WORCESTER POLYTECHNIC INSTITUTE

Demining Autonomous System

A Major Qualifying Project

Submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted By
Malek ElShakhs
Alexander Hagedorn
Troy Howlett
Eamon Oldridge
Maggie Raque
Jeremy Wong

Submitted To
Craig Putnam
Nicholas Bertozzi
Markus Nemitz

Abstract

Every year at least 7,000 people around the world die from abandoned, unexploded landmines leftover from times of conflict. Current demining methods are expensive, dangerous, and slow. To combat this issue, our team is continuing the project to create an operational autonomous demining system. The system will consist of three parts: a rover, a drone, and a base station. The rover will be able to search a user defined area for unexploded landmines. When the rover finds a landmine, it will record the location. After the search is complete, the drone will then fly to the locations of the landmines and will drop a small payload onto the mines to detonate them. The base station acts as the communication link between the rover and the drone and provides a user interface for the operator to control the system. The goal of our project is to have a functioning, relatively inexpensive system that can increase the safety and efficiency of global humanitarian demining efforts.

Acknowledgements

We would like to thank our advisors first and foremost: Nicholas Bertozzi, Craig Putnam, and Markus Nemitz for supporting us in this challenging endeavor through this difficult time. Their guidance and insight was integral to the success of our work. Thank you. We would also like to thank Kat Crighton, whose administrative work allowed us to flourish, and whose passion bound the entire department throughout the year.

There were many others who supported us along the way without whom, in small or large part, this would not have been possible:

Connor Miholovich - a passionate and experienced drone connoisseur, for providing the expertise to achieve the quick turnarounds on drone testing that were required and for invaluable problem solving insights.

Andrew VanOsten - a member of Demining Phase IV, who went above and beyond with video call instruction and debugging, and even providing written guides on the specifics of the project.

Ian Anderson - for the instruction and support in Washburn labs necessary for the mechanical success of the project. The welding lessons so generously provided were a foundation of our progress.

Armando Uribe - an engineering representative from UAV Systems who worked closely with the team to resolve not only our issues with the drone that we were provided, but even provided some resources for our own custom systems.

The Demining Phase IV Team - whose work we built on.

Thank you all.

Table of Contents

List of Figures

1.0 Introduction

1.1 Problem Statement

Abandoned, unexploded landmines are a major concern for human life and wellbeing. While an actual number is hard to obtain, the International Campaign to Ban Landmines (ICBL) reports that in 2018 there were 6,897 casualties due to landmines, 71% of which were civilians (ICBL, 2019, p. 54). One majorly affected demographic is children, accounting for 40% of all casualties with a known age, and 54% of civilian casualties with a known age (ICBL, 2019, p. 58). However, even these statistics are likely lower than the actual number, since these are based on only the reported numbers, and the ICBL states that even estimating the gap between reported and actual casualties is not feasible due to conflicts that started in 2015 (ICBL, 2019, p. 54). There have been a total of over 130,000 reported casualties since 1999 (ICBL, 2019, p. 55). Landmines inhibit the development of infrastructure, trade, and impact mental health in affected communities. The goal of our project is to develop a relatively cheap, safe, and efficient system capable of clearing landmine contaminated areas in order to prevent as many civilian casualties as possible. Not only will this reduce the number of civilian casualties due to abandoned, unexploded landmines, but it will also reclaim useful land. Demining efforts are important in helping conflict stricken countries recover.

1.2 Project Description

The project mission is to create an Autonomous Demining System operable by a non-technical user. The demining system specializes in the PMN-1 landmine. The PMN-1 is an anti-personnel blast mine developed in the 1950s. They are one of the most widely used mines because of their low cost and simple production. They have a metal trigger, detectable by a metal detector. These mines are relatively deadly, containing about five times the explosive material of other mines of similar sizes (Landminefree.org, 2017). It is reported to be in use in at least 20 countries especially in the Middle East, Africa, and Southeast Asia (Smith, 2019).

2.0 Background

2.1 Related Work

2.1.1 Other Methods of Demining

The main three methods of demining are manual, animal assisted, and machine demining (Mikulic, 2013) (Habib, 2002).

Manual demining involves trained personnel using sensing equipment like metal detectors, radar, and probes to identify the location of a mine, and then remove or remotely detonate it (Mikulic, 2013). Manual demining relies heavily on metal detectors, but about one third of all anti-personnel landmines are metal free. Additionally, the soil in minefields is usually saturated with metal fragments, debris, and litter. In some environments, only about 0.1% of all signals belong to mines (Habib 2002). Manual demining is not only slow, but dangerous. When Afghanistan was demining PMN-1 mines, about 3% of their deminers experienced accidents (Trevelyan, 2000).

Animal demining is usually carried out by dogs, which are 10,000 times more sensitive to explosive residue than man-made detectors (Habib, 2002). Other animals, insects, and even bacteria have also been used though. African Pouched Rats are a relatively new tool being used, and may be better suited than dogs. They have better senses of smell, cost less to maintain, mature quicker, can be transported easier, and are resistant to tropical diseases (Habib, 2002). However, animals can be overwhelmed when there is a high density of landmines, and cannot work long hours.

Machine demining involves either tilling or flailing the ground to detonate the mines under a protective shield attached to a vehicle (Mikulic, 2013). Flailing the ground detonates the mines by triggering their pressure sensors. The two tilling designs either crush mines under a till, or uproot them and bring them in front of the till to be more easily flailed. Usually, these detonators are attached to armored vehicles. The heavier the vehicle, the higher its speed, but the lower its versatility (Mikulic, 2013). Excavators can also have their end effectors changed to these detonators, and can be much more versatile. Mechanical demining is relatively efficient, but is significantly gaited by terrain restrictions, environmental disruption, and low consistency (Mikulic, 2013) (Habib, 2002). It is best used supported by manual or animal teams that do a secondary pass over mechanically cleared areas. Speed, safety, cost, and versatility are all factors when assessing a demining solution. Figure 1 shows the speeds of different demining methods. All of the speeds are in m^2 per day in favorable conditions (Mikulic, 2013).

	Manual	Dog assisted	<5 ton Machine	5-20 ton Machine	>20 ton Machine
Primary Pass	75 m^2/day	500 m^2/day	725 m^2/day	1550 m^2/day	2500 m^2/day
Secondary Pass	450 m^2/day	1500 m^2/day	*N/A*	*N/A*	*N/A*

Figure 1: Efficiencies of various contemporary demining methods

2.1.2 Other Autonomous Robotic Solutions

Many current autonomous robotic demining solutions are primarily developed for military use and are high cost systems, which are too expensive for many communities most in need. While some efforts have been made to create affordable and accessible robotic demining systems, they are not yet as effective or readily available as is truly needed. One of these efforts led by K. T. M. U. Hempala and R. P. Razzoli (2012) in a joint project between the University of Sri Lanka and the University of Geneva. They worked on creating a design for a demining robot that could be made from materials and farming equipment that local residents of their target areas would already have access to and know how to operate.

The primary goal of the 2012 project was to create a demining robot that would be effective for use in “mine belts" as well as randomly buried mines. The design does not locate buried mines and is focused on the removal of the buried mines as opposed to the detonation of them, which was seen as less reliable for total clearance. This system implements several innovative ideas but has several flaws that ultimately limit its effectiveness. This system is highly dependent upon ground conditions and drops in both efficiency and effectiveness in non-ideal terrain. Because the system is not locating buried mines there may be ones that are missed and the system can not guarantee 100% clearance.

