HackRover

Capstone MegaDoc

Justin Heinzig, Danny Kha, Jacob Park, Ryan Miller, Bahja Adan, Long Ly
Advisor; Mentor; Sponsor: Pierre Mourad

Contents

Foreward

Abstract

The document begins with an executive summary and project requirements documentation, merely preface content not mentioned in the Table of Contents. Categorized in the content are the four defining tracts of this MegaDoc: (A) Design Strategy, (B) Experience Design, (C) Requirement Documentation, and (D) Concept Development. Additional supplemental sections cover test reports and references.

Executive Summary

HackRover is a cyber-defense-based student research project aimed at making, breaking, and hacking mobile platforms, encompassing rovers, drones, and anything that moves. Our goal is to practice and develop skills to help defend our increasingly connected world of embedded systems, where the safety of people and communities is being placed in the hands of autonomous machines.

Our plan is to pioneer a new type of hacking competition - named the HackRovoer Autonomous Robotics Competition (HARC) - where participants attempt to infiltrate self-driving rovers and collect round-winning information. There is no hackathon in the world that is centered around autonomous machines, yet according to a 2019 Global Autonomous Driving Industry Outlook report, 1 in 4 cars are expected to have autonomous capabilities by 2030. The competition we're creating isn't exclusive to UW Bothell, as tentatively schools around the globe can set up chapters. Succeeding in creating such an event would result in an excellent pedagogical tool.

Each iteration of the competition the theme changes, which influences competing rovers' designs, puzzles and props within the competition, and game rules. This iteration's theme is disaster relief; participants are tasked with designing rovers which find themselves in disasters like collapsed buildings or tornado paths. They'll build rovers for a hypothetical company which would sell to local governments and emergency services in disaster prone-regions. Rovers would provide mostly scouting and minimal medical function, either/both seeking out victims (such as under rubble) or/and dispensing minimal medical aid. Obviously the competition is yet to exist officially, so we act as a lone team following the themes set for ourselves.

This year we completed the construction of a disaster rover (first iteration), nicknamed Mimir, as well the competition puzzle panel, a prop which the rover interacts with. The rover we designed consists of three tiers stacked onto a drive base: the bottom tier houses micro-controllers and electronic power distribution; the middle tier houses the microcomputer and communication devices; and the top tier houses the LiDAR camera system and its control Stewart platform. The puzzle panel contains small electromechanical devices mounted by velcro to a velcro-clad, wooden puzzle; these devices are currently interfaced by an Arduino Mega and custom designed, PCB power distribution system (PDS). See figure 1 are the two devices completed last quarter. This quarter, we continued iterative improvements on the project by delivering a second version of the main PCB of the rover and began development of a Simultaneous Location and Mapping (SLAM) system. SLAM is a crucial first step implementing towards the Autonomous part of HARC since its completion opens up opportunities for more advanced automation.

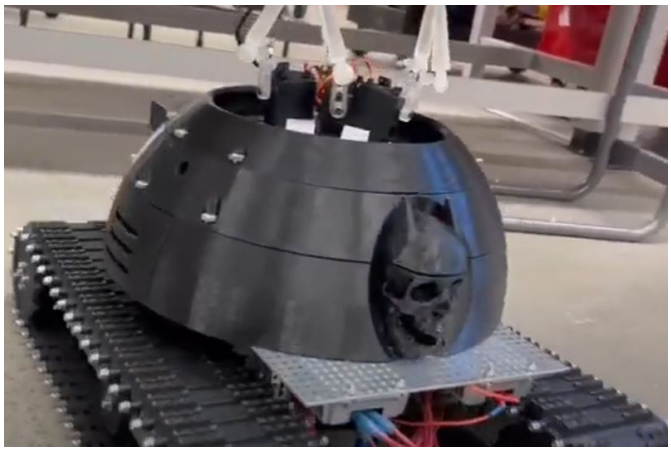

Figure 1: The rover

Project Requirement Documentation

This year we completed the construction of a disaster rover (first iteration), nicknamed Mimir. The rover we designed consists of three tiers stacked onto a drive base: the bottom tier houses micro-controllers and electronic power distribution; the middle tier houses the microcomputer and communication devices; and the top tier houses the LiDAR camera system and its control Stewart platform. See figure 1 are the two devices completed last quarter. This quarter, we continued iterative improvements on the project by delivering a second version of the main PCB of the rover and began development of a Simultaneous Location and Mapping system.

A Design Strategy

A.1 User Insights and Research

A.1.1 Consumer Journey Map

The unique nature of our project makes consumer analysis difficult, as product consumption occurs at multiple levels: faculty, students, companies, disaster responders, etc. Here we'll speak lightly of most and provide consumer journey maps for students and disaster responders: the key subjects of our capstone project.

When structuring education, university faculty seek useful pedagogical tools; such tools will provide interdisciplinary education, equip students with life skills, and possibly generate research/products that enrich the name and wealth of the school. Students, specifically engineering ones, look to develop their skills while in school through participation in clubs, projects, and research. They're often pulled in many directions as their educational appetite is wide but options are slim/focused.

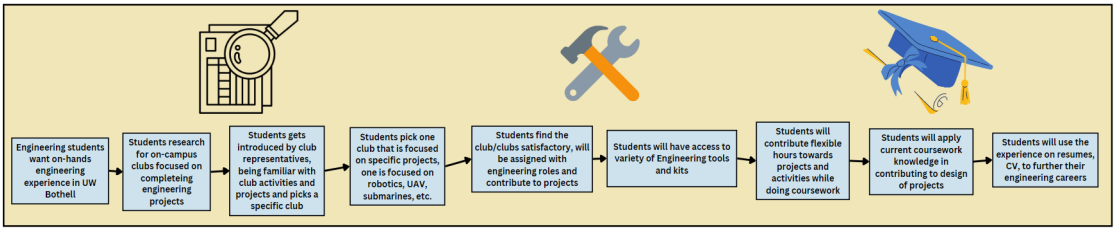

Figure 2: The Consumer Map of a Student

The student should look for extracurriculars, see options, have to pick an option, each option should have different range of benefits.

Seen in figure 2 is a student consumer map cataloging the journey a student experiences as they seek extracurricular, educational activities which empower them. A key takeaway is that the student is limited by time and must select one club with a narrow purpose. Further, these clubs demand a wealth of previous knowledge and experience to properly participate, raising the bar to entry. Broadening the scope of work for any given organization and lowering the bar to entry are obvious improvements.

Robotic companies that develop disaster rovers are limited by the opportunities available to them and niche of their field. Disaster relief is a philanthropic endeavor for the most part, where the only money to be made is from tech savvy local governments (an already fickle customer) or when disaster relief can be re-purposed as a virtuous advertisement. Minimal money exists in prize form through government sponsored competitions or events, like DARPA's disaster relief competition. Beyond monetary opportunity, research is hard to come by; like an ouroboros, the lack of monetary gain causes people to ignore robotics research in disaster relief. This makes constructing disaster relief systems difficult all around. All of this makes R&D even more difficult, as relying on robotic systems is a difficult and risky endeavor. You're never sure when a disaster will strike, and circumstances are unique each time. As a responder you must delegate quickly and make use of limited resources in saving as many lives and limiting damage as possible. Rovers are dependent on a human operator as well as some uptime; further, their operation needs to be clean and quick to permit a disaster responder to complete the same or better task without major risks.

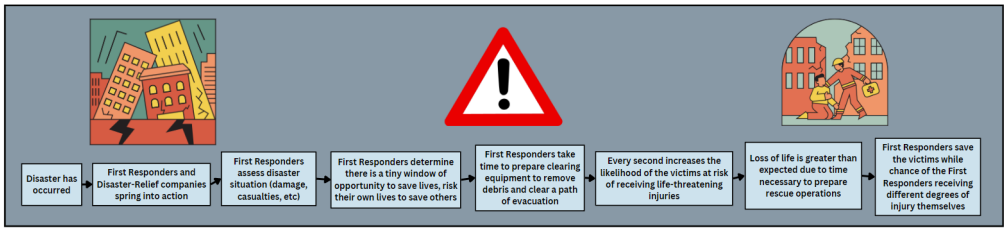

Figure 3: The Consumer Map of a Disaster Responder

The disaster responder should be presented with a disaster, have to quickly assess damages, delegate resources in a timely manner, save people, make risks, possibly emphasize the risks they might take, and

talk about the time it takes to accomplish their goal.

Seen in figure 3 is a disaster responder consumer map cataloguing their response to a generic disaster event. Of emphasis is the risk to responder's life, a time crunch, and quick decision making with limited resources and information. Systems or methods that might remove the human operator from danger, improve operation efficiency, and maximize life-saving information and resource use are obvious improvements.

A.1.2 Stakeholder Framing

- **Stakeholder:** *Government Relief Organizers:* The bottom line of these individuals is effective distribution of force and resources; inability to do this casts doubt on their ability. Any cost effective solution which improves how they distribute resources, or reduces resources needed will be a boon to them.
- **Stakeholder:** *Disaster Relief Operators:* Like the government organizers above, these individuals need to manage a swath of resources; albeit smaller, with more attention. Tools that can sub in for more expensive resources, have shorter uptimes, improve information gathering, accomplish rescue, and maximize human life (operator or victim) are beneficial.
- **Stakeholder:** *UW Bothell Students:* UW Students are the primary beneficiaries of our product. Prior we made clear the intent to create a product that serves UW Bothell students first and foremost, and whose repercussive effects are benefits to disaster response. Providing students with extracurricular engineering activities that are easy to get into while simultaneously serving broad interests will nix the problem of narrow-focus clubs with steep learning curves.
- **Stakeholder:** *UW Capstone Core Team:* Our organization already exists, but the product we create will have bearing on our success. The ability to serve a need within the school has direct correlates to further funding and participation.
- **Stakeholder:** *Pierre Mourad:* Pierre derives benefits from the product in the form of reputation of his person, outreach for access to students (for other projects potentially), and access to robotics systems. Our performance in this capstone cycle will affect trust in his decisions.
- **Stakeholder:** *UW Faculty:* Like Pierre, the UW faculty has a stake in this project. They've provided plenty of resources and space which might be used for something else. Further, a successful project might provide a powerful pedagogical tool in the future.
- **Stakeholder:** *UW Student Technology Fund:* The STF is the primary contributor to our organization, providing more than 3000 dollars in hardware technology. The exchange of this technology was meaningful use, future requests need to be paired with demonstration of this good use.

A.1.3 User Scenarios and Personas

Three Personas:

- *UW Student seeking a club* A prototypical UW engineering student seeking out extracurriculars to further their knowledge of mechanical, electrical, and/or computer engineering.
- *UW Faculty seeking new educational resource* A UW faculty member desiring to contribute to the education of their students while generating value within the school.
- *Disaster responder* A disaster responder on the ground attempting to respond to an earthquake that resulted in the collapse of multiple buildings with people inside.

Scenarios:

- A student looks at a roster of available clubs and organizations to participate in with the aim of furthering their engineering ability in multiple fields and equipping them with industry-standard skills. They're met with a limited selection containing clubs that exist but don't operate well, and others which are popular; all of these clubs have a niche focus that forces the student to sacrifice some of their interests if they invest their time. The student picks a popular club and is met with a great amount of competition and steep learning curve. Everyone wants to contribute and do

something cool, but resources are limited, so competition ensues. To stand out the student must struggle to learn a massive amount of content; by the end their skills may no longer be useful. Trying to latch onto existing projects within the club is difficult because documentation is sparse and the student can't understand decisions that were made. These circumstances aren't different for others so the project sits in gridlock: afraid to move forward in case precious resources are destroyed during the process of improvement, and a lack of confidence in how to move forward, since decisions require immense background knowledge (which isn't well documented). The student spends most of their time jumping hurdles just to do minimal meaningful engineering, get discouraged, and use the club as a resume filler.

- A faculty member has to balance their desire to improve their students lives with forwarding their own career. They could create a product which is a horrible pedagogical tool and hyper-focuses the education of their student, but which benefits their own reputation and school image. Alternatively they create product which is great for educating students, but doesn't draw any attention or produce meaningful results/research.

- A disaster responder is provided a small troop of men to command with limited expensive resources to save as many lives as possible in a short amount of time. If they spend too much time deliberating, hundreds may be crushed or suffocated under rubble; if they act too quickly manpower and resources will be spent inefficiently, hundreds may still die. Information collection and action must also be balanced. Any work they do also places their own team-members lives' at risk.

One Scenario Environment

The environment of focus is the disaster responder since it's scenario which motivates most development decisions within the project. The student is important but accessory to the disaster responder. We've repeatedly emphasized above the time and resource intensive environment responders participate in, increased consideration to human life, and limited information. Furthermore, disaster responders are beholden to government organizers who provide resources for disaster response. Despite being the users of our product, the responders must be able to convince higher ups the acquisition is sensible and cost-effective.

A.1.4 User Insight Report

UW students need a product that address their lack of interdisciplinary, extracurricular activities. Activities provided by the product must be friendly to students of any educational background and not require them to climb a steep learning curve; documentation of past activities should be plenty yet easy to read. Clerical work should be minimized, while real engineering work is maximized. Any participation should result in industry skills being learned.

UW faculty members need a product that is both pedagogically developed and useful for producing reputable work and/or research.

A given of our capstone is the development of some rover that addresses disaster responders' issues, beyond additional product components which address UW students' needs. We've made it very clear that **disaster responders** need a rover design that's cost effective, as to convince government organizers of acquisition; has a short up-time and is easy to control, to maximize efficiency; and, can provide vital information, to streamline rescue missions and better distribute resources.

A few meetings with our sponsor, Prof. Pierre Mourad, shed light on both the school environment and state of disaster relief robotics. It seems Bothell suffers from limited engineering extracurriculars, many of which are too narrow (as per our previous mention); the only solid 'rover' project is TrickFire, which sees heavy member competition. Furthermore, few pedagogical tools exist for students; many try to learn through clubs, but those often require extensive background understanding. Pierre Mourad also discussed his understanding of disaster relief rover design: companies would rather send a specifically configured rover into the situation substitute for a responder, but close by. Rovers would need to be easily configurable or provide enough data under small up-time to legitimize their use in the field. Rovers should also have some autonomous function so they don't have to be babysat by their operators.

A.2 Problem Understanding

A.2.1 Product Assumptions

- For UW Students: A product that addresses their lack of interdisciplinary, beginner-friendly extracurriculars, which improves their education, equips them with industry-standard skills, and provides an outlet for creative design.
- For UW Faculty: A product that addresses their lack of pedagogical tools & reputable student projects, providing benefit in the form of an excellent learning tool they might direct students towards while simultaneously generating projects that advance the school's reputation.
- For disaster responders: A product that addresses their time & resource constraints while responding to disasters, and concern for human life, by providing life-saving information (or even saving lives) with little up time, small cost to direct supervisors, and ease-of-use.

A.2.2 Functional Assumptions

- For UW Students: A product that addresses their lack of interdisciplinary, beginner-friendly extracurriculars, which improves their education, equips them with industry-standard skills, and provides an outlet for creative design, via that product's pedagogical methods, emphasis on industry practice, and interesting activities.
- For UW Faculty: A product that addresses their lack of pedagogical tools & reputable student projects, providing benefit in the form of an excellent learning tool they might direct students towards while simultaneously generating projects that advance the school's reputation, via that product's pedagogical methods, and activities that generate documentation and design research.
- For disaster responders: A product that addresses their time & resource constraints while responding to disasters, and concern for human life, by providing life-saving information (or even saving lives) with little up time, small cost to direct supervisors, and ease-of-use, via that product's data collection systems, simple configuration, cheap yet robust construction, and intuitive controls.

A.2.3 High Level Usability Constraints

- For UW Students: An accessible, well funded, educational product that addresses their lack of interdisciplinary, beginner-friendly extracurriculars, which improves their education, equips them with industry-standard skills, and provides an outlet for creative design, via that product's pedagogical methods, emphasis on industry practice, and interesting activities.
- For UW Faculty: A well-documented, effectively managed, economically efficient educational product that addresses their lack of pedagogical tools & reputable student projects, providing benefit in the form of an excellent learning tool they might direct students towards while simultaneously generating projects that advance the school's reputation, via that product's pedagogical methods, and activities that generate documentation and design research.
- For disaster responders: A short up-time, easy-to-use product that addresses their time & resource constraints while responding to disasters, and concern for human life, by providing life-saving information (or even saving lives), and small cost to direct supervisors, via that product's data collection systems, simple configuration, cheap yet robust construction, and intuitive controls.

A.2.4 Final Need Statement and User Outcomes

A short up-time, easy-to-use device whose design takes into account the needs of students who wish to learn as well as meets the most basic informational needs of disaster responders.

This is a tricky two-part needs statement: (1) the rover portion of the product for the disaster responders, and (2) the product satisfying the needs of the students. The aim was to direct student efforts and learning to the construction of the physical product; this directed effort is accomplished through the pedagogical tool which is the competition.

A.2.5 Hypothesis Statement

For the disaster responders, the product is going to be a rover. For simplicity of prototyping we're going to design an earthquake disaster response rover. We believe a system which is dust and rubble resistant, capable of semi-autonomous action, can collect and relay data, is cheap to manufacture, and easy to setup will help in collecting information and making decisions for disaster response teams. It may also potentially serve functions traditionally held by human responders which may preserve human life.

This physical product will be prototyped within some educational setting which acts as the abstract, student-serving component of the product. The educational setting is going to be a competition, but the specifics of the guidelines are unclear as of now. The competition would be managed by our organization, which is both a club and capstone.

A.2.6 Possible Inventions and Business Model

Possible Inventions:

Current systems for disaster relief are overly-expensive, shallow attempts at a design that jerry-rig existing structures/systems to solve issues of disaster relief. What's more is that these systems rarely provide meaningful help, have large uptimes, or return limited data back. For example, aerial systems show snapshots from a bird's eye view, but little more; they're expensive to acquire, time-consuming to deploy, and require a skilled operator. Some might argue that visual feedback is enough to trace access paths and gauge the scope of the damage, but it gives a limited understanding of victim locations, danger points, etc. Another example: most land rovers - despite being capable in their locomotion - return limited useful sensor data (compared to drones), and the current state of actuative technology is limited. The biggest thwart to effective disaster technology, however, is the context in which it's developed. Frequently made by heavy industry tech giants, the return on investment is minimal and opportunity costs are great; to them, the effort is best served elsewhere. The only place to turn to, then, is University Students who are willing and wanting to work for the low cost of a good resume and contribution to society. Here effective organizations need to be made to mobilize their talents into the creation of disaster relief robots; the best scenario is a robotics competition. Most existing competitions see students participating in events that see them designing rovers tailored to useless/limiting activities, or just working with a confining kit and no creative freedom at all. Though some break the mold, they're few and far in between, and none focus on disaster relief. To further robotic, disaster relief tech, effective student organizations must be created.

The first obvious 'invention' of this capstone is a disaster relief-oriented competition, which we call the HackRover Autonomous Robotics Competition (HARC). Traditionally it existed to provide an outlet for interdisciplinary engineering practice and an exploration of IoT; but, as the competition itself has no context there is a point for improvement. This new iteration emphasizes the creation of puzzle panels with elements and a point system that inspires the design of autonomous systems with capable actuation, computer vision and vision processing, capable locomotion, short up-time, easy configuration, and quick access.

There are some hacking competitions, and even disaster robotics competitions, (such as the one hosted by DARPA); but, all of these are largely philanthropic endeavors. Certainly we have the UW system as our foundation, and will likely receive meaningful funding from them, but this only offsets the cost of physical hardware they deem absolutely necessary. To make this organization really stand out and be financially viable we theorized a few 'inventions': (1) a payment scheme for participants, (2) providing design favors to local companies in exchange for funding, and (3) providing student research favors to local companies in exchange for funding. The first is obvious, participants in the competition will pay a fee, but this will likely only offset food, advertising, and more. The second and third see UW students dedicating physical design efforts, or research efforts proportional to the money gifted by a local company. For example, say that Boeing wants a unique end effector which performs a task they might see in their factory, and donates $4,000, while Lowes wants a software architecture reflective of potential AGVs in their warehouse and donates $6,000; HackRover may or may not form capstone inspired by these requests, but would dedicate 40% of their extra manpower to Boeing's desires, and 60% extra to Lowes.

The second main invention is a modular rover framework. In the past, the biggest hurdle for all robotic systems has been the software architecture, which must be uniquely tailored to the hardware being employed. With the help of some middleware Robotic Operating System (ROS), we can create individual software nodes and elements which can be knit together in as little as a day to create unique

Figure 4: Disaster Relief Robotics competition hosted by DARPA

robotic software stacks. Our goal, then, is to create an easily understandable, well-documented, modular rover framework that supports a near-endless number of arms, locomotive systems (traditional mostly), cameras, and other sensors, limited only by the hardware being used.

The third invention is a physical disaster relief rover. Founded on the modular rover software framework (discussed as invention two), the design of the relief rover would only concern a small group of electrical and mechanical engineers configuring various pieces of third-party hardware into a physical rover capable of speedy, robust locomotion, payload delivery, actuation, and sensing. All control (arm and locomotion), sensor data processing, and additional actuative capabilities are the subject of the well-designed software framework mentioned.

