x 5.75" ID Aluminum Round Tube 6061-T6-Extruded, 1 ft	Material	$60.01	1	$60.01
0.375" Aluminum Plate 6061-T651	Material	$23.81	1	$23.81
0.125" Aluminum Sheet 6061-T6	Material	$27.38	1	$27.38
Carbon Steel Acme Lead Screw, Right Hand, 1/4"-16 Thread Size, 1 ft.	Fastener	$4.35	2	$8.70
Carbon Steel Acme Hex Nut, Right Hand, 1/4"-16 Thread Size	Fastener	$2.34	4	$9.36
Grade B7 Medium-Strength Steel Threaded Rod, 1/4"-20 Thread Size, 1 ft.	Fastener	$4.29	1	$4.29
High-Strength Steel Hex Nut, Grade 8, Black-Oxide, 1/4"-20 Thread Size, 100-count	Fastener	$6.00	1	$6.00
High-Strength Grade 8 Steel Hex Head Screw, Black-Oxide, 1/4"-20 Thread Size, 25-count, 1”	Fastener	$9.23	1	$9.23
High-Strength Grade 8 Steel Hex Head Screw, Black-Oxide, 1/4"-20 Thread Size, 25-count, ½ “	Fastener	$8.57	1	$8.57

Note: prices are accurate as of last revision of this section on 12/6/21.

5.4 Subsystem - Rover Payload

5.4.1 Overview

The Rover Payload subsystem will be a small, radio-controlled robot designed to meet the requirements of the FAR 1030 competition: 10ft of driving distance with live video feed. Additionally, taking inspiration from the more advanced NASA Perseverance Rover, we have implemented more advanced robotics features to extend beyond the bare requirements of the competition.

The Rover Payload is designed to fit inside the Payload Canister, being released after the Canister completes its landing. After release, the Rover Payload will be responsible for traversing desert conditions and providing a live video feed of its surroundings. To facilitate these efforts, the rover will make use of advanced state estimation to track our position and orientation over time, as well as make use of exteroceptive sensors to map our environment in real-time. With these two capabilities, we are capable of a simple form of autonomous navigation, referred to by the Perseverance team as “blind drive.” When provided with a waypoint, our rover will travel autonomously to it, mapping its surroundings and avoiding obstacles as best it can.

Figure <ROVER_BLOCK> shows the block diagram of the rover system:

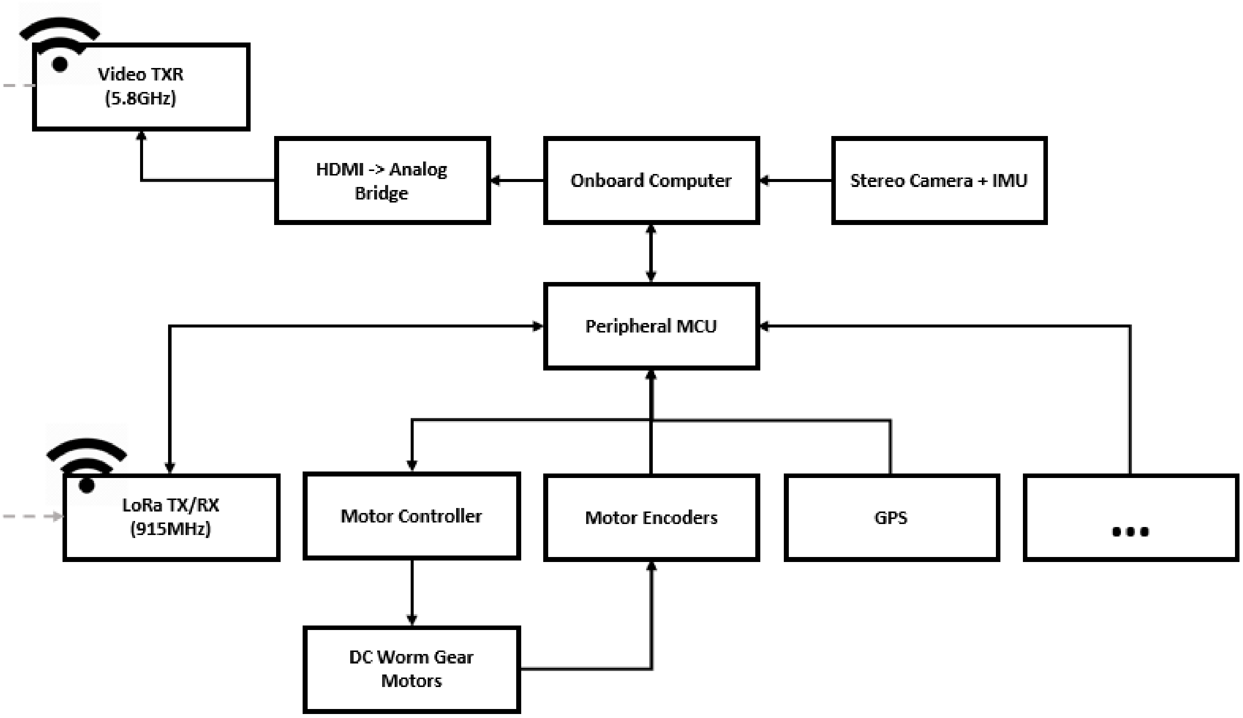


Figure <ROVER_BLOCK>: block diagram of rover system.

The software onboard the rover will be developed using ROS (see section 3.2.4.1) as a framework. ROS will allow us to quickly prototype and create a working system, due to its exhaustive library of tools and open-source software. Significant prior experience means that we can efficiently leverage ROS to rapidly expand our feature set. Further leveraging ROS, we can quickly develop a complete physics-enabled simulation of the rover that will utilize the same code base as our real-life rover. The use of software simulation will allow us to develop software before we have the rover entirely assembled and parallelize our work.

The mechanical design of the rover is heavily constrained by the tight dimensions and weight restrictions imposed on the canister by the design of the rocket we will ride on [R1.0]. Since the rover needs to fit inside the canister, we must meet those requirements as well.

5.4.2 Data Streams and Sensor Integrations

An important consideration in the design of the rover is what data streams we have available to us, as they will determine what information we can use to make decisions. This section attempts to identify the exact data streams that will be used, their source, and how exactly they will interface with and contribute to the rover system.

In general, sensors will be connected to our Jetson Nano SBC through a data pin (such as GPIO) or specialized connector (such as MIPI-CSI2). That pin will be read with firmware, such as Jetson.GPIO, which will provide sensor data as objects to be manipulated in our program. Then, this firmware will be wrapped in a ROS node that can translate the message to their ROS message equivalent and publish them as they're available. In this way, we ensure that the entire system has access to the sensor inputs simultaneously through the ROS pub-sub framework.

Data streams can be separated into interoceptive (about the rover) and exteroceptive (about the environment around the rover).

5.4.2.1 Interoceptive Data Streams

One of the key motivations behind our interoceptive data streams is the tracking of our rover's position and orientation, called the *pose*, in real-time, a process called *odometry* [C47]. Odometry will answer the question "where are we now, relative to where we began?" Our ROS software stack will include ekf_localization_node, a software module that implements sensor fusion for heterogeneous streams of data [C45]. Through it, we can use an Extended Kalman Filter (see section 3.2.3.1.3) to generate a robust state estimate to be used in our localization and mapping efforts. This state estimate can then be published as a nav_msgs/Odometry [C50] message for use in other parts of our software system.

Inertial Measurement Unit (IMU):

The inertial measurement unit selected in section 3.3.3.2 will provide information about our angular velocity i.e., the rate of change of our orientation (roll, pitch, yaw), our linear acceleration in the X-, Y-, and Z-dimensions, as well as information about our magnetic heading (see section 3.2.3.1). These measurements can then be integrated with respect to time to generate rotation (from angular velocity) and position (from linear acceleration). These data points combined comprise the pose of our rover, an important component in our overall state estimation, localization, and autonomous driving efforts [C44].

If we lived in a theoretical world where our initial pose was known exactly and our sensors are perfect, we could use this single IMU sensor stream on its own. But we don't live in that theoretical world: our initial pose will not be known, and our sensors will be noisy. So, it is necessary to have more data points than one IMU can provide. Using more than one IMU allows us to use sensor fusion, hopefully providing a more accurate measurement than any one unit can manage on its own. Luckily, our selected stereo camera (section 3.3.3.1) also includes an IMU. Through fusion of both the stand-alone and camera integrated IMUs, we can get a better pose estimate that is greater than the sum of its parts.

To access pose information from the standalone IMU, we'll use the UART communication protocol to read data travelling from the IMU to the GPIO pins of the Jetson Nano SBC. This UART communication will then be wrapped in a higher-level ROS node that will handle the translation of the data into a ROS message, in this case a *sensor_msgs/Imu* message [C46], that can then be passed as an input into ekf_localization_node.

Wheel Encoders:

Wheel encoders (discussed in section 3.2.3.4) will measure the motion of our wheels relative to our chassis. They do this by tracking the rotation of the wheels via "ticks," where 48 ticks equal one complete revolution of the wheel. Integrating this information over time allows us to track how the wheels (and by extension, the rover) have moved since system origin [C22]. This is another source of odometry information that can be fused into our EKF-based state estimation.

While initially we planned to use motors with built-in Hall Effect encoders, we found that they were simply too noisy to reliably track wheel rotations. We pivoted to using the capacitive encoders selected in section 3.3.5.4, initially chosen for use as part of our canister. These encoders use a two-pin, two-channel communication scheme that will be connected to the GPIO pins of the peripheral MCU. The encoders are accessed via GPIO polling with a frequency determined by a user-configured timer, currently set to about 250Hz. We found, through experimentation, that that rate gave use the smoothest response from the encoders without overwhelming the MCU, which has other tasks to handle simultaneously.

Like the other peripherals attached to the peripheral MCU, the data generated by the encoders will be pushed along a serial connection to the SBC at a rate of about 20Hz with the "$ENC" prefix. The ROS node that communicates with the MCU, pico_bridge, will translate the encoder data into two ROS messages (std_msgs/Int64), one for each wheel. Our custom odometry node, encoder_odom, will then translate these ticks into an odometry message, in this case *nav_msgs/Odometry* [C50], for the rest of the system to use. It should be noted that this message is identical in form to the final message output of the ekf_localization_node node.

Global Positioning Unit (GPS):

Our GPS unit (selected in section 3.3.3.3) will provide periodic, discrete absolute position measurements. These measurements will be world-fixed i.e., where we are on Earth, rather than relative to our starting point like our other interoceptive streams. The GPS data can be used to augment our state estimation by acting as an input to our EKF, implemented with ekf_localization_node.

The selected GPS module will be connected to our peripheral MCU via a UART interface (9600-8-N-1). When the GPS unit has updates for us (roughly 1Hz), it fires an interrupt that reads the data from the GPS and transmits it through the serial connection to the SBC with the prefix “$GPS.” All data from the peripheral MCU is read by the pico_bridge node, which will take those messages marked with “$GPS” and publish them as ROS messages for the rest of the system to consume.

5.4.2.2 Exteroceptive Sensors

Exteroceptive sensors provide data about our environment. Rather than being used (solely) for state estimation, exteroceptive sensors allow us to generate a representation of our environment, such as a costmap, that can be used for mapping and autonomous navigation.

For this rover, the sole exteroceptive sensor is planned to be a stereo depth camera, which can provide two data streams:

Video:

The stereo camera is a camera, and thus provides a video stream for us to access. While it’s certainly possible to perform some object detection and recognition using robot vision algorithms, in this case we only plan to route this video through the rover SBC to the Hyperion GHz transceiver to be sent to the ground station. The rover itself isn’t planned to make use of this data stream for any state estimation, landmark detection, or mapping.

Depth Cloud:

The stereo camera can be used for more than just video streams. Since we have two identical cameras separated by a known distance, we can generate a cloud of depth points, providing volumetric info about our surroundings (as discussed in 3.2.3.3).

The ROS package move_base provides the costmap_2d software node, which allows us to input sensor streams like depth clouds that are then used to generate a rough 2D map of our environment. Using this map, we can use our path-planning algorithm of choice to navigate through it to our goal. The depth cloud generated by our stereo camera will be the lynchpin of our “blind drive” autonomous waypoint navigation.

5.4.3 Software Design

5.4.3.1 Software System Overview

The software system for the rover will consist of multiple layers. The first of these is the “firmware” layer, where code that interfaces with individual sensors will be. These “hardware interface nodes” will generate the data streams that will be fed into the upper layer. The upper layer contains the ROS-enabled controls and state representation software that will enable rover functionality. In this layer, the controller receives sensor data, user input (through RC), as well as the results of any mapping or state estimation, and uses these input streams to make decisions. The controller will then call services to actuate the system, whether that’s turning, moving to a waypoint, or reconfiguring parameters on the parameter server. The relationship between the upper and lower layers is depicted in figure <SOFTWARE_OVERVIEW>.

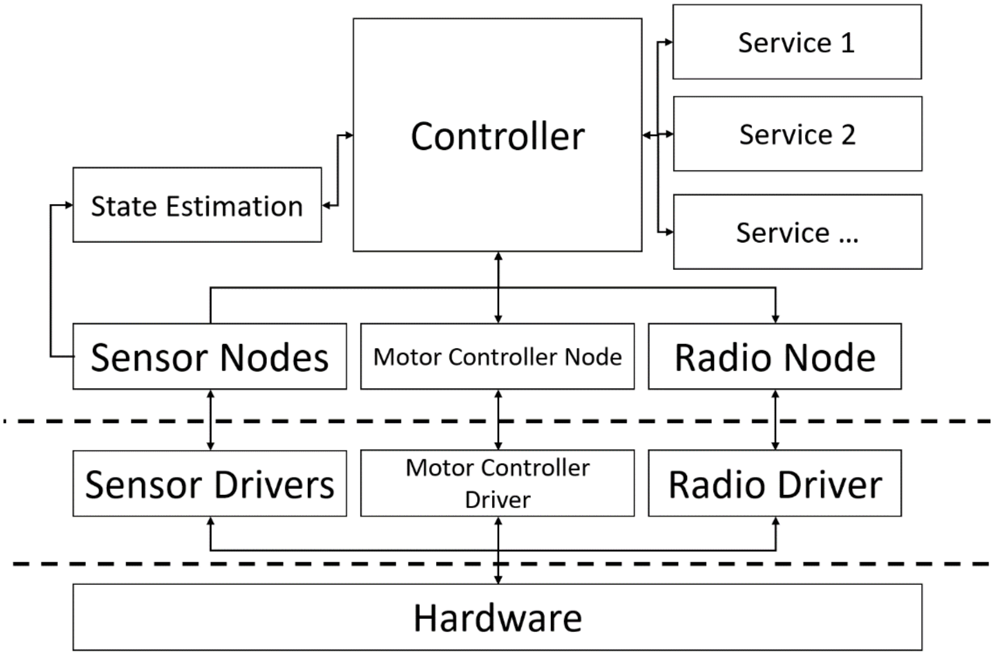


Figure <SOFTWARE_OVERVIEW>: the software “stack” that comprises our rover system

The software system of the rover will leverage the Robot Operating System (ROS) for inter-process communication. ROS allows for simple, scalable robotics software and promotes ease-of-communication between software modules. Further, ROS provides many software packages “off-the-shelf” that will allow us to rapidly expand our feature set. A more complete discussion of ROS and its capabilities can be found in section 3.2.4.1.

Hardware interface nodes will connect the data from our sensors, motor controllers, and radio receivers to the rest of the system. In these nodes, data pulled from drivers is converted to a standard format (ROS message) and connected to the ROS system through calls to provided ROS libraries in C++ or Python, as in Figure <5.4.2_NODE>. In this way, the hardware details are abstracted away from the controller, allowing for a more modular design that can also leverage ROS software modules.

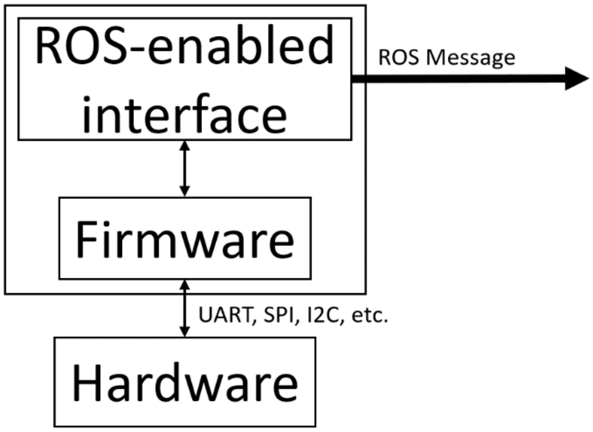


Figure <5.4.2_NODE>: The structure of a single hardware interface node.

The software controller itself will use a simple state machine (see figure <STATE_MACHINE>) to determine behaviors based on context. Sensor streams, controls data, and user inputs received through ROS messages will be used to coordinate the motion of the robot. This state machine is further explained in section

To facilitate development, a software simulation of our rover will be developed. ROS comes bundled with the Gazebo simulator, which is capable of directly interfacing with ROS systems using a set of ROS program wrappers called gazebo_ros_pkgs [C37]. The software onboard the simulation of the rover and its real-world counterpart could be developed in parallel and have near-identical codebases. The ability to test every aspect of our software solution ahead of time is crucial considering the compressed timeline of development (less than 2 semesters) and the difficult conditions surrounding real-world integration testing (having to launch a rocket first). The rover simulation will be explored further in section 5.4.2.2.

ROS also provides a convenient method for visualizing data streams in real-time via the rviz program [C36]. rviz allows us to visualize ROS messages, like those published by our sensor nodes and controller, in three dimensions. Anything from the current position of our wheels to video streams from our camera can be displayed graphically. Further, rviz configurations can be saved and loaded later, allowing us to create “standard” GUIs for data inspection using relevant sensor streams in convenient configurations. rviz will extend our ability to test our rover in simulation and in real-world operation and is used as a key software component of our ground station subsystem (see section 5.5) for viewing debugging data during development.

To meet our telemetry logging requirement [R4.8], make use of the ROS built-in *bag* filetype. Bag files are used to store ROS message data for later use [C34]. To log telemetry, we can use a program like rosbag [C35] to capture streams of ROS messages and store them in bags. Later, these bag files can be loaded into a data visualizer like rviz and "played back" for debugging and verification. Further, since bags can log complete sensor streams (including time published), actual data can be injected into our simulations for greater real-world parity.

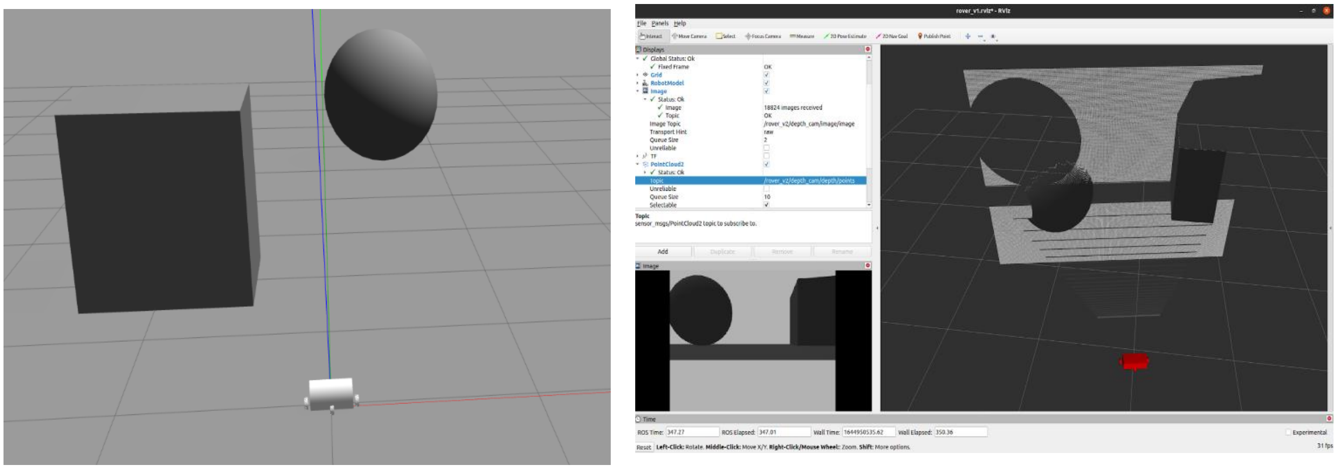
5.4.3.2 Simulation

Using Gazebo and ROS, a simulation of the rover was developed with two purposes: as a proof-of-concept for the state estimation approach and the data input streams, and as a means of no-risk testing with high real-world software parity.

To act as a proof-of-concept, the rover simulation was designed to match the real-world rover as closely as possible. Therefore, it uses only sensor streams that will be available to the real-life version: depth camera, IMU, GPS, and a near-perfect odometry source that will simulate the output of our wheel encoders. By operating on the assumption that our sensor data will be in a standard ROS message format, we can then develop the system without having the hardware readily available. Configuring ROS systems is a time-consuming process, so being able to begin as soon as possible is a serious benefit.

The second goal of the developed simulation is to act as a development ground for the state machine and controls software. Since we have access to both user inputs and our data streams, we can begin building the controller long before we have a working physical prototype. After this development, we should be able to directly transfer this software, unchanged to the physical robot. Along with some bespoke ROS-to-driver programs, the controller will be ready to perform in the real-world with very little reconfiguration.

Below is a screenshot of the Gazebo simulation in its early stages, as well as the rviz window showing our data inputs in action:


rover_v1.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
Image
Time
Reset

✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	☑
RobotModel	☑
Image	☑
✓ Status: Ok	
✓ Image	18824 images received
✓ Topic	OK
Image Topic	/rover_v2/depth_cam/image/image
Transport Hint	raw
Queue Size	2
Unreliable	□
TF	□
PointCloud2	☑
✓ Status: Ok	
Topic	/rover_v2/depth_cam/depth/points
Unreliable	□
Queue Size	10
Selectable	☑

Topic
sensor_msgs/PointCloud2 topic to subscribe to.

ROS Time: 347.27 ROS Elapsed: 347.01 Wall Time: 1644950535.62 Wall Elapsed: 350.36 □ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options. 31 fps

Figure X: a) the Gazebo physics simulation of our rover, with obstacles loaded; b) the rviz window showing outputs of the simulation. Note obstacles being detected by camera stream in lower left and point cloud to the right.

ROS provides packages for simulating common sensors like stereo depth cameras, IMUs, and LiDAR, in Gazebo. Placing these simulation plugins on a rough model of the robot, made drivable by further plugins, we can move the rover through a simulated world and generate data streams that match our expected *actual* data inputs in both format and (ideally) content. These sensor models can be configured to match the hardware sensors, improving the fidelity of the simulation, and preventing nasty surprises down the road. The overall fidelity of the simulation is proportional to the amount of time we spend tweaking it, meaning that we can get better and better results by taking the time to carefully model our own sensors, perhaps even developing our own simulation plugins as necessary. Using our robot model, its sensor streams, and its motor plugins simultaneously, we can run our controller in simulation with a relatively high-level of real-world to simulated world parity.

All the data generated by these simulations can then be visualized or stored for later use. Leveraging these simulations, we can gain a large amount of data that can be used for later introspection, allowing us to design the “final” rover more intelligently.

Another useful feature of Gazebo simulation: the ability to create “world” files that save and restore the complete working state of a particular environment. Every obstacle, light source, and actor within the world will be placed exactly as they were when the world file was created. This provides us a useful means of reproducing testing scenarios, meaning that we can more meaningfully debug by removing a potential source of variation. Further, it opens the possibility of regression testing for specific functionalities. If we know that our rover is capable of autonomous navigation in a specific world file at a specific date, any future changes to the rover can then be tested within that world file to ensure that we’ve not broken any already existing functionality. In a software system as complex as this, being able to point to specific scenarios in which we *know* our software works is a powerful debugging tool when we encounter odd behavior or system-breaking bugs.

5.4.3.3 State Machine

The state machine depicted in figure <STATE_MACHINE> below will coordinate the behavior of the rover over its lifetime. Each state of the state machine will be responsible for specific behaviors of the rover, and they will transition to other states based on pre-defined inputs. We implemented this state machine using SMACH, a Python library intended for this purpose. By integrating this SMACH program with ROS’s Python bindings, provided by rospy, we can use this state machine as the high-level “executive” to make decisions about what to do and when.