2.2 Previous Work

2.2.1 Previous Work on Base Station

The base station consists of the base station laptop, a wireless link to communicate with the rover and drone, and a real-time kinematic positioning unit (RTK unit). Previous work designed the base station to use ROS for high-level planning and communication with both the rover and the drone. It will define and hold the essential mapping information such as the boundaries of the minefield to be cleared and the location of the detected mines which the rover has scanned. It will then use a pathfinding algorithm to map out an ideal path for the rover to follow in order to cover the entire minefield, sending these waypoints to the rover as it requests them. When the rover identifies a mine, it will send the GPS coordinate back to the base station. The base station will keep track of all identified mines to send to the drone once the rover is finished sweeping the minefield. Once the rover has completed its search of the minefield and moved to a safe location, the base station will command the drone to launch and provide it with the mine locations. The drone will travel to each location and drop a payload on the mine to detonate it, then retreat to the associated escape location. Once the drone has cleared all mines or run out of payloads, it will communicate with the base station and return to be recharged and restocked, then restart the process until all mines have been neutralized.

The system requires input for designated minefield boundaries, needs to be manually restocked, and may need to be aborted. Therefore a user interface was developed for the application to allow a non-technical user to communicate with the system. This UI was built by the previous team, but not completed. The end goal for the UI is to provide a simple and intuitive means for a user to complete the necessary setup for the system to start without issues and provide understandable feedback to the user during the process of demining. A secondary purpose is to also provide a way to quickly abort and recall the rover or drone in the event of an emergency, or even take manual control of them as the situation dictates.

As our team took control of the project, most of the User Interface work was cemented in place, the UI framework was complete with a Google Maps API display for selecting minefield bounds. However, the UI was still being developed and required much fleshing out. The API no longer functioned correctly upon picking up the project, and had not been fully completed and tested previously. The UI also still needed added functionality to communicate with ROS in order to send commands and receive data. The rover pathfinding had to be implemented as it, along with the rover code, had been planned but delayed due to COVID-19.

2.2.2 Previous Work on Rover

The rover has two essential parts that will allow it to do its job properly: the four-bar arm, which has a SURF 1.2 metal detector, and a navigation system, composed of the drivetrain and RTK. The four-bar arm uses a pair of linear actuators to raise and lower the height of the metal detector. The metal detector is attached to a linear screw system at the end of the four-bar arm as shown below in figure 2.

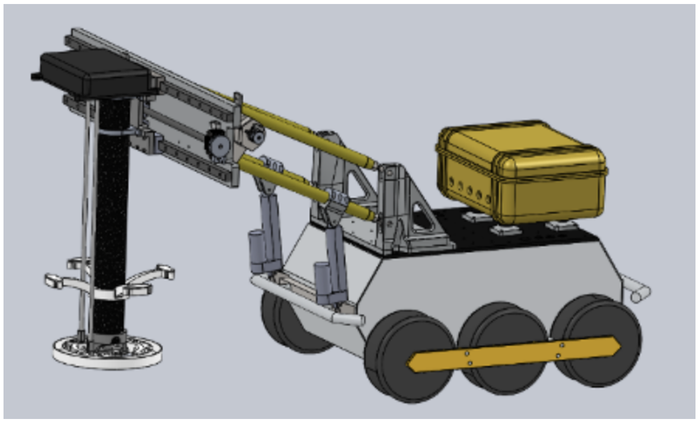


Figure 2: Rover with Four-Bar Arm and Metal Detector

The linear screw system provides the lateral motion for the metal detector to sweep in front of the rover. It is powered by a Vex CIM motor attached to a lead screw. The metal detector input is processed by an Arduino Uno. A Teensy 3.5 manages the Uno, all the motors and actuators, and all the sensors except the RTK. An onboard computer running ROS manages the RTK and communicates with the base station to receive waypoints to move between. The first waypoint sent directs the rover to the correct location to start following its path through the minefield. Once it reaches the start location, it will start a grid search pattern where it moves the linear screw system from one side to the other, scanning for mines. The rover will then move forward the distance of the metal detector scanning radius and perform the sweep function again. While the linear screw system moves the metal detector laterally, four ultrasonic sensors will measure the distance from the metal detector to the ground and adjust the angle (pitch and roll) and height of the metal detector to keep it parallel and close to the ground. This adjustment ensures maximum detection radius, depth, and accuracy. When the metal detector is triggered, the rover will send a message back to the base station indicating that there is a suspected mine at the current RTK coordinates. The RTK will be mounted directly atop the arm which moves with the linear actuation, guaranteeing that the coordinates sent are the same as where the detector was set off. It will then repeat, inching forward until it reaches the waypoint.

The planned navigation system for the rover would also employ a LiDAR sensor to scan for obstacles. The data collected from the LiDAR will be processed in ROS, then the path planning will be updated to avoid obstacles in the minefield. The detected mines as well as objects identified by the LiDAR will be factored into the base station's path planning. The rover will traverse the field by sweeping adjacent strips according to the planned path until the entire section has been covered. Then it will follow waypoints to a safe location outside of the minefield to ensure it is undamaged during the drone's detonation operation.

2.2.3 Previous Work on Drone

The drone system consists of three main subsystems: flight, mission control, and payload delivery. The flight subsystem consists of the physical drone and its low level control, including electronics such as motor controllers and the flight controller. Demining Phase IV had integrated an ArduPilot Pixhawk flight controller into a prebuilt drone base. The mission control subsystem is in charge of communicating with the base station, and directing the drone between waypoints during the mission. Previous work utilized an RTK unit for precise localization and a Teensy 3.5 microcontroller as the on onboard computer. Phase IV wrote a flight control program to be run on the Teensy in C++, and a partner program on the Base Station. Finally, the payload subsystem accurately deploys payloads to remotely detonate the landmines. The dropper inherited from Phase IV was 3D printed and had a stepper motor mounted, but no electronics were implemented and it had not been tested. The dropping mechanism is attached to the bottom of the drone, and is run by the Teensy 3.5.

The drone is able to navigate to the target location by receiving waypoints from the base station. The base station sends two waypoints to the drone, the first is the location of the suspected mine detected by the rover, and the second is an efficient escape location in the direction of the next mine. The escape location is a safe distance away from the mine so the drone speeds away to clear the area of potential shrapnel. Once at a mine waypoint, the drone will steady itself in a hover, and make sure to be as close to the original mine coordinates as possible before dropping the water balloon to detonate the mine.

The previous team working on this project determined that the force from a 0.5lb water balloon dropped from 20 ft above ground level is sufficient to detonate the PMN-1 landmine (Santos et al., 2020, p. 59). When the drone runs out of payloads it will return to the base station and land, where it can be reloaded and recharged to continue the mission. Once it is done clearing all suspected mines, it will return to the base station and wait to clear the next section after the rover identifies another set of coordinates.

Unfortunately during the final test flight last year the drone crashed and due to COVID 19 the team was unable to replace it before the end of the school year.