Considering the exacts of our 'rover' - (a term I use here liberally) - we must consider the context and needs surveyed in previous sections. We're designing some product for earthquake, building collapse, and warzone environments; the product must have a short uptime, be easy to control, and relay important information; finally, it must be affordable for local governments. Surveying existing products on the market, we see massive, clunky, humanoid robots that are trying to solve every problem - analogized to a universe key. In our opinion these endeavors are foolhardy, not just for us but the people doing them; more, the costs are exorbitant. We're going to control our appetite and try for something that fits a niche. Here a simple rover, which we'll define simply as 'A vehicle that navigates rough terrain' is going to be our best bet. We were inspired early on by biologically inspired designs like worms or spiders; but, scrapped the inspiration after reflecting on the manufacturing difficulties and costs associated with the rubber plastics used, as well as the control programming complexity. We then thought a drone would be a great idea: relaying LiDAR data from a birds-eye-view; but the LiDAR weight alone would drive the size (and thus costs) way up. The obvious conclusion was a ground-based rover, with tank treads that might traverse uncertain and difficult terrain. If we design out of inexpensive thermoplastics - in price ranging from PLA to Nylon fiber - and stock aluminum components, we could easily sell to local governments.

Our Solution and Invention

An intuitively controlled disaster relief rover which considers the time & resource constraints of disaster responders - with its short up time and inexpensive construction - while providing life-saving information from its sensor stack and various feedback mechanisms. The rover will be developed within an undergraduate robotics competition, where competition props (electromechanical) and guidelines will serve as pedagogical tools, and all student activities are documented in LaTeX and akin to IEEE standards.

Choosing to develop the rover within a competition setting both satisfies the needs of a pedagogical tool for UW Bothell students and produces the desired rover. The competition, we're calling the HackRover Autonomous Robotics Competition (HARC) will be held at Bothell, managed by the HackRover club and capstone project. The annual nature of the competition will serve as a force for iteration, making the design better year over year. The addition of hacking serves to penetration test rovers so they can't be taken advantage of while performing life-saving tasks. In a good-faith effort to bring the competition to life we will also be making puzzle-panels, which are competition props that the robotic systems interact with. These puzzle-panels are themselves pedagogical tools designed to teach students the basics

Figure 5: Biologically inspired robot

of mechatronics design. Puzzle elements are laid out on about four floors, accessible by stairs/elevators. Students operate rovers, attempting to solve puzzles and gain points while navigating the building effectively and hacking/slowing other competitors' rovers. Points are awarded based on solutions to puzzles, specifically designed to encourage rover designs that would be capable in disaster relief contexts. Despite shortcomings early designs, the rover will serve as disaster relief in limited contexts and will provide an enriching educational engineering experience for UWB students now and for the future. The practice of documenting in LaTeX will not only provide students with industry-standard skills, and aid at generating a wealth of information which might be used by other developers; but, it will also speak to the high standards of practice, and aide in on-boarding students with ease, lowering the learning curve. This good documentation practice and logging of information may come in handy in the future if we act on the business plans discussed above.

A.2.7 Planning Roadmap

Initial Product: A competition with a rulebook, at least two rovers (one disaster-relief oriented), and four puzzle panels.

The first competition encourages student involvement, a clear understanding of the framework, and expectations for the iterative methods used in developing our systems. The first two rovers created will be designed to prototype a 'skeleton' software framework that exemplifies the principles of the software organization we desire. Physically these rovers will serve different purposes: the first to simply prototype the competition, drive around Discovery Hall, and make use of our available technology, and the second to provide an early model of what a disaster relief rover could be. Focusing more on the latter robotic design, an early disaster relief rover would have quick, rugged locomotion systems, effective sensor data storage, feedback, and/or processing, capable payload delivery, and meaningful arm actuation. At this point in the 'product' development, we wouldn't focus too hard on waterproofing or extremely robust chassis. The obvious adopters here are going to be UWB students, capstone students, Pierre Mourad, STF, and only philanthropic companies.

Product Additions: *An improved, R and D-friendly, competition.*

The competition will eventually be formatted to address the desires of tech companies nationwide. Interested investors can trade capital for limited control over the direction of the competition and design requirements of involved rovers. As the competition and participants grow this offer would be more lucrative, and the potential for outsourcing cheap but quality R& D increased greatly.

Product Additions: *Specialized Software Nodes.*

Mentioned implicitly through the document is the notion that software nodes 'providing/extending capability' of the rover may be created. A feature of ROS is that functionality can be compartmentalized to a web-like software architecture. Drastic changes in the hardware do not require extensive recreation of the software; instead, nodes are repurposed and reorganized to achieve a framework that reflects the physical system. A skilled ROS operator with knowledge of the nodes' functions could create a completely new system within a day, for a totally unique rover. Future product additions could see 'stacks' of nodes created for unique disaster relief settings: building collapse, fire, aqueous, etc. Each of these stacks would come with the necessary code to configure context-specific locomotion, sensing, and

actuative systems. Instead of purchasing an expensive system that 'does it all', clients could buy just one software stack suited to their disaster response expectations. The rovers' modularity will have different packages that a client can use to configure their rover to their specification. For example, if there is a need for a LIDAR system then there is a module that would integrate the LIDAR system for a client. This would already be done physically as a unit that would attach with one connector to the main rover board and the software would just have to have slight code changes to adapt to the new module.

B Experience Design and Human Interface Design

B.1 User Workflow

Before addressing UX in depth, a quick review of the state of the systems. The last capstone sent forth a rover with hardware components ordered according to an (almost) ideal rover architecture, however no software (at least not in ROS) was in place. All control was exclusively done through a script loaded onto the Raspberry Pi (RBPi) which enabled a user to control the rover via a Bluetooth game controller. The controls were hardly programmable, and what existed was limited (no control over LiDAR, arm, etc). Before any mind can be paid to serious UX, the software framework has to be created; each node of the rover must be in contact with others via the Bluetooth to enable us to create useful control scripts and paradigms.

Despite the little mind paid to UX, there are some basic expectations and workflows, for both the competition and rover. Within the organization/competition, students will form teams for competing. Each team will construct at least one rover which is capable of wireless access and control, and is based on ROS; we want to maximize creative potential, so almost no design is off limits. They will be able to get puzzle component software libraries and construct/prototype puzzles of their own for their rovers to interact with. They will study up on hacking and common system penetration techniques, writing/developing their own (or defenses) in anticipation of game day. Finally, they will tie all their efforts together with HackRover organization standardized working papers (based on IEEE formats) that be sent to the organization, which summarizes their work.

On game day, teams will stay in command rooms, where they can semi-autonomously control their rovers. Each team will be distributed random vulnerabilities that reflect real life vulnerabilities. Through the competition all will try to:(1) solve puzzles with the rovers, (2) hack other teams' rovers, and (3) defend their rovers from hacking, to gain points.

Moving beyond the student organization we inspect the workflow of a disaster responder using our rover. The rover will turn on, connect automatically to a dedicated computer, and on that computer a launch file will be run. This launch file will associate all internal components with each other, and the rover with the outside world (GUI, communications and controls, etc). The launch file can either be booted up through SSH from another computer to start the processes, or can be automatically run by the rover itself. During testing and development of the rover, we recommend that the launch file be booted up through SSH from another computer, and in practical applications that the rover has the launch file set to automatically execute as the rover commences its boot sequence. In a short time, the disaster responder will be able to send the rover out in autonomous, or semi-autonomous mode to collect life-saving information. The rover, being rubble resistant and small, will travel in to precarious situations, looking for trapped victims, relaying the info back to the operator. This workflow does, of course, reflect an ideal product far into the future that we're working towards; but, it is possible with time and dedication.

B.2 Environmental Analysis

The environment of the rover is two-fold: competition space with simulated events/circumstance, and confined disaster spaces. The web-based UI and simple wireless control serves to empower the competitor's experience during the event. However, it should also easily map onto the disaster location. Good communication with the rover system as it operates in distant, wireless connected, cramped spaces are a must. Ease of control programming and use is a must. Good sensor and visual feedback is a must. Robust traversing of the environment is a must.

Assuming all these 'musts' mentioned in the last paragraph, the three-pronged solution to satisfy them is: an effective and functional software architecture, a reliable mechanical system, and a long-lasting power distribution system. The software architecture, created in ROS, will be made with modularity and programmability in mind. The mechanical system will transcend its simple motor-shaft design into the realm of geared drive trains and intelligent chassis configuration. The power distribution system will consider as many power-saving mechanisms and electrical stop-gaps to failure as possible. The resulting system should be easy to program, deploy, control, and service.

B.3 Cultural Factors Mapping

Our device needs to save lives, end of discussion. Cultural factors are of comparatively less importance unless they help us do that better. The two factors mappings we genuinely believe will empower saving

lives are: (1) intuitive controls based off video games, and (2) friendly rover design.

Most remote-operated technology uses joysticks and an array of push-buttons, incredibly unintuitive and often requiring hours of training for operators. Looking to the pop-culture side of controls we see video game systems. Sony has sold 180 million PS2s, while Nintendo has sold 102 million Wii's; both these consoles defined a generation of children, and trained plenty of potential rover operators by popularizing their consoles control paradigms. We might harness this power by using Wii Motes or Playstation/Xbox controllers to control the rover; in fact early iterations will use an Xbox controller, and we've discussed controller mappings for Wii motes in depth as a team.

Figure 6: A motion sensitive Wii-mote (right) and nunchuck (left)

The other factor mapping - friendly rover design - sees us making our rover amicable and easy to approach by all sorts of people. From built in displays to lovely shapes, when the disaster rover finds a victim it might restore some hope and provide comfort. Industrial designs may cause anxiety and concern upon first contact.

B.4 Information Architecture and Hierarchy

For startup: The rover will have a modified boot up sequence so a launch script can be executed automatically. This launch script will comprise of turning on the ROS core network and any specific ROS nodes that are vital to the rovers operation in the beginning. The boot up sequence for most Linux based computers such as the NVIDIA Jetson are easily modifiable through boot configurations, which differs depending on what Linux distribution is being utilized.

For use: Intuitive Bluetooth pairing button combinations, or browser-based logins will allow the user to quickly access the rover. The startup launch, mentioned prior, would have enabled the rover to receive simple communication. Perhaps a UI for selecting controls before use will ensure the correct systems are selected. From there all controls should be either intuitive, well documented, visually displayed, or easily accessible from the file system of the rover.

During use: The user is assumed to be interacting with the rover at this point. All inputs will forward from the control scheme directly to the master node hosted on the Xavier (or Nano). This outputs control commands to the necessary sections (LiDAR, arm, etc), for which feedback is received. This feedback is processed as we deem fit and then manifest as either actuation of the physical rover or display on a desired monitor. For example, a control input may be processed through some control script, then modified by state conditions of an actuating device to provide some output. Alternatively a change in the environment, or press of a button will alter what sensor information is displayed on some feedback monitor. We plan on having only certain data be displayed in an orderly fashion, rather than providing the user the ability to select what they'd like to see. The reason is that only limited information will be displayed back as it already is, and we'd rather not muddy the use or development process (in creating such a system). Shutdown after use will be the reverse of startup, just as easy to accomplish.

On the Electrical Architecture, digital wiring between components are essential for having the rover to operate correctly and safely. In the first major architecture, the rover needs the connections between the different hardware to be correct, one misplaced wire could lead to severe failures for the rover. Failures

can involve, hardware breaking down electrically which causes damage or the rover moving in the wrong direction. Knowing which wire connects from hardware to a specific node is important. The connections between the Arduino Mega and the Motor controllers involve enable and digital signals. There are wires that are in charge of directing digital communication to allow the software within a hardware to operate correctly with other hardware.

B.5 Interface Design

The most base and simple design for interfacing with the rover is obviously mouse and keyboard. This default is assumed with ROS, as it runs in its own terminal and accepts typed inputs. Work will need to be done to create a GUI that accepts keyboard/mouse input, but it's the minimum priority.

The powerful aspect of ROS is its modularity, we might design an input node that sends commands to a control mapping node, which translates and forwards to a control script node. If, on a whim, we decided to change the control paradigm, we would substitute the input node with another, say an Xbox controller or Wii remote. The rest of the framework would continue to work so long as our mapping node was configured correctly. Beyond keyboard and mouse, we do plan to use the Xbox controller. Operators would be able to press just a few buttons to get the rover into an active, controllable state.

Feedback from sensing devices, (like LiDAR or encoders) is streamed to a monitor output. This provides a sort of 'commander center' for the operator to get information back. ROS has options to allow LiDAR or encoders to interact with ROS motion nodes, such as cmd vel which is a built in ROS node that commands velocity.

The built in ROS command velocity node is a quick and easy way to integrate a built-in API that allows users to utilize teleoperation. Instead of hand creating a velocity API that takes in movement commands and outputs velocity commands for motors, we are choosing to use cmd vel from ROS. This allows us to quickly integrate and test with our rover, and can give us understanding on how others may also quickly implement their own solutions with cmd vel.

B.6 Graphic User Interface Design

The GUI, mentioned in passing in the previous section, is known as our command center. In it's most base form it's designed to provided sensor feedback and minimally processed data. The GUI itself is hosted on the rovers main computer, and accessible wirelessly via a browser. At a minimum we plan for it to stream direct visuals and a heatmap from the LiDAR camera, (which processes and sends data in ROS nodes of its own). We'd also like to have encoder data to understand the more data (such as speed) at which the rover is operating at.

The nature of our GUI being designed within a ROS framework means future modifications can be made: informative post-processed statistics, sensor additions, and more. Specific 'profiles' might be engineered as simple scripts, where more or less information is provided.

The GUI was built using HTML and CSS while the backend that connects and is hosted as a ROS node is built utilizing Flask in Python. The mentioned code is referenced in the official HackRover GitHub Rover repository.

We did name it 'command center', implying some form of control. In its current state, and for our nearest plans, there isn't this aspect. The keyboard/mouse combo, or Xbox controller manage all controls; but, we have discussed the very real possibilities of adding settings that can: (1) modify rover parameters in real time, and (2) switch between control paradigms and profiles. For example, as a user I may find the proportional, joystick speed control too wiley, so I tap a switch in the command center that allows me to set a constant speed. Another example, I may like working with Xbox for most operations, but would like to enter into a more privileged mode with keyboard and mouse, or maybe, my coworker prefers a Wii remote; the command center should allow for seamless switching between these control paradigms. We would also want to implement that a controller is connected directly to the laptop viewing the webserver is able to control the rover through that paradigm.

B.7 Best Use Practices Report

There are virtually no 'Best Use Practices' for any mechanical or electrical subsystems of our rover. We designed with the intent of offloading physical configuration from the user; all best use falls in the lap of human-computer interaction, and our computer engineers.

Minimum best-use expectations are relying on custom designed launch scripts for initial setup. We want the user to avoid typing into any terminal, or plugging directly into the rover with computer. This

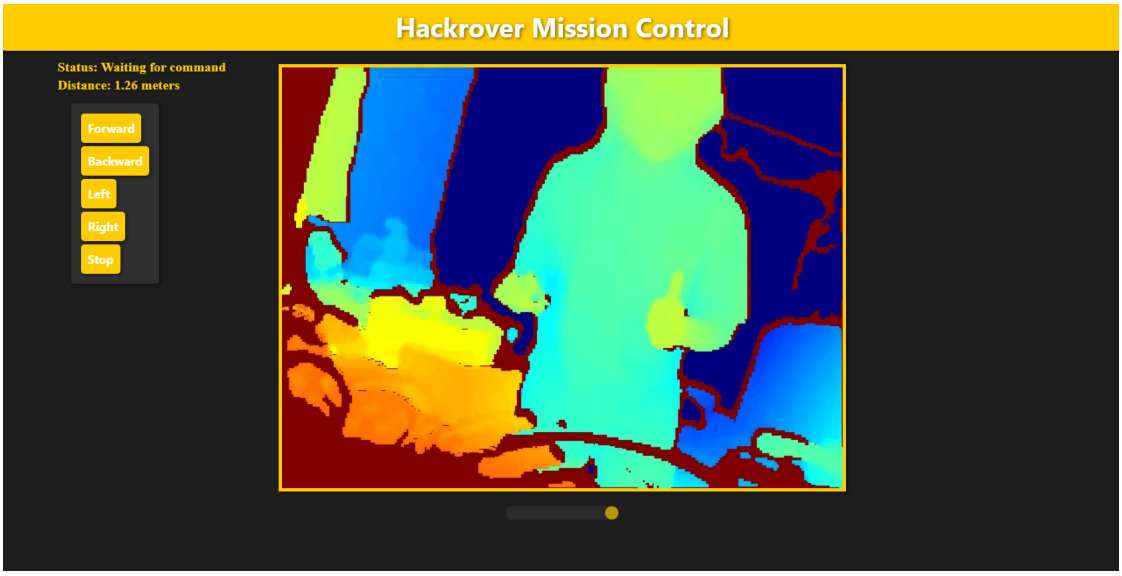

Figure 7: HackRover mission control with a depth heatmap activated

decision acts to simplify the setup process, and also hide information. Hiding information is important because it removes the opportunity for making careless mistakes.

Once the launch scripts - activated with simple turn on or button press - setup the rover, the user should spend all their time within the command center GUI, and the controller in their hands.

There are some limitations and vulnerabilities that we have thought about with hosting a webserver based off of the WiFi that the rovers computer is attached to. The first would be that hosting a website on a public WiFi that anybody can access is inherently dangerous in terms of cybersecurity. Secondly, there are bypasses that have to be done to allow the rovers computer to host the webserver such as allowing any user that is connected to the computers IP address to modify serial ports, through Linux change modification commands. Lastly, a Denial-of-Service (DoS) attack can flood the public IP address that the rovers computer is hosted as with a high volume of traffic, which would overwhelm the rovers resources and causing the website to become inaccessible to legitimate users.

B.8 Usability Requirements Document

The first product will be a robust rover system with a modular, programmable software architecture created in ROS and hosted on the central Xavier or Nano computer. Booting the rover system will be straightforward, and all background tasks for effective startup will be offloaded from the user via automation by simple scripts. The user will be able to easily control the rover from one of many control paradigms, and see useful output on a monitor through a custom designed GUI.

Our primary objective is to ensure that users can easily navigate and interact with the interface, enabling efficient and intuitive control of the rover's functionalities. By adhering to a set of usability requirements, we aim to optimize the user experience and maximize mission success.

1. Intuitive Interface:

   The Mission Control GUI should feature a visually appealing and intuitive design that enables users to quickly grasp the various functions and controls. The layout should be logical and organized, with clear navigation paths and easily identifiable icons and buttons. Prioritizing simplicity, the interface should minimize clutter and present information in a concise and understandable manner.

2. Ease of Use:

   To ensure the widest possible user base, the GUI should be accessible to individuals with varying levels of technical expertise. It should offer both basic and advanced control options, allowing beginners to easily initiate missions while providing advanced users with advanced functionalities.

3. Responsiveness and Real-Time Feedback:

   The GUI should provide real-time feedback to users, ensuring that they are aware of the rover's status and any ongoing operations. Response times to user commands should be minimal, allowing

for immediate feedback and preventing any perceived delays or latency. Real-time telemetry data, such as battery levels, location, and sensor readings, should be displayed prominently, facilitating informed decision-making.

The form factor of the rover itself will be akin to last year: a PCB interfacing the Xavier/Jetson with an RBPi and motors; and a LiDAR and communication stack communicating with the Xavier/Jetson. All UI will be handled through ROS.

C Requirement Documentation

C.1 Tiered Schematics

C.1.1 Design Decisions and Rationale

The final plans for the build phase will - at a minimum - include the creation of one first iteration disaster rover (Mimir). Experiencing success with these sooner than later will prompt us to improve upon the base design with further iterations (which introduce more electronics, better mechanical housing, cleaner code), as well as create more puzzle panels and competition rovers. With the success of the first iteration rover, this sequence's final build plan is to improve on the power distribution system and add computer vision functionality. As a stretch goal, the drive and movement system are being phased out from the Arduino to a Raspberry Pi.

Rover

The rover will operate in earthquake, warzone, and building collapse environments. it needs to be rubble resistant, have a low center of gravity, and stocky. The electrical systems that power it must be reliable, robust, and deliver clean energy. The software framework that underpins it must be modular for future development or rapid on-site changes. This modular software framework will also translate into the physical of the rover design, with mountings and electronic connections that are modular.

The rover also sees the addition of a novel, camera control method the form of a Stewart platform. Traditionally the Stewart platform has been a parallel robot with 6 linear actuators. The significance is that this device can move in 6 DOF, and reverse kinematics can be used to calculate actuator position - simply plug in desired position and a math model determines actuator output. Recently roboticists have been popularizing a Stewart platform that uses rotational servos - as opposed to linear actuators - and the designs/models are promising, especially for their cost effectiveness. We plan on a Stewart platform controlling the LiDAR camera.

C.1.2 Tier 1 Schematics

Mechanical Team

An early mapping of the rover seen in figure 8 reflects what we actually intend to build. As you can see, the chassis is divided into drive base and housing, where housing is further subdivided into tiers. The size of components mostly dictates the function/organization of tiers; but, conveniently electronics and microcontrol will occur in the bottom tier, higher level microcomputer control and networking will occur in the second layer, and sensing with the LiDAR will occur at the highest layer.