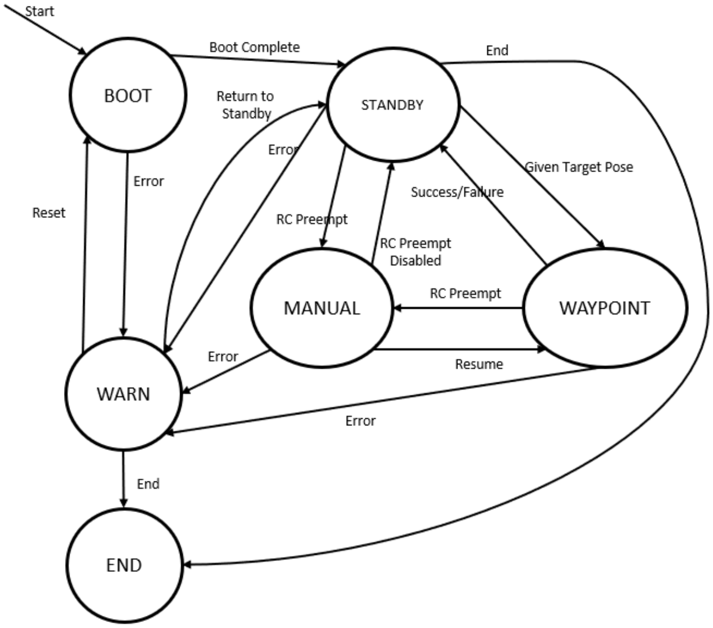


Figure <STATE_MACHINE>: the "executive" state machine of the rover, implemented in Python with SMACH and rospy.

Below is a brief description of each state in the diagram.

BOOT: Initialized with the Start signal, the controller begins its lifetime in the BOOT stage, where we ensure that the controller has all necessary input streams (e.g., RC commands and sensor data) and verify our ability to command the rover with calls to each software component. The BOOT state will not allow the rover to move on to dynamic behaviors without getting input from each sensor or being directly overridden by the operator, making it an important safety precaution, especially when the rover is operating at a significant distance from the ground station.

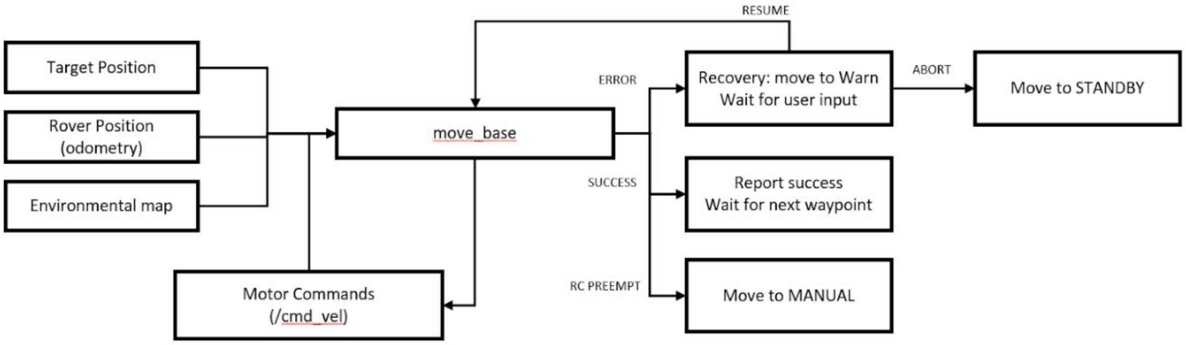
STANDBY: After successful Boot, the default state will be STANDBY, where the controller shall wait for command inputs. The STANDBY phase is where the rover will spend any time that it isn't directly executing an operator command. From STANDBY, we can choose whether to manually drive the rover with RC commands by transitioning to Manual/RC state or provide a navigation waypoint and allow the rover to (attempt) autonomous navigation in the WAYPOINT state. After completion of any commands, the FSM falls back to STANDBY.

MANUAL: if we provide the rover with an RC message and RC preempt, we transition to the MANUAL state. Here, we can command the rover with discrete "movement units" of forwards/backwards translation or left/right rotation. The approach used here doesn't require real-time control. After commands are sent, the rover will complete the movement provided without further input, then it will wait for further commands.

WARN: At any time/state during operation, a thrown error can transition the controller to the Warn state, where the rover will wait for the operator to diagnose the problem and either return to Standby or End execution of the controller.

END: The End state ensures the graceful termination of all processes to ensure the controller can be restarted without needing to reboot the SBC.

WAYPOINT: in the WAYPOINT state, our FSM attempts autonomous waypoint navigation. During waypoint navigation, the plan is to use ROS Navigation stack, a group of software packages designed to provide autonomous navigation to robots. When given a waypoint, the robot should navigate towards it without user intervention. One benefit to this approach is allowing the user to send commands asynchronously while the rover handles the details of driving as best as it can. If it runs into issues, then we switch to manual to help it along. This approach allows us to transmit less data less frequently and frees up our (limited) bandwidth for telemetry data. This is roughly akin to the Perseverance Rovers “blind drive” functionality, designed to deal with an 8-minute data delay from Earth to Mars. Proper functioning of this Waypoint state requires that the robot be properly configured to use the ROS move_base package. move_base will take in a target pose and generate motor commands to move us to that position. But, the effectiveness of this approach will depend on the quality of our mapping, so it’s important to provide a Warn state that gives us the chance to take control as needed e.g., if the rover gets stuck, or cannot find a valid path to the target location. The internal structure of the Waypoint state is shown below in figure <WAYPOINT_STATE>. Further details about the autonomous navigation system can be found in section 5.4.3.4.



5.4.3.4 Autonomous Navigation (“Blind Drive”)

One of the stated goals of this rover is the ability to “blind drive” like the NASA Perseverance rover: when provided a waypoint, we should autonomously navigate to it to the best of our abilities. This is a means of ameliorating the effects of long-distance communication with limited bandwidth. While our range needs certainly aren’t as extreme as those of a Mars rover, we still struggle with the same issues regarding communication latency. This makes our rover's ability to operate without real-time control an important means of safely executing its mission.

To implement our blind drive autonomous navigation, we'll make use of the ROS Navigation stack: a linked framework of ROS nodes designed to provide autonomous navigation to simple wheeled robots in roughly two-dimensional environments [C52].

When provided with an odometry source, such as the odometry generated by our interoceptive sensors, and exteroceptive sensor data such as point clouds, the Navigation stack will create multiple maps of our environment: a local map of our immediate surroundings relative to the rover and a global, world-fixed map of everything the rover has seen. Using this map, the Navigation stack can take in a pose target, such as the one we provide in our Command message and passes it to the move_base node. Properly configured, move_base can handle the specific motor commands necessary to reach a desired pose.

To integrate all that functionality into our controller state machine, we wrap the Navigation stack in our Waypoint state. Beginning from Standby, a pose target will transition the controller into Waypoint which will then handle the inputs and outputs of the Navigation stack by itself. This includes publishing/routing it's input pose as well as taking responsibility for the status returned by move_base as the rover moves. We keep careful watch for errors with the Warn state, so that issues with blind drive turn over control to the user.

5.4.3.5 Peripheral Microcontroller Software

Our rover requires quite a few peripheral devices to function as designed: a GPS, LoRa module, wheel encoders, and a motor controller. To interface with these peripherals and make our design robust to last-minute SBC changes, we've decided to use a "peripheral microcontroller." The peripheral MCU will coordinate data to/from our peripherals and pass it through a serial connection to the SBC. To do so, we had to develop software to run on that microcontroller. In this section, we provide the implementation details of that software.

The peripheral MCU software is written in C, with convenience functions and macros provided by the Raspberry Pi Pico SDK. While the use of Pico-specific functions makes our code less portable, it provides the advantage of faster development and a more robust (read: less likely to mysteriously break) implementation.

The peripheral MCU is intended to provide the SBC a single point of access to the array of peripheral devices on-board the rover. The embedded software on the peripheral MCU interfaces with each of the components attached to the MCU (e.g., wheel encoders, LoRa transceiver, motor controller, etc.) and exposes an interface where the rover can send commands and receive data. That interface between the rover software and the peripheral MCU is implemented as a simple serial connection (115200-8-N-1), sending plain-text messages in pre-defined message formats. On either side of the serial connection, message prefixes determine how a specific message is handled. For example, the “$MTR” prefix denotes a motor PWM command to be carried out on the peripheral MCU by sending PWM commands to the motor controller, while the “$CMD” prefix denotes a message from the Ground Station, received by the LoRa transceiver and passed through to the SBC for interpretation. On the MCU side, each message received is passed through a switch-statement that determines how the message is handled based on the prefix.

The details of how the peripheral MCU interfaces with each component follow:

The LoRa transceiver: the LoRa transceiver communicates via a UART (115200-8-N-1) protocol. Since messages can be sent/received at any time during the lifecycle of the peripheral MCU program, reading from and writing to the LoRa transceiver is handled on the Pico’s second core. This approach allows us to constantly poll for newly available data without sacrificing the responsiveness of other components. When a message is read in, it's added to an inter-core queue for asynchronous processing by our main core. Likewise, when our main core generates a message to be written to the LoRa receiver, it’s added to an inter-core queue to be handled asynchronously by the secondary core.

For data *from* the LoRa transceiver, we assume that all data we receive is a command for the rover, so we prepend that data with “$CMD” before passing it to the SBC. For data *to* the LoRa transceiver, we assume that that data comes from the SBC and is intended for the ground stations, so we prepend that data with “$TLM.”

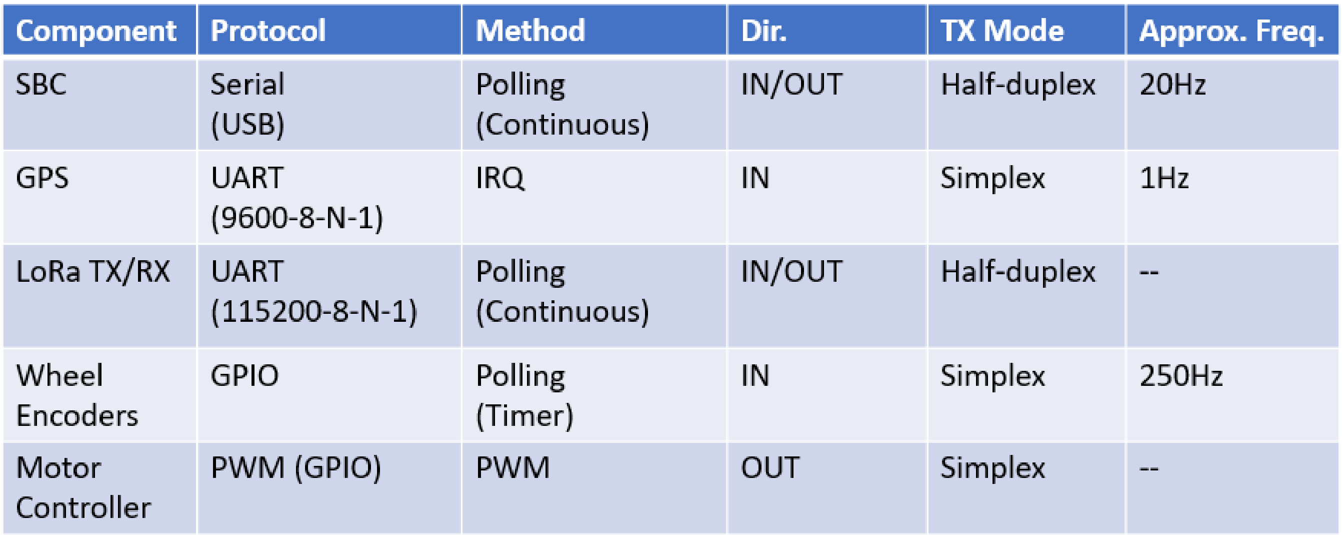
The motor controller: the Cytron MDD10A motor controller onboard the rover is configured to use four pins for controlling the motors: two digital (binary) GPIO pins for controlling motor directions, and two PWM pins for controlling motor speeds. Using functions provided by the Pico SDK, incoming motor commands (those with the “$MTR” prefix) are unpacked into variables and passed to a convenience function that sets the pin outputs. This is open-loop control: there’s no data output by the MCU with the “$MTR” prefix, so the rover has no way of directly knowing what PWM values are being used. Instead, this information can be derived from wheel velocities and the “linear” relationship between PWM and wheel speed.

The wheel encoders: the wheel encoders generate two waveforms on channels A and B, one that tracks rotations, and one that tracks direction. In concert, we can track the velocity (speed and direction) of the wheels. By configuring a timer at a high frequency (~250Hz), we can poll both channels and generate wheel ticks. These wheel ticks get passed to the SBC during the main loop, at a more reasonable 20Hz, after being prepended with the “$ENC” prefix.

Ultimately, the wheel ticks that are output by the encoders get turned into RPM readings on the SBC, then wheel velocities. After being fused into a “unicycle model” of our robot’s current position and velocity, these wheel velocities are an integral part of our odometry solution, and thus our mobile navigation solution.

The GPS module: the GPS module communicates via UART (9600-8-N-1). Data is formatted as standard NMEA sentences. Software interrupts are configured to fire every time data is available to be read from the GPS (at roughly 1Hz). After being read in, data is prepended with the “$GPS” prefix and passed to through the serial connection to the SBC be processed by the rover software.

Figure <P_MCU_COMMS> summarizes the communication between the peripheral MCU and other system components.



Component	Protocol	Method	Dir.	TX Mode	Approx. Freq.
SBC	Serial (USB)	Polling (Continuous)	IN/OUT	Half-duplex	20Hz
GPS	UART (9600-8-N-1)	IRQ	IN	Simplex	1Hz
LoRa TX/RX	UART (115200-8-N-1)	Polling (Continuous)	IN/OUT	Half-duplex	--
Wheel Encoders	GPIO	Polling (Timer)	IN	Simplex	250Hz
Motor Controller	PWM (GPIO)	PWM	OUT	Simplex	--

Figure <P_MCU_COMMS>: a summary of the communication protocols connecting peripherals to peripheral MCU

5.4.3.6 Other Software Nodes

To get a complex system like this working, we had to develop quite a few “support” programs, intended to process input signals, generate data, or interface with other programs as needed. These support programs are detailed in this section.

pico_bridge (Python): the “rover-end” of the serial connection to the Pi Pico; acts as the “single point of access” interface for every other program in our system. In essence, this program consists of a “main” loop and multiple callbacks.

The main loop reads/writes data to the serial connection with the Pico. Data read from the Pico is parsed based on the prefix (detailed in section 5.4.3.5), then is converted to the appropriate ROS message before being published on a pre-defined ROS topic. Other nodes in the system are configured to read from these topics for updates. It’s this interface that, for example, propagates wheel encoder ticks from the MCU to the rest of the system.

For data intended to be written *to* the Pico, we have a group of callbacks, one for each topic of interest, each running in its own thread. When we receive a message on a topic of interest, the associated callback fires – processing the data and packaging it for transmission to the Pico. This interface is how we send motor control messages, as well as telemetry data for transmission back to the ground station.

It's important to note that as a single point of access, this program also acts as a single point of failure. If this program stops working mid-execution, it means that we’ve lost access to almost all our sensor data, as well as the ability to control our motors or send data back to the ground station, which is less than optimal. As such, we’ve taken great pains to proof this program against unexpected death. This includes “sanitizing” inputs, to ensure that we don’t accidentally try to access information that isn’t there or try to process erroneous data. It also means taking great pains to log each error as it happens, so that we can find the root cause and tackle it, often using try-except statements or explicitly defined default values.

Errors reported by the peripheral MCU with the prefix “$ERR” often have to do with incorrect formatting of inputs, issues with the hardware configuration, or straight-up program crashes. These errors are propagated to the system using a special logging topic, /rover/errors, which serves as a real-time diagnostic tool, as well as an important source of data in retrospective examination.

imx219_stereo_cam_node (CPP): this custom CPP node interfaces with the Waveshare IMX-219 stereo camera, publishing camera streams and IMU data as ROS messages to be consumed by the rest of the system.

Initially, this program handled both reading frames from the cameras and publishing IMU data. However, we found significant latency issues with the camera streams that resisted debugging, and lead to a 2-3 second latency in the frames. That is, if you move your hand in front of the camera, the camera wouldn’t show it for a few seconds. So, we made the decision to use a third-party ROS node to handle camera frames and left this node to publish IMU data only.

To publish IMU data, we make use of a C library provided by Waveshare. We access the IMU data at a user-defined rate, then use that data to populate a ROS message, sensor_msgs/IMU. This data is then propagated through a pre-defined topic to the rest of the system.

telemetry_pub (Python): packages information about the current state of the rover into a custom-defined Telemetry message that then gets pushed to the peripheral MCU for transmission to ground station.

For telemetry, we have a defined telemetry message that contains the critical information about the state of our rover, including its current position in local (odometry) and global (GPS) frames and its current FSM state. This data is published by multiple software nodes in our system, and just needs to be collated into the Telemetry message and pushed to the Pico for transmission to the GS. telemetry_pub simply subscribes to our topics of interest using callback functions, collates the information as it comes in, then publishes a Telemetry message on a defined topic every 10s. pico_bridge is subscribed to that same topic and when it receives an update, it converts it to a format the Pico can understand and sends it down the wire.

encoder_odom (Python): given encoder ticks by way of the peripheral MCU, generates a nav_msgs/Odometry message containing an estimate of our current position and speed.

We need to turn wheel ticks from the MCU into a position estimate. To do so, encoder_odom subscribes to the wheel ticks’ messages, and every time it receives an update, it uses these ticks in combination with information about wheel radius and ticks-per-revolution to generate a wheel velocity for each wheel.

We model changes in position with a “unicycle” model i.e., it rotates in place and only goes forward and backward for translation. Using the wheel velocities we calculate, we can generate our change in x-, y-, and theta- (rotation about central axis) directions. We integrate (sum) this data over time to generate our current position. We publish updates at about 20Hz for the rest of the system to consume.

motor_control (Python): a node that converts the contents of a ROS Twist message into appropriate commands for our motor controller (using the Motors custom message definition).

This node generates Motors messages using a desired velocity as an input, as well as information from wheel ticks and geometric information about the wheels and the rover, such as wheelbase length and wheel radius. It makes use of a linear approximation of the relationship between PWM and motor velocity to determine PWMs based on an incoming velocity command. This approximation was determined experimentally, rather than derived.

5.4.3.7 Custom Message Definitions

To implement our system, we had to generate a few custom ROS message types to pass necessary information back and forth. These messages are described in detail below, and summarized in figure X.

5.4.3.7.1 Telemetry Message

```
# state machine state
std_msgs/String state
# relative (odom) pose
geometry_msgs/PoseStamped pose
# gps fix
sensor_msgs/NavSatFix fix
```

The Telemetry message definition

The Telemetry message is mean to capture the current state of various mission-critical attributes of the rover. As of this writing, it contains the current state of the executive FSM, as well as the rover position in relative (odom) and absolute (GPS) frames.

This message is published by telemetry_pub before ultimately being passed to the Pico via pico_bridge for transmission to the ground station.

5.4.3.7.2 Heartbeat Message

```
time time
```

The Heartbeat message definition

The Heartbeat message was designed to act as a simple means of determining that the ground station and rover were still connected in cases where no data was being actively sent otherwise. This was intended to prevent timeouts on either side of the connection It simply contains the current time (from the ROS clock), but it could feasibly contain any type of data, such as a static text string.

This message wasn’t used in practice, as we found that we never ran into a situation where transmissions were sparse enough to risk timeouts.

5.4.3.7.3 Cmd Message

```
std_msgs/Header header
geometry_msgs/Point target
rover_msg/RC rc
# geometry_msgs/Twist motors  # motor commands
std_msgs/Bool start         # if true, Rover exits BOOT state (if all is well)
std_msgs/Bool cancel        # if true, cancels current actions
std_msgs/Bool shutdown      # if true, stop robot execution
std_msgs/Bool rc_preempt    # if true, preempts current actions with motor commands
std_msgs/Bool pose_preempt  # if true, preempts current actions with waypoint target
```

The Cmd message definition

The Cmd message acts as the operator input to our rover system. Commands are parsed from the Pico into the Cmd message format, where our FSM state machine then implements the commands.

Currently, the message contains: a header which includes a timestamp, to stop us from operating on potentially old commands; a target point which tells the rover where to go in WAYPOINT; a custom RC message (defined below) for the rover to execute in MANUAL state; and a set of flags for further command of the rover.

5.4.3.7.4 RC Message

```
int8 forward
int8 reverse
int8 left
int8 right
```

The RC message definition

The RC message contains four integers, one for each direction of movement. Generally, it’s carried in the Cmd message. When the rover is in MANUAL state, it uses RC messages to determine how many discrete “movement units” it will travel in those directions.

5.4.3.7.5 Motors Message

```
std_msgs/Byte   dir1
std_msgs/Int8   pwm1
std_msgs/Byte   dir2
std_msgs/Int8   pwm2
```

The Motors message definition

The Motors message contains the data necessary to drive the motors, roughly matching the configuration of the motor controller. This message exists to act as a go-between for the motor_control and pico_bridge nodes. The contents of this message are passed to the Pico, which uses these fields to interface with the motor controller hardware.

5.4.4 Electrical Design

This section contains details concerning the electrical design of our rover, including information about our power distribution and our printed circuit board.

5.4.4.1 Power Distribution

The rover electrical system requires three separate power domains: a 5VDC power domain to power the Jetson Nano and its peripherals, as well as the wheel encoders; a 3.3VDC power domain to power the LoRa transceiver; and a 12VDC power domain to power the GHz transceiver and drivetrain. We generate these power domains by converting the voltage input from our 2S battery pack using step-up and step-down converters.

The power hierarchy of our rover is summarized in the power tree shown in figure <ROVER_POWER_DOMAINS> below.

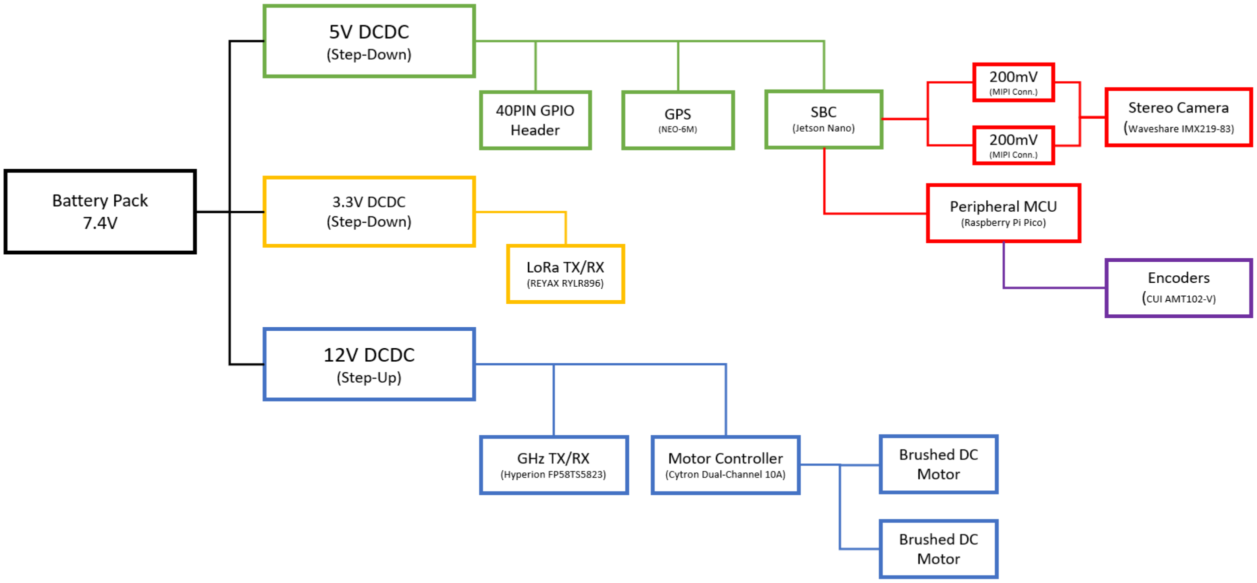


Figure 1:The power tree for our rover

5.4.4.2 Printed Circuit Board (PCB)

The printed circuit board (PCB) we designed for this rover serves two purposes: it distributes power to the components of our rover and serves as the interface to our peripheral microcontroller.