2.2.4 Evolving Project Design

Since the project has been ongoing for several years, there have been necessary changes to the equipment to improve the efficiency of the system and keep it up to date. This year, the team received a new rover base and were able to use a new drone as well. The previous rover base had been the source of issues with turning precisely, which were exacerbated by the heavy load it carried. These turning issues would have been inefficient and dangerous for the mission. The wider turn radius would mean that the rover may have entered unsafe locations, such as other sections of the minefield, when turning to scan the next strip of the current minefield section. The imprecise turns would have potentially caused the rover to get off track and unless there was live location checking using the RTK unit. This could also cause the rover to scan outside of the designated minefield section and fail to scan the whole section while marking it as scanned, possibly leaving undetected mines behind. The team also used a new drone due to the difficulty in acquiring parts to repair the extensive damage to the previous drone caused by the crash late last year. The difficulty getting parts was partially due to the drone being a discontinued model, and partly because of trade relations between the United States and China, where the parts were available. This made repairing it unfavorable since it would require printing or machining of the broken parts as well as any parts needed in the future. The team was generously offered to share a drone with Professor Markus Nemitz, who was looking to acquire a heavy payload drone for his own research and became an advisor for this project.

These major project changes are not unique to this year, the last project team rebuilt the entire four-bar and sensing arm assembly. This involved replacing the painting mechanism to mark mines with the current RTK coordinate markings. They also changed the payload dropping system on the drone completely, from sand bags to a revolving water balloon dropper with a higher capacity.

3.0 Methodology

3.1 Problem Formation

3.1.1 Acceptance Criteria

This section describes an acceptably working system, and was decided based on the skills of our team, the speed of our progress in previous work, and the accomplishments of previous teams on this project.

1. The mine detecting apparatus is attached and functioning on the new robot base.

2. The C/C++ robot code is completed so the rover can use the metal detecting apparatus, drive, turn, and navigate using its encoders and dGPS.

3. The ROS code for the robot allows it to communicate with the base station, and manages the path planning and the basic functions of the robot listed above.

4. The rover can communicate with the base station to coordinate completely sweeping an area defined using the Google Maps API for both concave and convex polygons. This means the path planning takes it across the whole field, and it is able to identify suspected mines and avoid them during the process.

5. The accuracy necessary to consistently detonate PMN-1 mines from an altitude of 20 ft. with our payloads is identified, and our dropping mechanism achieves this necessary accuracy.

6. The drone is able to receive mission instructions from the base station, navigate to a number of mines equal to its payload capacity, and drop its payloads.

Although these are the goals for our project, they do not constitute a complete demining system. Trials involving the complete system, as well as more field realistic tests for each subsystem, would still need to be completed.
The rover should be able to sweep areas at rates at least comparable to those of manual mine sweeping, which is between 5-150 square meters per day (Guide to Mine Action, p.137).

3.2 Rover System Updates

3.2.1 Mounting the Metal Detector on the Rover

3.2.1.1 Modeling The CERBERUS Rover

Previous phases of the project utilized a Clearpath A100 Husky platform as the metal detecting vehicle of the system. The last DAS MQP phase encountered mobility issues involving the capabilities of the rover to turn while bearing a heavy load. The Husky rover was replaced with a treaded rover base from a past CERBERUS WPI project to improve the maneuverability of the rover with the metal detecting apparatus. A Solidworks CAD model was created of the rover to allow for design and fitting of new components as well as analysis of these components and the system as a whole. First, measurements of the main, central frame of the rover were taken and recorded along with notes about frame geometry. The frame of the rover was determined to be made from one inch steel square tubing with a wall thickness of one sixteenth of an inch based on the measurements, color, texture and weight of the frame. This information was then used to generate a CAD model with accurate dimensions and shape as shown in figure 3 below. The same measure-and-model method was then used to create the wheel housings, the wheels, and treads. Additional components including a bottom plate and batteries were added to give a more accurate depiction of the robot and its weight.

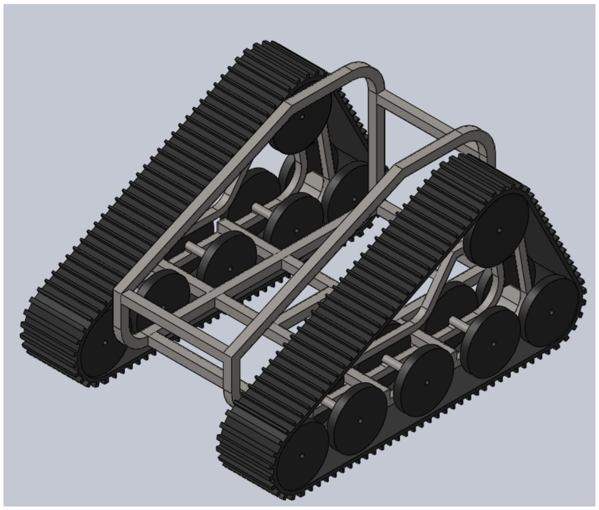


Figure 3: Rover CAD Model

3.2.1.2 Designing The Mounting Attachments

There needed to be a flat platform on which to mount the four-bar's anchoring support on top of. In addition, a lower horizontal bar was needed in front to connect the linear actuators for the four-bar mechanism. In order to adapt the CERBERUS base to fit the arm designed for the

Husky, the connection for the linear actuators would need to be approximately 5” in front of, and 7.5” lower than the connection joint for the four-bar linkage. To provide the flat platform for the anchoring support, a 90 degree frame made from square steel tubing was designed with angled cuts to match the angled slope of the frame, where it would be welded as shown in figure 4 below.

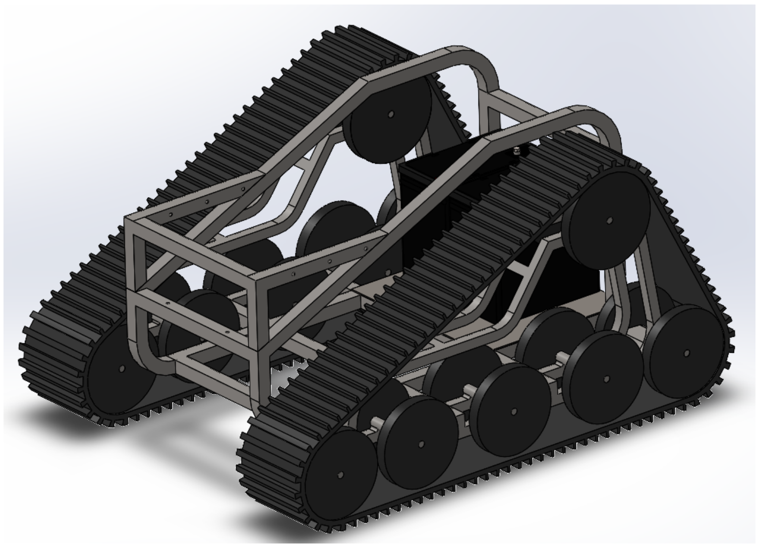


Figure 4: Rover with Mounting Platform

The linear actuators are attached to the frame using steel square tubing with pairs of welded steel tabs that sandwich the frame and are secured with a 1/4-20 bolt as shown by label A in figure 5. To attach the four-bar anchoring support to the newly framed platform, the bottom plate that holds the four-bar mechanism was drilled and tapped so twelve 14-20 1.75” bolts could be bolted on, sandwiching the one inch square tubing as shown by label B in figure 5 below. Six aluminum tabs that are 1/4” thick are used to secure the plate to the mounting frame.