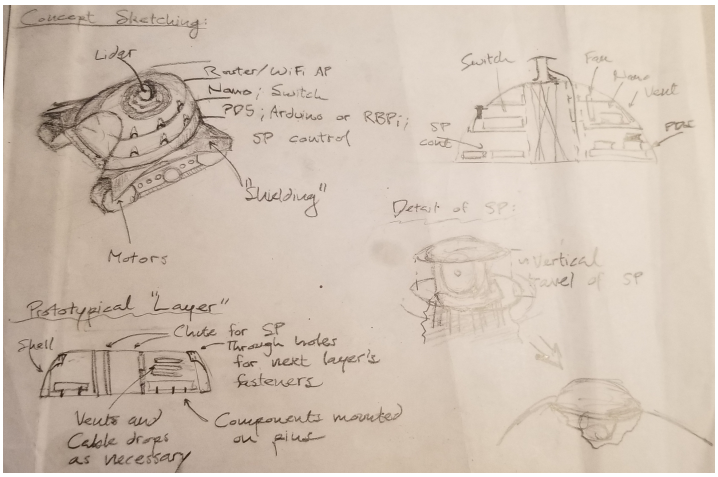

Figure 8: A sketch detailing early design of the rover

Electrical Team

The rover will need a power source that can provide enough reliable and steady power to all hardware. A power distribution system will be needed to safely distribute enough power to all hardware and also, the size of the PDS will be dependent on what the final size of the mechanical system will provide for the PDS to fit in. The PDS has also been updated to allow the use of wall power or battery power.

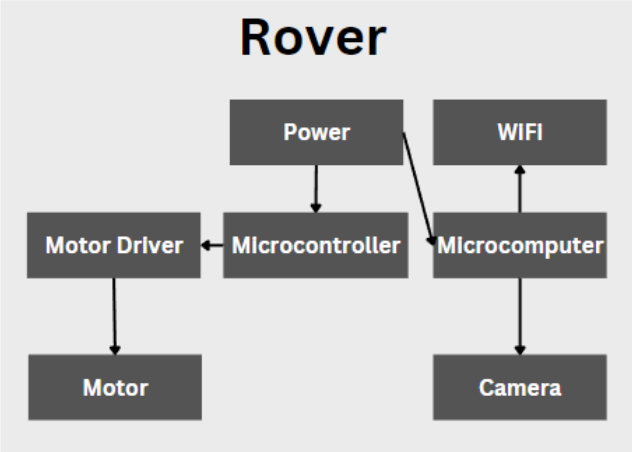

Figure 9: Tier 1 schematics of the rover

Software Team

The basic diagrams are shown below for the software team. These show us the major nodes of the ROS framework which consist of the Intel Realsense LIDAR L515, SLAM package, Web Server Node, Teleoperation Node, and the ros core from the Jetson Nano. Web Server and Teleoperation are currently nodes, but they may be sorted into independent packages for modularity. We also introduce a second diagram that shows the controller system that is connected in the fashion shown in the diagram. For the third diagram 12, the package displays distinct components that are necessary for the function of capturing its environment and mapping the environment that the rover can utilize and human eyes can recognize. The point cloud library is responsible for rendering physical obstacles onto the Rviz visualizer as a immense collection of colored points. Working in tandem, the Ceres-solver library is responsible for non-linear optimization, which assists Octomap library in "stitching" the series of created points and rendering them correctly in 3D space. Notably, the camera captures the depth of the environment at a set rate, and the Jetson Nano computes and visualizes the result using all the listed SLAM packages. For our stretch goal of implementing a comprehensive control system for the Stewart platform and drive system, 13 shows the interaction of the Wii mote, which is adapted into electrical signals for the Arduino microcontroller. The microcontroller then moves the Stewart platform accordingly. 14 is our current stretch goal implementation to move the Stewart platform and drive system. This model removes the important Control Selector Service that allows multiple types of controller to connect to the control systems. Please reference the Summary, Conclusion, and Future Perspective for the full outline.

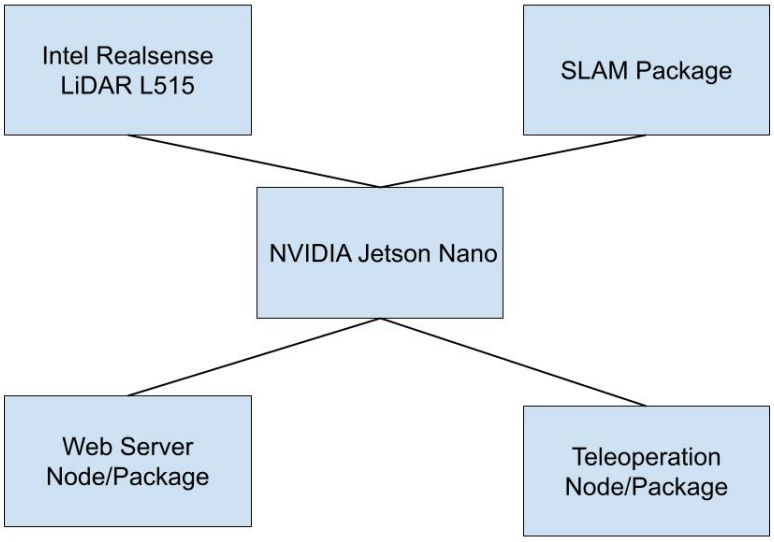

Figure 10: Dominant nodes of the ROS framework

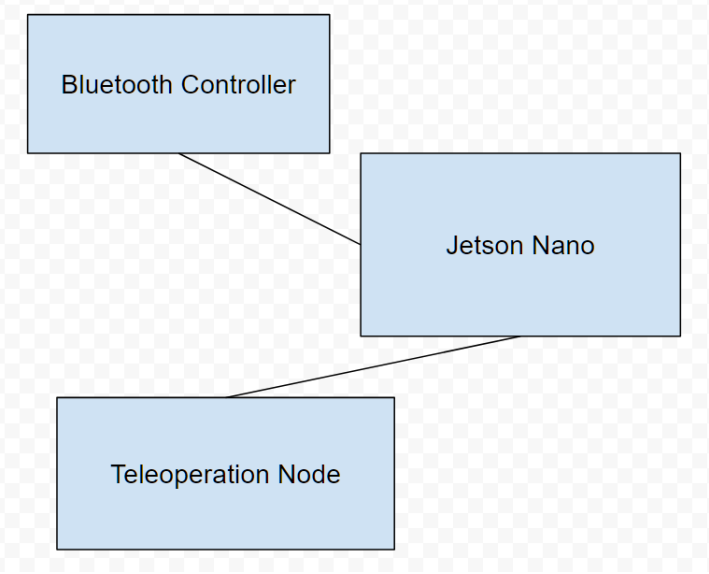

Figure 11: Controller system design diagram

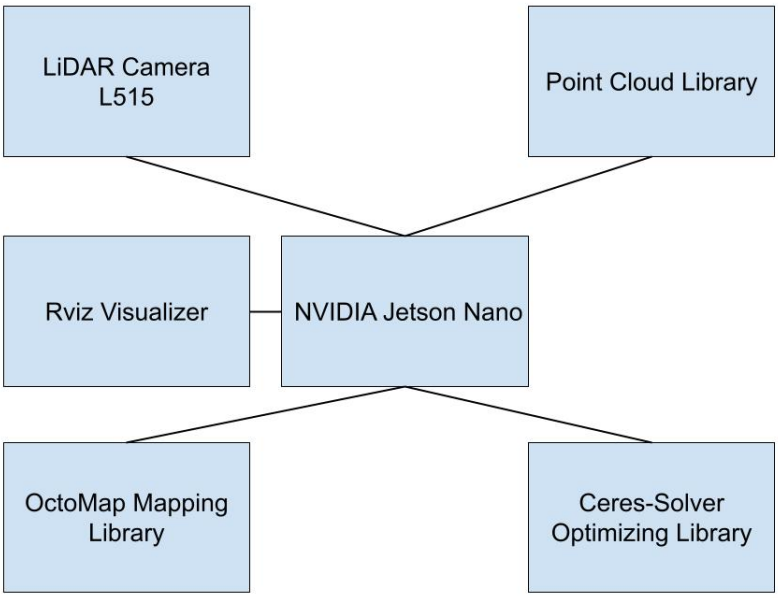

Figure 12: SLAM package design diagram

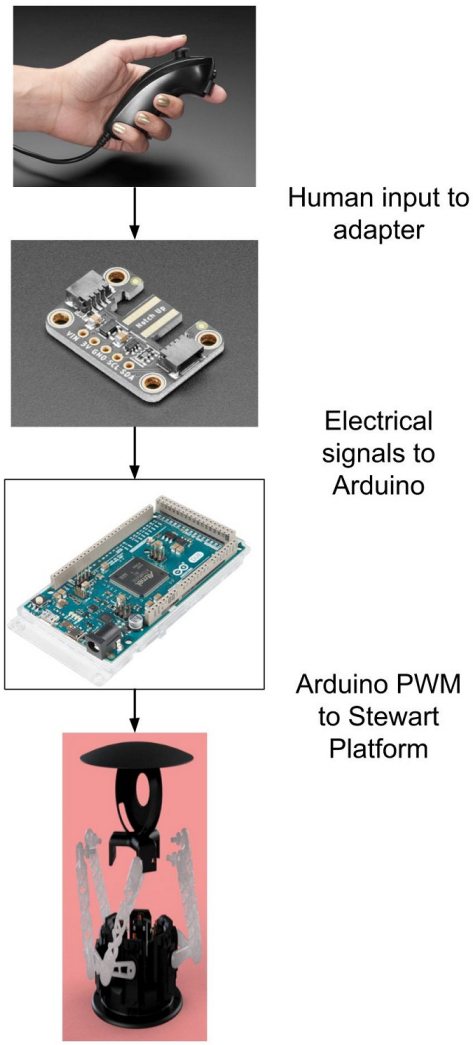

Figure 13: Stewart Platform Physical Architecture and Use Case

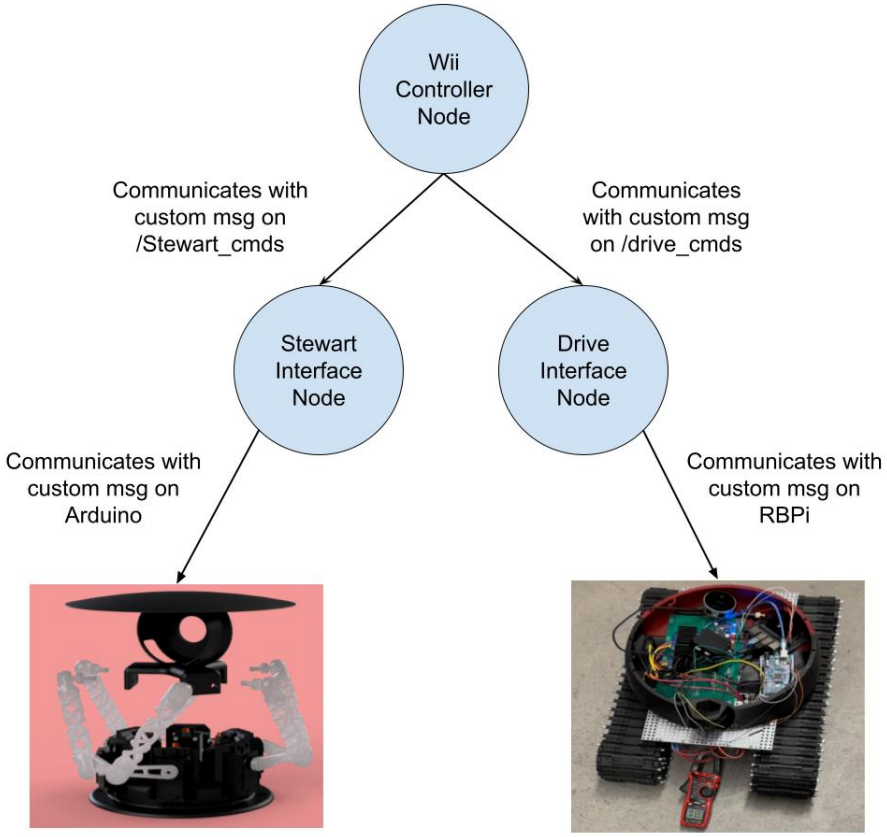

Figure 14: Control Software Architecture In Progress

C.1.3 Tier 2 Schematics

Mechanical Team

Provided thanks to retrospect, we can see a better visual of component placement in Fusion360. Here the same tiered 'territories' exist within the rover design. Components included in the images in figures 15 and 16 should give some idea of function.

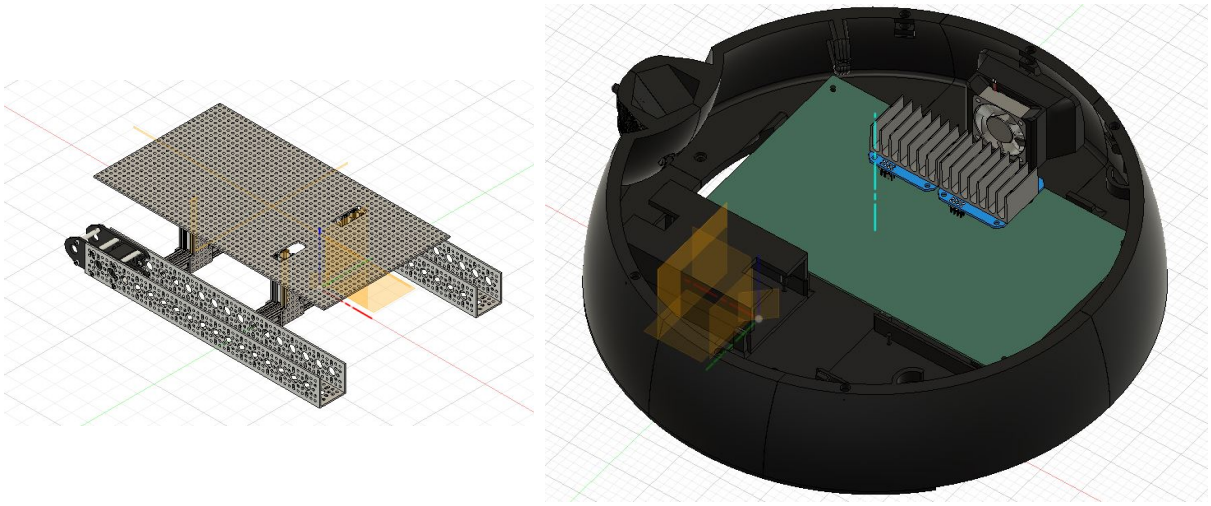

Figure 15: The drivebase without treads, and rover bottom tier

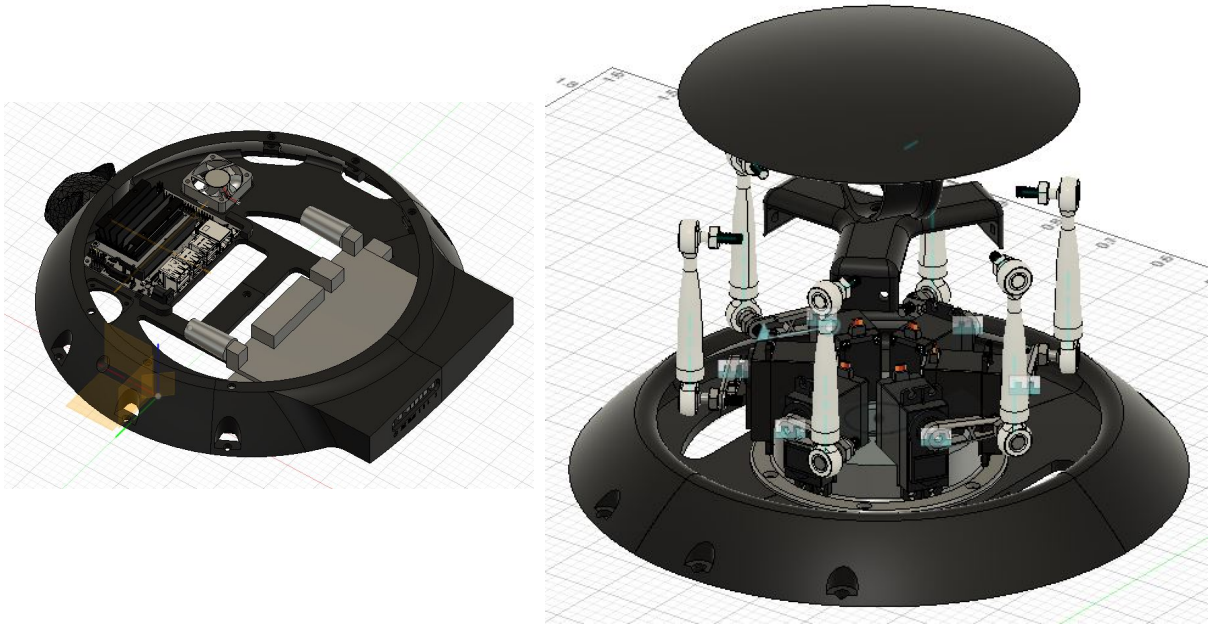

Figure 16: The middle tier and top tier (with Stewart platform

Electrical Team

The simplified Rover Electrical Schematic details the connection from the power source to the rest of the electrical components that are present in the system. The hardware involved is essential for controlling the rover. The Jetson Nano is a microcontroller that has a discrete graphics processing unit which means it handles AI and image processing particularly well. A Wifi component is built into the Nano which can be controlled wirelessly without having to control it with a physical wired connection. The Raspberry Pi is a microcontroller which is coded to control the motor controllers. The Nano is commanded by the user which the user would send a command to the Nano, and depending on the command which will be sent to the Raspberry Pi. This scenario is when both the Nano and the Pi talk to each other with the Nano being controlled by the user have the final say. This relationship works the same way with the Arduino in the Stewart platform. The signals that are going to be sent to the motor controllers in the motor system, contain directional/control signals. The motor controllers act like a mini-computer which receives the signals from the Arduino and transmits those signals to the motors which as a result, runs the motors. The Srduino with the Stewart platform carries the LiDAR camera which monitors live footage and sends back the information in a wireless connection back to the user. The Stewart platform is formed with many small stepper motors that reorient the position of the platform from the Arduino signals.

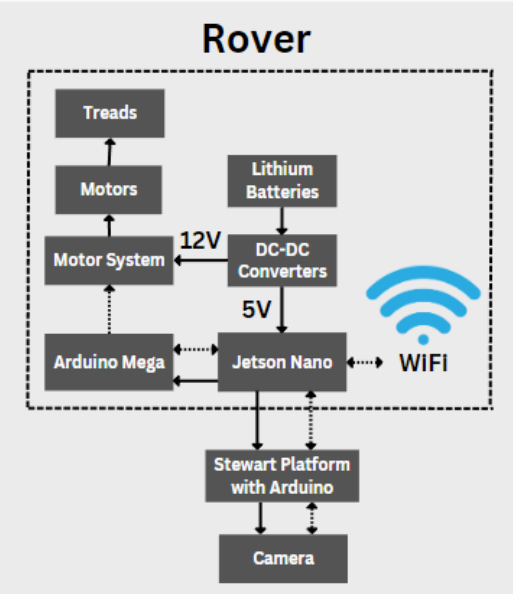

Figure 17: Rover sketch detailing connection of components

Software Team

Figure 18 details the interaction between the Intel RealSense LIDAR L515 with the NVIDIA Xavier. The ROS node that is used as the publisher of the LIDAR image data to then be sent to a subscriber to the image data uses image_ transport as the library to export the images from the LIDAR to the processing node in a low-bandwidth compressed format to reduce latency. The image processing node has computer vision models within the node to process the image using OpenCV, which is then sent to the Xavier as data that the Xavier will use for further motion logic or displaying the image to the user.

The 18 details the interaction between controllers (Xbox, Playstation, etc type controllers) with the NVIDIA Xavier. There is the package that contains the motion logic that the previously mentioned LIDAR interacts with to send motion controls to the Xavier as well as a ROS node that takes controller inputs and processes them to be sent to the motion logic package. The data going from the controller to the ROS controller is analog data that Linux can already understand and there are pre-existing ROS libraries to use controllers. The controller node itself will send out motion commands to the motion logic package which will then send motion data to the Xavier. We also want to ensure that the controller node itself won't do any illegal inputs so we want to create a connection between the motion logic package and the controller node in case of inputs that were not friendly to the rovers motion.

The RQTGraph details the all nodes that depend on one another to run a specific package. As shown, the camera runs its ROS wrapper nodes under /camera. The input captured from the camera passes to the custom-made ssl slam nodes, which process them into point clouds, then into /odom node for positional data, and eventually mapping on /map node and visualizing in ROS's rviz node. The processed data in /map node can and shall be utilized for future projects.

20 displays in further detail on our implementation of our drive and Stewart control system. Sections containing the nodes are color-coded for clarity of where the nodes should be located: green is Nvidia Nano, blue is Arduino, and salmon pink is Raspberry Pi.