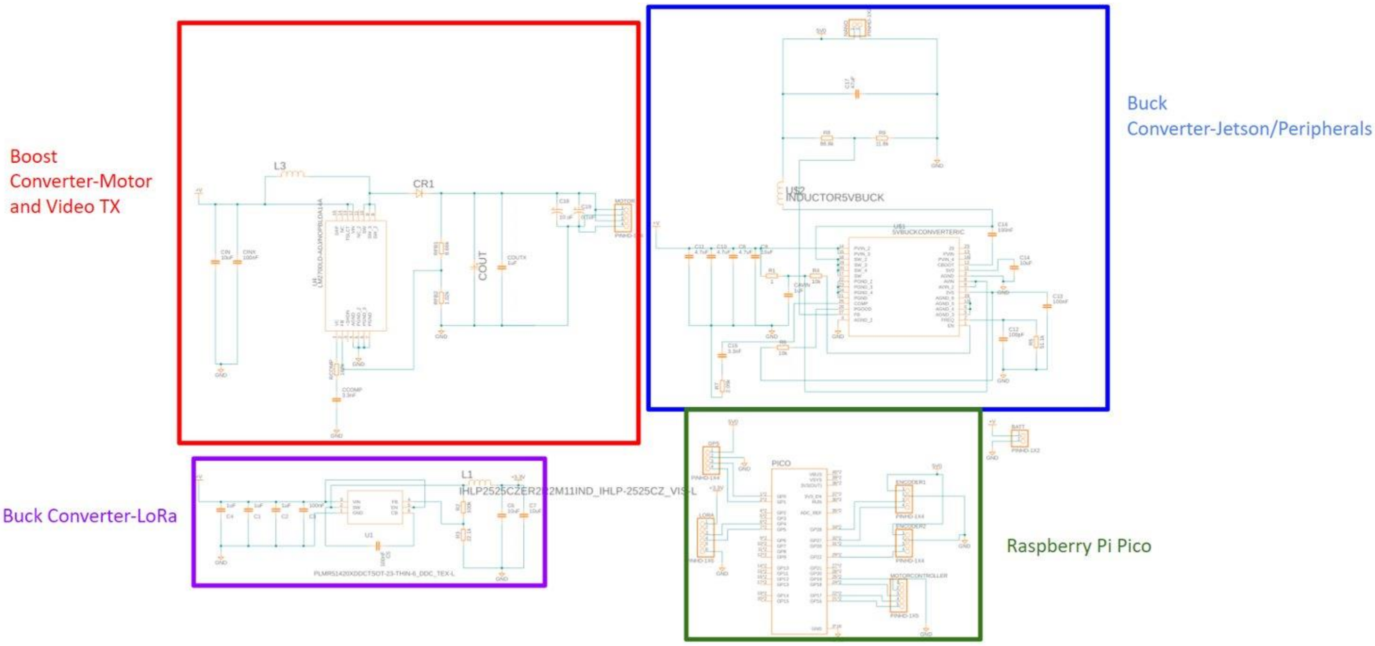


Figure <PCB Schematic>: shows the final PCB schematic for our design

The annotated total PCB schematic is shown in figure <PCB_SCHEMATIC>. The three DC-to-DC converters are explained in greater detail in section 5.1.

The interface to the Pico microcontroller is summarized by the connection schematic in figure <PICO_SCHEMATIC>. The PCB is designed to allow us to use headers to connect to all the peripherals, giving us flexibility in where those peripherals are placed in the final rover prototype.

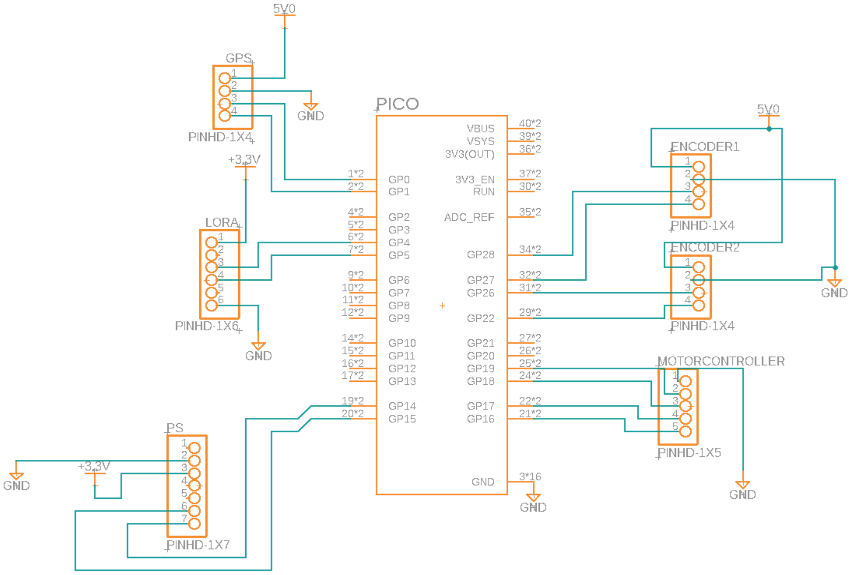


Figure <PICO_SCHEMATIC>: schematic showing the connections to our peripheral MCU

The overall PCB layout is shown below. In the top right, we have the 5VDC converter, which provides our 5VDC output to the SBC (connection NANO), as well as the GPS (connection GPS) and our two encoders (connections ENCODER1 and ENCODER2). Below the 5VDC converter is the 3.3VDC convertor, which provides power to the LoRa module (connection LORA). Finally, the 12VDC convertor in the bottom left provides power to the motor controller (MOTOR) and the video transmitter.

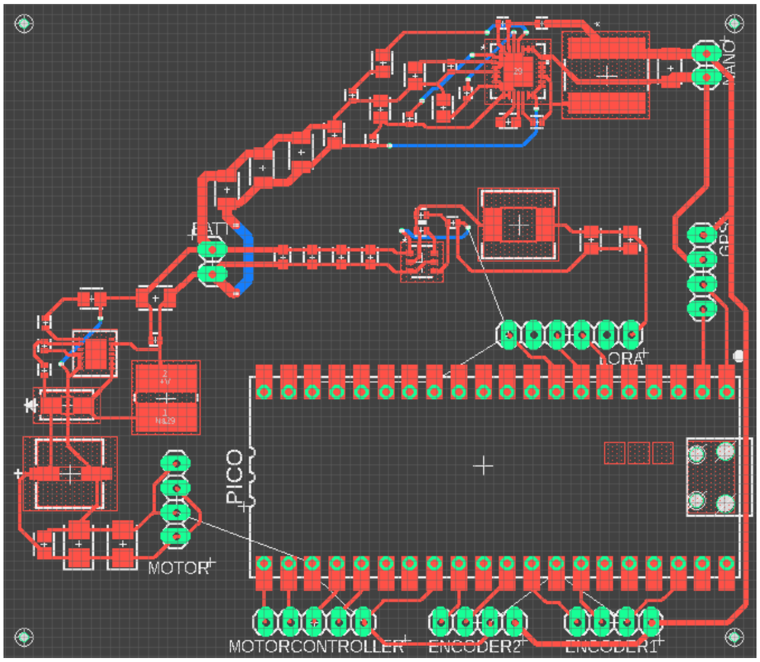


Figure <PCB Layout>: shows the final PCB layout of our design

5.4.5 Mechanical Design

WF: this section is mainly archival. During prototype construction, we made significant changes that rendered most of this section obsolete. As such, the mechanical drawings called out in this section were scrapped for actual prototype building. For proper information about the construction of the prototype, please refer to section 6.0.

The mechanical design of the rover was constrained by the size and shape of the Arcturus Payload Bay and the dimension requirements derived from it [R1.3]. With less than 12.7cm as our maximum radius (since the canister will take up a non-zero amount of that space), the only way to fit all our components is to design our rover to be “short and wide.” This allows us to keep our radius quite narrow (roughly 10cm as currently stands) while not sacrificing on any of our electronics.

We started with a conceptual design of the rover to determine what components we would need, and roughly where they might need to be placed. The conceptual design is shown below in figure <ROVER_CONCEPTUAL>.

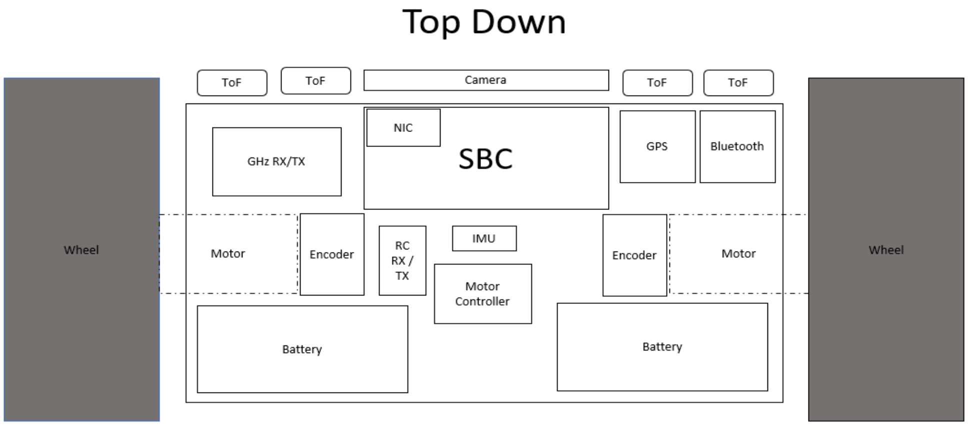


Figure <ROVER_CONCEPTUAL>: A conceptual design of the rover intended to facilitate the modelling process.

The conceptual design was then translated (as best as possible) into a full 3D CAD model using Fusion 360, shown below (figure <CAD_ROVER_TOTAL_INTERNAL>):

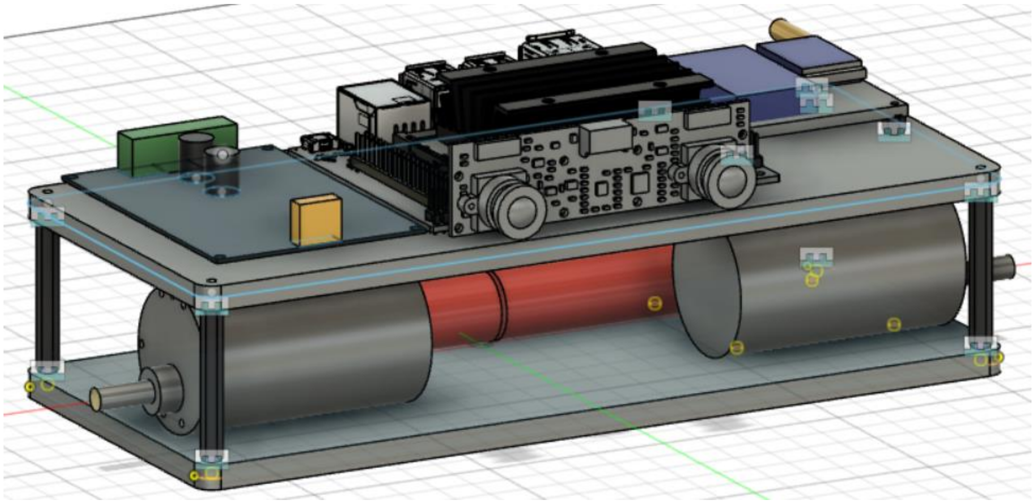


Figure <CAD_ROVER_TOTAL_INTERNAL>: The 3D CAD model of our rover and its internal components.

The CAD model was built using models of parts that matched our actual components as closely as possible in dimension to ensure we got as accurate a representation of the final product as possible. Further, attempts were made to ensure that all fasteners and mounting hardware were accounted for to minimize the amount of “hacking” that would need to be done during prototyping and final assembly.

The 3D CAD model was then translated into a set of mechanical drawings to determine final dimensions, as shown in figure <ROVER_DRAWING>

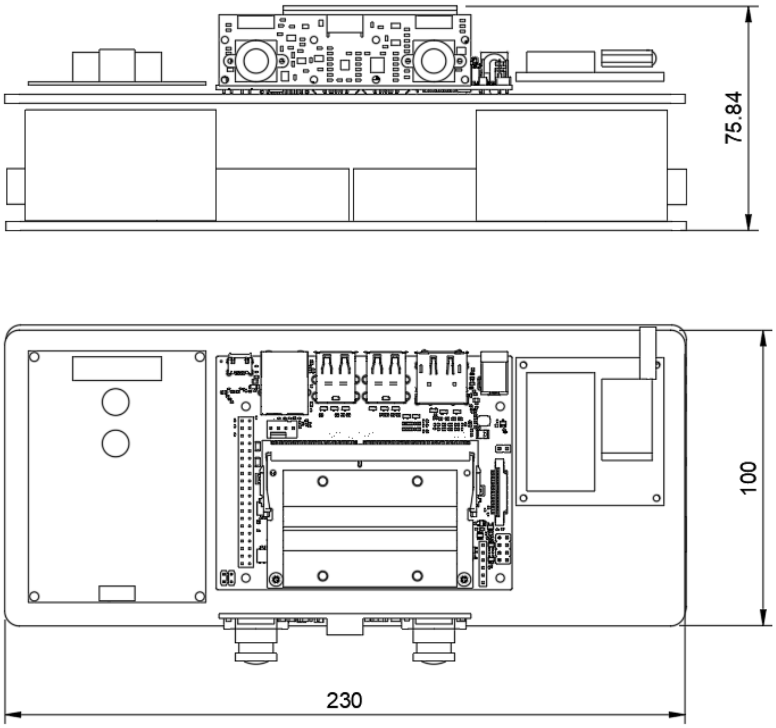


Figure <ROVER_DRAWING>: the auto-generated mechanical drawing of our rover, intended to show final dimensions.

As it stands, the dimensions of the rover meet our size requirement [R1.3], with its radius being only 10cm and length at 23cm, we have plenty of space for both the rover and the Payload Canister to fit within the payload bay. Significant "dead space" was left between major components to aid in passive cooling, provide space for wiring, and aid in easy assembly/disassembly. Should our design requirements change, this design can be compacted further via aggressive cable management and the introduction of an active cooling mechanism.

Creating the 3D model clarified how all our components would fit together internally. Ultimately, we decided on a multi-level design, using PVC foam board to create the "plates" that components will sit on. These plates will be separated via 3mm hex standoffs (shown above), providing stability with minimal material. PVC foam board was selected due to its relative lightness and strength when compared to other feasible materials such as plywood (too heavy), acrylic (heavy and difficult to cut), or even 3D printed rectangular volumes (too weak).

Batterie and motors will sit on the lowest level, away from all our sensors, our SBC, and our motor controller. This effectively separates the batteries from all our sensitive components and potentially provides protection in the case of a battery malfunction, such as a leak or overheating. This also serves to lower the center of gravity of the robot, since the batteries and motors are the heaviest components by a wide margin. Lowering the center of gravity should help stabilize the robot while it moves. Further, should we have space in our weight budget, we have the space to employ a ballast, such as lead pellets, to further lower the center of gravity and stabilize the robot. The motors will be mounted directly to the baseplate and the sides of the enclosure via a motor mount. Whether that mount will be a custom design and fabrication, or a COTS part is yet to be determined. Batteries will be removable. The lower level is shown below in figure <ROVER_MOTOR_BOARD>.

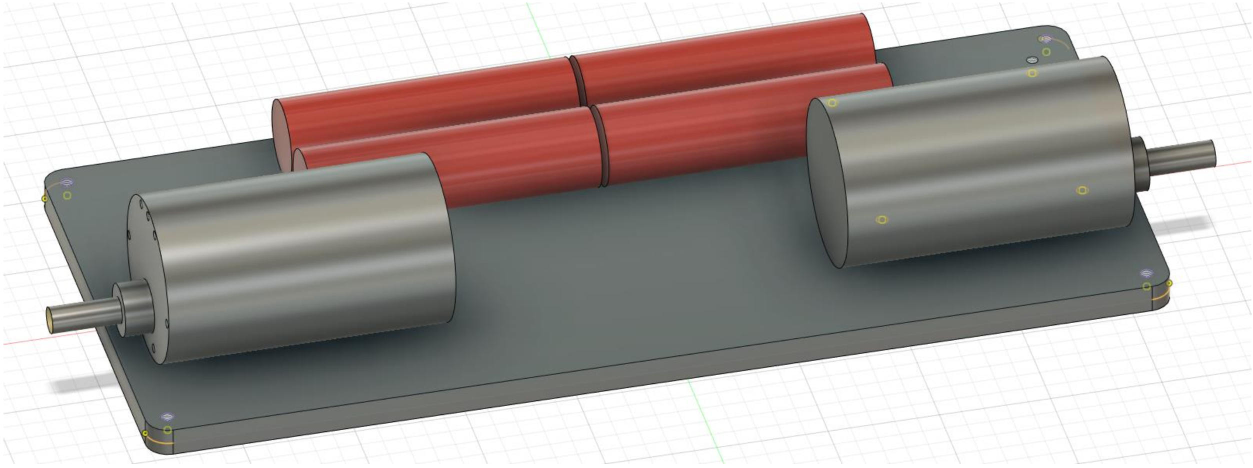


Figure <ROVER_MOTOR_BOARD>: The 3D design of the "lower level" of our multi-tiered design, housing the batteries and the motors.

On the upper level, the SBC, motor controller, and all exteroceptive sensors will sit. This includes the stereo camera, the RF communications components, the IMU, and the GPS, as shown below in figure <ROVER_BASE_BOARD>.

The RF communications components, namely the REYAX LoRa and Hyperion GHz modules, sit on the right side, on their own miniature baseplate. Since both modules will likely have antennas that need exterior access, the standalone plate allows us to be more flexible with our placement of these components, as well as providing potential space for the GPS and IMU underneath it, should we switch to using taller standoffs. The motor controller sites on the left side of the baseplate and will be connected to it using either plastic standoffs or rubber spacers. There will be a through-hole or cutout in the baseplate to allow the wiring for the motors to be passed through to the lower level. The SBC will be in the center of the upper level, partially to simplify the connections to other components, and partially to keep its weight central to stabilize the rover. The SBC will also be connected to the baseplate with standoffs or spacers, with the explicit intention of minimizing potential vibrations. Whether or not this ends up being a serious concern for the solid-state components on the Jetson Nano remains to be seen. Finally, the Waveshare stereo camera will be connected by a (not-yet-designed) custom mount that will connect it to the baseplate and potentially the case of the rover. It's important that the Waveshare camera be central to the robot, not only to make sensing easier, but because the integrated IMU will provide the most useful data if it is placed on or near the axis of rotation.

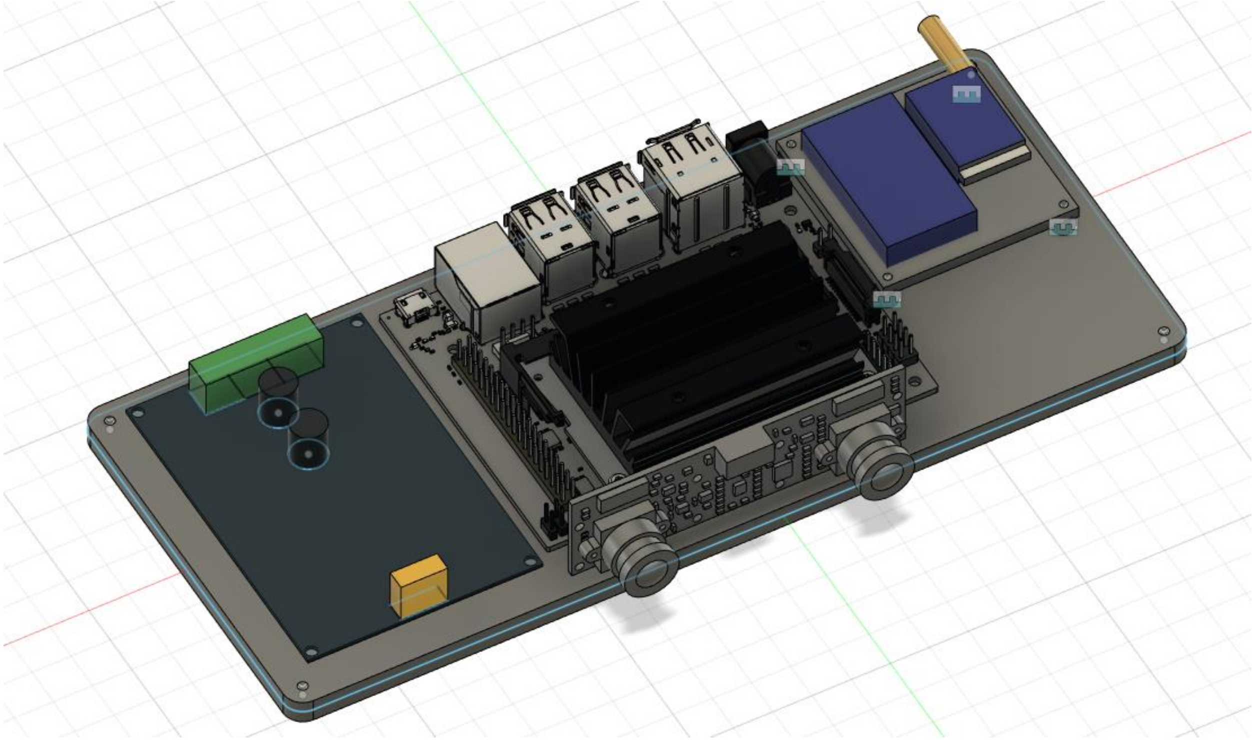


Figure <ROVER_BASE_BOARD>: 3D model of the "top" level of our multi-tiered design, housing sensors, SBC, and communications components

For the rover "case" that our components will sit in, the dimension constraints (<6in diameter) influenced us to design the rover as "short and wide," with the longest axis being perpendicular to the wheels. This makes the purchase of any commercial-off-the-shelf (COTS) rover chassis difficult, since frequently they have the longest axis parallel to the wheels. The small size restricts us even further, as very few of the ubiquitous frames can fit within the dimension constraints given us. Buying a COTS case, such as a Pelican case, and adapting it to act as the enclosure for our robot was also considered. However, finding one with the proper dimensions proved an insurmountable task. Therefore, we've decided to make use of 3D printing to print a case that we've designed. This is feasible only because of the relatively small size of our rover and our access to large-bed 3D printers. A larger structure would require printing in many separate pieces, potentially further decreasing mechanical strength. However, this 3D printing approach comes with caveats: 1) it's possible that even though our print is relatively small, a 3D print won't have the durability to survive the forces of launch, deployment, and landing. While it's possible to print with a large infill percentage to increase strength, only so much can be done to overcome the inherently fragile nature of 3D printed volumes.

The case itself will be designed to match IP50 as closely as possible, to allow us to operate in a sandy/dusty environment. A complicating factor is the need for our internal components to have access to the exterior of the case. The wheel shafts, the comms equipment antennas, and the camera lenses all need to be outside of the case, while still connected to the components on the interior. This will require creating through-holes in the case, and potentially compromising its dust-resistance.

Another (currently unsolved) factor is wheel placement. Initially, two large wheels on either side of the rover were envisioned. These wheels would have a radius roughly equal to the maximum radius of the rover. However, during software simulation, it became clear that this approach had serious complications. The two wheels' large separation leaves the main body of the rover too vulnerable to accelerations, causing it to tip and wobble back and forth during movement. This makes mapping and state estimation unpredictable, or potentially impossible, and leaves the rover vulnerable to flipping backwards during forward acceleration. While it's possible that a low enough center of gravity could ameliorate this issue, a third stabilizing structure on the back of the rover will likely be required to maintain a constant attitude for the rover sensors. As it stands, that stabilizing structure will likely be a 3D printed "foot" or "sled" to sit on the back end of the rover and stop it from tipping backwards.

5.4.6 Bill of Materials (BOM)

The complete bill of materials (BOM) for the proposed design of the Rover Payload is as follows:

Function	Name	Price/unit	Quant.	Total Price	Cust. Cost
SBC	Jetson Nano 4GB Dev. Kit *	$99.00	1	$99.00	$0.00
Camera	Waveshare Stereo Camera	$44.99	1	$44.99	$0.00
MCU	Raspberry Pi Pico	$3.60	1	$3.60	$0.00
GPS	GPS NEO-6M	$11.59	1	$11.59	$0.00
Motor Driver	Cytron Dual Channel 10A	$21.18	1	$21.18	$0.00
Motors	Greartisan DC 12V 250RPM Worm Gear	$28.99	2	$57.98	$0.00
Wheel Encoders	ENC-AMT102-V	$23.86	2	$47.72	$47.72
Wheels	Dagu Wild Thumper Wheels 120x60mm *	$17.95	2	$35.90	$0.00
Batteries	Zeee 2s 7.4V 4600mAh LiPo	$27.89	1	$27.89	$0.00
BMS Circuit	ACEIRMC 4A 2S BMS *	$2.00	1	$2.00	$0.00
Boost Converter	ACEIRMC XL6019 5A DCDC *	$3.33	1	$3.33	$0.00
Enclosure	Zulkit ABS Project Box IP65	$9.99	2	$19.98	$0.00
LoRa TX/RX	RYLR896 LoRa	$24.47	2	$48.94	$0.00
Video TX	TS5823 Transmitter	$8.99	1	$8.99	$8.99
Video RX	5.8 GHz downlink Receiver	$24.60	1	$24.60	$24.60
HDMI->Analog	HDMI2AV Upscaler (modified)	$10.99	1	$10.99	$0.00
Connector	JST-XH 2.54mm Connector Kit *	$8.99	1	$8.99	$0.00
Connector	Amass XT30U Pair	$1.10	5	$5.50	$0.00
	* parts were scavenged, rather than purchased				
			Total	$483.17	81.31

Note: prices are accurate as of last revision of this section on 4/23/22

It's important to note that the BOM represents only the MSRP, off-the-shelf cost of these components. However, quite a few of our components were scavenged from the Robotics

Club lab or furnished by team members from their personal supplies. In this way, the total cost to the customer was significantly decreased.