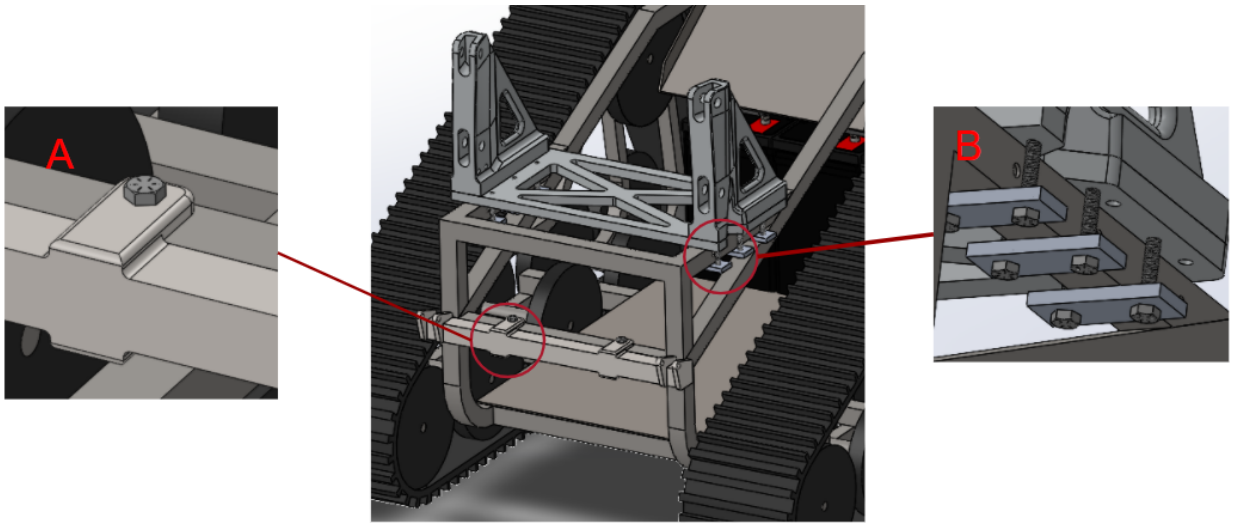


Figure 5: Four Bar Attaching Components

This connection allows for a sturdy and easily removable connection so that the four-bar mechanism can be removed from the base if necessary for transport or repairs. The location that the plate is meant to rest on is etched into the metals so that it can be returned to the same place every time.

3.2.1.3 Analysis of Mounting Components

To analyze the mounting components, both handwritten and computer simulated stress calculations were completed to ensure that the stress on each of the components did not exceed the von Mesis yield criterion for each of the materials. To analyze each component the solidworks models were run through ANSYS. This pThis provided a comprehensive understanding of the stresses throughout each of the individual parts. An example of one of the ANSYS analysis is shown below in figure 6. To confirm that the numbers the program provided, we completed simplified 3D static hand calculations. For our hand calculations we simplified it to the force applied on the projected surface area of the fastener holes. We know that this is an oversimplification but since we were only using the hand calculations to confirm our program analysis, we were only looking for the hand calculations to be within 30% of the numbers from ANSYS.

Material: 6061-T6 Aluminum

Max Stress = 83.7 Psi

Yield Strength = 35,000 Psi

FOS = 418

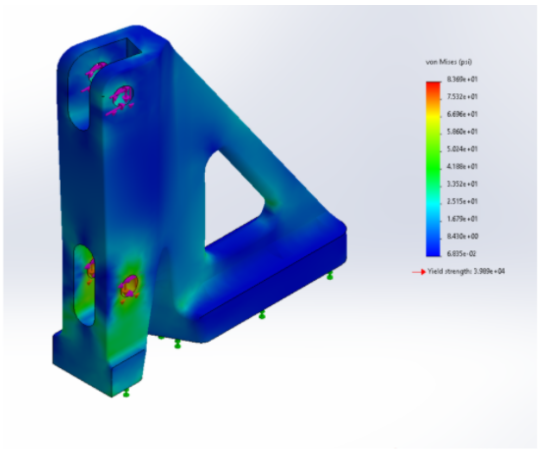


Figure 6: ANSYS Analysis of Left Angle Bracket

3.2.1.4 Weight Reduction In Four-bar Design

The metal detector extends beyond the front of the rover frame and has a significant weight of 50 lbs. In order to move the center of mass back towards the center of rotation for the rover, which in turn allows for more control and mobility of the rover, the weight of the four-bar support design needed to be reduced. To reduce the weight of the four-bar assembly we redesigned the top front plate to remove around 7 lbs of material while still being able to mount the linear slide mechanism that allows the metal detector to oscillate from one side of the rover to the other. This design was completed but was never fabricated.

3.2.1.5 Fabrication

To create the mounting platform that is needed for mounting the four-bar assembly to the CERBERUS base we needed to weld together the one inch square steel tubing. The team decided to utilize TIG welding for all welds that were used, because it allowed for more precise welds and gave us more versatility in the thickness of steel we ended up using. Once the steel frame of the mounting plate was welded together, as shown below in figure 7, the angled cuts of the square tubing were welded to the steel frame of the Cerberus rover.

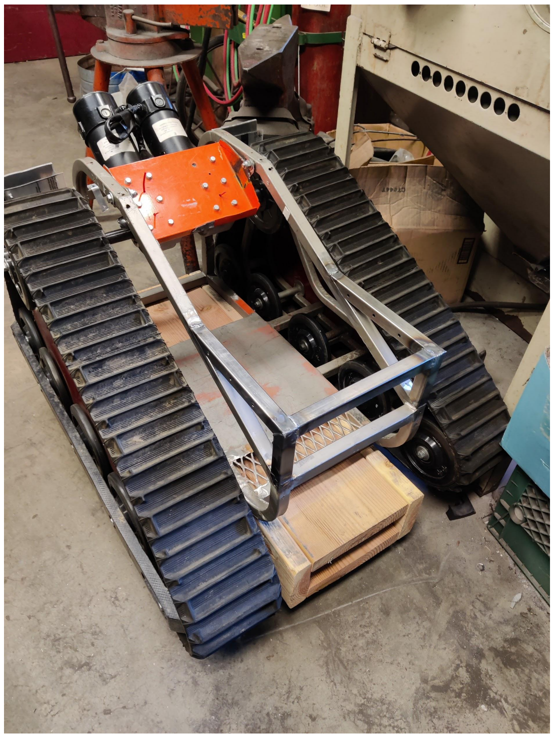


Figure 7: Mounting Frame Welded to Rover Frame

Another two holes we drilled into the front, steel, cross bar, and four identical tabs were welded onto a front steel mounting bar that supports the linear actuators. The bar and tabs were then mounted to the CERBERUS base as shown below in figure 8, this allowed for the bolts to be used to attach the front bar that is connected to the linear actuators of the four-bar mechanism.

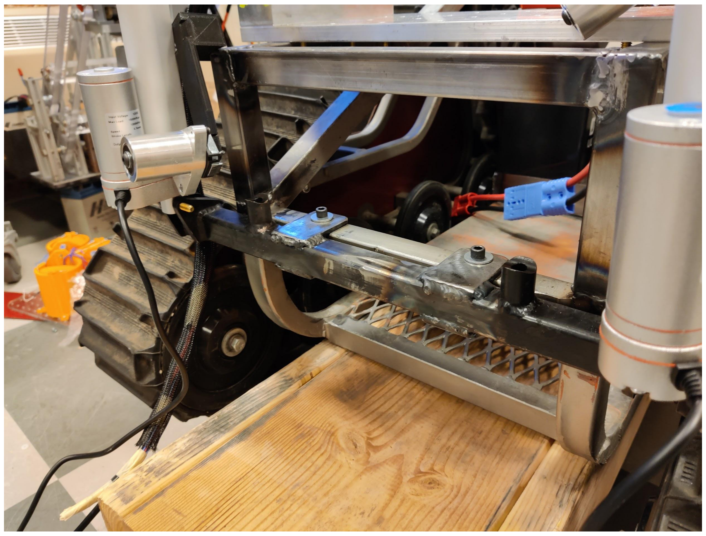


Figure 8: Modified Front Bar

The final modifications to the rover base was a mounting box that was made out of sheet metal was folded and welded onto the rover base to hold the two 12V batteries during operation.