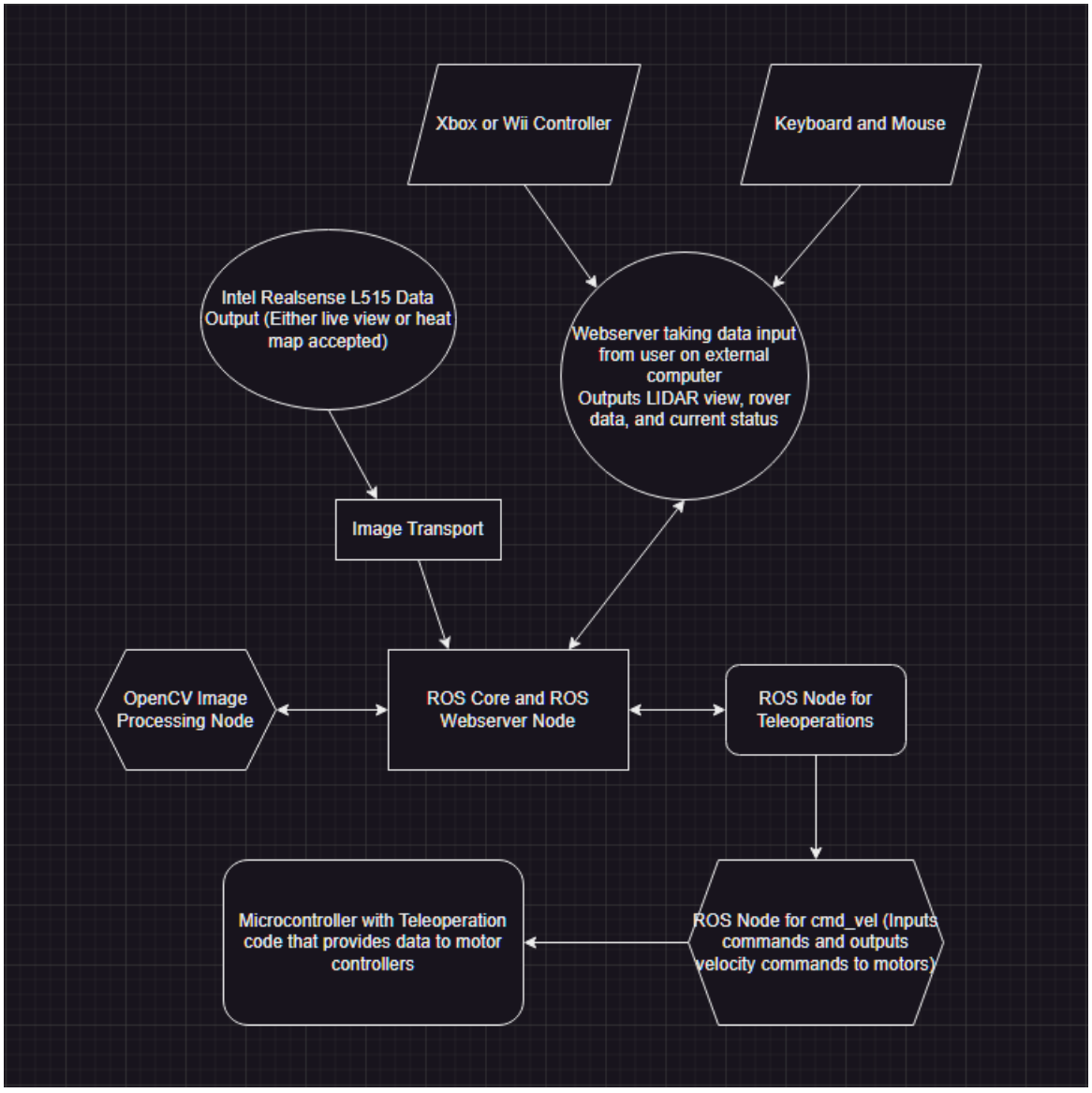

Figure 18: Entire ROS Setup in Detail

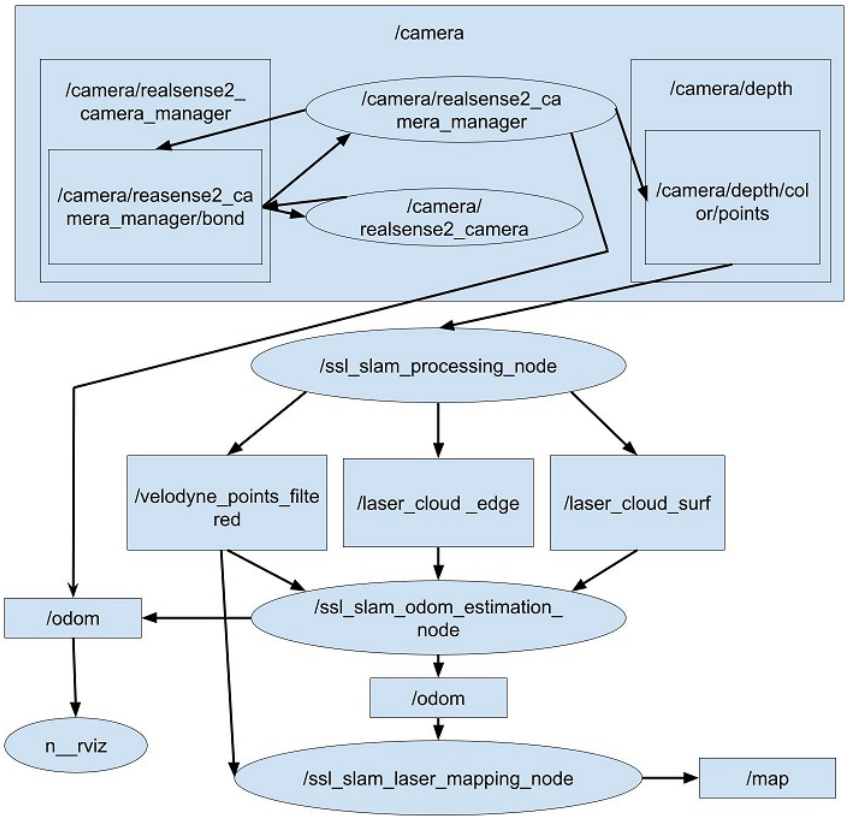

Figure 19: SLAM Nodes in Detail

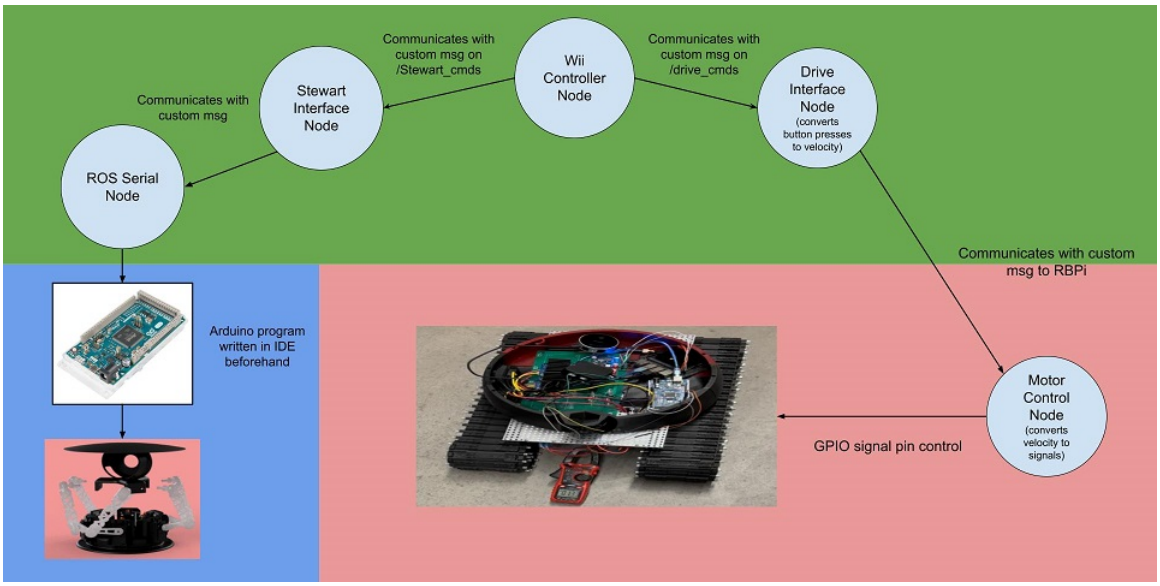

Figure 20: Stewart Platform and Drive Control System in Detail

C.1.4 Tier 3 Schematics

Electrical Team

The rover will be powered by lithium batteries that can provide enough power to the whole rover. The 12V DC-DC buck converters will output 12V to power the motors. The small buck converters, convert the 12V to 5V which powers the Jetson Nano, Raspberry Pi, Stewart platform, and other components.

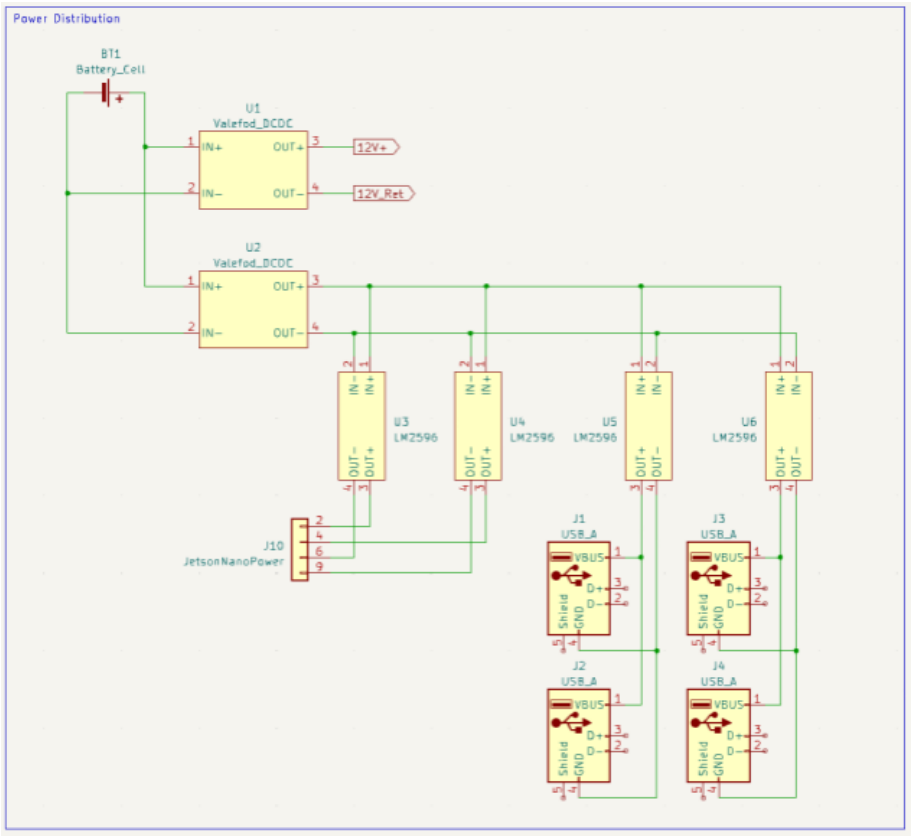

Figure 21: A schematic of the rover PCB

The power will be distributed with buck modules to power each component at its needed level. The power, when stepped down, will then be delivered to the Raspberry Pi, Jetson Nano, Motor controllers, and other components which will run the whole rover. This PCB is constructed to use four motor controllers but in this project, two are used to operate the rover. This allows future teams the option of using four motors.

The four USB ports output 5V which are used to provide power to any additional add-ons to the rover, which means different hardware can be added. In our case, we added 20mm case fans.

In Figure 22, on the right, you may see a single example of the connection of a motor controller and motor encoder and on the left, you may see how all four motor controllers and encoders connect to the Raspberry Pi.

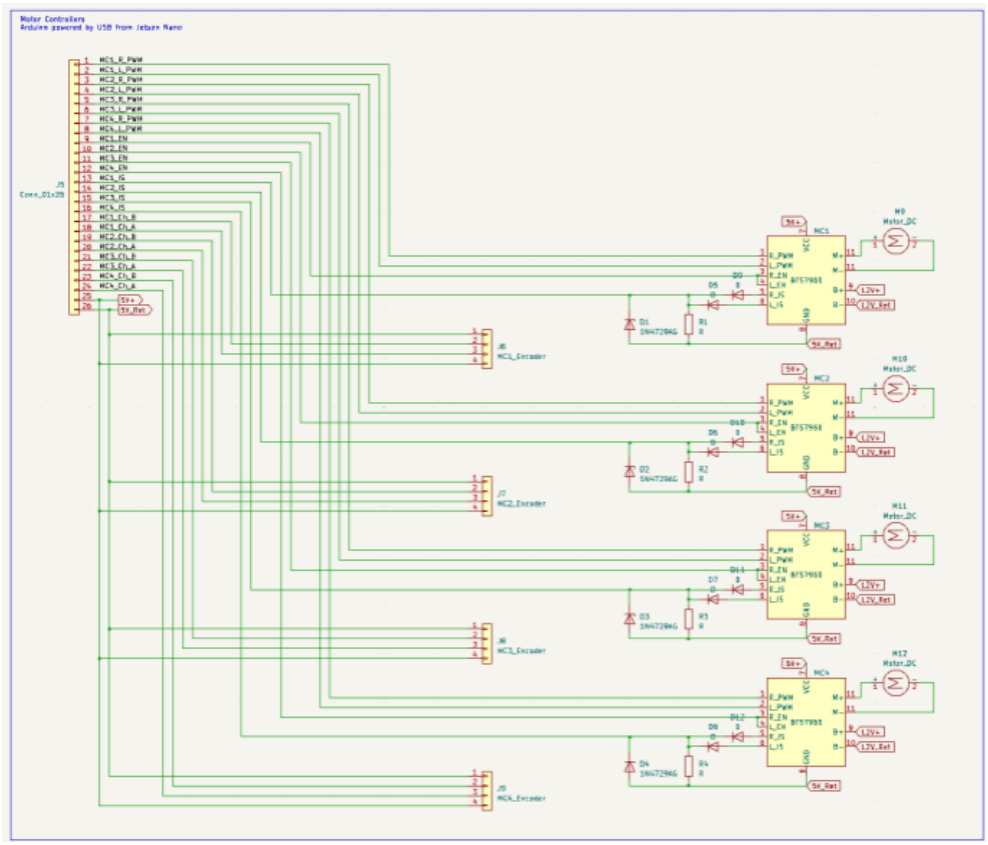

Figure 22: A schematic of the rover PCB

Software/Computer Team

Starting with the LIDAR integration the main connection is between the publisher and subscriber node of the LIDAR image. The publisher node will be outputting a compressed formatted image to the subscriber through the image_ transport API. Within the processing node (subscriber to the image data) we can use OpenCV to translate the image that is being collected from the publisher to be created into a model that the rover can use to identify what is being shown in the image. This can be done with a point-cloud model and computer vision techniques. After the image is processed, the image itself is sent to the Xavier to display to the user and the image data with motion commands is sent to the motion logic, where the motion logic will then determine whether the images motion data is relevant to motion objectives. The latter is a back and forth process between the subscriber processing node and the motion logic component. The second part with Xavier for this capstone project is the controller node. The main connection will be between the controller node that takes in controller information and the motion logic. This will also be a back and forth process between the two since the motion logic may not want to perform the actions that the user is inputting into the controller. A launch script is also created to simplify launch operations for the rover. This allows us to run the rover without having to connect a screen, mouse, and keyboard to set up the rover with the correct environment. For SLAM, these are the custom-made files for creating the ROS nodes. As there are multiple files for executing SLAM, the source code won't be displayed, but they execute in accordance to 19. This is the src directory in ssl slam package responsible for running SLAM. Running SLAM only require these few command lines in Terminal. Currently, the Stewart platform and drive system controls do not have a Tier 3 architecture since the details of stretch goal are still in progress.

```python
def start_pipeline():
    global pipeline
    pipeline = rs.pipeline()
    config = rs.config()
    config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
    config.enable_stream(rs.stream.depth, 320, 240, rs.format.z16, 30)
    pipeline.start(config)


def generate_color_frames():
    global pipeline, running
    while running:
        frames = pipeline.wait_for_frames()
        color_frame = frames.get_color_frame()
        if not color_frame:
            continue
        img = np.asanyarray(color_frame.get_data())
        ret, buffer = cv2.imencode('.jpg', img, params=[cv2.IMWRITE_JPEG_QUALITY, 90, cv2.IMWRITE_JPEG_OPTIMIZE, 1])
        frame = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

def generate_depth_frames():
    global pipeline, running, distance_value
    align = rs.align(rs.stream.color)
    colorizer = rs.colorizer()
    while running:
        frames = pipeline.wait_for_frames()
        aligned_frames = align.process(frames)
        colorized = colorizer.process(aligned_frames).as_frame()

        depth_frame = aligned_frames.get_depth_frame()
        if not depth_frame:
            continue

        # Calculate distance value at the center of the frame
        width, height = depth_frame.get_width(), depth_frame.get_height()
        distance_value = depth_frame.get_distance(width // 2, height // 2)

        # Convert depth frame to heatmap
        heatmap = cv2.applyColorMap(cv2.convertScaleAbs(np.asanyarray(depth_frame.get_data()), alpha=0.03), cv2.COLORMAP_JET)

        # Convert heatmap to RGB format
        heatmap_rgb = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)

        # Encode image as JPEG with optimization
        ret, buffer = cv2.imencode('.jpg', heatmap_rgb, params=[cv2.IMWRITE_JPEG_QUALITY, 90, cv2.IMWRITE_JPEG_OPTIMIZE, 1])
        frame = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Figure 23: Python Code For Creating ROS Pipelines and Generating Image Frames

```
screen -dmS roscore bash -c "source ~/.bashrc && roscore"

sleep 10

screen -dmS rosserial bash -c "source ~/.bashrc && rosrun
rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200"

sleep 5

screen -dmS webserver bash -c  "source
~/.bashrc && cd webserver && sh weblaunch.sh"

sleep 5
```

Figure 24: Shell Script For Booting Rover Automatically

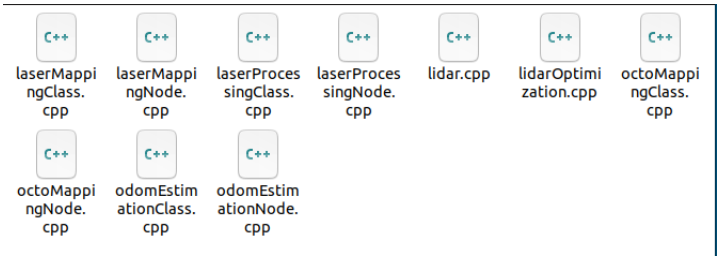

Figure 25: src directory in ssl slam package