5.5 Subsystem - Ground Station

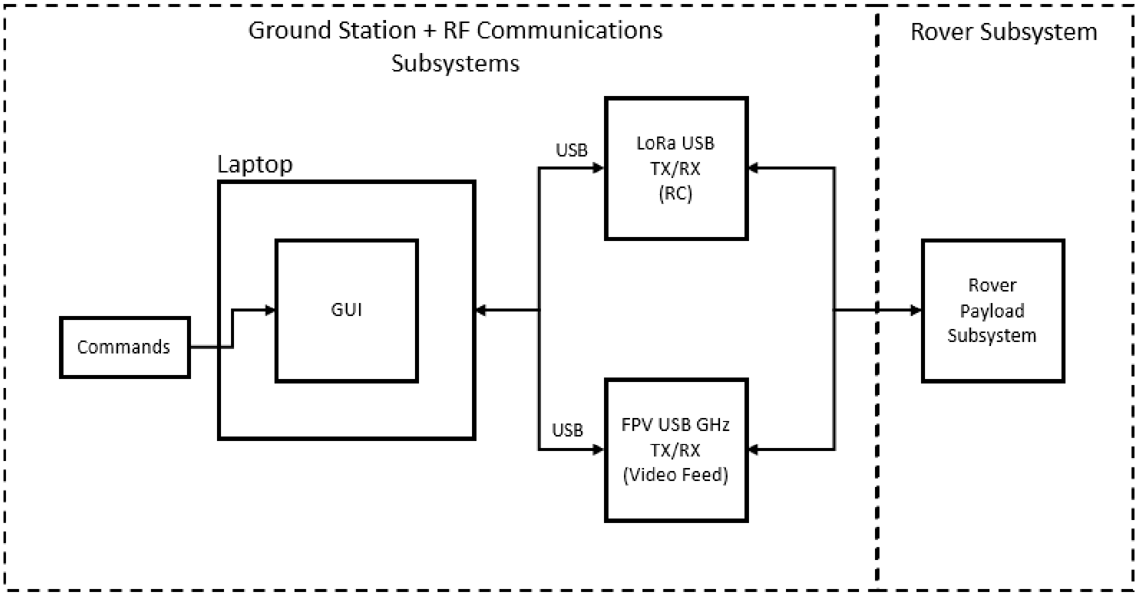
Before we go any further, we must explain what a ground station is. A ground station is basically a control station that use an antenna and associated electronic equipment for transmitting, receiving, tracking, or controlling equipment or messages. The ground station for this project will be used to receive video feed, GPS location, and allow us to control our rover.

For the software, we utilize a GUI maker called Qt designer to make a layout of the GUI. Specifically, we use PyQt5, which is a python compatible version of QT version 5 to convert that GUI into python code to edit it and add extra features such as necessary. We also use ROS, specifically rviz and rqt, to help facilitate the development of the ground station, for a more in-depth explanation for how we are going to use ROS look at Section 5.4.2.1.

For the hardware, we’re using a laptop, an FPV Receiver 5.8 GHz LoRa USB receiver, a joystick or possibly keyboard controls on a laptop, and a LoRa 915 MHz transceiver. Now the reason why we are using an FPV Receiver is what we use to get a video feed from the rover. Now the reason why the FPV is a 5.8ghz receiver is because the project does not allow a 1.0 GHz to 2.0 GHz signal because of the interference they would cause. However, because of the limitation due to size, weight, and power we’re aiming to get a 480p video resolution. In addition, the LoRa will be used by us to send a receive signal to control the rover. However, the signal at which we will do this for the LoRa will be 915 MHz, the reason why we are using the 915 MHz range for the LoRa USB transceiver is that the 70-centimeter band, which is the frequency range 420-450 MHz, is banned due to the problems it may cause with interference. Other than that hardware won’t be too much of an issue because ROS has support for various kinds of sensors.

5.5.1 Overview

Construction of the ground station will start with making a basic GUI in a python GUI maker software called Qt maker. From there we convert it into python code using PyQt5 and edit it in a text editor. Once the ground station code is finished, we use ROS, specifically the rviz and rqt packages, for software. Rviz is a 3D visualization tool for ROS. We use rviz to interact with the sensors as well as make a GUI with the aid of rqt. The hardware side will handle the communication side for the rover. Since the rover will be less than a thousand meters away, we don’t need long-range communication devices instead will use short-range communication devices such as an FPV receiver at 5.8 GHz to receive a video feed from the rover and a LoRA usb receiver at 915 MHz on the rover to send a receive controls, and laptop to run the software necessary for the project respectively.

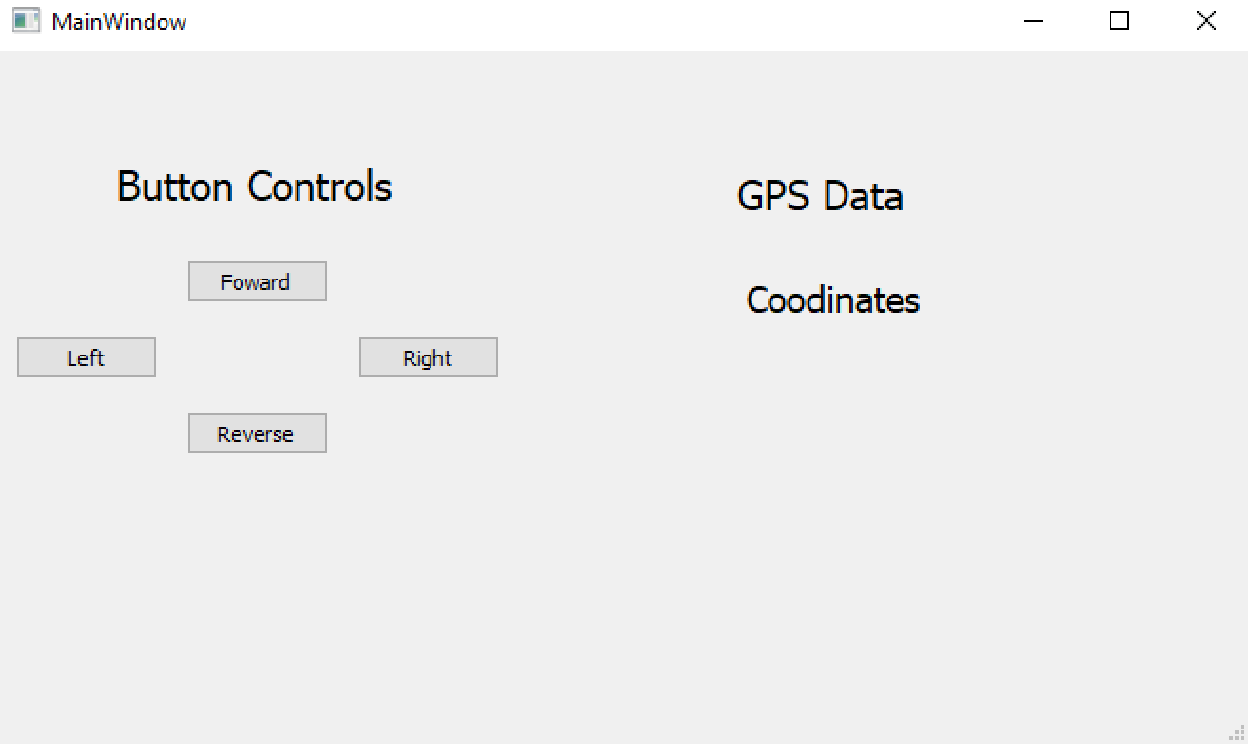


5.5.2 Software Design

For the software part of the ground station, we developed a custom GUI to be run on a laptop that will run Ubuntu Linux 18.04 natively or through a virtual machine. The GUI will be created using Python, its built-in modules, and external pip-installable modules. In our final design, the external modules that we employed were: PyQt5, to create the layout and GUI elements, Pyqtlet, to create user-interactable maps, PyQtWebEngine, to embed those maps into our GUI, PySerial, for an easy way to interact with serial ports, and Qt_material, for its elegant, modern pre-built stylesheets. The ground station will also be equipped with VLC to display the video input from the SkyDroid FPV receiver’s serial port. As we developed the code for our peripheral MCU and communication modules, we had to make significant changes to the ground station for seamless integration. As shown below, we made two revisions to our initial design.

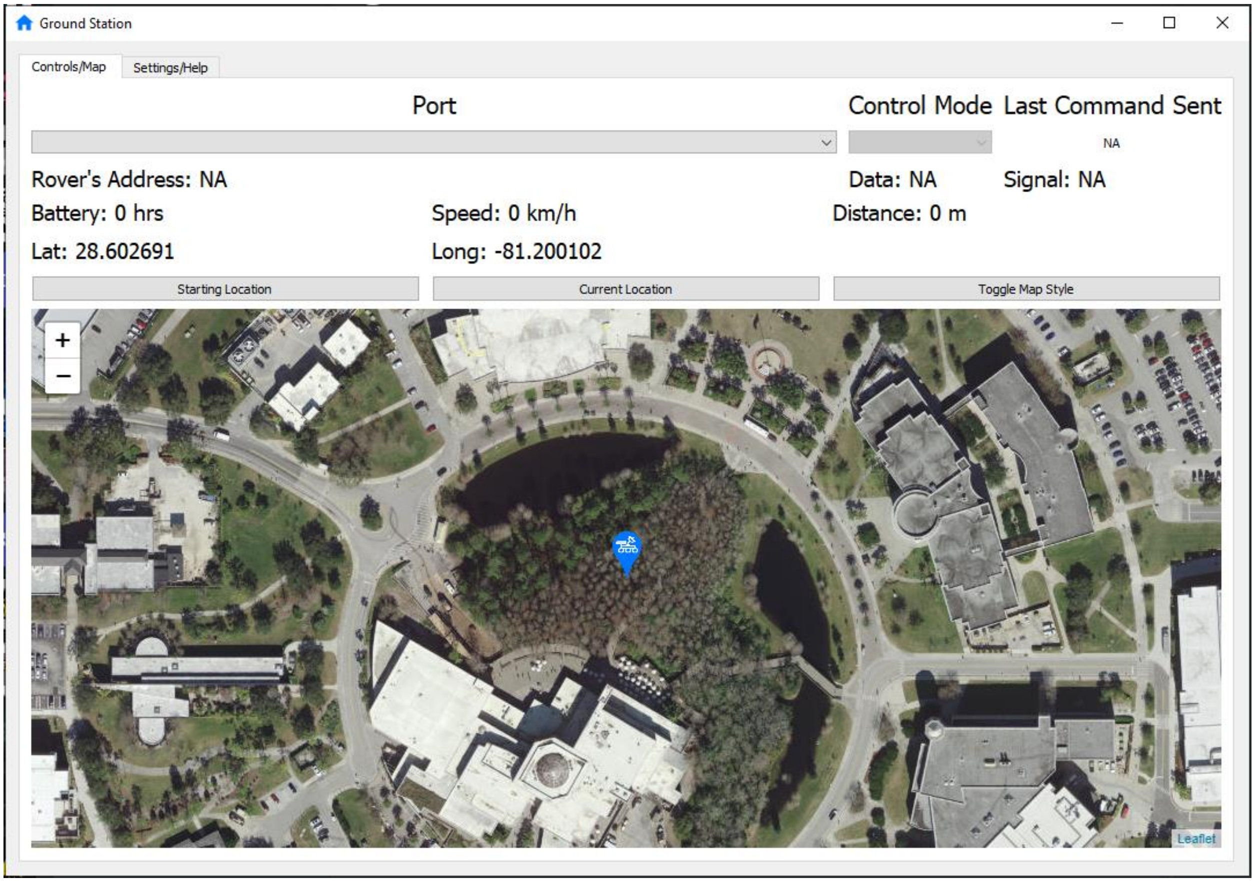
5.5.2.1 First Prototype Design

Although lackluster, our first prototype allowed us to get familiar with the PyQt5 library. In the picture below, you can see that all we had were four buttons for traveling and three labels. The buttons did not have any functionality linked to them, and the “Coordinates” label wasn’t dynamically updated since no GPS data was received from the ground station from the rover. Looking back, the first prototype was a naïve representation of what we thought the ground station software should look like but got us moving in the right direction.



5.5.2.2 Second Prototype Design

Moving on to our second prototype, the amount of improvement is clearly visible. A lot of the core functionality in our final design is present in this revision. Starting from the top, there are two tabs to reduce cluttering everything on one window. In the "Controls/Map" tab, we have two lists that allow the user to change the port to send data the LoRa transceiver and the desired control mode. At startup, the port list gets populated with the serial devices attached to the computer. Once a port is selected, LoRa AT commands are sent to configure the LoRa for rover communication. In this prototype, there were only two control modes: "Blind Drive" and "Manual". Both control modes are shown in 5.5.2.3 Final GUI Design. If "Blind Drive" is selected, two QLineEdit fields are displayed along with a travel button. The user can then enter two latitude and longitude coordinates and send them to the rover with a click of the travel button. If "Manual" is selected, four directional buttons are displayed. When clicked, they send a command to the rover to move in that direction. Then we have nine dynamic labels that display information about the following: last command sent from the ground station to rover, the rover's LoRa address, any data received from the rover, the signal strength of the LoRa connection, the remaining battery of the rover, the current speed of the rover, the distance the rover has traveled since boot, and the rover's latitude and longitude. All that's left in this tab are three map-related buttons and the interactable map. Using QtWebEngine and Pyqtlet, we were able to create a map that can be dragged and resized by the user. Two markers are shown, and they indicate the ground station and rover's location. The aptly named "Starting Location" and "Current Location" buttons pan the map between those two locations. The final button allows the user to toggle between a minimalistic, dark map and a world view map. In the “Settings/Help” tab, the user can change the serial port, LoRa port, and map options. This tab was later renamed to “Settings/Debug” and is shown in 5.5.2.3 Final GUI Design. All of the modifiable options are initiated to sane defaults at startup, so no modification is necessary. The serial port options that can be changed are baud rate, byte size, and timeout. The LoRa has much more configurable options. These options are spreading factor, bandwidth, coding rate, programmed preamble, addresses of the ground station and rover, network ID, band, and baud rate. The “Set All” button can be used to configure the LoRa with these options. The first 4 options are initialized with optimal values determined through extensive LoRa testing. The “Set Parameters” button configures these four options. The LoRa info button can be used to check the LoRa’s current configuration. This tab also allows the user to send custom commands and view their response. The last option is a map option that when set, automatically pans the map to the rover’s current location. The ground station also keeps logs of following: rover coordinates, messages sent by the ground station, messages received by the ground station, and other rover telemetry. They are stored at text documents in the appropriately named “logs” folder. Finally, the application gets its own icon and window name. All images used by the ground station software are in the “images” folder.



NA

Rover's Address: NA

Data: NA

Signal: NA

Battery: 0 hrs

Speed: 0 km/h

Distance: 0 m

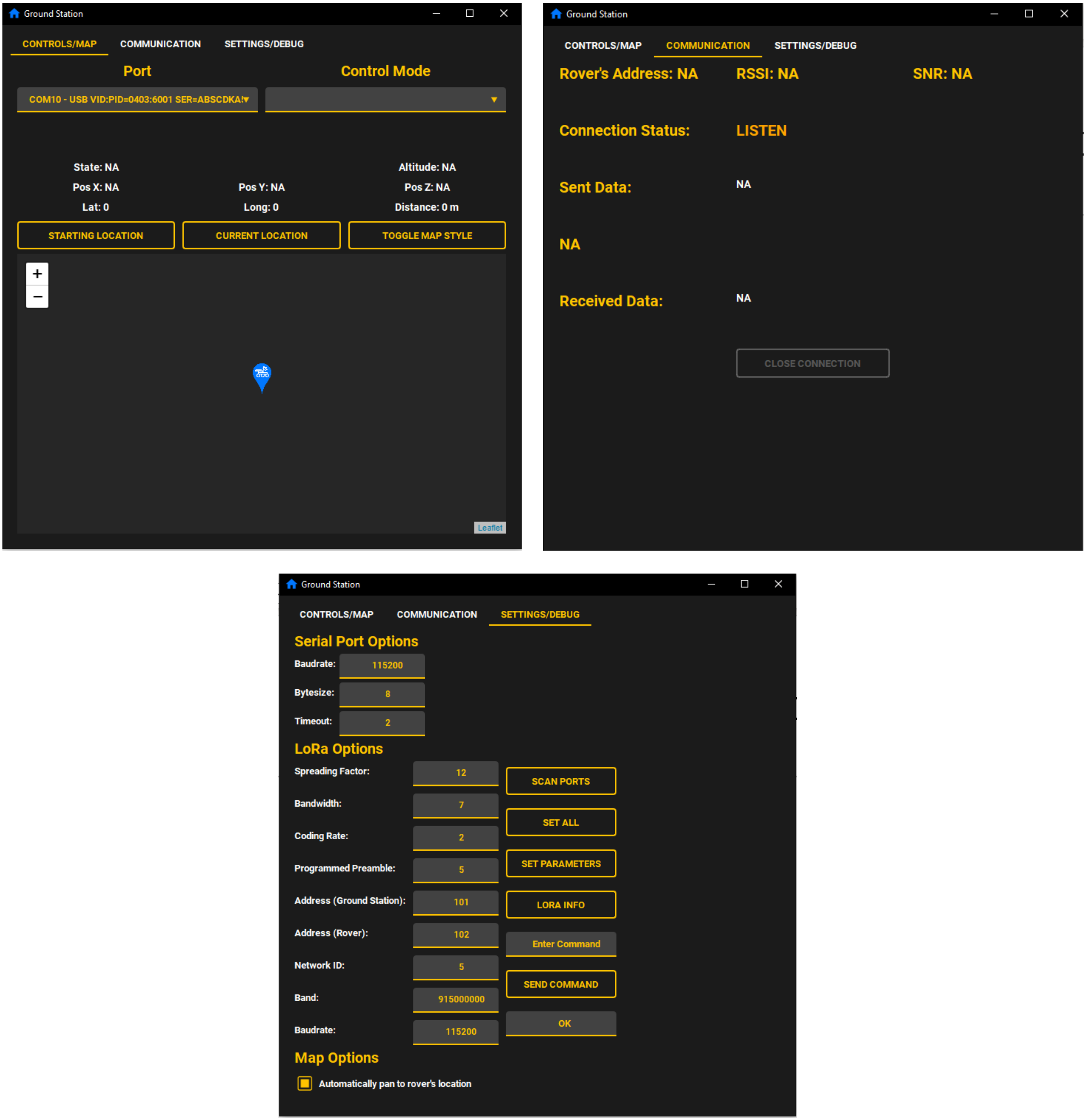
Lat: 28.602691

Long: -81.200102

5.5.2.3 Final GUI Design

The final design for the GUI has three tabs. The first tab is the controls/map section, the second tab is the communication, and the third tab is settings/debugging. The first tab deals with the initial setup and rover telemetry. For all intents and purposes, this is the ground station's primary tab. To begin operating the rover, the LoRa port must first be selected. This will configure the LoRa and establish a connection with the rover. Afterwards, one of the following control modes can be selected: Blind Drive, Manual 1, and Manual 2. Blind Drive takes two latitude and longitude coordinates and transmits it to the rover. In terms of accuracy for longitudinal and longitudinal positioning it is down to 5 decimal places for accuracy. Manual 1 will relay XYZ coordinates, as well as rover state parameters, to the rover. Manual 2 allows travel by specifying the number of units and cardinal direction in which the rover should move; these directions are relative to the rover's orientation. Any incoming telemetry will be shown below the controls. Further down, we have two buttons that will pan the map to the position of the ground station and rover. The last button will toggle the map style between a minimalistic, dark map and a realistic, world map. The map shows the current location and path of the rover. The second tab allows us to see the state of communication between the rover and ground station. It will also tell us if the message needs to be retransmitted or the connection is lost. The third tab allows us to change settings related to the LoRa serial port, LoRa configuration, and map. By default, the most optimal LoRa parameters are set. If needed, we can use this tab for debugging hardware or communication issues.

Below is our final ground station design's three tabs. Control modes will be discussed later in section 7.3 Controlling the Rover.



Port

COM10 - USB VID:PID=0403:6001 SER=ABSCDKA!

Control Mode

State: NA

Altitude: NA

Pos X: NA

Pos Y: NA

Pos Z: NA

Lat: 0

Long: 0

Distance: 0 m

Rover's Address: NA

RSSI: NA

SNR: NA

Connection Status: LISTEN

Sent Data: NA

NA

Received Data: NA

Serial Port Options

Baudrate: 115200

Bytesize: 8

Timeout: 2

LoRa Options

Spreading Factor: 12

Bandwidth: 7

Coding Rate: 2

Programmed Preamble: 5

Address (Ground Station): 101

Address (Rover): 102

Network ID: 5

Band: 915000000

Baudrate: 115200

Map Options

☑ Automatically pan to rover's location

Figure <GUI_FINAL>: screenshots of the final design of the GUI

5.5.3 Hardware Design

A USB to TTL serial adapter will be used to connect the RYLR896 transceiver board to the ground station computer. LoRa offers significant flexibility in terms of data transmission, allowing for data transmission rates from a low of 11 bit/s to a high of 253 kbit/s. The data transmission rate can be configured indirectly by changing the bandwidth, spreading factor, and coding rate. In general, as the data rate increases, the range decreases. It is necessary to balance the data rate with the desired range of the rover.

Now for the video feed that will be on a sperate screen and not through the GUI. The video feed will be handled by Skydroid FPV receiver USB stick. There will be a sperate screen for displaying video from the rover. Now the FPV receiver will be send a low-resolution video of about 480p to the command station on a 5.8 GHZ frequency. There will be a video recording of the video feed so if the user wants to examine the data later, they may do so. There are applications to connect, record, and view on an FPV.

5.6 Deployment

All our work on this rover is for naught if we don’t have a successful deployment from the Arcturus rocket platform. In figure <DEPLOYMENT_FLOW> below, the general flow of the deployment is pictured:

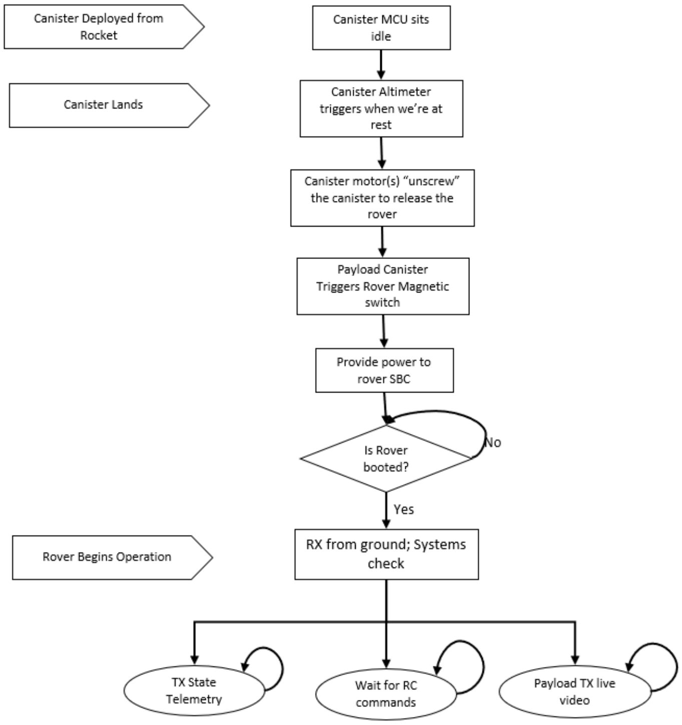

Canister Deployed from Rocket
Canister MCU sits idle
Canister Lands
Canister Altimeter triggers when we’re at rest
Canister motor(s) “unscrew” the canister to release the rover
Payload Canister Triggers Rover Magnetic switch
Provide power to rover SBC
No
Is Rover booted?
Yes
Rover Begins Operation
RX from ground; Systems check
TX State Telemetry
Wait for RC commands
Payload TX live video

Figure <DEPLOYMENT_FLOW>: The flowchart of events that leads from canister deployment in the air to rover deployment on the ground.

Before launch, the rover needs to be loaded into the Payload Canister, and the Payload Canister must be loaded into the Payload Bay onboard Arcturus. After this loading, the time that the rocket may “sit on the tarmac” waiting for launch is not yet known, so it’s important that we do our best to conserve energy. To this end, the rover will not be booted until after the canister makes its landing. The canister MCU and electronics will sit in a low-power idle state until they’re needed.

In the air when the canister is deployed from the rocket, an onboard altimeter will track our elevation (and elevation delta). When we come to rest, the change in elevation will fall to zero, triggering the “unscrewing” of the rover from the canister. As the rover is slowly released, the canister will trigger a magnetic reed switch inside the rover, closing the power circuit and booting the SBC.