3.2.1.6 3D Static Truss Analysis

In addition to our calculation of the attachment points to confirm that the fasteners would go beyond the max von Mesis force, we conducted an analysis of the four-bar mechanism to find the internal forces in each member, the reactionary forces in each joint, and the deflection of the assembly; both when it is at rest and when it is turning. To complete these calculations we made a series of assumptions to slightly simplify the system, but every assumption we made resulted in higher internal forces on the system, that we could be confident that if it could handle the forces we calculated, it would be able to handle the actual stress it would be receiving. To calculate the resultant force for the rotational acceleration, we needed to find the force that the treads would be able to apply to turn the robot and create the rotational acceleration. This force needed to be found experimentally because the company that the Cerberus base was originally purchased from refused to provide any information about the operating capabilities of the drive motors. To find this force we drove the robot into a scale that was propped up against the wall and measured the total force that the robot was able to apply before the treads started slipping. This experiment is demonstrated below in figure 10.

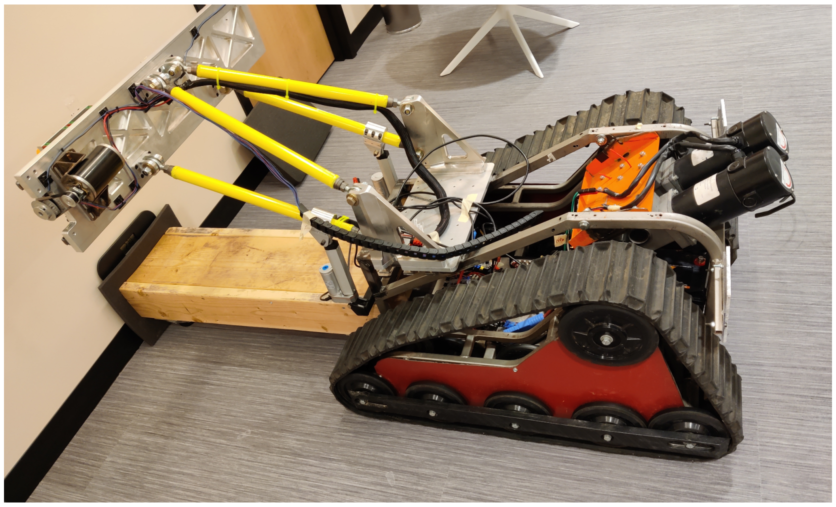


Figure 9: Resultant Force From Treads Test

This test is very dependent on the surface that it is being conducted on, but the force that we found far exceeds the limits we are putting on the rover while it is operating. Because of this the resultant calculated acceleration and force from the metal detecting arm rotating that we calculate will be higher than the actual one, and as such it is structurally safe under our calculated load. In addition, we understand that not 100% of the force that the treads apply when they are driving in a straight line. Part of the force that the treads are able to apply will go toward overcoming the friction force to get the front and back part of the treads to slide over the ground as the treads rotated around the center of rotation. We did not take this into account when conducting our calculations because if we did, it would only decrease the resulting force at the metal detecting arm, and it is very dependent on the type of ground the rover is covering, so to be on the conservative side in our calculations, we decided to not take it into account. Another simplification we made was to assume that the members in the four-bar mechanism were massless, and looked at the entire mass of the four-bar mechanism and metal detecting linear slide and based our calculation off the assumption that all of the mass is being applied at the center of mass of the linear slide. This allowed us to simplify our hand calculations and still accounted for the weight of the steel tubing. The final assumption we made is that we assumed that the centripetal force from the velocity of the rover rotating was negligible when compared to the resultant force due to the acceleration from the motors. Since we have capped the rotational velocity of the rover at one full rotation in 8 seconds, or 0.79 radians per second, to allow for precision in the autonomous system. The resulting centripetal force would only be approximately 3 lbf at max speed and is negligible.

3.2.1.7 Calculating the Resultant Force

To calculate the force that is applied to the system from the rover turning, we needed to find the acceleration of the metal detector as it rotated around. To calculate that, we used the maximum force that the treads can apply to the system, and used that to calculate the rotational acceleration of the system from the torque on the system. To do that we needed to calculate the moment of inertia of the rover. To calculate the moment of inertia we simplified the rover down to a rectangle rotating around the pivot point with the center of mass 2.5 inches in front of the center of rotation. The calculated moment of inertia was 28 kg*m^2, the resultant angular acceleration of the rover was 14.7 rad/s^2. After we had the rotational acceleration of the system we could find the linear acceleration of the metal detector and its resultant force. The linear force that is applied from the rover turning ended up being 71.7 lbf. Figure 10 shows a diagram of the system to help make the described calculations clearer.

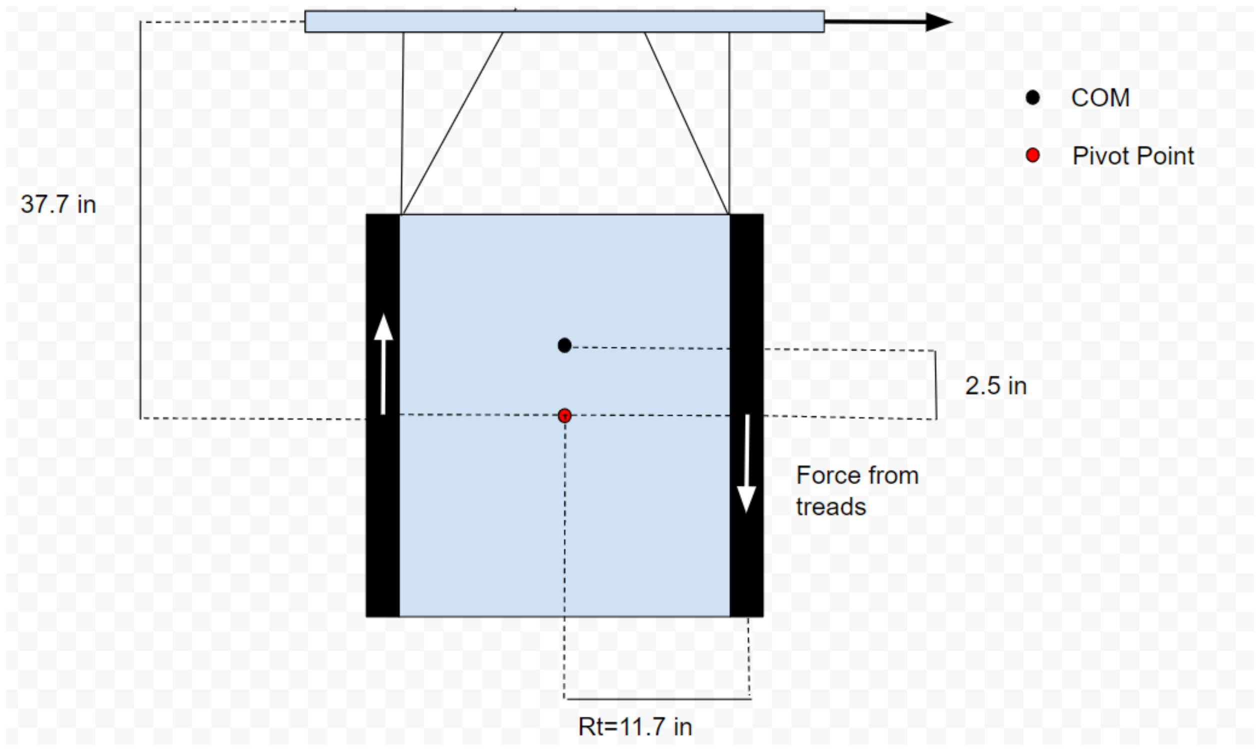


Figure 10: Diagram for Rotation of the Rover

3.2.1.8 Mounting Plate Fastener Pull Out Strength Analysis

Aftering running the 3D Truss Analysis, we were able to calculate the force that was going to be applied to the 1/4-20 bolts that hold the mounting plate for the four-bar to the CERBERUS frame. From there were were able to calculate the maximum pull out strength for the 1/4-20 bolts to see if they would be able to endure the load that would be applied to them from the weight of the four-bar.