```
jetparty@uwb:~$ source /opt/ros/noetic/setup.bash
jetparty@uwb:~$ source ~/catkin_ws/devel/setup.bash
jetparty@uwb:~$ rosrun rqt_graph rqt_graph
jetparty@uwb:~$ roslaunch ssl_slam ssl_slam_L515.launch
```

Figure 26: Command lines responsible for running SLAM

C.2 Estimation of Cost of Goods

The final estimation of cost falls around 1,500 dollars. Overtime, the cost changes due to the major design decisions have to be made by electrical and mechanical leads before a final order can be placed. An each orders are done on a weekly basis with each section having a total shown below.

The estimate costs of goods of the Electrical components falls to 915 dollars which includes miscellaneous components that are already in our inventory that costs around 400 dollars. The Mechanical components falls to 233.24 dollars """(include also, estimation costs of miscellaneous mechanical parts already in inventory)"""

Amazon	MIN CI 20mm DIA Neodymium Magnets	N/A	https://www.amazon.co	1 Set	$22.49
Amazon	BluexYellow DIYmall OLED Module 0.96 inch I2C IIC	N/A	https://www.amazon.co	3 Screens	$30
Amazon	HUAYY 6 Inches Width 1 Yard Length | Hook and Loop Self-Adhesive Material	N/A	https://www.amazon.co	2 Sets	$30
Amazon	TX RGB LED RGB Light Strip 12V	N/A	https://www.amazon.co	1 Roll	$15
Amazon	ELEGOO Ultrasound Sensor	HC-SR04	https://www.amazon.co	1 Set	$9.99
				TOTAL:	$107
Miscellaneous	Various PCB quotes, electronic components, and mechanisms			Unknown	$400
Amazon	Arduino Due with Headers	N/A	https://www.amazon.co	1 unit	$55.03
Amazon	Metal Gear Servo Motor	N/A	https://www.amazon.co	1 package of 6	$30.20
Amazon	Micro Servo Motor	N/A	https://www.amazon.co	1 package of 10	$21.45
Amazon	CASOMAN Double Sided Tool Organizer	N/A	https://www.amazon.co	3 unit	$71.16
Amazon	Valefod 8-36V to 12V buck module	N/A	https://www.amazon.co	2 unit	$64.72
McMaster Carr	1-4" 20 Plastic Knurled Thumb Screws	91185A394	https://www.mcmaster.c	1 pack	$8.60
				TOTAL:	$251.16
JLCPCB	Custom PCB	N/A	N/A	5	$39.61
AutoDesk	Fusion360 License	N/A	https://www.autodesk.c	1	$56.00
Amazon	9V, 1A, Arduino Compatible Power Supply	N/A	https://www.amazon.co	3	$24.24
Amazon	5V 4A AC Power Supply for NVIDIA Jetson Nano	N/A	https://www.amazon.co	2	$36.64
Harbor Freight T	SAE & Metric Tap and Die Set, 60 Piece	N/A	https://www.harborfreig	1	$44.99
Amazon	Bluetooth Label Maker	N/A	https://www.amazon.co	1	$25.89
				TOTAL:	$227.37
Amazon	GEZICHTA 30W 120V Electric Vacuum Solder Sucker Iron Tool Desoldering Pu	N/A	https://www.amazon.co	1	$33.29
Digi-Key	M20-7831046 20 Position Receptacle Connector	952-1837-ND	https://www.digikey.com	10	$20.29
Digi-Key	1N4733A-T50A Zener Diode	488-1N4733	https://www.digikey.com	20	$5.96
				TOTAL:	$59.54
Amazon	Ratchet Crimping Tool Kit with Connectors	N/A	https://www.amazon.co	1	$30.20
Amazon	4 pack of brushless, USB, 5V DC cooling fans	N/A	https://www.amazon.co	1	$10.78
Amazon	LiPo Battery Charger	N/A	https://www.amazon.co	1	$61.49
				TOTAL:	$102.47
				Electric TOTAL:	$915
				Mechanic TOTAI	$233.24
				GRAND TOTAL:	$1,148.02

Figure 27: The Bill of Materials

C.3 Schedule

We make use of an online, free, Gantt software accessible to anyone. A screenshot of only a portion of our schedule can be seen below. Our project is split into mechanical, electrical, and computer sections, each with 'sub-projects' which have their own schedule proscribed with dates, information, background, etc.

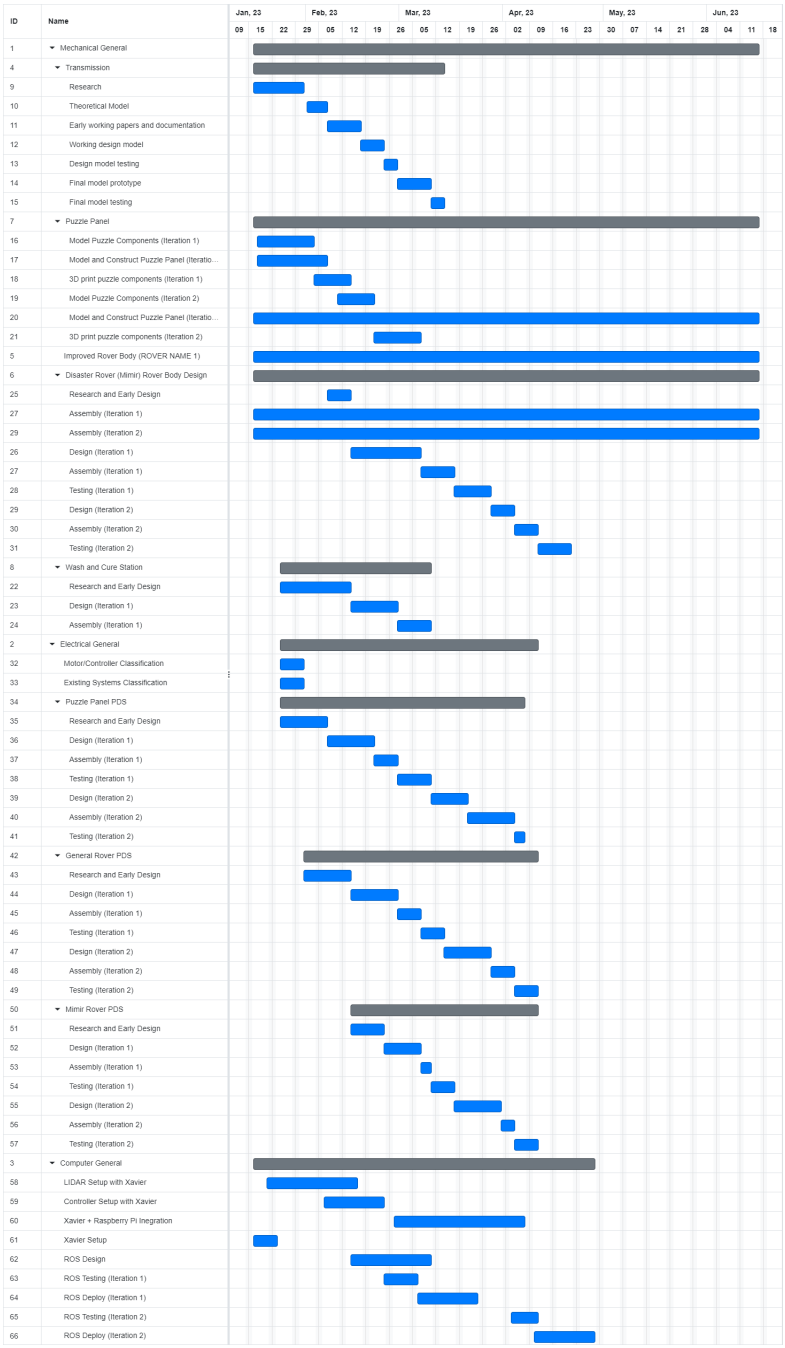

Figure 28: Sample of the Gantt Chart

Compiled Resources

Throughout the project's planning and build phases, daily scratch notes, capstone meeting minutes, project notes, tutorials, and finally our megadoc were compiled into two locations: our Github HackRover server (not HackRover Robotics, our defunct server) and "Complete Drive" folder on Google Drive. The Google Drive contains our overarching goal, methodology, direction, and various information on components like datasheets. Previous capstone notes are also located there. Our Github is our dumping ground for code-related files. This ranges from C++, Python, LaTeX, and its variants, which are separated out into Puzzle Panel, LaTeX, and Rover repositories.

D Concept Development

D.1 Technology Mapping, Risk Analysis, and Feasibility

The minimum product we'll create is a disaster rover, for which we have all the needed components. Functionality will be ensured with the software created to operate well-planned electromechanical systems. Usability will be ensured with effective documentation and simple controls. A well ordered rover with ease of control and robust build will be necessary and capable for further business; though, our client is the UW, so we have little concern for how this would sell.

The disaster rover would need to traverse rugged terrain. A compact frame with a well-set center of gravity and tank treads would allow it to path in any environment. Falls or tumbles would be mitigated by a 'domed' upper chassis that ensures it always lands on its base. Operators will be able to connect to it easily and wirelessly, (ideally through a browser), and control with easy to understand, video-game based control schemes. All electromechanical systems would interface through robot operating system (ROS) housed on a central, Xavier computer. Focusing on electrical and mechanical systems independently: electronics will convey clean, modular power to various components through LIPO batteries. Clean and modular, I stress again, are the focus; devices like the Xavier have stringent voltage and current regulations. Modularity is desired, as future iterations will see different parts added or subtracted; ideally the only aspects affected here are battery life. Everything is contained on a central PCB, improving physical efficiency further. The mechanical systems need to be robust and sourceable. Through the use of an aluminum frame, high end FDM and resin plastics, and (hopefully) casting, we'll be able to create a protective housing for all electrical and computer components. Motors will be mounted directly to the frame with little (or no) geared transmission, depending on the speed and torque output of what motors exist. A small frame is preferred to minimize stresses. Further research into the mechanical engineering of motion mechanisms is needed before a better description can be provided.

The greatest risks are addressed in order: computer, electrical, and mechanical. The computer model only addresses the function of interdependent components in extremely ideal conditions. A disaster environment won't be perfect, and the rover needs to get itself out of sticky situations if communication is lost or electromechanical systems aren't behaving perfectly. The electrical system could fail to provide stable enough current or voltage, or locales of high power draw could result in heating that leads to failure. It's possible the arrangement of components doesn't permit a long enough battery, or results in excess inefficiencies. The mechanical systems may fail about shafts or joints, and the production of gears (as they'll be printed) may be outside of tolerance.

To address the stated risks we need effective testing and redundant systems. Computer models need to simulate circumstances where power delivery or mechanical failure causes issues. Further, boot scripts and backup communication protocols need to exist. Electrical systems have to be tested under every conceivable driven condition, with a focus on clean power. Mechanical systems need to be simulated, constructed with high factors of safety, and modeled physically with test rigs.

D.2 Component Technology Research

Rover To address environmental constraints imposed by the context of disaster relief, our rover will have a rubble-resistant chassis design, and encased, centralized electrical PDS. We will also try to achieve modularity mechanically, electrically, and software wise. To achieve mechanical modularity, the drivetrain will be constructed of stock aluminum parts, and the housing will see a tiered system with adjustable mounting and plenty of open space. In addition to these two major rover aspects, we will have a rotational servo actuated Stewart platform that controls our LiDAR camera; all other components on it will be 3D printed.

To achieve electrical modularity the battery connector will be common, connecting into two buck converters. These buck converters are responsible for stepping down voltage to power logical (5V) and power components (12V). The unique property here is any battery with the same connector may be used. The PCB we designed itself is the PDS, providing power to all logical and actuative components. On the PCB we will add extra ports and adjustable buck modules for unanticipated, future modifications, or alternative designs. Though our rover only uses two motor controllers, the PCB will be designed with four so it may be repurposed for other designs.

To achieve software modularity we will be developing our architecture in ROS. Each component, interaction, data processing procedure, etc can be represented as a modular 'node' that can be added or removed from the final build and launch script.

D.2.1 EE Circuit Models

The current Electrical Circuit Models are made to power at any voltage level, providing different modular of power that can power different hardware. Each system will both utilize the same LM2596 Buck Converters, but the Rover PDS will include multitude of components. With this in mind, the more components there are in an electrical system, the more risks of electrical hardware failure can occur.

Figure 29: LM2596 Buck Converters

- The LM2596 buck converters are designed to step down the voltage that is needed by the electrical components to be able to functionally run. And it's adjusted by a potentiometer that controls the voltage level output.

Risk identification

Rover PCB

For the Rover PCB and the following components that will be used in the Rover are listed here. And also, the risks involved that may entail with solutions that can help mitigate those risks for both electrical systems.

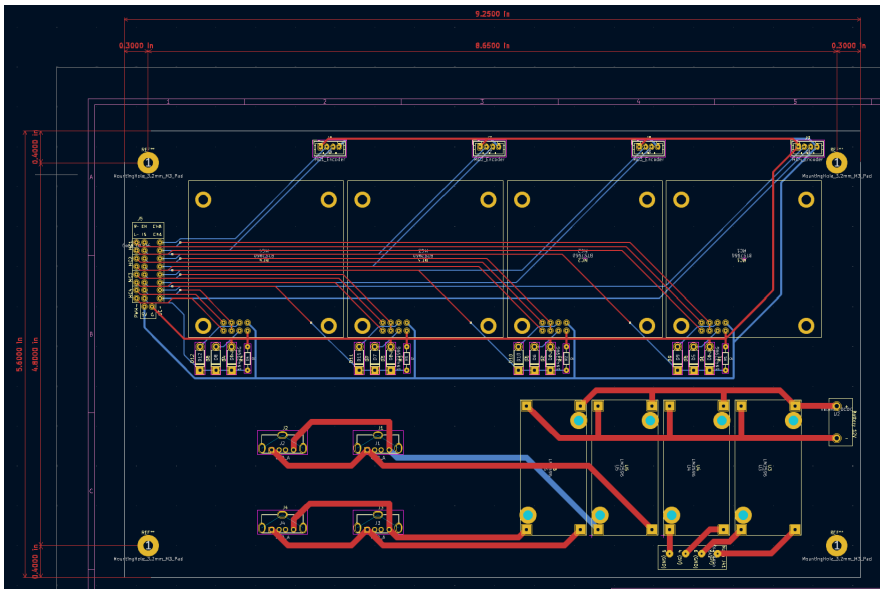

Figure 30: Power Distribution System of Rover

- Shown above is the power distribution system PCB of the rover that is designed to distribute power to every electrical hardware from a single power source or battery. And as well as, provide the signal connections for the hardware as they are essential in controlling the functionality of every component.
- Raspberry Pi is a microcontroller that is designed to send command signals from an encoded software to the motor controllers that control the drivetrain to the motors.

Figure 31: Raspberry Pi

Figure 32: Jetson Nano

Figure 33: 12V DC-DC Buck Converters

- The Jetson Nano is a small/micro computer that is designed to run multiple neural networks like image processing, object detection, etc.
- The 12V DC-DC buck converter is used to step down any voltage larger than 12V down to its assigned level of 12V.

Figure 34: LM2596 Buck Converters

- The LM2596 buck converters are designed to step down or drop down the voltage that is needed by the electrical components to be able to functionally run. And it's adjusted by a potentiometer that controls the voltage level output.

Figure 35: BTS7960 Motor Controllers

- The motor controller is used to receive signals from the Raspberry Pi and delivers the resulting information to the actual motors of the drivetrain.
- The 12V planetary motors are used to drive the whole rover which are controlled by the BTS7960 motor controller through digital signals from the Raspberry Pi.

Figure 36: 12V Planetary Gear Motors

Risk identification

- One of the major risks or dangers for the electrical system is the probability of overheating which will cause a fire hazard. This will cause permanent damage to many of the components on the rover. The battery and the components that needed the most power are at risk of this danger, so sufficient protection for these parts will lower the risk of this danger.
- The risk of the motors running too much or insufficient torque will cause different amounts of currents to be drawn from the battery. This current spike will cause damage to the electrical hardware that is directly connected to the battery and cause a fire hazard. The mechanical components will also face damage as a result.

The electrical circuit or model will be tested with the mechanical models, and the testing will involve making sure connections are stable and all electrical components are functioning correctly and correspondingly communicate with each other.

Future testing of electrical components and their functions:

- The connections between components
- PCB functionality
- Jetson Nano connection
- Raspberry Pi and Drivetrain connection
- Battery and power/current performance

The main risk of the electrical circuit or model is that the electrical connection between any electrical components has failed. The risk mitigation can involve:

- Rewiring/replacing wires, checking every connection between every electrical component.
- Replacing old/faulty components with new ones can decrease the risk of circuit failure.

Another risk of the electrical circuit is the uneven distribution of power to any electrical component through voltage/current spikes and voltage/current noise. The risk mitigation can involve:

- Adding new electrical components that are used to mitigate or decrease voltage/current spikes.
- Snubber diodes, bypass capacitors, transistors, and any other component that can prevent undesirable effects.
- Adding fuses can be used as a fail-safe to prevent the failure of the entire electrical system.

Another risk is any wire disconnecting during a test run which can cause an open circuit with the current not flowing in the intended direction and flowing towards volatile electrical components. The risk mitigation can involve:

- Adding new electrical components that are used to block current.
- Blocking diodes and transistors can protect volatile components by preventing or blocking current from flowing backwards to a vulnerable component.

Taking consideration of these risks, we are able to update the PCB design to include these essential components that mitigate the risks. Diodes are placed in the PCB to protect volatile hardware like the motor controllers from voltage spikes. And also, blocking diodes are used to block any current that is going to the opposite way.

Protecting the hardware components that are mentioned and listed above, would ensure that the rover will be able to drive around in normal conditions without any electrical difficulty.

The components and the wiring shown in figure 37, includes the potential wiring that will take place when all of the components are included. This is the early stage of the assembly as we placed the PCB to fit into the housing that the mechanical team made. This would allow us to figure out how to wire the entire rover without making any final decisions from the beginning.

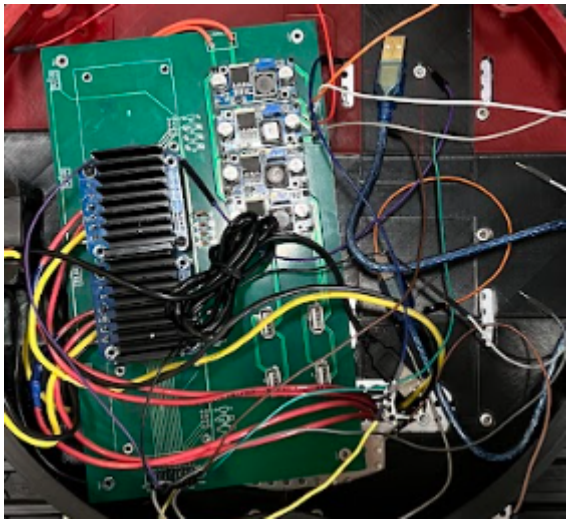

Figure 37: Early Electrical System of Rover

Figure 38 shows a much later or up to date assembly of the rover with the PCB connected and all of the components wired as well. This includes adding the LiDAR camera which eventually will be inserted into the Stewart platform in the later stages of the assembly. But the test of the electrical system is essential to make sure everything both hardware and software work together.

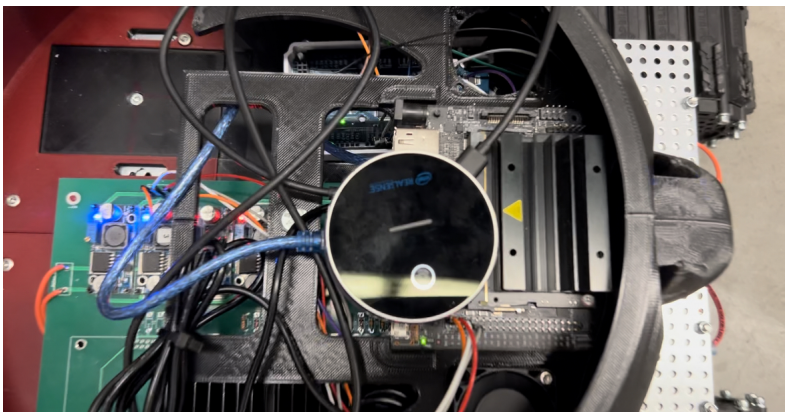

Figure 38: Latest Electrical System of Rover

Stewart Platform

The Stewart platform houses and controls the motion of the LiDAR. Early designs of the electronics are presented below. The first iteration we created was simply proof of concept, and to prototype the control software. It used a GikFun solderable breadboard as the base for our power electronics and signal electronics, and all components. An early mockup of the electrical schematics can be seen in figure 39. We do have plans to have the control board be professionally printed, or seek out a third party servo controller, if possible.

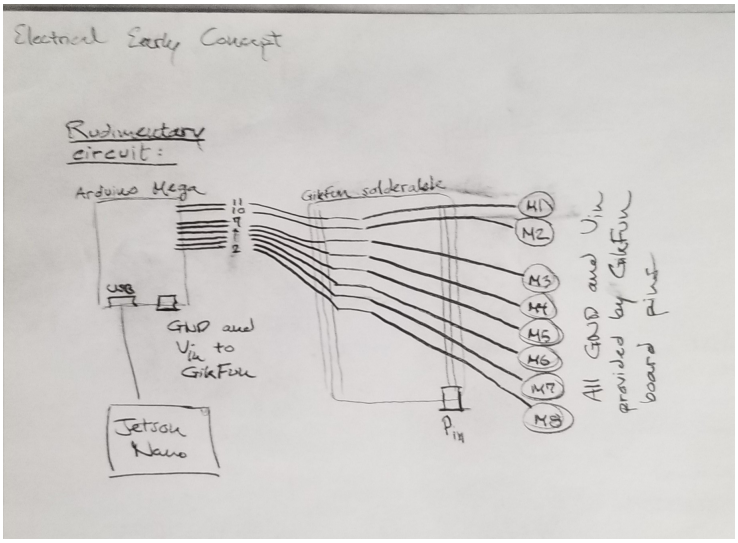

Figure 39: Rudimentary electrical sketches

The Stewart platform was controlled by an Arduino Due. The Due is effectively a beefier version of the Arduino Mega: it has the same footprint but faster processing speed and bigger memory. The math model-based script we're using to forward angle inputs to Stewart platform actuation is extremely intricate and process intensive. A visual of the Due can be see in figure 40

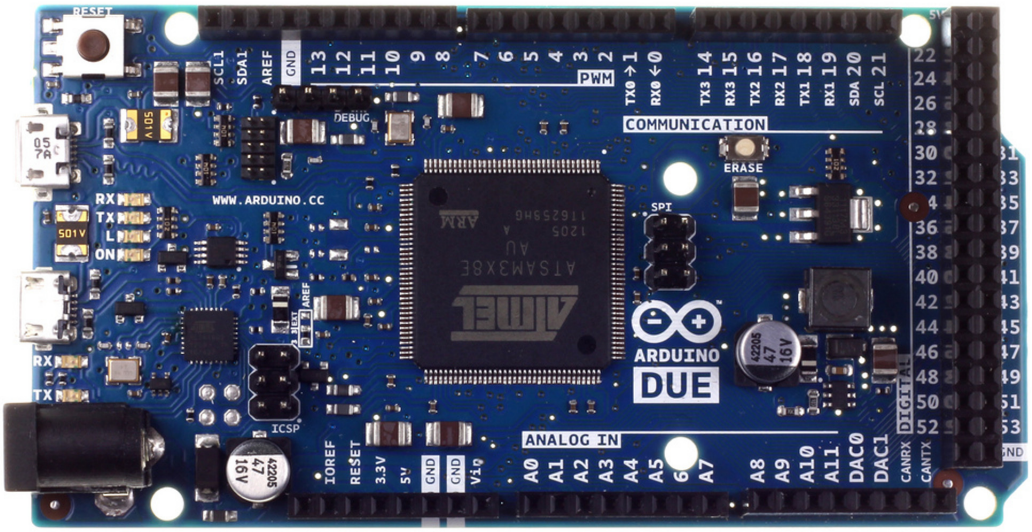

Figure 40: A top down of the Arduino Due

D.2.2 Mechanical Models

Mechanical modeling was completed incrementally and with lots of thought on a day by day basis. Plenty of notes were taken, and it would be a waste to ignore them. That being said, content must be streamlined in the MegaDoc presentation. I present various models throughout all the stages.

Rover

The rover can be split into its drive base and component housing. Seen in figure 15 is the drivebase without treads, assembled in Fusion360. The drivebase was adopted from the previous year's design, so details of component selection aren't extremely relevant here. Motor were changed, however, as analysis of performance revealed last year's team made poor selection resulting in low efficiency; this analysis is

elaborated upon in the methodology section. We swapped out the 612RPM motors for 118RPM motors, seen in figure 41; you might believe the speed would drastically decrease, but the 612RPM motors were being run at such poor performance they were already 1/12th their quoted maximum. The 118RPM motors were significantly more efficient and roughly the same speed.

Figure 41: The motor used in the rover

The component housing was design per our specifications and mention before. We have three tiers: bottom, middle, and upper. The bottom tier houses the PDS, and microcontrollers for the motors and Stewart platform; see this tier in figure 15. The middle tier houses the Jetson Nano - the master microcomputer running ROS - and the 3-in-1 networking device; see this tier in figure 16. The top tier houses the Stewart platform, it being discussed in a later section; see this tier also in figure16.

Stewart Platform

The Stewart platform houses and controls the motion of the LiDAR. We're using the same LiDAR as years prior, so we don't feel the need to discuss it in detail or present images; it's a Intel RealSense LiDAR L515. Early designs of the Stewart Platform are presented below. The first iteration we created was simply proof of concept, and to prototype the control software.

Early on I decided to print the housing, linkages and ball joints. Most parts were custom designed, but some were 3D models that were sourced from McMasterCarr, then modified, as seen in Figure 43. The printing was completed on both resin and FDM printers.

A final first iteration can be seen in figure 44. It includes and top and side view. This version worked out well for prototyping, but wasn't our final.

In the second iteration we made sure the platform mounted to the rover. We also improved the structure by making the ball joints and linkages permanently connected and printed with a stronger resin. A housing for the LiDAR was created on an FDM printer. The final version can be seen in figure 16.

Risk Identification

Risk was assessed constantly and iteratively as I designed all aspects of this project. For example, linkages were designed with a more expensive resin, but upon later risk analysis determined to be too frail, and strengthened by printing as one piece. As this project is constantly developing, the design isn't complete, and there exists some risks we tried to mitigate.

- Heat Distribution: The motor controllers and microcomputer will generate plenty of heat. The early iterations didn't consider this, but it became a major risk point when tabletop tests showed pretty hot heat fins. It's not a major risk; but, to address it (were it to get worse) vents and airflow tracts were built into the component housing, and 40mm fans were employed to convey heat. See one example in figure 45

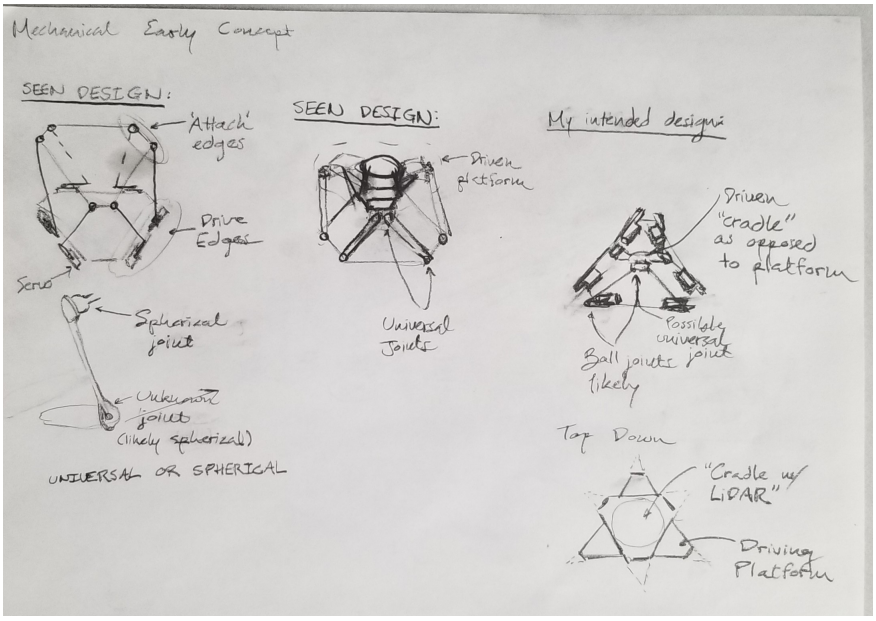

Figure 42: Mechanical sketches seen at the right

- Component Slippage: Housing components is difficult when they can't use threaded fasteners, or their size relative to the chassis is large. As the rover moves there is a fear that components slide, slide and pop out of where they should sit. To mitigate this risk, alignment posts or walls that take advantage of the component's geometry were added. In certain places flexible tabs were also added. It's not perfect, but it should be a point of inspection for the future.

D.2.3 Software Models

The software system of the rover will be centered around using the Robotic Operating System - ROS, which is an open source robotics middleware. The system is designed to live on the NVIDIA computer and a Microcontroller on the rover and consists of multiple sections of code that interact within the ROS environment. The software system is used to provide command and control for the rover.