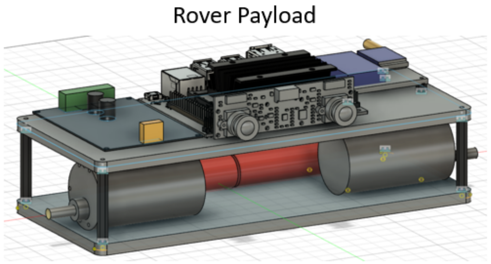
After systems checks indicate all is well and we successfully exchange Heartbeat messages (section 5.2.2) between the rover and the ground station, the rover can begin its ground operation. At this point, the canister becomes effectively inert.

5.7 Summary of Design

Our project consists of three major components: the rover payload, it’s launch/landing canister, and the RF comms and ground station software to connect users on the ground to the rover at the landing site.

The Payload Canister is designed to fit within the Payload Bay of the Arcturus rocket platform, containing the Rover Payload during launch, deployment, and descent. It’s been designed to withstand significant mechanical stressors, such as the forces of takeoff and non-ideal landings. Further, the canister provides a means by which to power on the rover after landing, ensuring that the rover does not waste its battery life sitting at the launch pad. Powering on when the rover is inside the canister at rest provides a “safe” place for the rover to power on and perform system checks, without worrying about aberrant data points due to forces measured during the trip down that might interfere with our state estimation.

After landing and being deployed by the Payload Canister, the Rover Payload becomes the focus of the system. It is the rover’s responsibility to traverse the terrain of the Mojave Desert and transmit back a live video feed to the ground station. The traversal will be handled either directly through explicit RC commands from operator, or via autonomous navigation towards user provided waypoints. While it operates, the rover is designed to generate an estimate of its own position alongside a map of its environment using a mix of interoceptive (IMU, GPS, wheel encoders) and exteroceptive (depth camera) sensors. Figure <ROVER_SUMMARY> summarizes the major functionality of the rover.



Rover Payload

- Deployed from Canister
- RC command
- Live video feed
- Uncontrolled terrain traversal
- Localization and mapping
- Autonomous “blind drive”

Communication between the rover and the ground station is carried out using the RF Communications and Ground Station subsystems in tandem. The RF Communications subsystems is comprised of the MHz and GHz transceiver pairs onboard the rover and at the ground station, as well as the software used to interface between them using predefined messages: Heartbeat, Telemetry, and Command.

6.0 Project Prototype Construction

This section details the construction of our final prototype, as well as a general build plan and photos of the in-progress and complete rover. Figure <PROTOTYPE> shows the completed rover prototype.

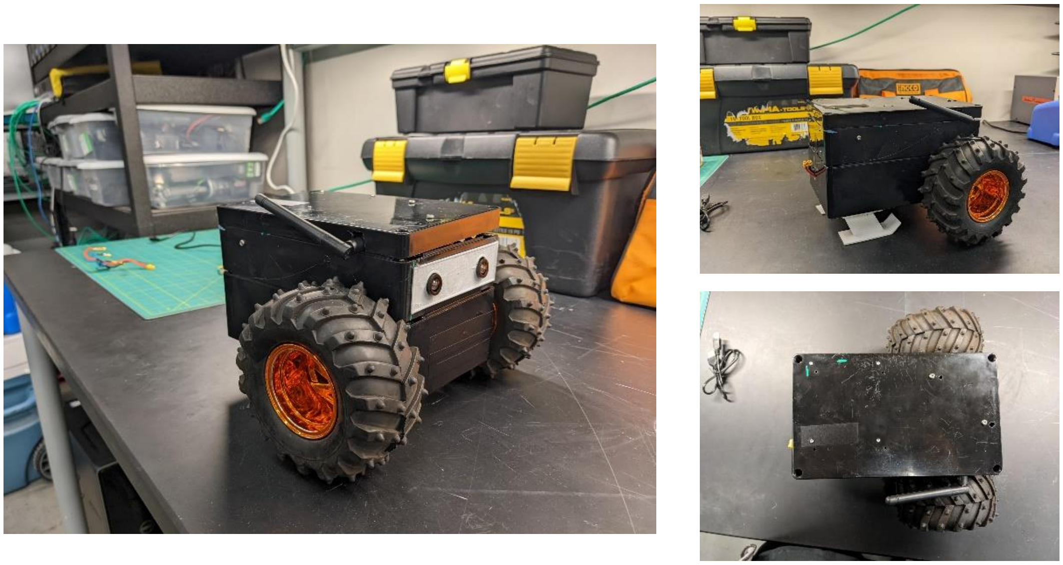


Figure <PROTOTYPE>: a) front view of rover prototype; b) side view; c) top-down view

Prototype construction took place in the lab space of the Robotics Club of Central Florida, a registered student organization at the University of Central Florida. The lab provides access to fully equipped electronics benches, hand tools, and machinery as shown in figure <LAB_SPACE> below.

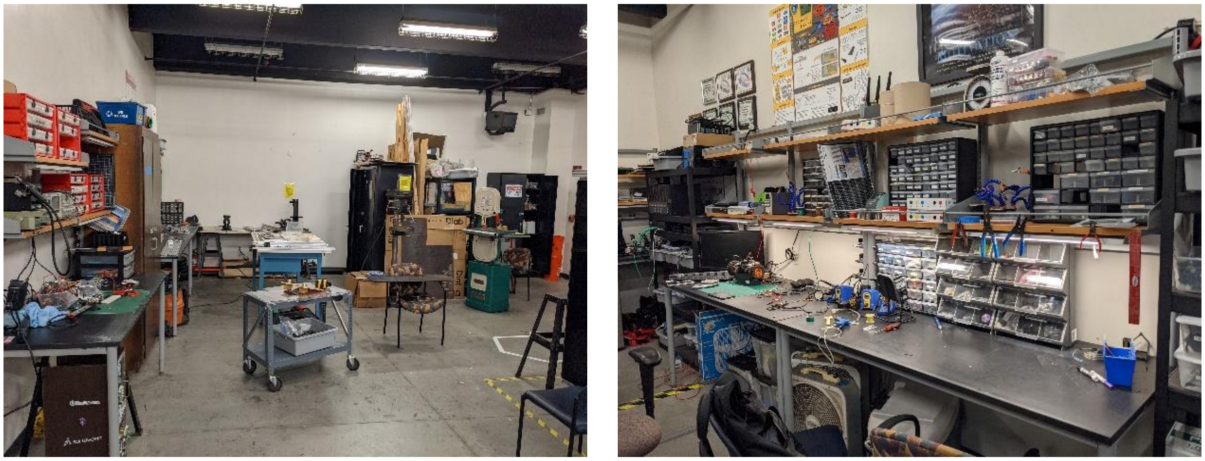


Figure <LAB_SPACE>: the facilities provided by the Robotics Club of Central Florida including a machine shop (left) and electronics benches (right)

6.1 Build Plan and Results

This is the prototype build plan, generated before we started construction of the prototype and updated to capture the lessons learned and action items from the process of construction.

6.1.1 Drivetrain Level Construction

The “drivetrain” of the prototype includes the motors (plus encoders), the motor controller, the wheels, and the battery pack, all contained in an IP67-compliant “project box.” These are the steps to assembling the drivetrain.

1. **Modifying the project box:** Remove plastic standoffs from bottom surface of interior of the box using rotary tool to make space for components.
2. **Mount motors:** Drill holes in either side of project box, as far forward as possible, to act as motor mounts. To do this, use a 3D printed hole template (shown in figure <PRINT_FIXTURES>a) to ensure uniformity. Drill holes at size that allows for clearance of M4 screws.
3. **Mount wheels to motors:** mount wheels to motors using a 3D printed shaft coupler, shown in figure <PRINT_FIXTURES>b. Use an M4 screw to securely attach wheels to shaft coupler. Use set screw on shaft coupler to securely attach shaft coupler to wheel.
4. **Mount wheel encoders**: using the interior output shaft, mount the through-hole capacitors to the motors. Secure them in place using either a 3D printed mount or with adhesive.
5. **Secure motor controller:** place motor controller underneath the motors. Wire the motors positive and negative leads to the motors M1A/M1B and M2A/M2B outputs. Be sure to wire one of the motors in opposite order of the other, so that they rotate in the same direction. Secure the motor controller in place with adhesive or use standoffs. Securely attach the control wires for the motor controller using a custom JST to Molex cable and adhesive.

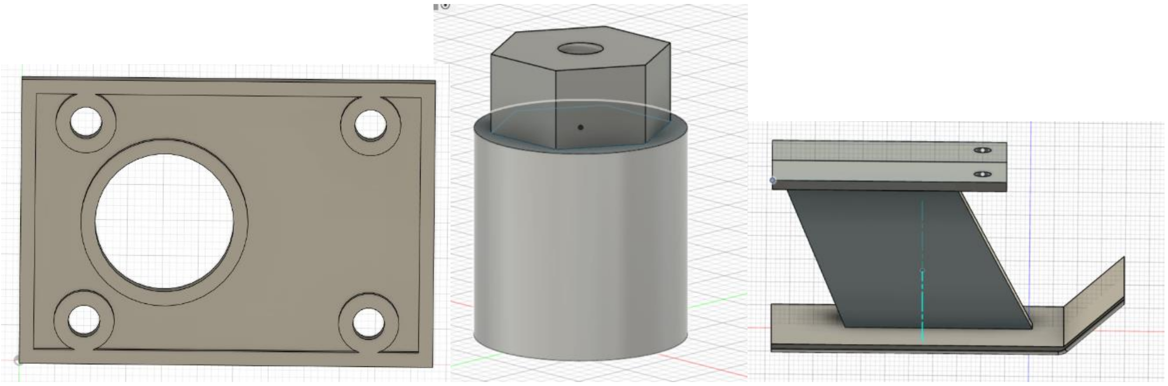


Figure <PRINT_FIXTURES>: a) hole drilling template for motors, b) shaft coupler for mounting wheels to motors, c) “skis” for bottom of rover.

6. **Place battery:** there should be significant space behind the motors and motor controller for the battery to sit. Affix it in place with either a 3D printed fixture (more secure), or hook-and-loop fasteners (more convenient).
7. **Mount skis:** mount the ski "feet," with design shown in figure <PRINT_FIXTURES>c to the bottom of the rover, with top plate of the feet roughly equal to the back of the container, using the M3 clearance screw holes or adhesive.

The final result of assembling the drivetrain is shown in figure <PROTOTYPE_DRIVETRAIN>.

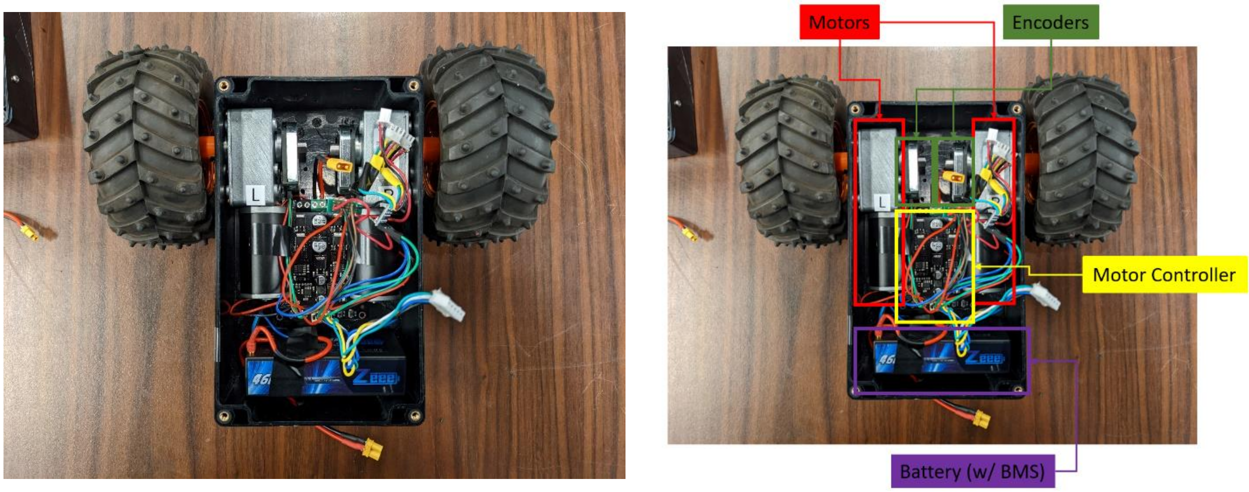


Figure <PROTOTYPE_DRIVETRAIN>: a) a picture of the drivetrain of our prototype; b) the same picture, annotated

6.1.2 Sensors and SBC Level Construction

Atop the drivetrain sits a second, smaller project box with the same footprint. It contains our SBC, stereo camera, and GPS. The steps to constructing the "upper level" follow.

1. Cut cable passthrough: in the rear of the box, cut a cable passthrough, i.e., a large rectangular hole large enough to pass the control wires of the motor controller, the battery power output wires, and the encoder wiring harness.
2. Drill eyeholes: in the front of the box, drill holes for the two lenses of the stereo camera to pass through. Use the 3D printed faceplate, shown in figure <ROVER_FACE>, as a guide, but drill the holes slightly larger than those in the faceplate.
3. Mount the SBC: drill holes to mount the SBC just rear of the drilled holes for the camera, leaving enough clearance for the camera between them. Drill for M2 clearance. Using the holes provided in the SBC PCB, mount the SBC using standoffs.
4. Place camera: place stereo camera (attached to SBC) in the holes drilled for that purpose, through the faceplate. Attach the faceplate to the rover using adhesive.
5. Mount GPS: drill a single M3 clearance hole in the side of the upper level. Use that M3 clearance hole and an M3 screw to mount the GPS module.

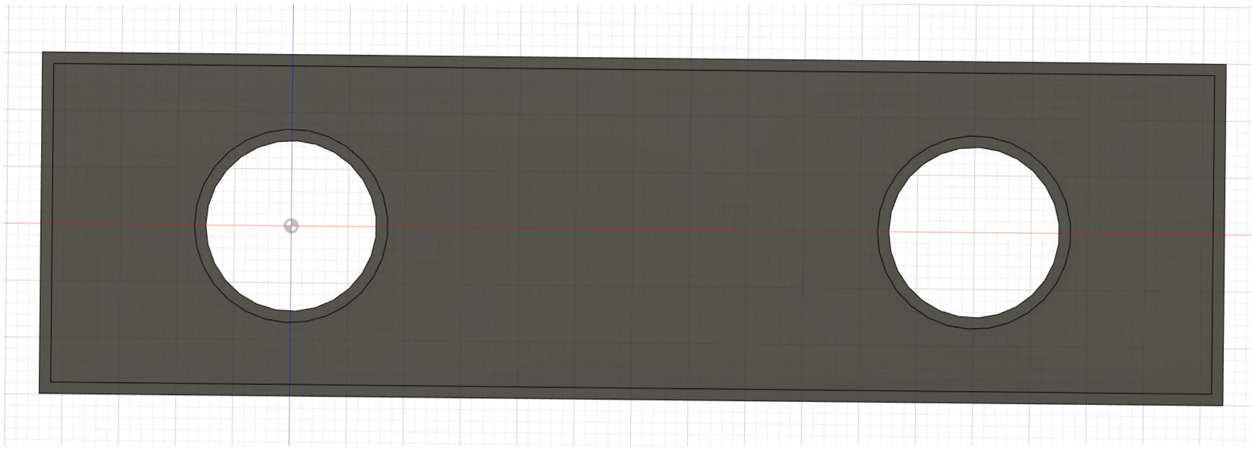


Figure <ROVER_FACE>: the design for a 3D printed "faceplate" that hold the camera lenses in place.

6. Mount 12V boost converter:
   WF: in our final prototype, our 12VDC power domain on the PCB did not work as intended. As such, we perf-boarded a commercial-off-the-shelf boost converter, attaching connectors for the GHz transmitter and motor controller power inputs. Drill M2 clearance holes in rear side of project box, using the perf-boarded boost converter as a marking template. Use M2 standoffs to mount the perf board.

The results of the assembly of the "upper level" of our prototype are shown in figure <PROTOTYPE_UPPER>.

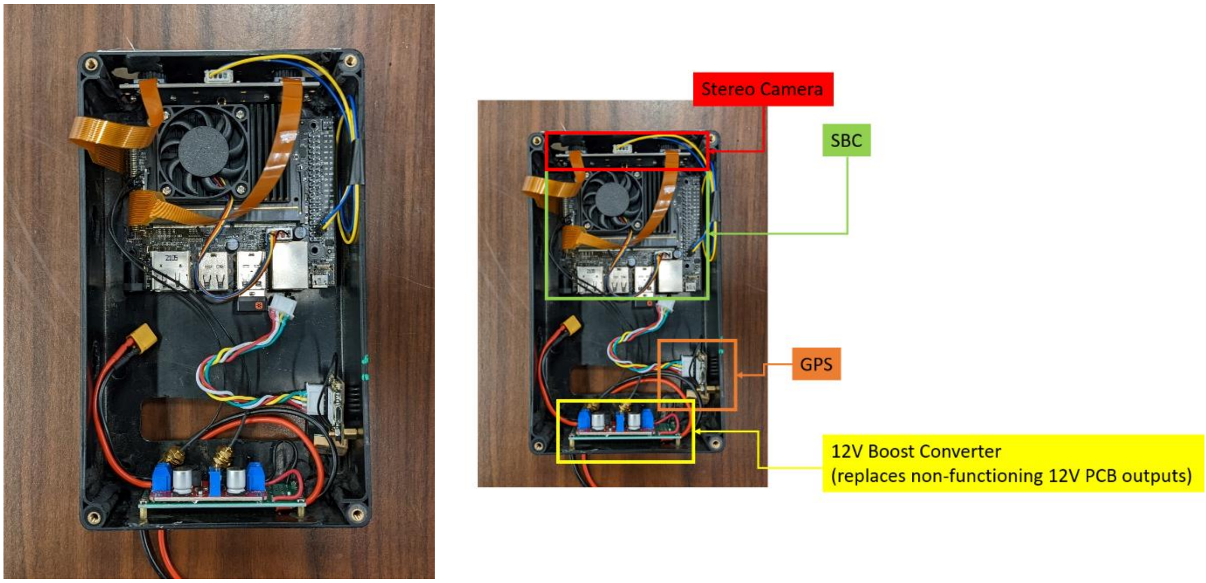


Figure <PROTOYPE_UPPER>: a) a picture of the assembled upper level of our rover prototype, b) the same picture, annotated

6.1.3 Transmitter and PCB Level (Lid) Construction

Finally, our video transmitter (plus HDMI->Analog bridge), LoRa transceiver, and PCB are mounted to the lid of the Sensors and SBC level project box.

1. Drill holes for HDMI->Analog bridge: using the bridge as a marking guide, drill M3 clearance holes to mount the bridge.
2. Drill holes for PCB: use the PCB as a guide to cut M2 clearance holes for mounting the PCB.
3. Cut passthroughs: using a rotary tool (or files), cut away material in the side of lid to allow for the GHz transmitter antenna, as well as the micro-USB connector for the PCB and an HDMI connector (these last two might need to be cut into the side of the Sensors and SBC level instead). This is a last-minute add, as we didn't have adequate space within the prototype for the PCB or HDMI cables.
4. Mount components: mount the PCB and HDMI->Analog bridge using standoffs. Since neither the LoRa or GHz transmitters provide holes for standoffs, fasten them with hook-and-loop instead.

The results of the assembly of the Transmitter and PCB level are shown in figure <ROVER_LID>.

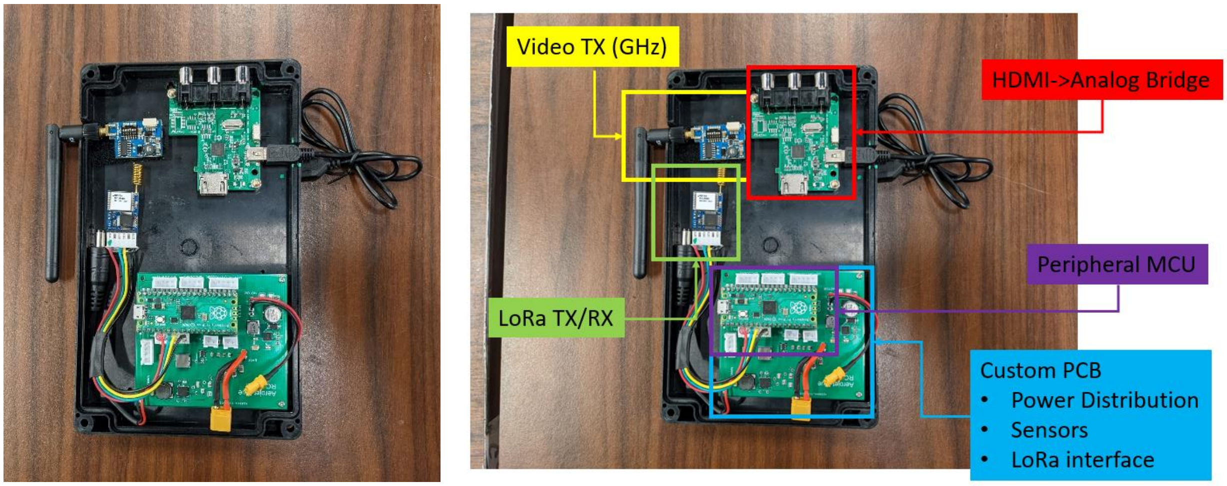


Figure <ROVER_LID>: a) a picture of the assembled lid of our rover prototype, and b) the same picture, annotated

6.1.4 PCB Assembly

This section details the assembly of the PCB, which we used to interface with our peripheral microcontroller, as well as provide power to the rover. The assembled PCB is shown in figure <PCB_ASSEMBLED>.

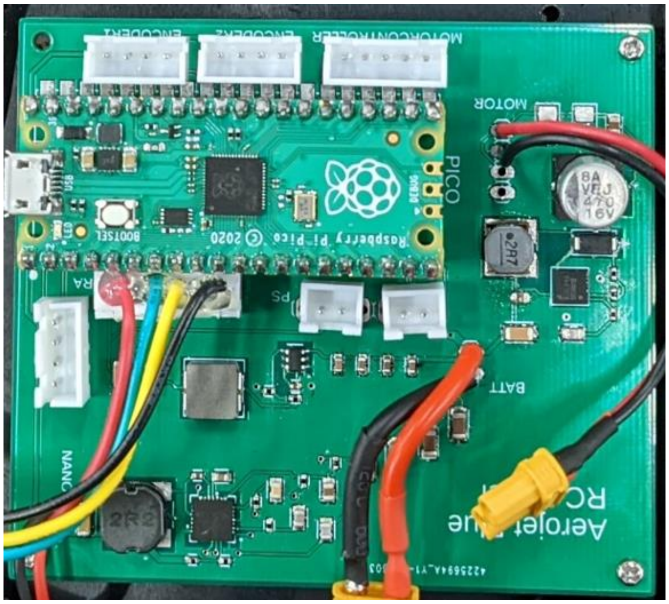


Figure <PCB_ASSEMBLED>: the assembled PCB of our rover prototype

To use our PCB as part of our prototype, we had to first attach the Raspberry Pi Pico into it’s provided through-hole pads. To do this, we first put male pin headers on the Pico itself, soldering them into place. Then, we soldered female pin headers into the PCB to match. When done, we could simply snap-fit the Pico to the PCB, as shown in figure <PCB_ASSEMBLED> in the upper-left.

To attach our peripherals and power connection to the PCB, we had to populate the pins with appropriate connectors. To do this, we used JST-XH (2.54mm) male pin headers for data and low voltage connections, and XT-30U power connectors for power connections.

The JST-XH male headers were attached at each of the through-hole pads we included for interfacing with the Pico. These connectors can be seen surrounding the Pico in figure <PCB_ASSEMBLED>. Using these connectors allowed us to make our own cable connectors with JST-XH female connectors on them, providing not only the flexibility of cutting to our preferred lengths, but also the convenience of swapping out damaged cables. Further, JST-XH connectors are locking, which gave our connections added security and strain relief, to stop them disconnecting or breaking.

The XT-30 connectors were attached by directly soldering their wires to the power output pads, as shown in figure <PCB_ASSEMBLED>. We took care to use different connector ends for the battery input and the 12VDC output to make it impossible to accidentally reverse bias our PCB and potentially damage its components.

7.0 Operator's Manual

This section is intended to serve as a simple operator's manual, briefly detailing how our prototype is meant to be interfaced with, and what behaviors to expect. This covers only the user-facing operations and isn't intended to be used for detailed troubleshooting or system design questions.

7.1 Booting the Rover

The rover is designed to remain on during launch/descent/deployment. As such, there's no "power switch," to power the rover, simply connect the battery to the battery input on the PCB, using the attached XT30 connectors. This should provide power to all of the components of the rover, including the SBC, peripheral MCU, and communications modules.