3.2.2 Rover Electronics

Just as the sensing arm hardware was transferred from the old rover base, the team also transferred the majority of the control system and electronics that were not a part of the off-the-shelf Husky system to the new rover base. Most of the sensing and onboard processing systems were able to remain unchanged, while modifications were made to the drive system to accommodate the new base. An XBee was added to communicate with the microcontroller for manual control of the drivetrain, four-bar, and metal detector, and a Logitech F710 was added to shortcut manual control of the drivetrain through the Hero Development board. In keeping with the previous team's efforts, we organized and documented the electrical system for ease of troubleshooting and repair as well as ease of understanding for future teams.

3.2.2.1 Combining the Husky and CERBERUS Electrical Systems

Integrating the electronics system with the new rover base presented a few issues regarding differing power requirements, particularly for the two drive motors. One of the primary reasons for using a different rover base was to achieve greater traction and power within the drive train. Last year only 12V and 5V supplies were required, but the drive motors on the new base require 24V. Additionally, the voltage conversion and power distribution components of last year's system were a part of the Husky platform and could not be transferred to the new base. The team added two voltage converters, 24V to 12V and 24V to 5V, to the system to create 24V, 12V, and 5V power supplies, as shown in figure 11 below.

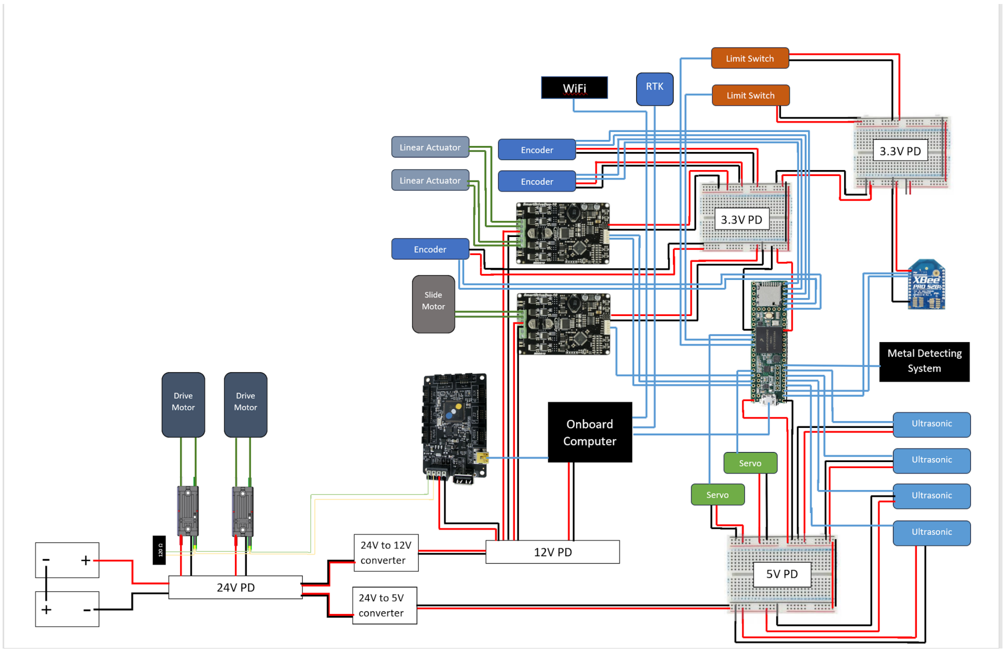


Figure 11: Rover Electrical Schematic

Another major difference from the old rover base is the style of the drive motor controllers. The Husky base used Jaguar motor controllers as part of the off-the-shelf system. The new rover base that our team received this year uses Talon SRX motor controllers. The primary difference between the two types is the way they receive signals. Jaguars use individual PWM signals while Talons use a CAN Bus system. Implementing CAN Bus signaling with the rover control system required the team to use a HERO Development board from Cross the Road Electronics to interface between the onboard computer and the CAN Bus system.

3.2.2.2 Electrical Redesign for Robustness and Durability

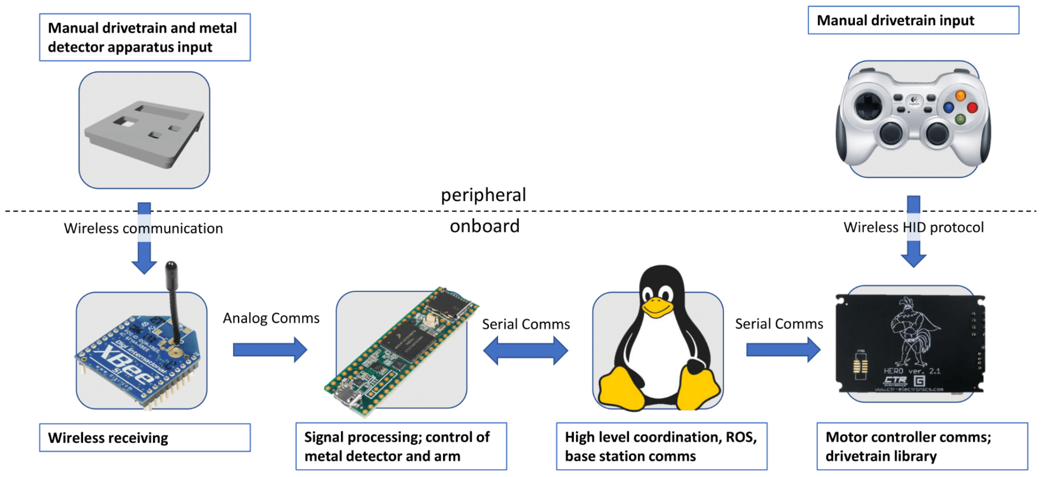
Continuing with the previous team's efforts to upgrade the overall system to make it more robust and high quality, our team is upgrading components of the electrical system to make them more durable and reliable. The primary upgrade will be the replacement of three breadboards being used for power distribution with perfboard circuit boards that have soldered connections. These connections will be less delicate and susceptible to coming loose with vibration from diving, which is necessary for the system to be able to function reliably in unpredictable terrain.

The team will also be using electrical connectors that create a solid and secure connection between as many components as is feasible to both ensure reliable connections and facilitate easy troubleshooting and quick replacing of components on short notice. This will be part of making the system more practical and accessible for real life scenarios and needed maintenance.