List of ROS packages (bound to change after testing and possible redesigns):

- LiDAR Package: This package consists of a publisher node for the LIDAR image data and two subscriber nodes for outputting the image data as either an imaging processing node or an image viewing node. The image processing node is able to process the image and send corresponding motion data to the main motion logic package. The image viewing node is used to display the image that is coming out of the publisher node to wherever the output is required live.

- Controller Package: This package consists of a publisher node for the controller data and currently one subscriber node that processes the controller data to be sent to the main motion logic package.

- Motion Logic Package: This package consists of a publisher node that takes motion data from the LIDAR or controller to determine the motion of the rover. The subscriber will be a middle node that connects to the Raspberry PI which is used as the motor controller for the motors on the rover.

Risk Identification

- A major risk is the LIDAR node creating a false LIDAR map of the surrounding area and thus sending false information to the motion logic package. We can mitigate this by adding in additional logic to prioritize using controller motion data compared to the LIDAR mapping data.

- Another risk is the motion logic not having enough logic within it to ensure that the rover does not make false moves that may endanger the rover itself. This risk can be mitigated through edge case testing and ensuring the motion logic corresponds correctly with rover movements.

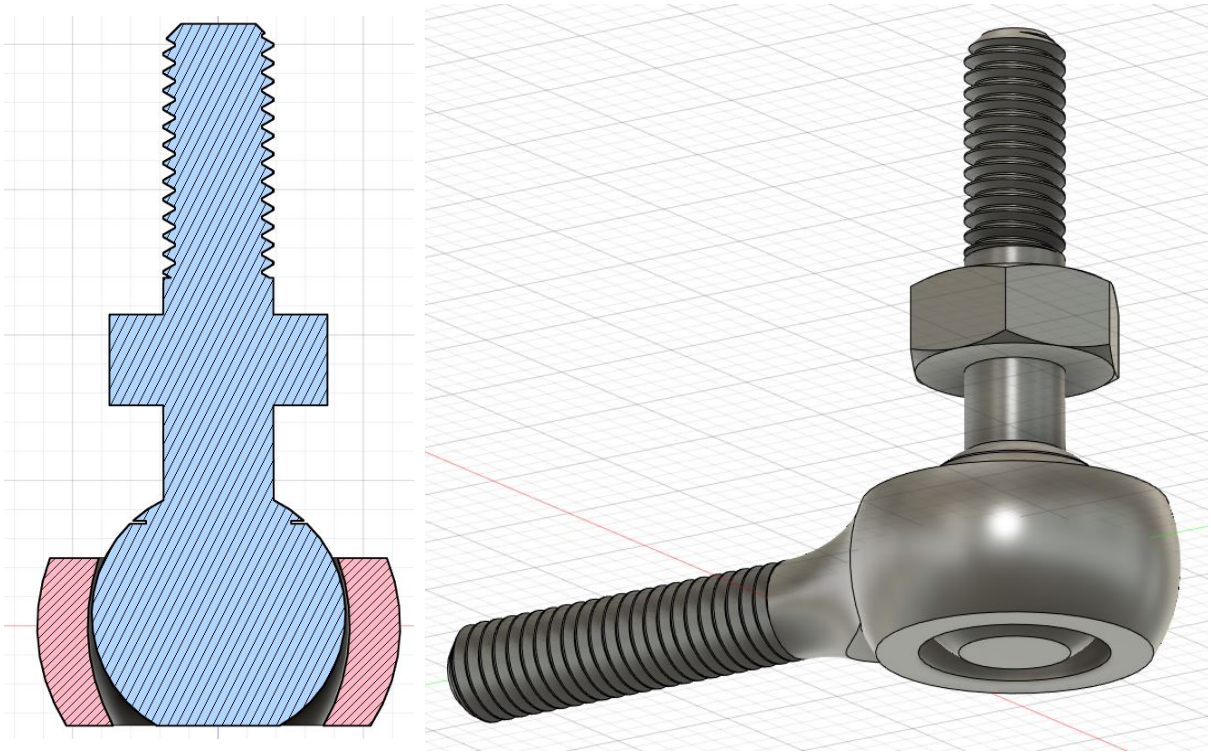

Figure 43: Section of the ball joint showing no interference, next to its 3D model

- The rover system depends on a fully functional user interface for all interactions. If one of the user interface functions does not work correctly, the specific functionality will be affected and not be easily accessed by the user through the application.

D.2.4 Human Interaction Models

The main interaction between the user and the prototype will be through a PC and possibly an Xbox controller which would control the rover. The PC will allow the user to start the rover through the ROS language and be able to test the prototype. The Xbox controller would be used to navigate the rover with joint sticks to do basic operations. And this controller works best for the user to control the rover and wouldn't require them to know ROS. The risk mitigation involves allowing the user to use the software without first-hand knowledge and being able to test the prototype with ease. This would also allow easy control and having it be user-friendly.

D.2.5 System Interface Modeling, Testing, and Risk Analysis

Testing the system interface of the rover, we would ensure that the connections between the interface are functioning accurately. Some of the components which we would be testing are the connection for the LiDAR camera.

The main risk of the system interface would be the failure in the connection between the devices. Mitigating the risk would involve:

- Testing each connection between systems and their interface functionality.
- Testing the LiDAR camera to ensure that the mapping is accurate and readable.

The testing of these systems will be finished around May and more progress will be done as the team continues to design and build the actual systems.

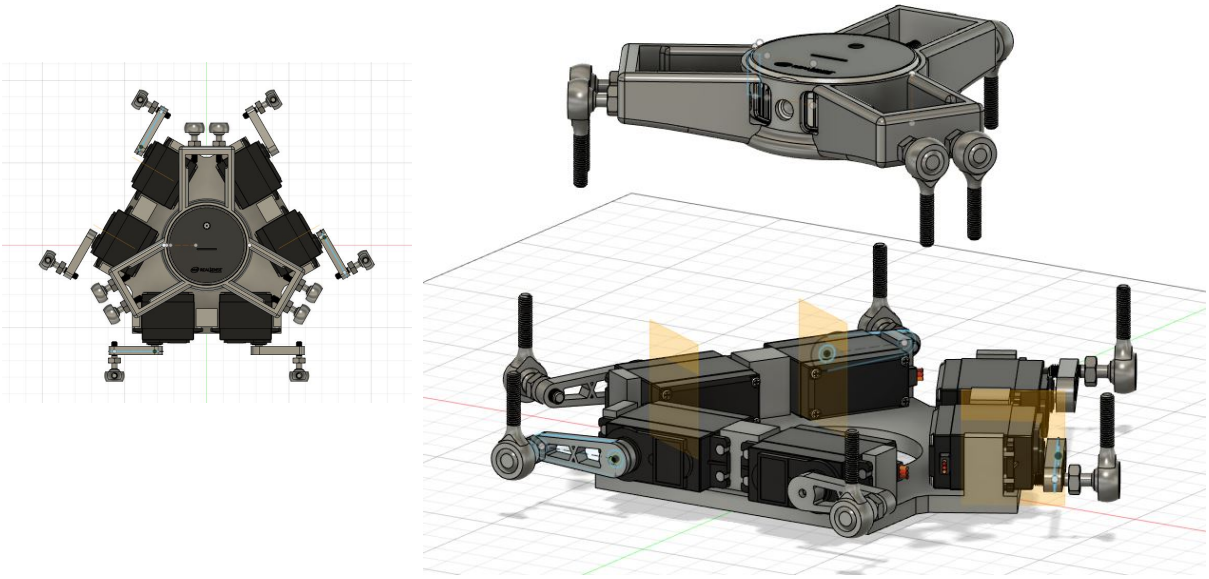

Figure 44: The final first iteration assembly

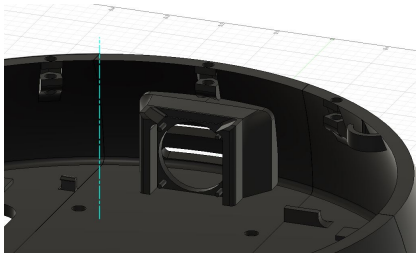

Figure 45: Shot of a fan mount and vents

E Methodology

E.1 Mechanical

Early Motor Selection

The lack of documentation from previous year's teams, and the dearth of UWB mechanical design education makes designing a rover hard or almost impossible. I scraped through websites, forums, and catalogs of books to try and find the best I can on rover design and drive train design.

Since the motor dictates the design we had to get its data sheet. All literature advises over-sizing motors and running them at 10-20% their stall torque in continuous run, split the difference and we should be working at 15%; further analysis will get a more precise value. This makes sense as maximum efficiency occurs roughly at 15% stall torque; we want to minimize battery drain. Doing this will set both the speed and torque output, the latter which will determine the weight of the rover. We don't expect the first iteration to drive up stairs, but when future iterations do, the incline increases net torque experienced; still we hope this won't creep up past 20%. Obvious future changes could be to accept normal operating torque of 10% of stall after reducing weight. The calculations for experienced forces driving up stairs in extreme conditions and on flat terrain still need to be completed at this stage.

The operation of the motor is also intimately related to travel speed and power consumption. We must determine early if the planetary gear motors provided by RobotZone have already considered losses to inefficiencies of gears, and how that will affect power consumption, but not sure this is possible. The rover travel speed is based on the diameter and angular speed of the tank drive sprocket. The angular speed of the tank drive sprocket is the motor's angular speed, which is not actually 612RPM, despite

being named so. Remind yourself that 612RPM is the no-load speed (this particular motor has 10% uncertainty by the way) and this decreases linearly with increasing torque. Operating at 15% torque we should actually expect to see 520RPM (85% speed). I will soon create accurate (enough) motor curves for reference, and calculate exact parameters on torque experienced and maximum weight allowed.

Generating Motor Curves

Motor curves are simple plots of torque vs current, power, speed, and efficiency. Speed is inversely proportional to torque, with the maximum value being the no-load speed (written on the motor) occurring at 0 torque, and the minimum value being 0 at stall torque. We have two potential motors for use, so I'll generate the curves for both: 612RPM and 118RPM motors.

The current has no-load and stall torque values; for our particular 612RPM (or 64.08 rads/sec) these are 0.5A (no-load) and 20A (stall). Mechanical power can be found as the product of torque (Nm) and angular velocity (rads/sec); this is different than electrical power, which is the product of applied voltage and drawn current. Efficiency is the ratio of mechanical power (output) to electrical power (input). Stall torque is 16kgF-cm, or about 1.57Nm.

With the above parameters I used Desmos to plot the motor curve of our selected motor. Torque units are in Newton meters, rotational velocity (red line) units are in rads/sec, mechanical output power (green) are in Watts, current (blue) is in amps. The **efficiency (black) has been multiplied by a factor of 50** for easy visualization; correcting and we see a maximum of 31.25% efficiency at 0.21/1.57 => 13.37% of the stall torque.

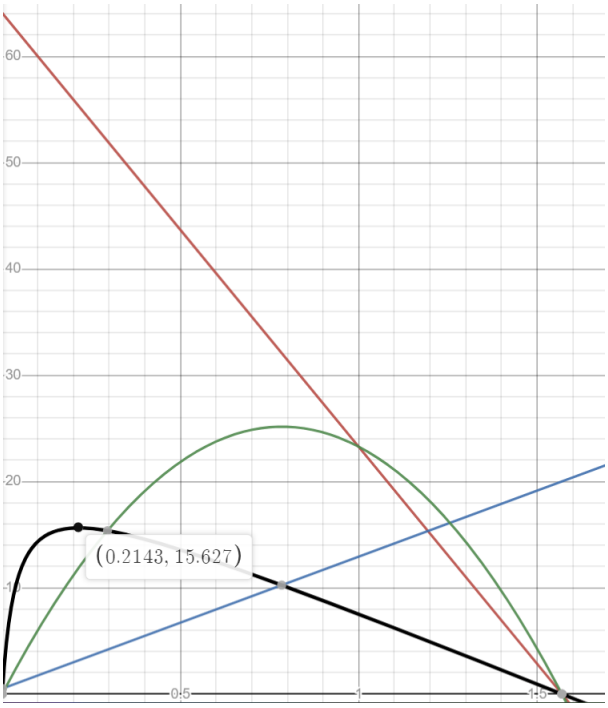

Figure 46: Motor curves with efficiency multiplied by a factor of 50 for scale

For our 118RPM (or 12.35 rads/sec) current values are 0.5A (no-load) and 20A (stall). Stall torque is 69kgF-cm, or about 6.76Nm.

With the above parameters I used Desmos to plot the motor curve of our selected motor. Torque units are in Newton meters, rotational velocity (red line) units are in rads/sec, mechanical output power (green) are in Watts, current (blue) is in amps. The **efficiency (black) has been multiplied by a factor of 20** for easy visualization; correcting and we see a maximum of 26.05% efficiency at 0.927/6.79 = 0.1365 => 13.65% of the stall torque.

Determining torque, maximum allowable size, and speed

See the motor curves above for useful context. When considering the 612RPM motors, we've determined a maximum efficiency of 31.25% if we run the motor at 13.37% of the stall torque, which is about

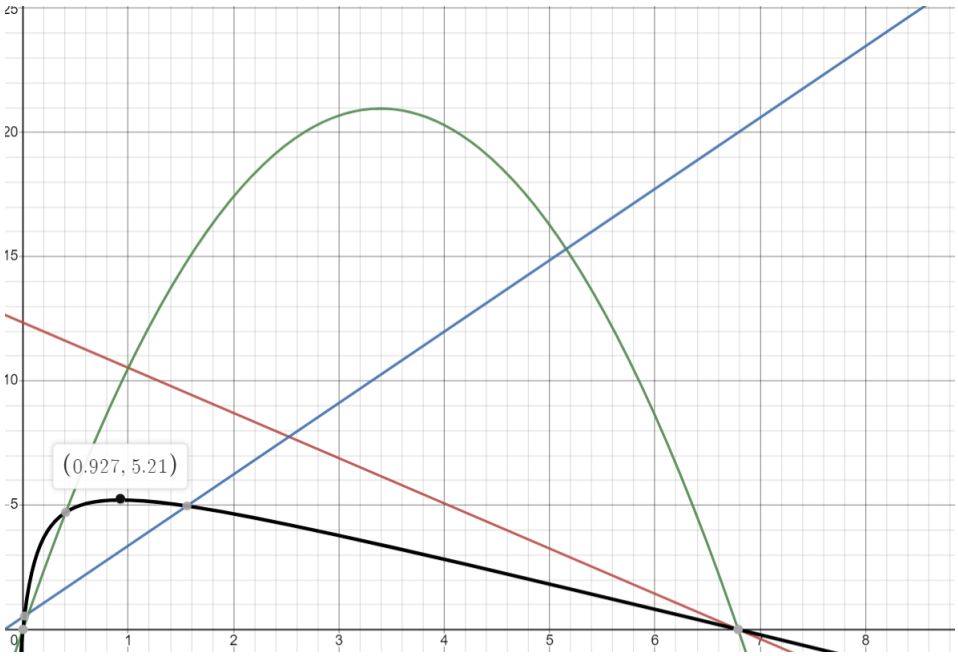

Figure 47: Motor curves with efficiency multiplied by a factor of 20 for scale

0.214Nm.

At 13.37% of the stall torque, we'd be running 86.63% of the no-load speed, or 526RPM, which is 55 rads/sec. The velocity of a tank tread is simply the linear velocity of it's drive sprocket. The current design, which we're building on uses 2400 Series goBUILDA tracks, and a 100mm diameter sprocket; thus linear velocity is $v = 55rad/s * 50mm$, a reasonable 2.75 meters per second, or a little over 8 feet per second.

If we assume we're going to drive this rover entirely flat at the maximum efficiency, our motor should experience 0.214Nm of torque. When moving a single 50mm radius sprocket, the force developed at the contact is 4.28 Newtons. From here things are a bit confusing, as from first glance the rover appears to have two of these motors, but the document (which is admittedly outdated) shows 4. If we have 2 our total force propelling the rover is 8.56 Newtons, 4 and it's 17.l2 Newtons. Move past the motors for a moment and determine what our rover must overcome to move. Consulting some literature, a liberal frictional coefficient for tread based vehicles should be 0.5 the weight; 0.7 will be our conservative. I'm also going to do the calculations for the rubber tank treads we might use in the future at 0.9 (around the value of rubber on dry concrete). The tank treads never roll, so static friction is a constant. The force to overcome is thus

$$F_{app} = \mu mg$$

We can assume that the motor will match this force, and solve for the only unknown (our mass). If we assume two motors outputting 8.56 Newtons: at the liberal friction coefficient estimate our ideal weight should be 1.75kg (3.8lbs), while for the conservative it should be 1.25kg (2.75lbs), and the future rubber treads about 0.97kg (2.13lbs). Obviously this is a massive weight reduction from the current 25 lbs rover, which would cause the motors to deliver 88% its stall torque, an incredible inefficiency. I really want my math to be wrong, but nothing from their MegaDoc explains the decisions they made. If we assume 4 motors and 17.12 Newtons, then the ideal weight should be liberal - 3.5kg (7.9lbs), conservative - 2.5kg (5.5lbs), and future - 1.94kg (4.26lbs); again this is a huge difference from the realistic 25 lbs rover, which would require the motors to deliver 44% its stall torque. Remember, our goal is 13%... Trying to read back through their document I see mention of a speed test, but they placed the data into an image that got corrupted and never submitted with the PDF. There is one mention I saw of 1.1m/s, and while I don't think it's the rover they built, it would correspond to about 70% of stall torque, causing me to lean on the side of 2 motors. I think what happened was this last team wanted a faster rover, and mindlessly piggybacked off the year-prior team's decision to use 118RPM motors without considering the torque specifications of the rover. If that rover uses 2 motors instead of 4, it's damn near a miracle it works, and it certainly wouldn't work climbing stairs.

The guaranteed takeaways are: (1) we need to drastically reduce the weight as much as possible to

reach peak efficiency, or (2) we need to swap out the motors for 118RPM motors which will result in improved efficiency, and possibility to climb stairs. Fortunately most of the logical components don't amount to collectively more than a pound, leaving between 4 and 6 pounds for the rest of the chassis, batteries, and Stewart platform.

Another future consideration is performance on stairs. The equation used above was flat only, however it would become

$$F_{app} = \mu mgsin\theta + mgcos\theta$$

for traveling up an incline of θ degrees. The increase here is appreciable when you consider how steep Discovery Hall's stairs are. Even 4 612RPM motors would be pushing their luck, we'd prefer 4 118RPM motors.

There are two situations that follow from this point: we continue running the two 612RPM motors, or swap out for 118RPM motors. Assuming we go with two 612RPM we see:

- Close to 88% stall torque in continuous operation at **current weight**
- Efficiency of 5% in continuous operation at **current weight**
- Increasing efficiency to 31.25% as we reduce from 25lbs to 3lbs
- Increasing speed to 526RPM and 2.75 meters per second as we reduce from 25lbs to 3lbs.

The current speed given the torque should be 12% of the max speed, or 73.44RPM (7.69 rads/sec), which corresponds to 0.38m/s. Their recorded value of 1m/s corresponds more closely to 30% of the max speed, but it's still hard to believe.

If we choose to swap out the 118RPM motors, at an ideal weight we'll be putting out 13.65% of the stall torque and attaining 26.05% efficiency. For this scenario we have the three friction coefficients: (1) liberal - 0.5, (2) conservative - 0.7, (3) alternative rubber treads for the future - 0.9. A single motor will develop 0.923Nm of torque, and on 50mm radius sprockets, this translates to 18.46Nm of force, for a 2 motor applied force of 36.92N. Ideal weights are thus liberal - 7.53kg (16.6lbs), conservative - 5.37kg (11.8lbs), and alternative - 4.18kg (9.21lbs). In our current scenario we assume 25lbs (11.3kg), resulting in a required force of 55.4N between two motors, or an experienced torque of 1.38Nm, which is pleasing 20.4% of the stall torque, and efficiency of 25.29%. This 25.29% is extremely close to the ideal efficiency of 26.05%. On the other end of the extreme, with rubber treads and 0.9 friction coefficient, we need 2.49Nm of torque, which is an alright 36.7% of the stall torque, and efficiency of 21%. This corresponds to a liberal speed of 0.49m/s, or conservative speed of 0.39m/s.

So what we see if we swap out the 612RPM motors for 118RPM motors is a mild decrease in speed, and massive increase in efficiency. At this current stage it makes the most sense to swap out motors and drop the weight. If we're able to drop the weight significantly we can go back to 612RPM motors.

Testing Speed and Efficiency

Knowing now that we've decided upon 2 118RPM motors, and implemented them, we must conduct tests. As the analysis shows above, for a 25lbs rover we should expect, conservatively 0.39m/s of speed. Conducting a speed test would grant insight into the efficiency and operating conditions of the motor. In figure 48 were the results of our test.

Trial	Distance (meters)	Duration (seconds)	Velocity (m/s)
1	3.8735	15.32	0.2528
2	3.8735	15.31	0.253
3	3.8735	15.29	0.2533
4	3.8735	15.30	0.2532
5	3.8735	15.33	0.2528

Figure 48: Results of our speed tests

From the test the rover is traveling 0.25m/s, this is about 2/3rds our conservative estimate. This is likely because our weight estimate of 25lbs was incorrect. In reality, the weight is nearer to 32lbs.

Fortunately, the underwhelming speed estimate wasn't 1/12th, as it was in past capstone cycles. Another possibility is that the speed was being throttled by the ROS software. Danny mentioned that ROS was setting the speed to a default of '1', and that he may be able to adjust it. It's very possible the software was limiting the speed.

Even assuming a maximum of 0.25m/s, we're operating close enough to our 25.29% efficiency, likely hovering around 20.00%. This isn't amazing, but it does mean the rover can handle trying situations that press towards it's stall torque. Note also that as you press towards the stall torque, current draw increases vigorously. We did one test where we drove the rover into a wall to simulate a load on the motors and the increase in current can be seen in Figure 51.

E.2 Electrical

LM2956 Performance

The data that we need to collect involves current and power performance when the puzzle panel power distribution system and the entire rover operates. The duration the batteries can last is inversely proportional to the current draw. We use a special type of ammeter called a clamp meter which allows us to measure current without interrupting the wire. Acquiring the current measurements help us determine the overall performance of the rover, its battery life, and overall efficiency.

In designing the PCBs for the puzzle panel and the rover, we used the industry standard KiCad EDA. The competitors are generally Altium and Eagle CAD which is owned by Autodesk. All three of these programs are used in the PCB manufacturing industry. Critically, however, KiCad is free and open source. Eagle CAD is $70 per month and Altium is $355 per month. KiCad is used by Adafruit who manufacture and distribute Arduino and Raspberry Pi adjacent boards, components, and accessories. KiCad is also used by DigiKey, one of the largest distributors of electronic goods online.

The testing for the results in measuring current performance and voltage leveling accuracy in the puzzle panel power distribution system was a success. Demonstrating the test, we utilized a stationed multimeter to measure and monitor the output voltage. In the demonstration, we tested one buck converter at a time, as it is more time-saving than to test the whole system for current/voltage monitoring. Since testing the whole system in a trial would require a ton of measuring equipment which we don't have at our disposal. In the picture below, figure 49 shows a demonstration of output voltage monitoring.

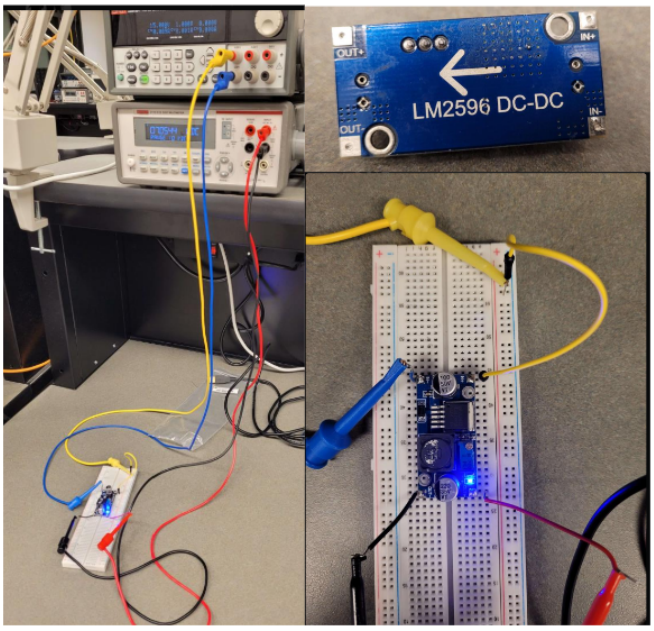

Figure 49: Buck Converter Test

To test this method, use the two pair of wire hooks from the power source and latch them onto the input side of the converter. The positive power connector should be clamped onto the input positive and the negative power to input negative. The same would be applied when using the two pair of wire hooks from the multimeter. Positive hook clamped to positive output, negative hook clamped to negative

output. With this method, we tested every buck converter. There were many that were broken, likely due to overuse from past capstone groups. The ones we found to work, we used for the rover PCB. The table below illustrates the results of one of the buck converters working well with a high input voltage and being able to step it down to any voltage lower than the input. It is also able to output the stepped down voltage which is essential in powering some of the hardware we have that requires low level voltage.

Description: Keeping the constant the same and adjusting the Resistance				
Input Voltage (v) = 12v				
Output Voltage (v)	Current (A)	Input Power (W)	Output Power (W)	Efficiency (%)
5	0.5	3.6	2.5	69.4
5	1	5.4	5	92.6
5	1.5	8.4	7.5	89.3
5	2	11.2	10	89.3

Figure 50: Results of our current/power tests

After getting the buck converters to work, we soldered them onto the rover PCB. We then monitored the power efficiency and delivery of the buck modules to their components. The results showed it was successful. Each buck module was able to output the specified voltage based on the adjustment of their potentiometers. And when powering the hardware, there were no signs of electrical failures. The most number of elements that are connected to the PCB are three and when they operate, they draw 0.3A to 0.5A. With other slots of power and USBs in the PCB that are available, additional hardware can be powered from the PCB. And with the performance of power not showing signs of failure, we can safely assume the rover PCB can handle powering multiple hardware and components at once.

Using the clamp meter, measuring the current/power performance of the rover was successful. We monitored the current levels by placing the clamp around 12-gauge wire connected from the batteries to the 12V DC-DC converters and it's demonstrated in the image below of figure 51.

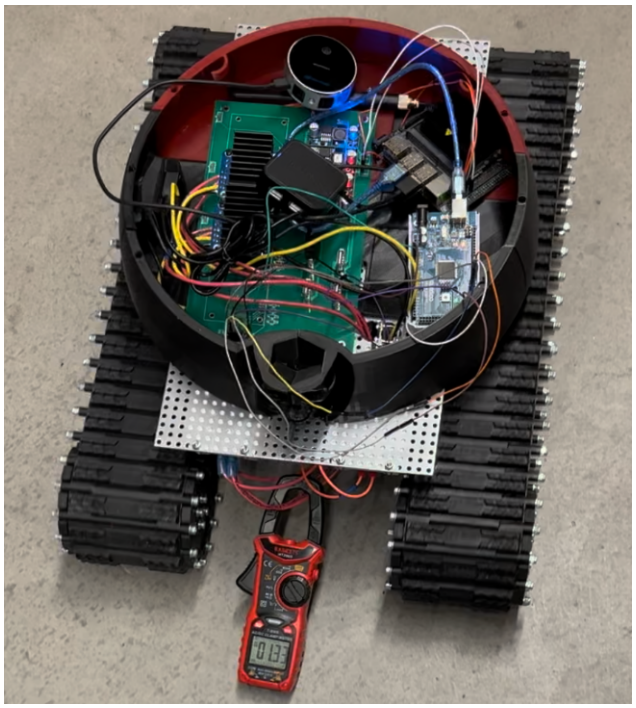

Figure 51: Rover Current Performance Demonstration

The power going into the motor controllers will run at 12V from the DC-DC converters during a

	Current (A)		
Environment	Trial 1	Trial 2	Trial 3
Smooth ground	1.83A	1.74A	2.0A
Dirt ground	2.54A	2.53A	2.70A
Hilly terrain	2.81A	2.92A	2.99A
Wall	3.32A	3.41A	3.32A

Figure 52: Rover Current Test Result

simple operational maneuver. The results are shown in figure 52. Conditions involved the rover riding on flat ground inside a building. And the rover used around 1.7A to 2.0A when operating on flat ground.

The next test involves operating the rover on dirt ground and driving over uneven terrain. Introducing dirt and uneven terrain would require the rover to draw more current since dirt and uneven terrain cause friction and resistant force for the rover to overcome. The results of the tests are successful with the rover operating over dirt in 2.5A to 2.7A and for uneven terrain, around 3.0A was used. The next major test involves moving the rover against the wall, to test the rover when it meets a maximum physical force. The overall current level reached to 3.4A which is almost twice the amount the rover used normally on smooth surfaces. These values help determine how long the rover could perform under each of these environmental conditions. Additionally, there were no electrical hardware failures through each test, which we can conclude that the entire circuit is protected from voltage or current spikes. The current used in these conditions is less than expected so we can safely assume the rover can handle harsher terrain and conditions.