After a brief delay while the SBC boots, the rover should automatically execute its launch file, spinning up all the software nodes it needs, and initializing the FSM into its BOOT state. At this point, the operator shouldn't need to handle the rover directly anymore; everything can be handled from the ground station.

7.2 Connecting the Ground Station

Now that the rover is powered up, it should automatically power the peripheral MCU on its PCB, and LoRa communication should be available (even before the SBC is fully booted). Now, we need to connect the ground station to the rover.

First, ensure that the second LoRa module is connected to a suitable computer via a serial to TTY adapter. Assuming the computer is running Ubuntu Linux, it should appear as a "file" within /dev/ named tty*.

If it hasn't been done already, create a local clone of the ground station repository from GitHub on the ground station computer. The repository is available at https://github.com/UCFSDBlueRover/sd_ground_station. Install the ground station dependencies by navigating to its root directory and running the following command:

```
$sudo pip install -r requirements.txt
```

Once a local version of the ground station is set up, it can be executed via the following command:

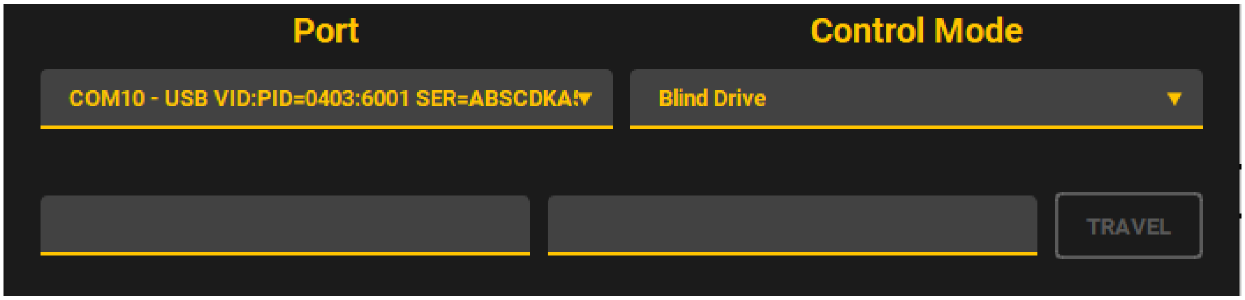
```
$python3 gui.py
```

At this point, the ground station software as detailed in section 5.5 should be available to the user. After selecting the correct LoRa transceiver on the home tab, the ground station should begin attempting to establish a connection with the LoRa module on the rover. Once that connection is established, it should be possible to command the rover by selecting a control mode on the Controls/Map tab.

7.3 Controlling the Rover

It should now be possible to control the rover from the ground station computer using the ground station software. There are three methods for commanding the rover:

BLIND DRIVE

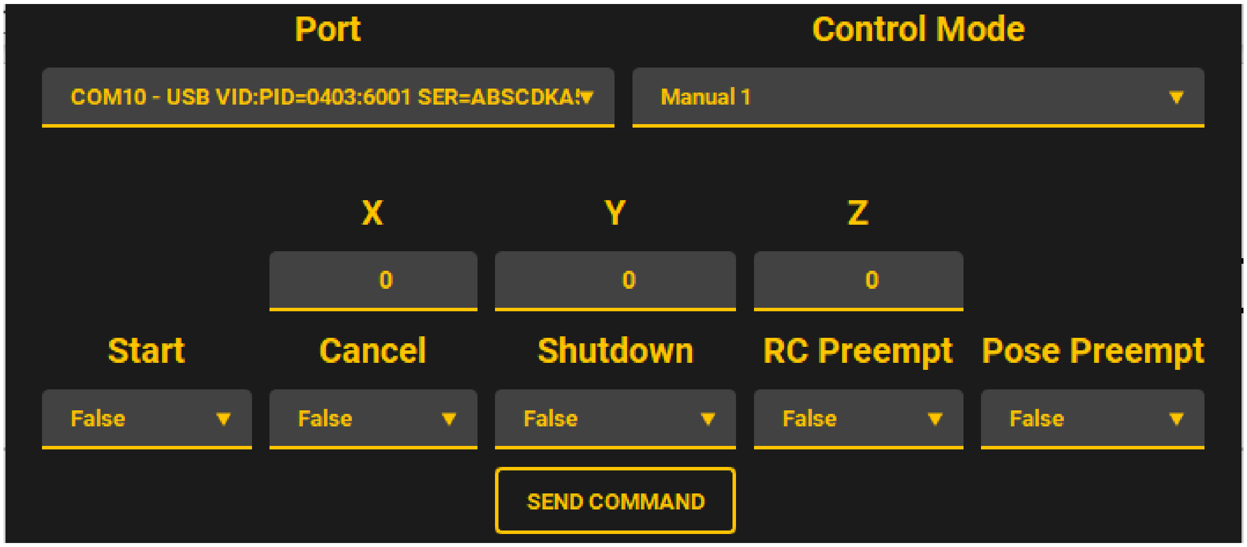


Port: COM10 - USB VID:PID=0403:6001 SER=ABSCDKA!

Control Mode: Blind Drive

Blind Drive can be selected using the Control Mode dropdown. To use this mode, first the user must enter the desired latitude and longitude coordinates. Alternatively, clicking the desired location on the map will automatically populate the coordinate fields with that location. The “TRAVEL” button is then used to communicate these coordinates to the rover.

MANUAL 1



Port: COM10 - USB VID:PID=0403:6001 SER=ABSCDKA!

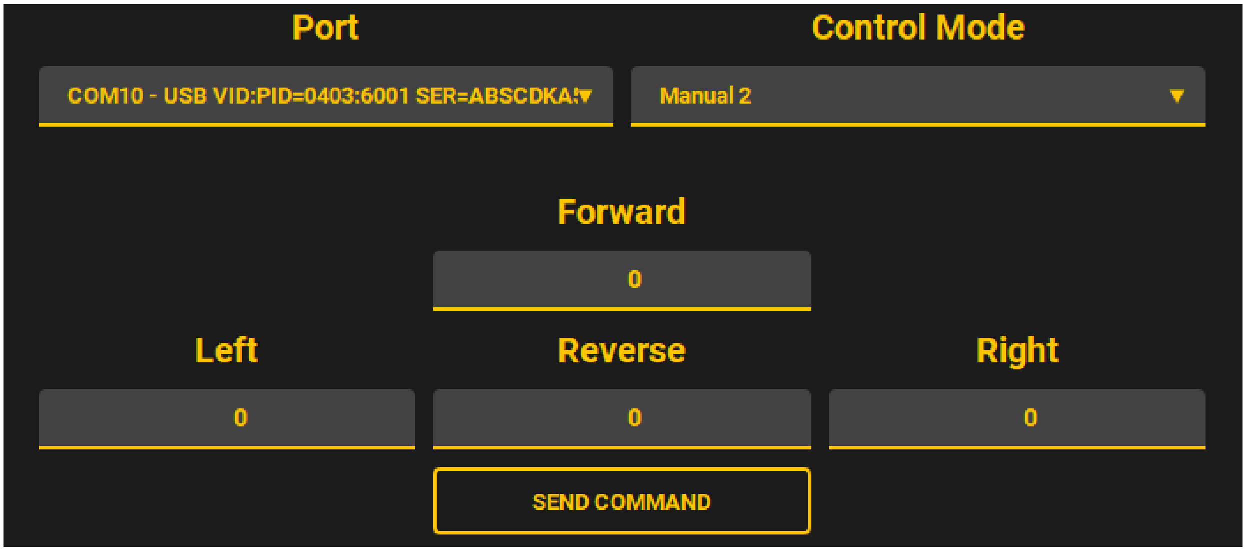
Control Mode: Manual 1

X	Y	Z
0	0	0

Start	Cancel	Shutdown	RC Preempt	Pose Preempt
False	False	False	False	False

Manual 1 can be selected using the Control Mode dropdown. X, Y, and Z coordinates relative to the rover’s starting position can be supplied to control the rover. Additionally, there are five other fields that can be used to affect the rover’s state. If true, the “Start” and “Shutdown” fields will initiate and deactivate the rover, respectively. The “Cancel” field tells the rover to return to STANDBY. The “RC Preempt” and “Pose Preempt” allow us to force the rover to transition between the MANUAL and WAYPOINT states. The “SEND COMMAND” button is used to issue the command to the rover.

MANUAL 2



Port: COM10 - USB VID:PID=0403:6001 SER=ABSCDKA!

Control Mode: Manual 2

Forward: 0

Left: 0

Reverse: 0

Right: 0

Manual 2 can be selected using the Control Mode dropdown. The operator can specify the number of "movement units" to move the rover in the forward, left, right, and reverse directions. The directions are relative to the orientation of the rover. The "SEND COMMAND" button is used to issue the command to the rover.

8.0 Project Prototype Testing Plan

8.1 Hardware Test Environment

The hardware testing will follow two stages: initial testing, where the basic functionality of the various canister/rover subsystems will be verified, and integrated testing, combining the canister/rover subsystems and testing them alongside the software in hardware-in-the-loop testing.

Initial testing will be done using resources provided at the UCF robotics lab. Modules will be connected to breadboards, powered by benchtop power supplies, and basic functionality will be confirmed to ensure that no components are defective. Care must be taken to prevent damage to the components during the initial testing phase. Benchtop power supplies can safely supply the required power to the circuit in the testing phase.

Preliminary measurements can be taken during the initial testing phase to determine the expected power use during normal operation. The crude initial power estimations will be refined as the typical behavior of components like the motor driver and A/V transmitter are determined. This will give a better estimate of average power consumption for these devices that will be more useful than the numbers given on datasheets.

Initial testing can also be used to determine the range of components like the A/V transmitter and LoRa transceivers. A basic setup can make use of USB to serial TTL adapters to connect to the LoRa modules and test the effective communication range of the LoRa modules using two laptops. Various operating conditions can be tested, and the parameters can be adjusted using AT commands on the LoRa modules to determine the optimal balance between range and data rate.

After initial testing is completed, integrated testing can begin, as the various submodules are connected. Starting with jumper wires and breadboards, communication between the Jetson Nano and its peripherals can be established, and the hardware/software interfaces can be tested. For the payload canister, the Arduino Nano and its accompanying modules will be tested.

The hardware testing environment will be focused on a “from the ground up” approach: the initial testing will establish that individual canister/rover subsystems function as expected, and the integrated testing will establish that canister/rover subsystems operate together as expected. A thorough, methodical approach will make it easier to troubleshoot problems as they arise and have a reliable final design.

8.2 Hardware Testing

Initial testing will be done on a component-by-component basis, to ensure the functionality of the canister/rover subsystems. After initial testing is complete, integrated testing will begin.

LoRa transceiver functionality verification:

The two RYLR896 LoRa modules will be connected to separate laptops using serial USB adapters. Using a serial monitor, the modules should first be configured for 915 MHz using the “AT + BAND = 915000000” command. This is very important to avoid transmission on an unauthorized frequency.

After addresses are assigned, both modules are given the same network ID so that they can communicate. Using the default spreading factor, bandwidth, and coding rate, a series of test messages can be sent. Initially, both laptops should be placed very close together. A simple handshake “hello” message will be sent from transceiver 1 to transceiver 2, and then from transceiver 2 to transceiver 1 to ensure that wireless communication is successful. After connection is established, transceiver 1 will send a series of dummy messages to transceiver 2. These dummy messages will be strings of 9 alphanumeric characters, to simulate GPS coordinates with six decimal points of precision, which will be the largest pieces of data sent from the rover to the ground station. Transmission of these dummy messages will be tested at different ranges. After communication is confirmed, transmission will be tested at 1 meter, 10 meters, 50 meters, 100 meters, and then 600 meters. Since the rover will be operating in a desert environment, it will not be necessary to test the range in an environment with heavy obstructions. If the signal fails to reach the 600-meter mark, it may be necessary to adjust the LoRa parameters, sacrificing a faster data rate for an increased range.

Pass: the below criteria are met:

- Data transmission is successful both ways between the transceivers
- No failures in data transmission occur during testing
- Communication occurs up to a range of 600 m

Fail: any of the above criteria not met.

FPV video transmission verification:

The camera will be connected to the Hyperion TS5823 transmitter, and both will be connected to benchtop power supplies. A computer with a 5.8 GHz USB receiver, running the ground station GUI software will establish a connection with the transmitter. Once the video signal is successfully received, several tests will be done to determine the range. The ground station computer will be gradually moved away from the transmitter to determine a maximum range for the video signal. The video transmission range will be lower than the LoRa communication range and will act as an upper limit on the operating range of the rover.

Pass: the below criteria are met:

- A video signal is received at the ground station
- The video transmission can achieve a range of at least 600 m without major obstructions
- Video transmission is continuous, and no significant interruptions occur during a 10-minute interval

Fail: any of the above criteria not met.

Rover Motors/Motor Driver testing and verification:

This will be among the most critical tests done to determine the power requirements of the rover. The motors, the motor encoders, and the motor driver will be connected using alligator clips. They will be tested in different conditions to determine a good estimate of average power and current draw, and the overall strength of the motors. Using the benchtop power supply, the current draw will be measured under various conditions.

The first test will be an “idle” test, determining the current draw of the motors and driver when they are idle and not moving. The voltage will be set to 12 V, the given voltage on the data sheet for the motor driver. The motors will be left in an idle state, and the idle current draw will be measured and recorded.

The second and third tests will involve testing both motors individually. If one of the motors or motor encoders is faulty, the fault can be identified at this stage and a replacement can be used. Once it is verified that both motors are functional, the current draw for each of the motors can be measured (the measurements should be nearly identical- if they aren’t, there’s likely a fault and more troubleshooting may need to be done). The motor functionality was confirmed, and current was measured at 0.26 amperes with no load.

The fourth test is to determine “stall” current- how much current the motor will draw when it is deliberately locked in place and prevented from rotating. During normal operation, the motor should not be subject to stall conditions, but stall current measurement is a useful way to determine the maximum current draw. The measured stall current is 0.73 ampere peak, and 0.56 ampere sustained. This gives an estimated power draw of roughly 7 Watts for sustained use.

The fifth test will have both motors running simultaneously. The current will be measured at this stage. After an initial current reading without any load attached to the motors, a “dummy load” will be tied to the motors, weighing roughly 4 kg, to exceed the expected total weight of the rover. This will ensure that the motors will be hardy enough to pull the rover, even in rough conditions. This gives a rough estimate of the “strength” of the motors, and their ability to keep the rover moving in rough terrain like the Mojave Desert.

Pass: the below criteria are met:

- Motors run consistently at 12 V
- Motors do not burn out when subjected to stall conditions
- Motors can apply enough torque to move a weight of 4 kg

Fail: any of the above criteria not met.

Rover Battery Test:

After the batteries are wired in a series-parallel combination with the battery management system, it will be necessary to test the functionality of the battery pack. Safety and reliability of the battery pack is critical to building a functional rover.

The initial test will involve charging the battery pack through a DC benchtop power supply. Following the test procedures for LiPo cells, the batteries will be charged up to 7.4 V at a constant current of 0.5 x 2 = 1 ampere, until they reach 7.4 V. After the battery back reaches 7.4 V, it will be discharged through a “dummy load” comprised of power resistors at a current of 2 x 0.2 = 0.4 ampere until it reaches the minimum discharge voltage of 6.6 V. This can be achieved by connecting two 10 ohm 50 W power resistors in series and connecting them to the battery pack.

Pass: the below criteria are met:

- Batteries charge up to charge voltage of 7.4 V successfully
- Batteries discharge to the minimum discharge voltage of 6.6 V successfully

Fail: any of the above criteria not met.

GPS Module Power-On Test:

To simplify later debugging efforts, we must verify that the GPS module works as intended without connecting it to the Jetson Nano or any other computer. We simply provide it power and check the data coming across the TXD (transmit) pin using a digital logic analyzer. If the GPS works as intended, we will receive valid hex data at a baud rate of 9600. This test has limited utility: while the logic analyzer can tell us if the GPS is sending data on the correct pins, it cannot parse the data to see if the module is accurate. All this test does is remove “the module doesn’t send data” from the list of possible issues.

Pass: the below criteria are met:

- Provided 5V power on VCC, the GPS powers on and transmits data on TXD.
- Data transmitted on TXD can be resolved to valid hex data at baud rate of 9600 (i.e., no framing errors)

Fail: any of the above criteria not met.

Artifact(s): N/A

Requirements Trace: N/A

GPS Module Data Test:

For this test, connect the GPS module to a computer using the built-in USB-B connector on the module. Using the u-center software package provided by u-blox, attempt to get data from the GPS. This removes "the GPS doesn't provide accurate data" from the problem space for later debugging.

Pass: all the below criteria are met:

- A reasonably accurate GPS fix can be demonstrated from within the u-center software.

Fail: any of the below:

- Failure to access the GPS data.
- Failure of GPS to get accurate position fix (if indoors, move outdoors)

Artifact(s): N/A

Requirements Trace: N/A

Rover Motor Controller and Motor Test:

For this test, we want to ensure that our selected motor controller, the Cytron Dual Channel 10A driver, works as expected with our selected motors. First, we connect the B+ and B- terminals of the driver to a power source capable of driving the inductive load of a DC motor, likely a battery. If voltage of the battery is higher than 12VDC, a step-down (buck) converter will guarantee 12VDC. Then, we connect our selected motors to the motor driver. Using the push buttons on the motor driver, we will drive both motors in both directions, measuring the current draw and voltage drop as we do.

Pass: all the below criteria are met:

- At a continuous voltage of 12VDC, the motor driver powers on as evidenced by the green PWR LED.
- When their respective push buttons are pushed, both motors can run in the forwards and backwards directions.

Fail: any of the above criteria not met.

Artifact(s): record of current draw for single motor and both motors running in both directions.

Requirements Trace: N/A

Canister Barometric Pressure Sensor Test:

To verify the Freescale MPL3115A2 barometric pressure sensor is working as intended, we will perform tests to determine its accuracy and data rate. In order to do this, we test various altitudes and plot the reported readings from the pressure sensor.

Pass: all the below criteria are met:

- The readings shown fluctuate at a maximum of 1 Hz.
- The difference between successive readings should not be more than ±0.3 meters.

Fail: any of the above criteria not met.

Artifact(s): record readings from pressure sensor.

Requirements Trace: N/A

Canister Motor Driver Test:

For this test, we will see if our two Allegro A4988 motor drivers function smoothly with our 9.6V power supply. To perform this test, we will setup the Arduino Nano on our breadboard and connect the motor drivers and benchtop variable power supply. After establishing a 9.6VDC connection and attaching the stepper motors, we will measure the current draw and voltage drop due to the drivers.

Pass: all the below criteria are met:

- The motor drivers are thermally stable when running the motors at the max rated current.
- The motor drivers can run the motors in both clockwise and counter-clockwise orientations.

Fail: any of the above criteria not met.

Artifact(s): record of current draw for both motors running in both directions.

Requirements Trace: N/A

Canister Motor Torque Test:

For this test, we will need to determine the required current and torque necessary to remove the rover from the canister. Although we had a rough calculation, we will try to find an optimal current to run the NEMA 11 motors at the required torque.

Pass: all the below criteria are met:

- The motors are able to eject a dummy load of 4.31 kg, accounting for the rover and canister.

Fail: any of the above criteria not met.

Artifact(s): record of the current input to the motors and resulting torque that performs best.

Requirements Trace: N/A

Canister Encoder Test:

In this test, we will need to analyze the function of our CUI Devices AMT102-V encoders. We will need to them to provide us with the accurate rotational displacement of the motors in order to determine if the rover is released. We will test this by spinning the motors and checking if the reported number of revolutions matches the actual experimental values.

Pass: all the below criteria are met:

- The encoders' reported values have less than 5% error when compared to the real value.

Fail: any of the above criteria not met.

Artifact(s): record of the most optimal PPR resolution to use.

Requirements Trace: N/A

Canister Battery Test:

This test involves the three LiFePo4 cells inside the canister. Like the rover batteries, these batteries will be charged with the benchtop power supply. We will test to see how long these batteries will last running the Arduino Nano at max capacity. We will also test the Nano under a realistic load with power-saving features enabled and motors at optimal speeds.

Pass: the below criteria are met:

- Batteries last more than an hour under max load.
- Batteries do not drop below the minimum value of 8V.

Fail: any of the above criteria not met.

Artifact(s): record of the max battery life under realistic settings.

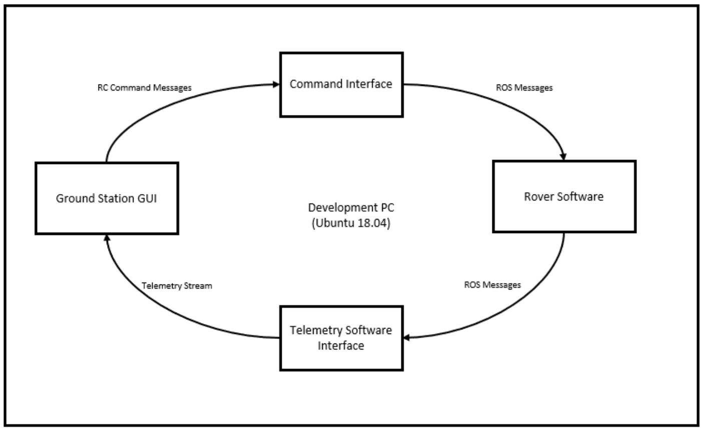
Requirements Trace: N/A

8.3 Software Test Environment

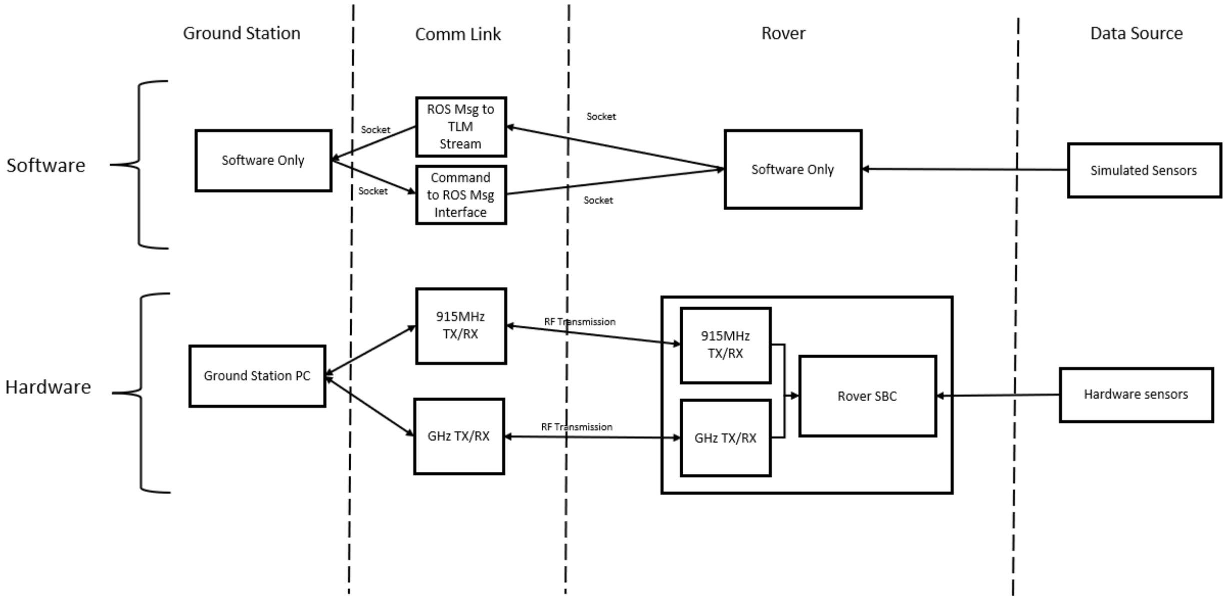
The software written for the rover and ground station will target Ubuntu 18.04 and by extension Linux4Tegra (based on Ubuntu 18.04). For the purposes of software testing, we must first verify the fidelity of a software simulation of the robot, and then use data streams generated by the simulation as inputs to our software stack. In this manner, we can test the software stack without consideration for the specifics of constructing the robot. So long as we are careful with how we manage inputs and outputs, the final software stack should perform in the real-world with close to one-to-one parity with the simulation.

The existence of a software simulation of our rover opens the door to more comprehensive hardware-in-the-loop (HWIL) and software-in-the-loop (SITL) testing. HWIL and SITL testing allows us to better isolate components of the system for testing, which will help to reduce the potential problem space for any debugging efforts brought about by these tests.

SITL testing can be done entirely on a single PC running Ubuntu 18.04. Both the ground station software and the rover simulation can be run on the same PC, using software “interface nodes” to translate messages into the desired format for communication between subsystems. In this testing configuration, message passing with sockets and ROS messages takes the place of RF communication links, but the actual rover simulation and GUI software will be unchanged. The interface nodes will convert messages from each source to the expected format of the destination, ensuring that each subsystem receives information in the simulation in much the same way as they would in the final product. This SITL configuration is depicted below in figure <SITL_DIAGRAM>.