3.2.3 Rover Control

The rover is controlled using the popular ROS framework. Previous work had also used ROS, but had relied on the professional libraries prebuilt for the Husky body. A ROS node network was designed with inspiration from previous work, and outlines future implementation of mapping, navigation, dGPS, and LiDAR nodes. The interface to the electronics uses C++ and Arduino and is called from the XBee manually. Code for the ultrasonic sensors, and the slider motor and linear actuators controlling the arm and sweep have been adapted for the new system but remain untested. The communication protocol between the rover and base station are being adapted from those written for the drone.

Because the XBee communication with the Teensy 3.5 was not implemented until the end of the project, the team investigated alternative manual control methods.



Manual drivetrain and metal detector apparatus input
Manual drivetrain input
peripheral
onboard
Wireless communication
Wireless HID protocol
Analog Comms
Serial Comms
Serial Comms
Wireless receiving
Signal processing; control of metal detector and arm
High level coordination, ROS, base station comms
Motor controller comms; drivetrain library

Figure 12: Rover control diagram

The team decided to interface directly with the HERO board with a game controller through a USB connection. This is because the HERO has a processor and can be directly programmed in C# through the NETMF framework. The development of the drivetrain library can be started on the HERO, and called by the controller. The same library can be called from the onboard computer in the end solution, so the team started the library to get basic driving but replaced the communication from the computer with a Logitech F710 wireless controller.

The HERO is designed by Cross the Road Electronics (CTRE) and communicates with the Talon SRX motor controllers over CAN bus. The team started by updating all the firmware on the HERO and both the Talons to the latest v-21.0 releases. Phoenix LifeBoat, the CTRE imager software, was used to update the HERO. LifeBoat then was able to identify the Talons through the HERO, and updated their firmware as well. The HERO board has a firmware level direct drive solution that requires no programming which was used to sanity test the electronics system. The direct_drive firmware on the HERO automatically recognizes attached Talon motor controllers and game controller input, then drives motors based on analog stick input with no coding needed on the user end. However, initial testing did not yield results. The team changed the HERO board to the NETMF firmware, and programmed it through the .NETMF framework addon in Visual Studio 2017. The team used an edited CTRE sample program to test the motor control, but this also yielded no results.

Through debugging, it was found that the XBox One controller that had been employed used too modern of a communications protocol, so the team switched to the Logitech F710 controller which uses HID USB protocol. It also has a DirectX protocol mode switch, which can be used as an emergency stop during operation as an added benefit. The team also found that the latest firmware versions implement special watchdog checks built for the FRC roboRIO onboard computer that had been preventing the motors from driving. Using Phoenix LifeBoat again, the

HERO board firmware was regressed to the latest non-FRC NETMF firmware, and the Talon SRX's were also updated to have the latest non-FRC firmware, v-11.11.

Using the changed firmware and the Logitech controller, one of the Talon SRXs successfully drove its motor and moved one side of the rover's treads. After thorough testing, it was determined beyond any reasonable doubt that the other older Talon SRX was not fully functioning. The team successfully replaced the dysfunctional Talon, and improved the C# drive code. These efforts resulted in the ability to manually control the rover's driving using the Logitech controller.

3.3 Systems Integration of New Drone

3.3.1 Tarot T-18 v2.1

Due to the terminal crash of the DAS Phase IV drone, our team investigated options for replacement. The specifications were long range, heavy payload capabilities, and ease of customization. Working in collaboration with Dr. Nemitz, the Tarot T-18 from UAV Systems International was selected. It was chosen for its 2 mile range and its 8kg payload capacity, as well as initial order customization support from UAV systems and an accessible electrical system. The Tarot T-18 is pictured in figure 13.

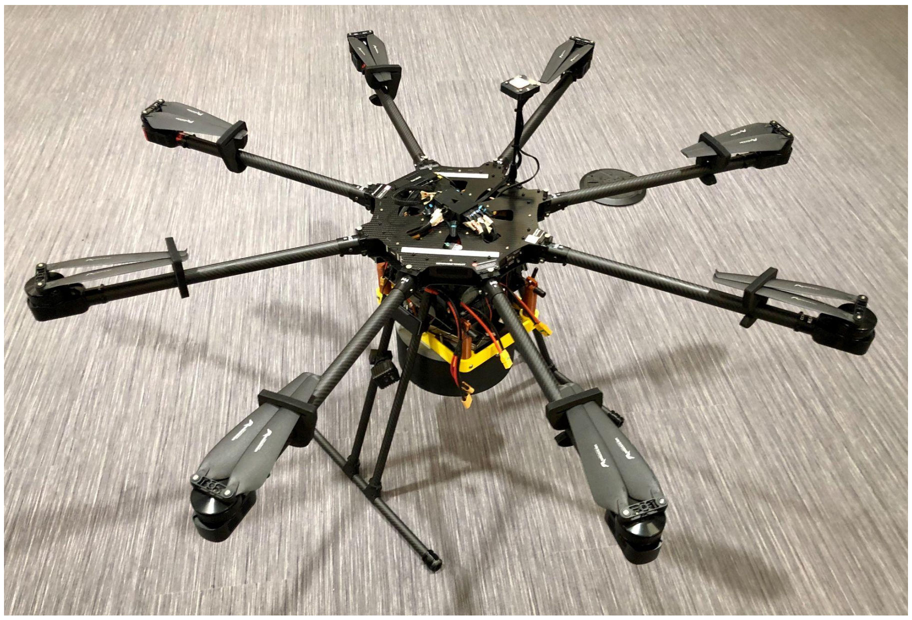


Figure 13: Tarot T-18 v2.1 Ready to Fly

3.3.2 Drone Electronic System

3.3.2.1 High-Level Electrical Overview

A major portion of the drone's upgrades was the integration of the onboard Teensy microcontroller, radio communication, and dropper mechanism with the electrical system of the new Tarot T-18 Drone. Figure 14 shows the latest electrical diagram for the drone.