E.3 Software

Learning ROS

In this section we talk about the process that occurred for the software team to learn, understand, and eventually utilize ROS. This year there was no single member of the software team that had any experience utilizing ROS, nor robotics in general. This created a steep learning curve for the individuals of the software team, and a majority of time was spent working on learning ROS and how it interacts with components. After studying ROS books and guides on the internet, the team chose to utilize ROS Noetic on Linux 20.04. This flavor of ROS was chosen due to the completeness of ROS Noetic as it is considered a ROS 1 architecture. This means that a majority of the APIs that are considered for this project had already been created, and would not have to be created by the software team themselves.

Implementation

After deciding on a ROS flavor, the team started to work on creating software and completing tasks. Tasks such as having a way to control the rover, viewing the LIDAR output, and teleoperation for the motors themselves. The team started by creating a website that is hosted on the NVIDIA Jetson Nano that is hosted a ROS node. Initially the team was unsure if Flask was able to become a ROS node since Flask is a general web framework that is written in Python. Through research and testing the team found that ROS Noetic has built in APIs that can interact with Flask code and generate a ROS node as a website. This allowed us to then complete the prior goals in this project. We were able to control the rover through the webserver, since the webserver became interactable to ROS on the Nano itself. The LIDAR live view was another pain point in our process, but through research, testing, and validation we were able to achieve a LIDAR live view that is sent through the image transport ROS pipeline. Meaning that the LIDAR becomes a ROS node and is able to send data through the built in pipeline that ROS builds.

A feature that was added for the LIDAR was having a switch that allowed both live and heat map view for the LIDAR camera output on the webserver. The heatmap is generated using OpenCV which is a Python package for computer vision. Through the OpenCV API we were able to manipulate and generate color maps for the LIDAR. We paired this with Intels provided Python RealSense API that can directly use our own Intel RealSense camera and all of its built in components within the device. We made use of the color and depth stream pipelines from the device to then utilize OpenCV to calculate and

generate heatmaps. Lastly we implemented a distance calculation that was made through built in API functions that provide us with distance based off the heatmap. We tested the distance measurements and were able to accurately read distance measurements from up to 8.2 meters, which was different from the 9 meters that Intel advertises. This may be due to environmental conditions as well as the code setup itself. We tested that the depth calculations for distance is based off of the center of the LIDAR camera itself.

For SLAM, a crucial decision to consider was which type of SLAM algorithm was best suited for rover and hardware resources. There were multiple types of SLAM packages such as gmapping or hector slam. The unfortunate part was with the short timeline; large amount of custom scripts required; and potentially incompatible hardware rendered many of the choices unfeasible. One of the choices that was reasonable was a Solid-State LiDAR (SSL) SLAM by H. Wang, C. Wang, and L. Xie. Their SLAM used the same L515 LiDAR that we are using and was suppose to be lightweight. This SLAM was a promising choice since the codebase and hardware were already established, which was feasible in our short time schedule, won't require excessive modifications, and was confirmed to be compatible with one of our existing hardware.

Rover Setup and Test

When the rover was built, we placed the NVIDIA Jetson Nano into the rover and began testing that all the components that was previously mentioned is able to interact with a physical rover. Prior to the physical rover being built, bench top test were done with a Arduino microcontroller, motor controllers, and one of the rover motors. This allowed us to test that we were able to control the motors through the Jetson Nano to the microcontroller to the motor controllers, and finally to the motors themselves. This also created an opportunity for the electrical team to record current data that the motors pull from the power supplies.

After the rover was built, we booted the system up with a non-headless display, meaning that we had to plug in a monitor, keyboard, and mouse to the Nano to set up the ROS environment. This became a hassle, therefore the software team chose to create a quick boot up launch script to automatically deploy the rover and set up the environment. Testing was done to ensure that the boot up time had not changed for the rover. We found that the average boot up time prior to the launch script was 17.8 seconds. After implementing the launch script, we found the average boot up time to be 30.5 seconds. Which was expected as the launch script contains sleep contains that add approximately 15 seconds of run time to boot up, allowing all environmental components to start up completely prior to completing boot up.

Invoking SLAM was straightforward since it involves three command lines on Terminal. With visualizing a point cloud, the main metric of success was being able to recognize the rendered object in real-time. As soon as Rviz visualizer opens and a point cloud begins rendering, a reasonable timeframe for the definition of real-time was being able to recognize the rendered object within 20 seconds. This can be further optimized to reduce the rendering time as one of the future projects. One of the obvious places to test SLAM was the mechanical engineering capstone room since this was the main location of conceptualizing and building the rover. Additionally, the capstone room was contained, full of obstacles, and reasonably small enough to render the room but large enough to investigate its range.

F Results and Discussion

F.1 Electrical

The electrical team was able to successfully design and have manufactured a main PCB that will both power the rover and control the main motors. During the assembly of the rover, many jumper wires and gauge wires were utilized in a tight housing. But as soon as the PDS is powered, all hardware connected were turned on. And this gives the software to be able to communicate with the digital system through the neural network. Also, when the rover was driving over distances, it showed no sign of hardware or electrical failure. Additionally, during an operation, the LiDAR camera is able to capture live footage which means it had enough power to operate. As a result, the rover PDS is able to provide enough power to variety of different hardware and provide clean digital communication paths for data transfer.

F.2 Software

The software team was able to conclude that all goals determined in the beginning of the project to be met by the software team were completed. Control, data view, and modularity goals for the rover to be operational had been all completed through this project by the software team.

SLAM integration with our hardware was a success since it was able to visualize recognizable objects within a reasonable real time of under 20 seconds. 53 is included for clarity of results.

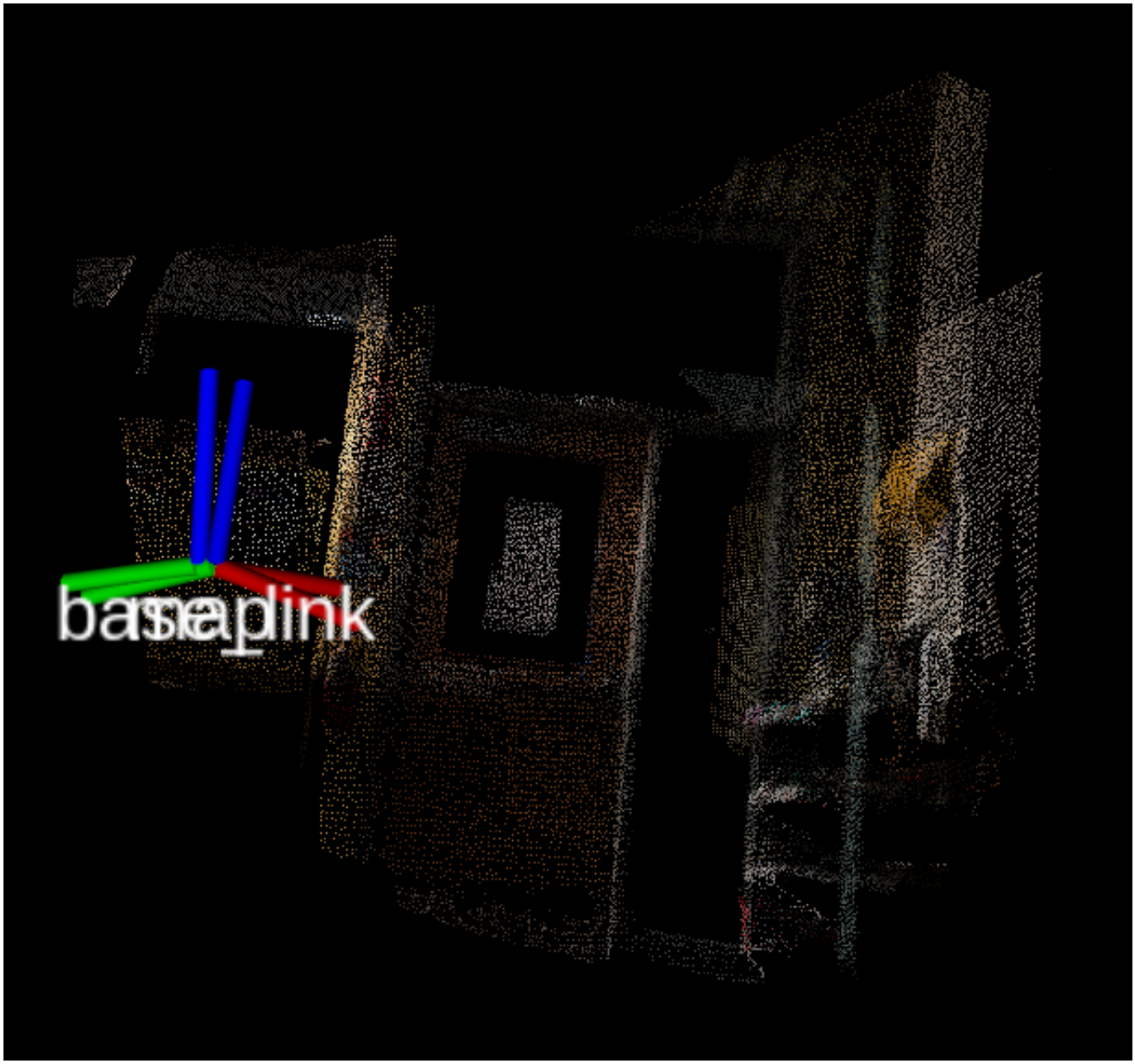

Figure 53: Render of Capstone Room Door

G Summary, Conclusion, and Future Perspectives

In kind with ordering seen in the document so far, we'll summarize our achievements then provide future perspectives.

- **Organization and Competition:** A well-ordered, public GitHub containing all our documentation, software, etc for future capstone groups and future teams was created. We gained more attention within the school, increased our size, and diversified the intelligence of our members.

The competition was made more concrete with better understanding of its proceedings, requirements, and scoring. Work must be done towards the advertising, aesthetic, and flare of the competition, such as through promotional videos, posters, websites, etc.

- **Rover (Mechanical):** The first iteration of only one physical rover was created. A drive train with two 118RPM motors navigated a housing containing a PDS, Stewart platform (for camera control), microcontrollers for controlling the motors and Stewart platform, a Jetson Nano microcomputer for high level control, and a networking device for interfacing all components within the rover to each other and the microcomputer with the outside world. Two iterations of the Stewart platform - the second mounting to the rover - were created. Effective documentation was created for everything.
- **Rover (Electrical):** The second iteration of the rover PCB was created, it interfaced the supply of power to all driven and logical components. The PCB interfaced motor controllers (which extended to the motors), any stray component that needed USB-based power, and the Raspberry Pi which sent motor commands. The power distribution for the Stewart platform was designed and constructed of solderable breadboard.
- **Rover (Software):** A software architecture in ROS was created, with basic function for the command GUI, LiDAR feedback, teleoperation, and network connectivity. SLAM integration allows the rover to formally process and visualize its environment and opens up to a greater number of possible interactions that utilize its computer vision.

HackRover and HARC is far from completion, and we must press on (ideally with increasing speed). The rover - too - needs major developments if we're ever going to prototype the event, or act as a authoritative source of information for robotic development. Below is a list of the most pressing future developments.

- **Organization and Competition:** The GitHub structure needs to transform into something of a 'template' other teams would follow (required for competition participation). We must officially constitute the organization with clear leadership, expectations and methods for membership, and channels of communication.

  The competition needs proceedings, requirements, and scoring formalized with written documents and exact metrics. We also need to improve outreach, namely outside of this campus, so we can legitimately compete. The Hacking aspect of the competition needs to have well documented vulnerabilities, how to implement them, how to perform hacks, and how to defend.
- **Rover (Mechanical):** Focusing first on the rover, either the weight needs to come down (to improve speed and efficiency) or more motors need to be added. The internal component mounting and wire routing needs to be improved for safety (avoid melting wires) and construction ease (cable access). A note, mostly the concern of electrical, is that the PDS needs to be reshapen to fit nicely into the chassis. Analysis and improvement of airflow should be conducted and improved in any way possible. The placement of the Stewart platform should be improved, as to lower it and make it more compact. Overall the fit of any components should be improved as best as possible.

  Additional rovers need to be constructed, as to prototype the competition. Thought should be placed towards a electromechanical, human controlled gear train for taking advantage of torque-speed transmission. Stair climbing and improved tank tread design is also a high priority.

  The Stewart platform will be a beast of its own. Research into best construction methods, and a plan for improved arrangement and manufacture needs to be developed. The linkages need to be better designed for support and weight; mounting of the servos must be improved.
- **Rover (Electrical):** Ideally, the second PCB design will be the last iteration of the PCB which controls the rover.
- **Rover (Software):** The architecture in ROS needs to be constantly developed with the addition of function-enhancing nodes. The most important nodes are teleoperation, modularized for a variety of control schema; and autonomy, driving based on a model from the input of multiple nodes. Networking with the use of the 3-in-1 Linksys device should be integrated, whether with ROS or without. The Arduino code controlling the Stewart platform must be improved, especially as to integrate it with the Jetson Nano and ROS. With SLAM's formal integration, there's a greater amount of potential projects, such as forming an optimized path planning algorithm; optimizing the

computer vision to produce greater imagery and reducing point cloud capturing/publishing latency; object detection for object avoidance; linking motor controls and navigation algorithms together to produce the first-iteration of self-driving autonomy; saving the point cloud map within a rosbag in order to retain a larger map; or possibly even coordinating arm control and the established computer vision. The current development of the control system is a simpler form of the intended design. The full proposed idea included the Control Selector service, so multiple different devices such as an Xbox controller and keyboard can also interface with the control systems appropriately. Additionally, they should allow multiple controllers to concurrently or simultaneously move the Stewart platform and drive system. 54 includes the service that oversees the other controller nodes.

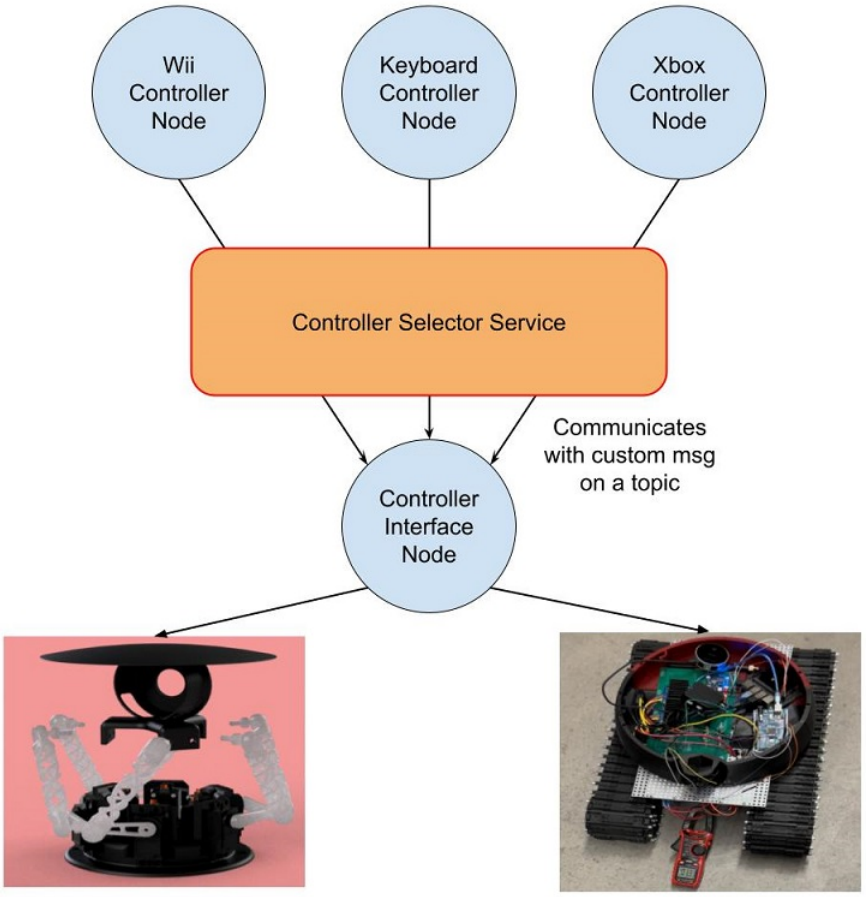

Figure 54: Full Proposed Control Software Architecture

H Acknowledgements

We would like to express our thanks to the Capstone Advisor and Sponsor, Professor Pierre Mourad for providing the guidance, support and flexibility in completing this project. And also, to members of the HackRover Club for providing their valuable participation and involvement in not only learning the knowledge to help further their educational career but completing the complex tasks of the project.

We would like to thank Professor Imen Hannachi for putting whole the team together to accomplish this project and as well as providing the tools to complete the write-ups necessary for documentation.

We are also thankful to the Mechanical Lab Coordinator Jacob Roth for providing us the equipment to put together our project and for coordinating our purchases of needed components to complete the project.

We would like to thank Professor Harry Aintablian for providing the valuable insight and feedback to the early stages of development for the electrical systems.

I Bibliography

- Technology, Electrical. *"Buck Converter - Circuit, Design, Operation and Examples."* ELECTRICAL TECHNOLOGY, 29 Sept. 2022, www.electricaltechnology.org/2020/09/buck-converter.html.
- Shojaei, Amir & nbsp; M. *"Interfacing BTS7960 43A Motor Driver Module with Arduino: [Step-by-Step]."* Interfacing BTS7960 43A High Power Motor Driver Module with Arduino, Electropeak, 24 May 2023, electropeak.com/learn/interfacing-bts7960-43a-high-power-motor-driver-module-with-arduino/. Accessed 09 June 2023.
- Data sheet for our motors here
- Ayodele, Abiola. *"How to Design a PCB Layout: A Comprehensive Guide."* Wevolver, 22 May 2023, www.wevolver.com/article/how-to-design-a-pcb-layout-a-comprehensive-guide.
- Franco, Sergio. *Design with Operational Amplifiers and Analog Integrated Circuits.* McGraw-Hill Higher Education, 2015.
- Sedra, Adel S., et al. *Microelectronic Circuits.* Oxford University Press, 2021.
- Brown, Stephen D., and Zvonko G. Vranesic. *Fundamentals of Digital Logic with Verilog Design.* McGraw-Hill Higher Education, 2014.
- A great example which includes a lot of mention of kinematics is seen here
- What might be called a treatise on Stewart platforms is found here
- An instructables on making a ghetto Stewart platform with Servos here
- All robotic devices need kinematics models to work; this article provides a kinematic model for Stewart platforms powered by rotary means (ie servo)
- Another good design example found here
- Though it uses linear actuators, has some good examples of tests that might be performed on Stewart platforms here
- Another good design example
- A good instructables; however, this one uses linear actuators
- A good design example with potentially useful testing procedures
- A lovely design that looks nice and has a nice circular platform
- No tutorial, just great inspiration
- *Parallel Robots* by J.P. Merlet
- *Introduction to Robotics: Analysis, Control, Applications* by Saeed Niku
- *Building Robot Drive Trains* by Dennis Clark
- *Constructing Robot Bases* by Dennis Clark
- *Making Things Move* by Dustyn Roberts
- Infineon Technologies, "High Current PN Half Bridge", BTS7960 datasheet, Mar. 2004 [Revised Dec. 2004]
- H. Wang, C. Wang, L. Xie, "Lightweight 3-D Localization and Mapping for Solid-State LiDAR"

J User Manual

J.1 Electrical User Manual:

IMPORTANT: Make sure no power source or battery is connected to any electrical system you are handling. Disconnect the battery or power away from the system to prevent electrical failures or accidents which may cause fatal injury and major damage to hardware.

Rover - To setup the hardware, we'll start with the power connections before the digital connections.

1. Power connections need both positive and negative polarities to complete a circuit. First, the motor controllers needed positive and ground connections to be powered in B+ and B- slots. The M+ and M- indicate the polarities to connect to the actual planetary motors.

2. Then, connecting the power to the hardware of the Jetson Nano, is by utilizing two separate jumper wires that power the Nano by 5V with two additional wires that are negative in respect to their positive counterparts.

3. The Raspberry Pi will be powered through a USB port that is built into the Power distribution PCB.

4. The LiDAR will receive power from the Nano through USB port in the Nano.

5. Figure 22: shows the programmer the correct pins to assign to each task to ensure proper usage of the main motors.

7. **IMPORTANT**: misplacing a digital wire connection can lead to severe consequences when giving a directional command to the rover. It can lead to hardware failure or operational accidents. So make sure the connection of each wire is correct and secure to prevent a disconnect when the rover operates.

J.2 Software User Manual:

When booting up the Jetson Nano, you want to first connect to the internet. The webserver can not be hosted without an internet connection. The webserver is hosted on the 5000 port of the IP address of the Jetson Nano.

For example: if the IP address of our Jetson Nano may be 172.10.3.20 and the webserver when it is operation will have a URL of 172.10.3.20:5000.

Through this you are able to SSH into the Jetson Nano and preform basic Linux operations and editing.

All the code can be seen in `https://github.com/HackRover/rover` The majority of the code for the rover seen above. In the repository, there is rover/mimir/webserver/ which houses the python server, weblaunch shell script, and the HTML/CSS template for the website. This is a framework for future users to base off their projects on. In the repository there is also rover/mimir/motorcontrol/src/ which houses the Arduino code for the motor controllers that can interact directly with the rover. The file system of the Jetson Nano is similar to how the Mimir Rover GitHub is currently built. The nodes that are created are the cmd velocity, webserver, teleoperation output, serial to Arduino, and LIDAR view pipeline nodes.

The software team found that the most difficult part was setting up the entire environment for the Jetson Nano to have all the APIs necessary to be able to work. After the environment is set up, coding becomes simpler with the APIs provided.

For activating and loading the SLAM package:

1. Figure 56: Open a terminal and run "roscore"
2. Figure 57: Open another terminal and enter "source /opt/ros/noetic/setup.bash"
3. Enter "source ~/catkin_ws/devel/setup.bash"
4. Run "roslaunch ssl_slam ssl_slam_L515.launch"
5. Figure 58: This should open the Rviz visualizer and after ~5 seconds of Rviz opening, the depth camera activates (can be confirmed with little red laser on camera) and maps the first batch of point clouds within 20 seconds.
6. Rviz can be saved to retain the map of the environment.
7. Figure 59: If the camera needs to be configured, an example of the launch file can be modified accordingly.

Webserver

Start with running