For HWIL testing, the simulated data streams in the rover simulation can be replaced or augmented with data from the actual sensors. Further, the RF communication links can be dropped into the system to allow the ground station and the rover simulation to run on separate computers. Further, the simulation itself can be completely swapped with the actual rover computer running its software, and that software can be augmented with simulated data streams. As depicted in figure <HWIL>, any component from the software stream (e.g., Ground Station or Data Source) can be swapped with a hardware component from the same column, or vice versa. This means we can test each part of the system in isolation before we test the whole system.



Software for the payload canister's Arduino Nano will be developed and debugged using the Arduino IDE. Since the Arduino IDE uses C++, we can debug each module separately and reliably.

8.4 Software Testing

These software tests serve as an assurance that the software system aboard and interfacing with the Rover Payload can meet the requirements we have set out. Failure of any of these tests would constitute a system that doesn't meet the minimum standard set out by our requirements and must be re-designed or re-implemented until it does meet that standard.

For each of these tests, the following is provided:

- A general description of the test and what functionality it verifies
- Pass/Fail criteria
- Artifacts: generated during/immediately following the test, serve as proof that test was passed and as a resource for checking test results later
- Requirements Trace: what, if any, requirement specification that the test "proves" the system meets

8.4.1 Rover Simulation Testing

The following tests serve to verify that the simulation of the rover developed for testing purposes is an accurate representation of the physical rover in realistic surroundings. In other words, we're proving that the simulation isn't providing a false sense of security. Further, we generate a "ground truth" that any observations made in the simulation may reasonably be extrapolated to real-world phenomena. All these tests require real-time inspection of the simulation or its generated data.

Rover Model Verification:

Using the simulated rover model, verify that dimensions, masses of components, and relative orientations of links (i.e., physical components of robot) all match the most up-to-date revision of the mechanical design for the real-world rover.

Pass: all the below are verified:

- Dimensions of model and links match mechanical design
- Masses of model and links match mechanical design
- Inertias of model and links match mechanical design (inertia of real-world rover estimated from CAD software or calculated directly via inspection)
- Orientation of all links (relative to model) match mechanical design

Fail: *any* of the above listed criteria is not satisfied.

Artifact(s):

- saved copy of model file in .sdf or .urdf format.
- screenshot of rover in simulation.
- screenshot of rover axes generated by a data visualization tool such as rviz.

Requirements Trace: N/A

Simulated Sensor Stream Verification:

Using the rover simulation, verify each data stream from the simulated sensors matches expected values. E.g., verify that when the rover is moved forward 1m in the x-dimension, that odometry reflects a 1m delta in the x-dimension.

Pass: all listed sensors return data that reflects simulation reality.

- Odometry
- GPS
- IMU
- Stereo Sensor (point cloud and raw image)

Fail: any above sensor does not accurately reflect simulation reality.

Artifact(s): bag or text file of confirmed sensor streams w/ screenshots or screen recordings of rover state at time of capture.

Requirements Trace: N/A

TF Frames Verification:

Using the rover simulation configuration, verify that there exists an appropriate TF frame for each component link (sensors, wheels, actuators) of the rover model. Further verify that these frames are in the correct orientation e.g., the camera frame faces forward.

Pass: there exists a correct TF frame for each model link.

Fail: *any* TF frame is missing or incorrectly oriented.

Artifact(s):

- TF tree (as produced by view_frames utility).
- screenshot of axes from a data visualization tool such as rviz.

Requirements Trace: N/A

Motor Commands and Vehicle Speed:

Using the rover simulation, send motor commands to the simulated rover using a supported method. Ensure that the rover moves in the correct direction and at a speed that matches real-world rover speed in both forward, reverse, and turning motions.

Pass: all supported methods (listed below) of motor command input generate motor movement in the simulated rover that matches (as nearly as possible) the real-world rover performance.

- Keyboard

Fail: any supported method fails to generate motor commands, OR simulated motor movement does not match real-world speed of rover OR motor movement does not match direction provided by input.

Artifact(s): bag or text file of generated motor commands, screen recording of movement of simulated rover.

Requirements Trace: R4.1

8.4.2 Rover Simulation SITL Testing

These tests serve to verify the functionality of the rover software stack in the previously verified simulation environment. Ideally, passing of these tests implies that the software stack works as intended, and any future errors in real-world performance can be effectively isolated to the hardware. These tests require real-time inspection of the simulation and its output data.

Localization:

Using rover simulation, move the rover through a simulated world with no obstacles. Ensure that the pose estimate generated by our EKF (likely ekf_localization_node) matches the absolute ground-truth measurements provided by the simulation as nearly as possible. Ensure that movement is continuous, with no time skips or teleporting. Further, attempt to include complex movement patterns, long distances, long pauses, etc. The more erratic the driving, the stronger the verification.

Pass: after driving the simulated rover, the output of our rover localization method matches the absolute ground-truth pose (position and orientation) furnished by the simulation.

Fail: output of rover localization does not match ground-truth pose (position or orientation.

Artifact(s):

- bag or text file log of localization output.
- bag or text file log of ground-truth simulation position generated by Gazebo.
- bag or text file log of input sources to localization method e.g., odometry, world-fixed position (GPS), IMU data, etc.

Requirements Trace: R4.5

Mapping:

Using rover simulation, generate a map of immediate surroundings in a world that contains various unique obstacles at known positions. Movement of the rover (if any) should be continuous i.e., done through motor commands, no simulation pauses or teleporting. Great care should be taken to see all sides of all relevant obstacles to ensure correct shape and size. Verify that the generated map and its mapped obstacles match the simulated obstacles in rough size and location.

Pass: generated map matches the immediate surroundings of the rover in the simulation world.

Fail: *any* of the following:

- Incorrect orientation of map relative to rover i.e., map and obstacles rotated relative to real-world surroundings and obstacles.
- Map does not contain all encountered obstacles.
- Map contains extraneous non-existent obstacles (noise).
- Size, rotation, or shape of obstacles does not match encountered obstacles.

Artifact(s): screenshot of generated map, screenshot of simulation world surroundings of simulated rover.

Requirements Trace: R4.6

Autonomous Navigation to Waypoint:

Using the rover simulation, provide a waypoint pose to the simulated rover. Ensure that the autonomous navigation function of the rover (likely move_base) correctly navigates to the target pose using motor commands. It’s important to ensure *a priori* that the pose is reachable via manual driving.

Pass: verify *all* the following:

- The rover takes a reasonable path to its target i.e., it travels via the most direct path.
- The rover reaches its target pose and comes to rest within a reasonable time frame. (Note: it is possible for the rover to “oscillate” endlessly about its target pose. This is NOT passing behavior.)
- The rover signals success upon reaching target pose.

Fail: *any* of the above criteria not satisfied.

Artifact(s):

- log of provided target pose.
- screen recording of successful manual navigation to target pose.
- screen recording of successful autonomous navigation to target pose.

Requirements Trace: N/A

8.4.3 Rover Simulation HWIL Testing

The HWIL tests are intended to test specific hardware components by looping them into the simulation, isolating them to ease troubleshooting. Each of these HWIL tests assumes that at least one component of the SITL simulation will be replaced with its hardware counterpart.

8.4.4 Rover Communications Testing

RF Translation Node - ROS Message to Telemetry Message:

Ensure that the RF translation node converts ROS messages to Telemetry messages correctly. The necessary ROS message streams (captured before test) should be fed to the RF translation node in real-time, which should then convert these messages into the proper format, and package them for transmission.

Pass: *all* the following criteria are met:

- Generated Telemetry messages are in the proper format, as defined in section 5.2.2.1.
- Generated Telemetry message fields contain necessary data from ROS data streams.
- Generated Telemetry messages are packaged for transmission as defined in section 5.2.2.1.

Fail: *any* of the above criteria not satisfied.

Artifact(s):

- bag or text file log(s) of ROS message streams that were used to generate telemetry messages.
- log of generated telemetry messages.

Requirements Trace: R4.9

RF Translation Node - Command Message to ROS Message:

Ensure that the RF translation node converts incoming Command messages to ROS messages correctly. The Command message(s) (captured before test) should be fed to the RF translation node, which should then convert the Command message into whatever ROS messages are needed (as determined by the content of the message).

Pass: *all* the following criteria are met:

- command message fields are unpacked into correct ROS message types as defined in section 5.2.2.3.
- generated ROS messages are in the correct format and contain correct data (type and value) as defined by the ROS message documentation.

Fail: *any* of the above criteria not satisfied.

Artifacts:

- text file log of Command messages used as input to the RF translation node.
- text file log of generated ROS messages.

Requirements Trace: R4.2

RF Translation Node - Heartbeat Message:

Ensures that the special Heartbeat message is converted from its ROS message format to the Heartbeat message format as defined in section 5.2.2.2. The ROS message that correlates to the Heartbeat should be fed into the RF translation node, which should then output the Heartbeat message in the proper format.

Pass: all the following criteria are met:

- ROS message is converted into the Heartbeat message as defined in section 5.2.2.2.

Fail: *any* of the above criteria not met.

Artifact(s):

- bag or text file log(s) of ROS message stream(s) that were used to generate the Heartbeat message.
- text file Log of generated Heartbeat message(s).

Requirements Trace: N/A

8.4.5 Rover Controller State Machine Testing

These tests verify the proper functioning of the controller, which is implemented as a finite state machine. They are based on the principle of state coverage (each state is tested at least once) and transition coverage (each transition is tested at least once) [C42].

As a secondary benefit, these tests also serve as explicit documentation of the desired behavior of each of these states and their transitions.

Overall FSM Structure Verification:

Use an FSM introspection tool such as smach_viewer [C43] to generate a visualization of the state machine structure. Verify that this structure is functionally identical (if not necessarily visually identical) to the most up-to-date revision of the FSM design. This verifies that the state machine has been implemented as designed.

Pass: FSM visualization is functionally identical to FSM design.

Fail: FSM visualization differs from FSM design, either in states or transitions.

Artifact(s): screen capture of FSM visualization.

Requirements Trace: N/A

BOOT State Testing:

The BOOT state of the FSM is intended to run a system check before transitioning to another state.

Test the BOOT state to ensure proper functionality, as well as proper transitions to other states.

Test 1: Provide all necessary data (ROS messages, user input, etc.) for a correct boot. Ensure that the BOOT state completes successfully, and transitions to STANDBY.

Test 2: Manually break a requirement of the boot sequence (one at a time). Ensure that the BOOT state fails and transitions to WARN. Repeat the test for each requirement of the BOOT state.

Pass: for every test above, all the following criteria are met:

- BOOT state throws the specified transition
- BOOT state fails/succeeds as specified.

Fail: *any* of the above criteria fails for *any* test.

Artifact(s): N/A

Requirements Trace: N/A

STANDBY State Testing:

The STANDBY state of the FSM is the “default” state where the controller will sit when not currently being commanded.

Test the STANDBY state to ensure proper transitioning to other states.

Test 1: Trigger the RC override and ensure that the state transitions to MANUAL.

Test 2: Provide a ROS message that contains a target pose and ensure that the state transitions to WAYPOINT.

Test 3: Introduce an error (for example, kill a ROS process that the STANDBY state uses) and ensure that the state transitions to WARN.

Pass: for every test above, the specified transition is thrown.

Fail: for any test above, the incorrect (or no) transition is thrown.

Artifact(s): N/A

Requirements Trace: N/A

8.4.5 Canister Software Testing

Tests need to be done to ensure that the software aboard the Arduino Nano can reliably transfer between the four states of BOOT, STANDBY, READ, and DEPLOY.

8.5 Mechanical Testing

Enclosure – Airborne Dust/Debris Infiltration:

Our rover is designed to operate in the wind and sand of the Mojave Desert. As such, it’s important to verify that the enclosure that contains all our sensitive electronics is (relatively) impermeable to dust and debris carried by the wind. IP50 specifies that “ingress of dust is not entirely prevented, but it must not enter in sufficient quantity to interfere with the satisfactory operation of equipment” [C51]. It’s up to the tester to determine

For this test, we will make use of a leaf blower (or other means of high-intensity airflow creation) and copious amounts of fine sand. With the enclosure of the rover securely closed, use the leaf blower to blow sand at the enclosure from as many directions as possible. When done, open the case and ensure that no more than a small amount of the blown sand has infiltrated the case.

Pass: all the below criteria are met:

- no more than a (tester determined) reasonable amount of dust/debris infiltration has occurred.
- The case is undamaged.

Fail: any of the below:

- Too much dust/debris has infiltrated the rover enclosure.
- The case was catastrophically damaged during testing.

Artifacts:

- A report of how much sand entered the enclosure by weight
- Photo of interior of enclosure immediately after opening

Requirements Trace: R4.4

Sustained Heat Testing:

The rover electronics are expected to perform inside a closed container in the Mojave Desert. It's reasonable to gather data about internal case temperatures during sustained loads before-hand, to determine if a more aggressive cooling strategy is appropriate. For this test, all the rover components will be placed inside a sealed (as well as possible) vessel. It need not be the actual enclosure for the rover, so long as it's sealed well and has roughly the same internal volume. Then, all electronics will be powered on and run at load: the Jetson Nano will be performing full-performance benchmarking, the motors will run at full speed, all sensors will be active. The entire system will draw power from our selected batteries. Temperature will be tracked over time, and the tester will ensure that temps do not reach dangerous levels, where dangerous is defined as "reasonably far outside of stated temps for all components." If temps rise too high, the tester is to stop the test before serious damage is inflicted on the SBC or sensors. Using this worst-case temperature data, we can determine the need for more/better cooling, whether that be in the form of case fans, copper pipe heatsinks, etc.

Pass: the rover runs for the entirety of its stated lifetime without significant overheating and without the need for tester intervention. This implies that our current cooling solution is sufficient.

Failure: either

- Temperatures within the case reach dangerous levels, forcing the tester to intervene
- Aberrant behaviors due to overheating begin to occur, such as CPU lockups, missed sensor readings, "twitching motors"

Artifact(s):

- The complete log of temperature vs. time for this test

Requirements Trace: N/A

Canister Random Vibration Testing:

This test is done to see what frequencies would cause our canister to fail. Various frequencies will be tested using a shaker system.

Pass: the canister does not fail at frequencies that are present inside the rocket.

Failure: the canister fails at a frequency present in the canister.

Artifact(s): record of the canister's response to various frequencies.

Requirements Trace: N/A

Canister Three-Point Flexural Testing:

This test is done to see the flexural response of the aluminum disks under various loads. We will test this with a flexural testing machine.

Pass: the disks do not have significant flexural strains under reasonable tensile loads.

Failure: the disks have detrimental deformations or cracks.

Artifact(s): record of the load at which the disks begin to fail.

Requirements Trace: N/A

Canister Compressive Stress Testing:

This test is done to see the maximum axial load that the payload canister can support. We will use a universal tester that can subjugate the canister to various loads.

Pass: the canister does not deform under reasonable axial loads.

Failure: the canister does not hold up under the approximate axial load of 76 lbf.

Artifact(s): record of the load at which the canister starts to yield.

Requirements Trace: N/A

9.0 Project Summary and Conclusions

Our project was to create a mobile robot capable of being operated at significant range and significant latency. As a team, we designed a suitably capable system from the ground up using selected components. We selected all our own parts, rather than using any pre-made robots or off-the-shelf kits. We designed our own LoRa communications protocol that allowed us to communicate with the rover at great distances. Further, we designed and developed our own custom GUI to handle this communication. Finally, we designed from scratch a mobile robot: selecting our drivetrain, sensors, and compute module; designing the electrical system; programming the robot using ROS and custom software; and assembling it to our specs.

As a team, we feel satisfied in the work that we turned in despite significant challenges throughout the past two semesters, including hectic schedules, school/work/extracurricular balancing, and personal issues. Despite some early struggles with communication, we came together and managed to develop a system that we're all extremely proud of.

WF: An important note about our final product: while we designed and parted out a canister for containing the rover, we did not construct it. This was due to a last-minute change by the Arcturus team. Whereas before each payload team was responsible for their own canister design, within the last two months or so, that changed to allowing us to use a single, unified canister designed by the Arcturus team. To consolidate our efforts around the rover, we made the decision to use the pre-built canister instead of our own. As such, the requirements surrounding the payload canister and payload sled are not relevant.

9.1 Requirements

This section is intended to serve as a discussion of how our final project met the requirements laid out in section 2.3 at the beginning of Senior Design 1.

First, the mechanical requirements in requirements section R1.0:

Req.	**Explanation**	**Critical Value**	**Pass/Fail**
1.1	Final weight of payload prototype including battery, wheels, and "skis" was 2.041Kg	2.041Kg	PASS

1.2	Fitting all components was a challenge and plans for a rev. 2 with more refined construction fell through.	Diam. = 9.31in
		Length = 9.055in
			FAIL

Mechanical requirements are a mixed bag. We were well under the weight budget, and we did well at staying within the (generous) length allowance. However, we struggled with packaging all the components and connectors within the space we had, and without the time to finish a second revision of the prototype, we simply couldn't get the radius down the required value.

And now, for the functional requirements in section R4.0:

Req.	Explanation	Critical Value	Pass/Fail
4.1	Robot is capable of driving under its own power, and easily exceeds 10ft of range	Distance > 10ft	Pass
4.2	We were able to send LoRa (RC) commands from the Ground Station (GS) to the Rover using the GUI	N/A	Pass
4.3	We can transmit live video, however, due to packaging issues we were unable to transmit video while the rover travelled around. Our range was good. However, overall, we didn't meet this requirement.	Range > 600m	Fail
4.4	Rover is constructed from IP67-compliant materials but cannot be closed fully and so cannot be called truly dust-resistant.	N/A	Fail
4.5	Robot can track position relative to origin to within about 0.2m	N/A	Pass
4.6	This was a stretch goal, but our rover couldn't map surroundings due to latency issues with the stereo camera not generating a depth cloud	N/A	Fail
4.7	R4.7 was for generating panoramas, but this wasn't functionality we actively pursued this semester	N/A	--
4.8	Rover does log all relevant sensor data, storing those ROS topics in .bag files, where they can be played back	N/A	Pass
4.9	Rover does transmit real-time telemetry data to the ground station using LoRa communications	N/A	Pass

Overall, we managed to meet a great deal of the requirements we set out in semester one, with some notable difficulties surrounding mechanical design issues. It's likely that with a second revision, or the addition of a Mech. Engineering major to our team, these could have been ameliorated. In the end, our team is satisfied with our performance on the functional requirements.

9.2 Skills Gained

Below, each group member summarizes the skills that they developed over the course of this project.

Wesley Fletcher

- Electronics
  - Soldering
  - Crimping/solder connectors
  - Working with LiPo batteries

- Embedded/MCU
  - Pi Pico SDK
  - IRQs, timers, void pointers,…
- Programming:
  - Deeper work in C, C++
- Robotics
  - Sensor selection, sensor input conditioning
  - Kinematic models/odometry
  - Mobile base control
- Mechanical
  - 3D CAD w/ Fusion 360
  - 3D printing
  - Fasteners, prototype assembly

James Ellison

- RF Communications
  - LoRa Protocol
  - Integration Testing-Video Tx
- Programming
  - Python
  - PyQt GUI design

Justice Cordova

- Electronics
  - Soldering
  - PCB design
  - PCB Assembly
  - Electronics Rework
  - Electronics Troubleshooting
- RF Communications

- LoRa Protocol
- Integration Testing-Video Tx

Joshua Kissoon

- Electronics
  - Soldering
  - PCB design
- Embedded/MCU
  - Pi Pico SDK
  - Multicore
- Programming
  - Python
  - PyQt GUI design
- Mechanical
  - 3D CAD w/ SolidWorks
  - Ansys simulation

10.0 Administrative Content

This section contains content related to the “administrative” details of our project, such as milestones, scheduling, and the budget.

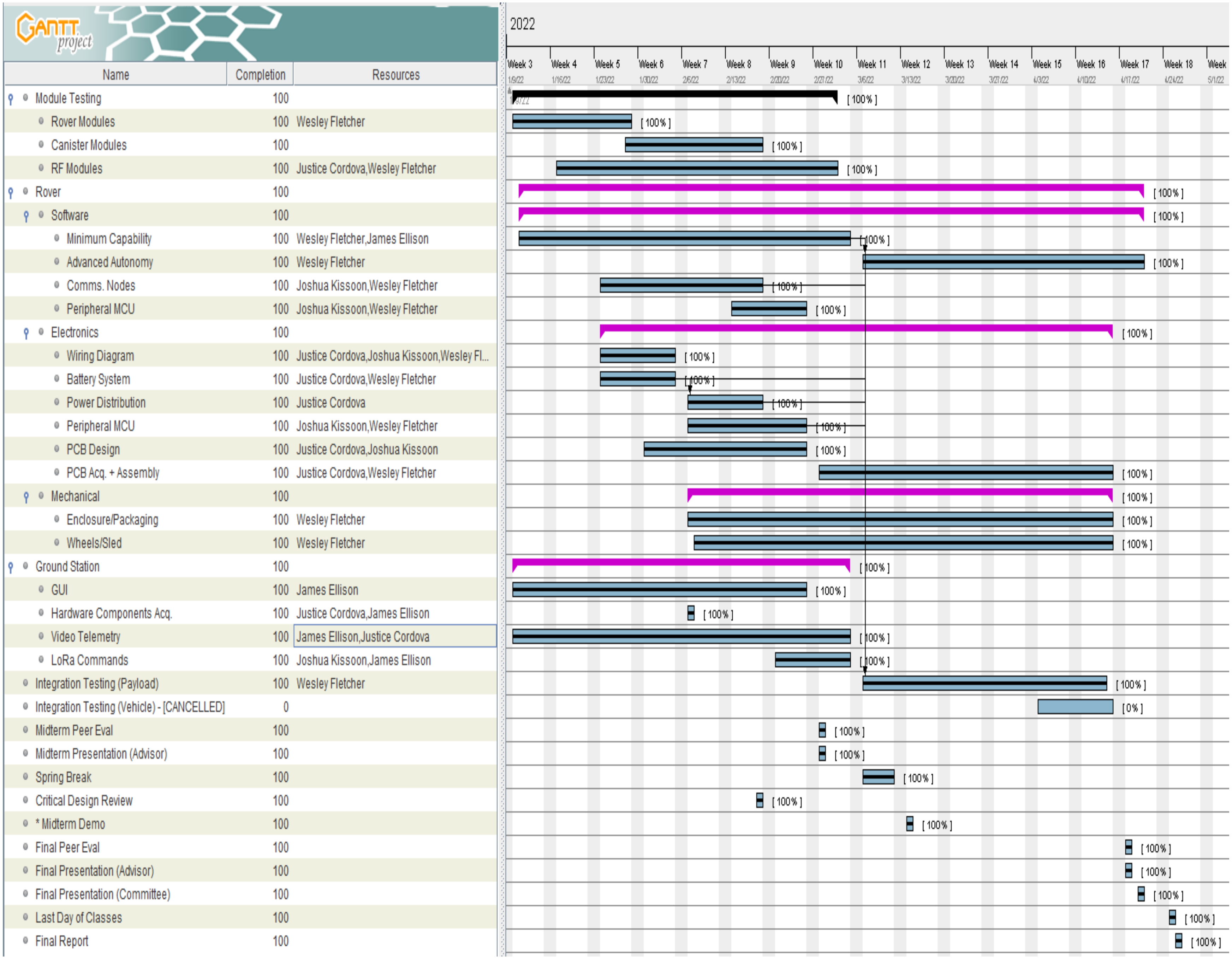
10.1 Milestone Discussion

The following table is a summary of the milestones we set out in Senior Design 1.

SENIOR DESIGN I	**Group/ Team Member**	**Start**	**Finish**	**Status**
Project Intro	Group 28	9/10/21	9/24/21	Completed
Role assignments	Group 28	9/24/21	9/30/21	Completed
Identify parts	Group 28	10/1/21	10/30/21	Completed
Project Reports	Group 28			Completed
Initial Documents	Group 28	9/10/21	9/17/21	Completed
Updated Initial Documents	Group 28	9/17/21	10/1/21	Completed
First Draft	Group 28	10/1/21	11/5/21	Completed
Final Draft	Group 28	11/5/21	11/19/21	Completed
Final Document	Group 28	11/19/21	12/7/21	Completed
Research, Documentation, and Design	Group 28	9/10/21	12/7/21	Completed

FPV Camera Module TX RX	Justice	9/10/21	11/5/21	Completed
RC Transceivers	Justice	9/17/21	11/19/21	Completed
Single Board Computer	Wesley	9/10/21	10/1/21	Completed
Microcontroller(s)	Joshua	9/10/21	11/5/21	Completed
Battery Management System	Justice	10/1/21	Jan 2022	Completed
DC/DC converters	Justice	9/17/21	12/7/21	Completed
Brushless DC motor	Wesley	9/17/21	11/19/21	Completed
Rover frame	Wesley	9/17/21	Jan 2022	Completed
Payload canister	Joshua	9/17/21	Jan 2022	Completed
Payload sled	Joshua	9/17/21	Jan 2022	Completed
PCB	Justice	12/7/21	Jan 2022	Completed
All terrain wheels	Wesley	9/17/21	11/19/21	Completed

For Senior Design 2, we fully switched to using a Gantt chart for project milestones. This Gantt chart is provided below.

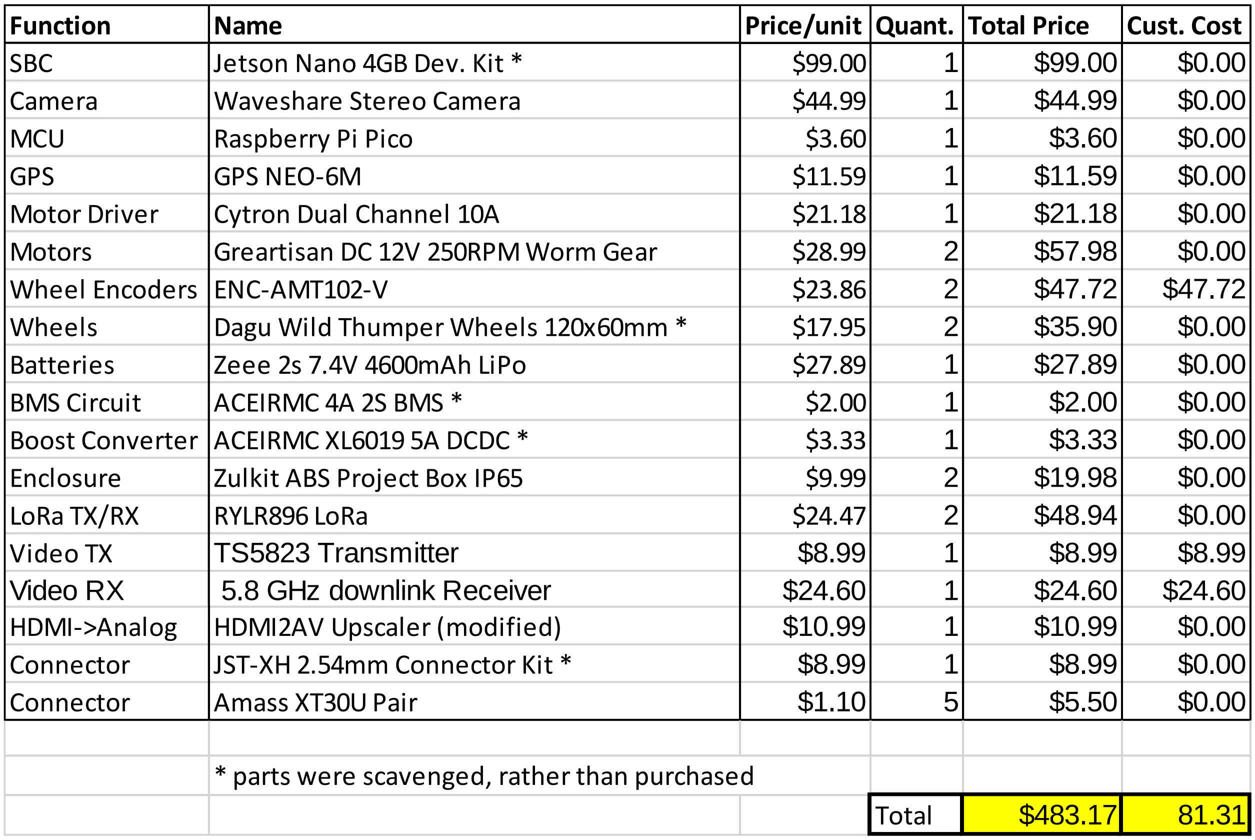


Name	Completion	Resources
Module Testing	100	
Rover Modules	100	Wesley Fletcher
Canister Modules	100	
RF Modules	100	Justice Cordova,Wesley Fletcher
Rover	100	
Software	100	
Minimum Capability	100	Wesley Fletcher,James Ellison
Advanced Autonomy	100	Wesley Fletcher
Comms. Nodes	100	Joshua Kissoon,Wesley Fletcher
Peripheral MCU	100	Joshua Kissoon,Wesley Fletcher
Electronics	100	
Wiring Diagram	100	Justice Cordova,Joshua Kissoon,Wesley Fl...
Battery System	100	Justice Cordova,Wesley Fletcher
Power Distribution	100	Justice Cordova
Peripheral MCU	100	Joshua Kissoon,Wesley Fletcher
PCB Design	100	Justice Cordova,Joshua Kissoon
PCB Acq. + Assembly	100	Justice Cordova,Wesley Fletcher
Mechanical	100	
Enclosure/Packaging	100	Wesley Fletcher
Wheels/Sled	100	Wesley Fletcher
Ground Station	100	
GUI	100	James Ellison
Hardware Components Acq.	100	Justice Cordova,James Ellison
Video Telemetry	100	James Ellison,Justice Cordova
LoRa Commands	100	Joshua Kissoon,James Ellison
Integration Testing (Payload)	100	Wesley Fletcher
Integration Testing (Vehicle) - [CANCELLED]	0	
Midterm Peer Eval	100	
Midterm Presentation (Advisor)	100	
Spring Break	100	
Critical Design Review	100	
* Midterm Demo	100	
Final Peer Eval	100	
Final Presentation (Advisor)	100	
Final Presentation (Committee)	100	
Last Day of Classes	100	
Final Report	100	

10.2 Budget and Finance Discussion

While the budget for this project hasn't been officially stated by the customer, it's been hinted at being around $500 dollars total. For a system with as many moving parts as this, we must stretch that money as far as possible to get the job done. For this reason, we focused on selecting as many parts that we already had available as possible, even if they might not be the absolute best choice for the job otherwise.

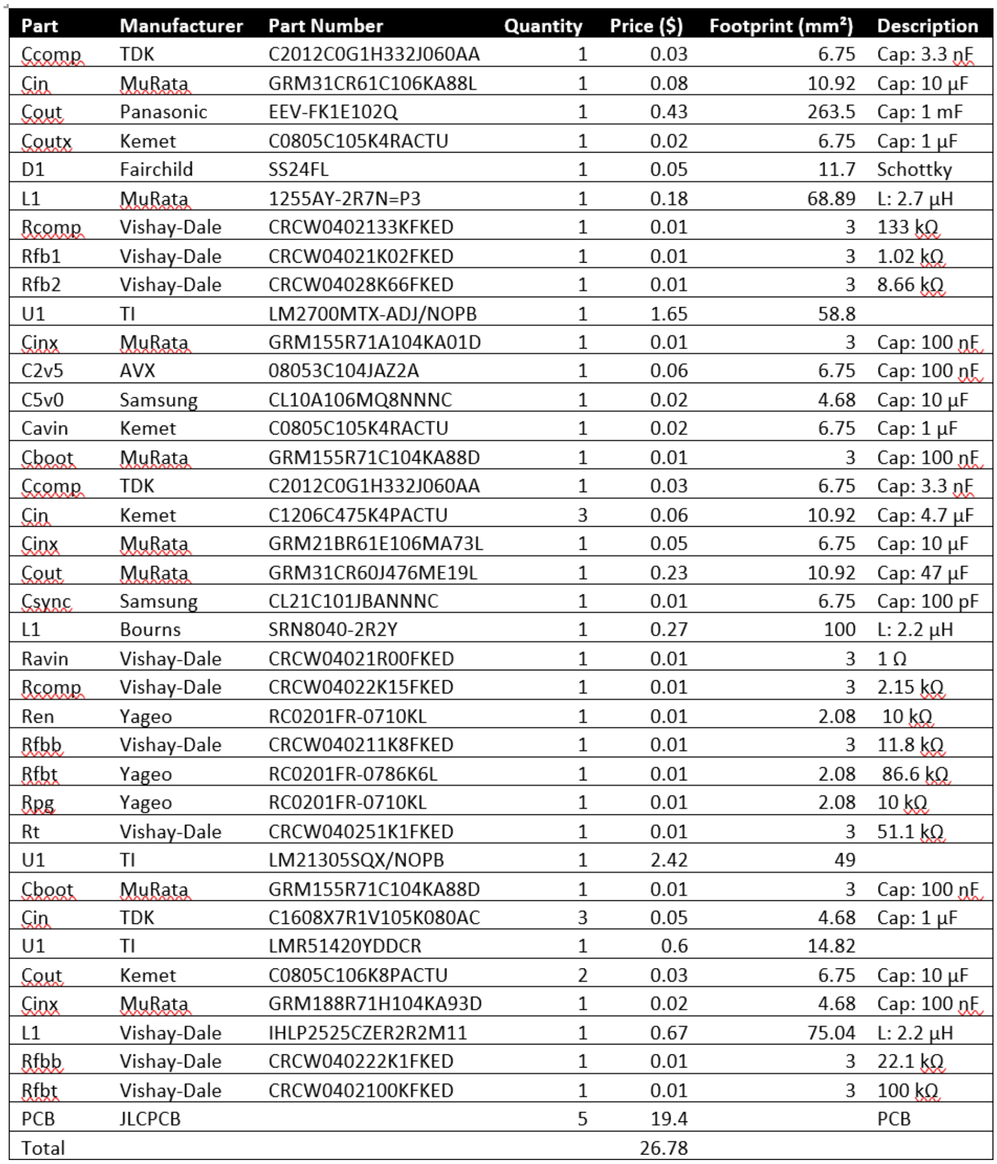
The final BOM for the rover is below:



Function	Name	Price/unit	Quant.	Total Price	Cust. Cost
SBC	Jetson Nano 4GB Dev. Kit *	$99.00	1	$99.00	$0.00
Camera	Waveshare Stereo Camera	$44.99	1	$44.99	$0.00
MCU	Raspberry Pi Pico	$3.60	1	$3.60	$0.00
GPS	GPS NEO-6M	$11.59	1	$11.59	$0.00
Motor Driver	Cytron Dual Channel 10A	$21.18	1	$21.18	$0.00
Motors	Greartisan DC 12V 250RPM Worm Gear	$28.99	2	$57.98	$0.00
Wheel Encoders	ENC-AMT102-V	$23.86	2	$47.72	$47.72
Wheels	Dagu Wild Thumper Wheels 120x60mm *	$17.95	2	$35.90	$0.00
Batteries	Zeee 2s 7.4V 4600mAh LiPo	$27.89	1	$27.89	$0.00
BMS Circuit	ACEIRMC 4A 2S BMS *	$2.00	1	$2.00	$0.00
Boost Converter	ACEIRMC XL6019 5A DCDC *	$3.33	1	$3.33	$0.00
Enclosure	Zulkit ABS Project Box IP65	$9.99	2	$19.98	$0.00
LoRa TX/RX	RYLR896 LoRa	$24.47	2	$48.94	$0.00
Video TX	TS5823 Transmitter	$8.99	1	$8.99	$8.99
Video RX	5.8 GHz downlink Receiver	$24.60	1	$24.60	$24.60
HDMI->Analog	HDMI2AV Upscaler (modified)	$10.99	1	$10.99	$0.00
Connector	JST-XH 2.54mm Connector Kit *	$8.99	1	$8.99	$0.00
Connector	Amass XT30U Pair	$1.10	5	$5.50	$0.00
	* parts were scavenged, rather than purchased				
			Total	$483.17	81.31

A more detailed discussion of the rover BOM is in section 5.4.6.

The final BOM for the PCB is below:



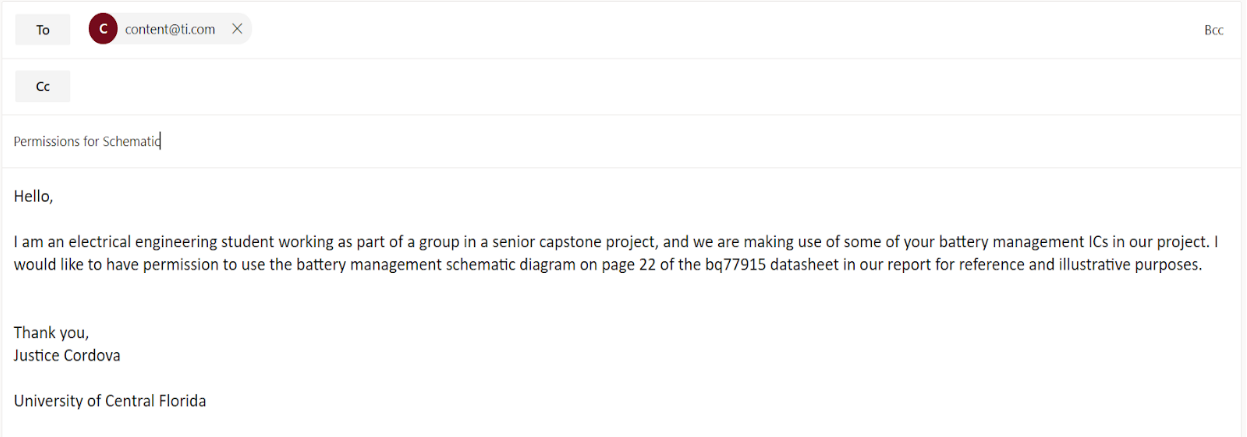
Part	Manufacturer	Part Number	Quantity	Price ($)	Footprint (mm²)	Description
Ccomp	TDK	C2012C0G1H332J060AA	1	0.03	6.75	Cap: 3.3 nF
Cin	MuRata	GRM31CR61C106KA88L	1	0.08	10.92	Cap: 10 µF
Cout	Panasonic	EEV-FK1E102Q	1	0.43	263.5	Cap: 1 mF
Coutx	Kemet	C0805C105K4RACTU	1	0.02	6.75	Cap: 1 µF
D1	Fairchild	SS24FL	1	0.05	11.7	Schottky
L1	MuRata	1255AY-2R7N=P3	1	0.18	68.89	L: 2.7 µH
Rcomp	Vishay-Dale	CRCW0402133KFKED	1	0.01	3	133 kΩ
Rfb1	Vishay-Dale	CRCW04021K02FKED	1	0.01	3	1.02 kΩ
Rfb2	Vishay-Dale	CRCW04028K66FKED	1	0.01	3	8.66 kΩ
U1	TI	LM2700MTX-ADJ/NOPB	1	1.65	58.8	
Cinx	MuRata	GRM155R71A104KA01D	1	0.01	3	Cap: 100 nF
C2v5	AVX	08053C104JAZ2A	1	0.06	6.75	Cap: 100 nF
C5v0	Samsung	CL10A106MQ8NNNC	1	0.02	4.68	Cap: 10 µF
Cavin	Kemet	C0805C105K4RACTU	1	0.02	6.75	Cap: 1 µF
Cboot	MuRata	GRM155R71C104KA88D	1	0.01	3	Cap: 100 nF
Ccomp	TDK	C2012C0G1H332J060AA	1	0.03	6.75	Cap: 3.3 nF
Cin	Kemet	C1206C475K4PACTU	3	0.06	10.92	Cap: 4.7 µF
Cinx	MuRata	GRM21BR61E106MA73L	1	0.05	6.75	Cap: 10 µF
Cout	MuRata	GRM31CR60J476ME19L	1	0.23	10.92	Cap: 47 µF
Csync	Samsung	CL21C101JBANNNC	1	0.01	6.75	Cap: 100 pF
L1	Bourns	SRN8040-2R2Y	1	0.27	100	L: 2.2 µH
Ravin	Vishay-Dale	CRCW04021R00FKED	1	0.01	3	1 Ω
Rcomp	Vishay-Dale	CRCW04022K15FKED	1	0.01	3	2.15 kΩ
Ren	Yageo	RC0201FR-0710KL	1	0.01	2.08	10 kΩ
Rfbb	Vishay-Dale	CRCW040211K8FKED	1	0.01	3	11.8 kΩ
Rfbt	Yageo	RC0201FR-0786K6L	1	0.01	2.08	86.6 kΩ
Rpg	Yageo	RC0201FR-0710KL	1	0.01	2.08	10 kΩ
Rt	Vishay-Dale	CRCW040251K1FKED	1	0.01	3	51.1 kΩ
U1	TI	LM21305SQX/NOPB	1	2.42	49	
Cboot	MuRata	GRM155R71C104KA88D	1	0.01	3	Cap: 100 nF
Cin	TDK	C1608X7R1V105K080AC	3	0.05	4.68	Cap: 1 µF
U1	TI	LMR51420YDDCR	1	0.6	14.82	
Cout	Kemet	C0805C106K8PACTU	2	0.03	6.75	Cap: 10 µF
Cinx	MuRata	GRM188R71H104KA93D	1	0.02	4.68	Cap: 100 nF
L1	Vishay-Dale	IHLP2525CZER2R2M11	1	0.67	75.04	L: 2.2 µH
Rfbb	Vishay-Dale	CRCW040222K1FKED	1	0.01	3	22.1 kΩ
Rfbt	Vishay-Dale	CRCW0402100KFKED	1	0.01	3	100 kΩ
PCB	JLCPCB		5	19.4		PCB
Total				26.78		

Estimated costs for the Payload Canister Subsystem:

Canister			
Function	**Name**	**Price**	**Acquired?**
Body Tube	6" OD x 0.125" Wall x 5.75" ID Aluminum Round Tube 6061-T6-Extruded, 12” Length	$60.01	NO
Bottom Disk	0.375" 8” x 8” Aluminum Plate 6061-T651	$23.81	NO
Other Disks	0.125" 12” x 12” Aluminum Sheet 6061-T6	$27.38	NO
Fastener	2 Carbon Steel Acme Lead Screw, Right Hand, 1/4"-16 Thread Size, 12" Length	$8.70	NO

Fastener	4 Carbon Steel Acme Hex Nut, Right Hand, 1/4"-16 Thread Size	$9.36	NO
Fastener	Grade B7 Medium-Strength Steel Threaded Rod, 1/4"-20 Thread Size, 12" Length	$4.29	NO
Fastener	High-Strength Steel Hex Nut, Grade 8, Black-Oxide, 1/4"-20 Thread Size, 100-count	$6.00	NO
Fastener	High-Strength Grade 8 Steel Hex Head Screw, Black-Oxide, 1/4"-20 Thread Size, 1" Length, 25-Count	$9.23	NO
Fastener	High-Strength Grade 8 Steel Hex Head Screw, Black-Oxide, 1/4"-20 Thread Size, 1/2" Length, 25-Count	$8.57	NO
Coupler	2 0.250" to 5mm 303 Stainless Steel Set Screw Shaft Coupler	$9.98	NO
MCU	Arduino Nano	$20.70	NO
Barometric Pressure Sensor	Freescale MPL3115A2	$9.95	NO
Motor Driver	2 Allegro A4988	$11.90	NO
Motor	2 NEMA 11 (11HS12-0674D)	$28.54	YES
Encoder	2 CUI Devices AMT102-V	$46	NO
Battery	3 LiFePO4 18650 Rechargeable Cell	$12	NO
	Total	**$272.61**	
	Total Minus Acquired	**$244.07**	

Appendix A - Copyright Permissions



Permissions for Schematic

Hello,

I am an electrical engineering student working as part of a group in a senior capstone project, and we are making use of some of your battery management ICs in our project. I would like to have permission to use the battery management schematic diagram on page 22 of the bq77915 datasheet in our report for reference and illustrative purposes.

Thank you,
Justice Cordova

University of Central Florida

Appendix B - Works Cited

1. Comparables/01_Throwable-Robots-TN_0616-508.pdf
2. Comparables/02_FirstLook-Datasheet-US.pdf
3. Comparables/03_IRIS_Brochure_1220v1(1)
4. Comparables/04_ReconRobotics_TB2-Spec-Sheet.pdf
5. Comparables/05_Vector-TRM-1
6. https://mars.nasa.gov/mars2020/spacecraft/rover/#Rover
7. https://mars.nasa.gov/mars2020/spacecraft/rover/wheels/#How-The-Wheels-Move
8. https://www.sbg-systems.com/inertial-measurement-unit-imu-sensor/
9. https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu
10. https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html
11. https://towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e#8457
12. Probabilistic Robotics, Thrun, Burgard, Fox
13. https://nerian.com/news/explanation-of-stereo-vision/
14. https://www.clearview-imaging.com/en/blog/stereo-vision-for-3d-machine-vision-applications
15. https://mars.nasa.gov/mars2020/spacecraft/rover/cameras/
16. https://www.stereolabs.com/solutions/robotics/
17. https://nssdc.gsfc.nasa.gov/planetary/chronology_mars.html
18. https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1971-045D
19. https://docs.idew.org/code-robotics/references/physical-inputs/wheel-encoders
20. https://www.cuidevices.com/blog/capacitive-magnetic-and-optical-encoders-comparing-the-technologies
21. https://www.anaheimautomation.com/manuals/forms/encoder-guide.php
22. Q. Lou, F. González and J. Kövecses, "Kinematic Modeling and State Estimation of Exploration Rovers," in *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1311-1318, April 2019, doi: 10.1109/LRA.2019.2895393.
23. http://www.matweb.com/search/DataSheet.aspx?MatGUID=b8d536e0b9b54bd7b69e4124d8f1d20a
24. http://www.matweb.com/search/DataSheet.aspx?MatGUID=dfced4f11d63459e8ef8733d1c7c1ad2
25. http://www.matweb.com/search/datasheet.aspx?matguid=61bd8c4763af44ab82793f78c89c9c77
26. Jetson Nano Developer Kit User Guide, p. 6
27. Raspberry Pi 4 Datasheet, p. 9
28. https://developer.nvidia.com/embedded/linux-tegra-r3251
29. https://www.tomshardware.com/reviews/arm-cortex-a72-architecture,4424.html
30. https://all3dp.com/2/raspberry-pi-vs-jetson-nano-differences/
31. Parts/IMU/DS-000189-ICM-20948-v1.3.pdf
32. Parts/GPS/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf
33. Parts/GPS/SAM-M8Q_DataSheet__UBX-16012619_.pdf

34. http://wiki.ros.org/Bags
35. http://wiki.ros.org/rosbag
36. http://wiki.ros.org/rviz
37. http://gazebosim.org/tutorials?tut=ros_overview
38. https://archive.thepocketlab.com/educators/lesson/how-does-pressure-sensor-work-physics-probeware
39. https://www.ros.org/blog/ecosystem/
40. https://aws.amazon.com/pub-sub-messaging/
41. Budynas, Richard G, et al. *Shigley's Mechanical Engineering Design*. New York, Ny, Mcgraw-Hill Education, 2020.
42. https://sttp.site/chapters/testing-techniques/model-based-testing.html#testing-state-machines
43. http://wiki.ros.org/smach/Tutorials/Smach%20Viewer
44. Manon Kok, Jeroen D. Hol and Thomas B. Sch ̈on (2017), ”Using Inertial Sensors for Position and Orientation Estimation”, Foundations and Trends in Signal Processing: Vol. 11: No. 1-2, pp 1-153. http://dx.doi.org/10.1561/2000000094
45. http://wiki.ros.org/ekf_localization_node
46. http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Imu.html
47. Fazekas, M.; Gáspár, P.;Németh, B. Calibration and Improvement of an Odometry Model with Dynamic Wheel and Lateral Dynamics Integration. Sensors 2021,21, 337. https://doi.org/10.3390/s21020337
48. https://github.com/sparkfun/SparkFun_Ublox_Arduino_Library
49. http://docs.ros.org/en/api/sensor_msgs/html/msg/NavSatFix.html
50. http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html
51. https://www.dsmt.com/resources/ip-rating-chart/
52. http://wiki.ros.org/navigation
53. https://forums.developer.nvidia.com/t/using-2-uart-simultaneously-in-jetson-nano/160557

Appendix C - Abbreviations

BLE	Bluetooth Low Energy
BMS	Battery Management System
CFD	Computational Fluid Dynamics
COTS	Commercial Off the Shelf
CTE	Coefficient of Thermal Expansion
EMI	Electromagnetic Interference
FAR	Friends of Amateur Rocketry
FEA	Finite Element Analysis
FPV	First Person View Camera
GPS	Global Positioning System
GUI	Graphical User Interface
HWIL	Hardware-in-the-Loop
IDE	Integrated Development Environment
I2C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
LiFePo4	Lithium Iron Phosphate
LiPo	Lithium Polymer
LoRa	Long Range
MCU	Microcontroller Unit
MEMS	Microelectromechanical Systems
PCB	Printed Circuit Board
PLA	Polylactic Acid
PPR	Pulses Per Revolution
RF	Radio-Frequency
RFI	Radio-Frequency Interference
ROS	Robot Operating System
RPM	Revolutions Per Minute
RTV	Room Temperature Vulcanizing
SBC	Single-Board Computer
SDK	Software Development Kit
SITL	Software-in-the-Loop
SPI	Serial Peripheral Interface
USB	Universal Serial Bus