```
roscore
```

in a new terminal to start ROS.

Then in a new terminal, navigate to the websever directory in the Jetson Nano and run the shell script

```
sh weblaunch.sh
```

Then in a new terminal launch the ROS serial server:

```
rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 _baud:=115200
```

That should be all the commands neccessary to start the server to interact with ROS and the Arduino as a motor controller. You can also run to preview active ROS topics

```
rostopic list
```

And also run the following to check out the /cmd_vel ROS topic which is where the velocity values of ROS are provided.

```
rostopic echo /cmd_vel
```

Webserver folder contains the website that is hosted on the Jetson nano that displays the real time LIDAR view. Run the shell script to launch the webserver.

The website features a camera view (both real time color and LIDAR heatmap view), current command status, distance measurements (from heatmap only), and control panel to control the rover.

Figure 55: WebServer Tutorial on GithHub

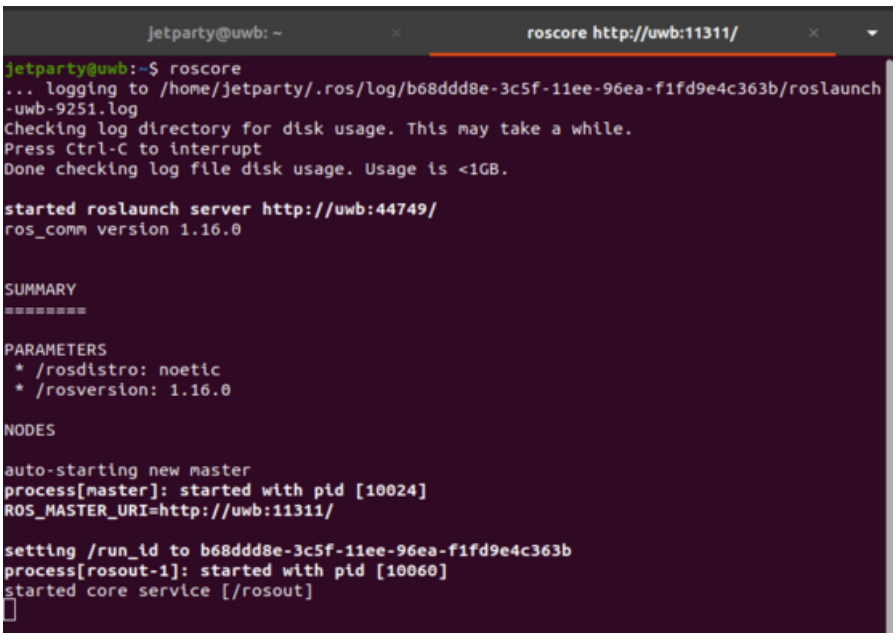

Figure 56: Initialize ROSCORE

```
jetparty@uwb: ~                                    roscore http://uwb:11311/
jetparty@uwb:~$ source /opt/ros/noetic/setup.bash
jetparty@uwb:~$ source ~/catkin_ws/devel/setup.bash
jetparty@uwb:~$ roslaunch ssl_slam ssl_slam_L515.launch
... logging to /home/jetparty/.ros/log/b68ddd8e-3c5f-11ee-96ea-f1fd9e4c363b/roslaunch
-uwb-11164.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://uwb:45183/

SUMMARY
========

PARAMETERS
 * /camera/realsense2_camera/accel_fps: -1
 * /camera/realsense2_camera/accel_frame_id: camera_accel_frame
 * /camera/realsense2_camera/accel_optical_frame_id: camera_accel_opti...
 * /camera/realsense2_camera/align_depth: False
 * /camera/realsense2_camera/aligned_depth_to_color_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_fisheye1_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_fisheye2_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_fisheye_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_infra1_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_infra2_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/allow_no_texture_points: False
 * /camera/realsense2_camera/base_frame_id: camera_link
 * /camera/realsense2_camera/calib_odom_file:
 * /camera/realsense2_camera/clip_distance: -2.0
 * /camera/realsense2_camera/color_fps: 15
```

Figure 57: Command Lines to Start SLAM

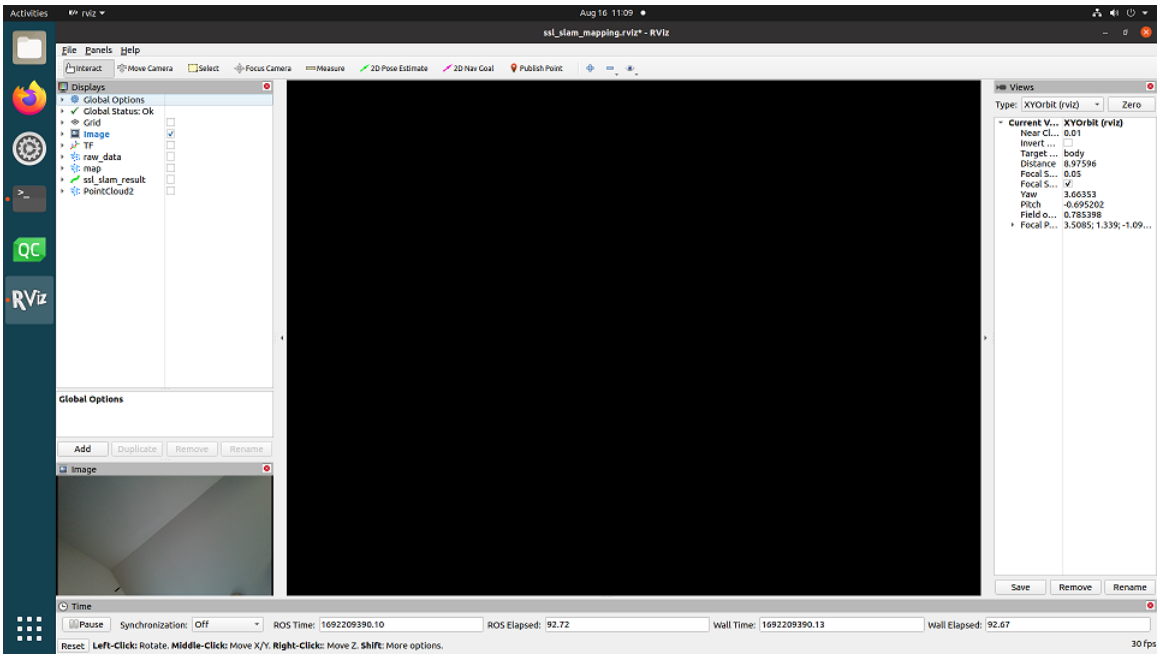

Figure 58: Rviz screen that starts to collect and publish point clouds

```xml
<?xml version="1.0"?>
<launch>

    <include file="$(find realsense2_camera)/launch/rs_camera.launch">
        <arg name="color_width" value="1280" /> <?ignore Originally 1280 ?>
        <arg name="color_height" value="720" /> <?ignore Orginally 720 ?>
        <arg name="color_fps" value="15" /> <?ignore Custom arg added in: original FPS is 30 ?>
        <arg name="filters" value="pointcloud" />

        <?ignore New additions to configure L515 camera for Jetson Nano ?>
        <arg name="depth_width" value="320" />
        <arg name="depth_height" value="240" />
        <arg name="depth_fps" value="30" />
        <arg name="enable_gyro" value="true" />
        <arg name="enable_accel" value="true" />
    </include>

    <param name="scan_period" value="2.0" /> <!-- Originally 0.1, seconds between scans-->
    <param name="vertical_angle" type="double" value="2.0" /> <!-- Originally 2.0 radian of angle-->
    <param name="max_dis" type="double" value="9.0" /> <!-- Originally 9.0, meters for max distance-->
    <param name="map_resolution" type="double" value="0.05" /> <!-- Originally 0.05, space between points in point cloud-->
    <param name="min_dis" type="double" value="0.05" /> <!-- meters for min distance -->


    <node pkg="ssl_slam" type="ssl_slam_laser_processing_node" name="ssl_slam_laser_processing_node" output="log"/>
    <node pkg="ssl_slam" type="ssl_slam_odom_estimation_node" name="ssl_slam_odom_estimation_node" output="log"/>
    <node pkg="ssl_slam" type="ssl_slam_laser_mapping_node" name="ssl_slam_laser_mapping_node" output="log"/>

    <arg name="rviz" default="true" />
    <group if="$(arg rviz)">
        <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz" args="-d $(find ssl_slam)/rviz/ssl_slam_mapping.rviz" />
    </group>

    <!--node pkg="hector_trajectory_server" type="hector_trajectory_server" name="trajectory_server_ssl_slam" ns="ssl_slam" >
        <param name="/target_frame_name" value="map" />
        <param name="/source_frame_name" value="base_link" />
        <param name="/trajectory_update_rate" value="10.0" />
        <param name="/trajectory_publish_rate" value="10.0" />
    </node-->
</launch>
```

Figure 59: Launch file that configures camera for SLAM