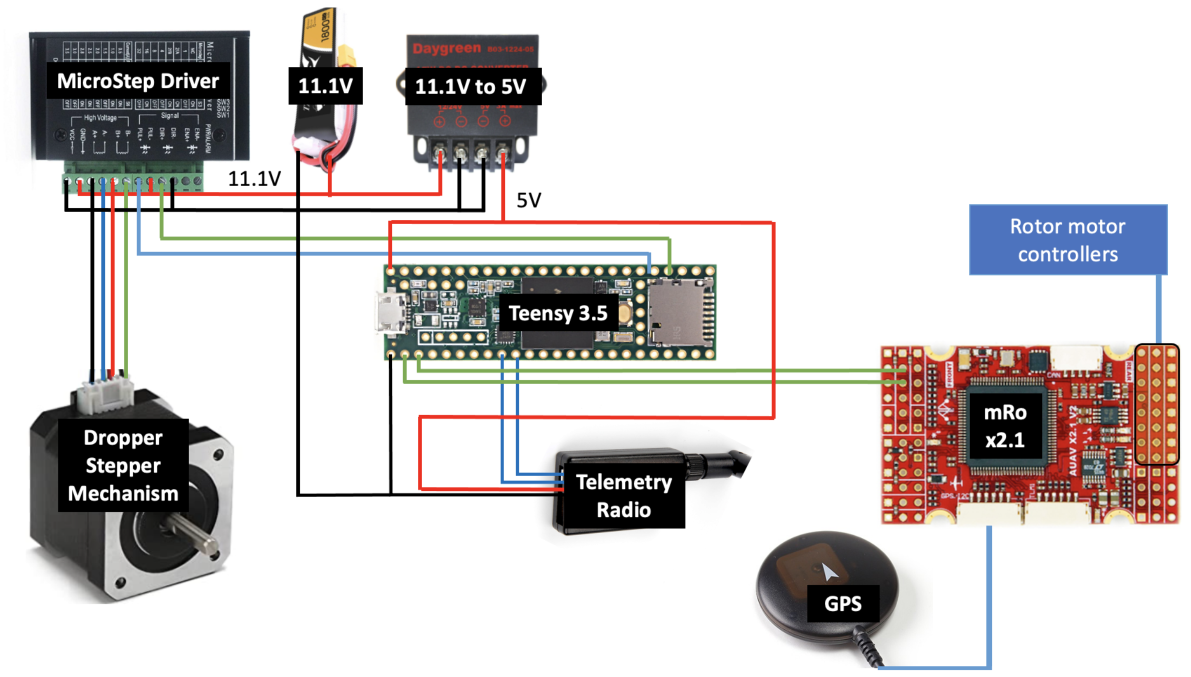


Figure 14: Drone Electrical Schematic

The stepper motor is powered with a 11.1V lipo battery and uses a MicroStepperDriver to achieve precise rotations from the stepper motor. Last year's team used a Pixhawk 4, while this year the new drone came with an mRo X2.1. This presented challenges in terms of trying to reconfigure the mRo to perform the same functions as the Pixhawk 4.

3.3.2.2 Flight Controller

The Tarot T-18 had been special ordered from UAV Systems International to use a Pixhawk 4 flight controller, the same as the previous iteration of the drone had employed. The Pixhawk 4 has an advanced STM32F765 processor, arguably the best flight controller on the market. It has 8 UART data ports, which are integral for connecting to the many planned peripheral devices. However, the Tarot T-18came with an mRo v2.1 flight controller installed instead. This controller's F4 processor is 25% slower, and it only has 3 UART ports. It also has a less precise internal compass, and has been known to face errors when running the latest ArduPilot firmware.

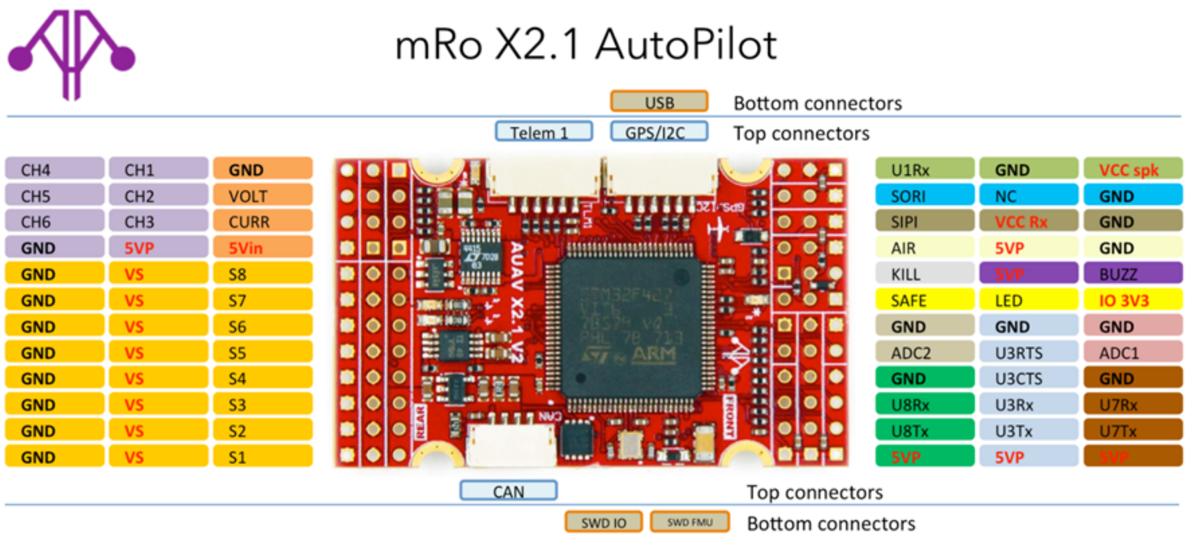


Figure 15: mRo X2.1 Diagram

The team investigated replacing the mRo board with a Pixhawk 4, but decided that the benefits were not worth the time needed to do so at the stage the project was at. UAV Systems was contacted to assist in reconfiguring the mRo board.

QGroundControl v4.1.2 was used in preparation to update the firmware of the mRo board to the latest ArduCopter release, v3.2.1. Then a UAV Systems engineer led the team through configuring the board. MissionPlanner v1.3.74 was used to set the full parameter tree, using a zip file provided by UAV Systems. Serial2 input was configured to take the MAVLink1 protocol through UART3 on the board, to communicate with the custom DAS mission control software on the Teensy 3.5 onboard computer. However, this may be the cause of the mavlink item transfer errors between the drone and the Herelink that were experienced.

3.3.2.3 Real-Time Kinematic GPS Unit

The system uses three precise GNSS modules. The modules are all ublox C94-M8P application boards with NEO-M8P-2 modules. These modules can achieve accuracies of 1-3cm relative to each other. One of them is set up as a stationary “base station,” and sends out correctional data to the two “roving” modules. The final design has one of the roving modules attached above the metal detector on the rover, and the other attached above the dropper mechanism on the drone. When the Rover discovers a potential mine location, it reports the location to the base station using the RTK unit. Once the Rover has swept the field, the Base Station reports the mine locations in an array to the Drone, which it navigates to using its onboard RTK.

The quickstart guide from ublox was followed to configure the three boards as base station and two rovers. U-center_v20.10 software was downloaded and used to initially configure the boards through USB connection. The u-center software was used to restore the boards to default settings before configuration attempts.

At this point, we started cooperating with the Snow Drone MQP, who had consulted for help with what positioning system to use for their drone. They decided to also use the same RTK modules.

Tests for the system consist of surveying-in the base station location through the u-center software, then connecting the rover modules to it. Only one rover will be tested with the base station at first, and will be moved to known distances within the range of 1 to 100 meters away from the base station to test the accuracy of the relative position. Then both rover modules will be connected to the base station and tested in parallel. The tests need to be completed in an open area with a clear sky.

In order to communicate with a Pixhawk 4 flight controller, the RTK unit on the drone needed to have its serial baud rate raised to 38,400 from its 19,200 default. This is done through the u-center software. To communicate with each other, the RTK units all need to have the same baud rate of 38,400, or the RTK corrections between the boards will not be recognized. In order to transmit this data, the radio baud rates of the modules need to be higher than that of the data within them, so are set to 48,000. The radios needed to be configured outside the u-center software using AT commands sent using PuTTY over a 9-pin serial cable.

3.3.3 Flight Control System Architecture

3.3.3.1 System Integration

The base station runs an application written in C++ that allows for user input to initiate and terminate the mission, as well as handling the path planning for the drone. Currently, dummy mine locations are hardcoded into the software. Once this application is integrated into the ROS network, coordinates for potential mine locations will be set by the Rover. This base station mission planner application uses the mine locations to generate a list of waypoint coordinates to send to the drone. The waypoints consist of mine locations where the payload drops will occur, and escape locations for the drone to move towards immediately after dropping a payload. The escape locations are planned so that they are in the direction of the next target. In order for the system to work coherently, there is a thorough setup process that occurs at the initialization of every mission.

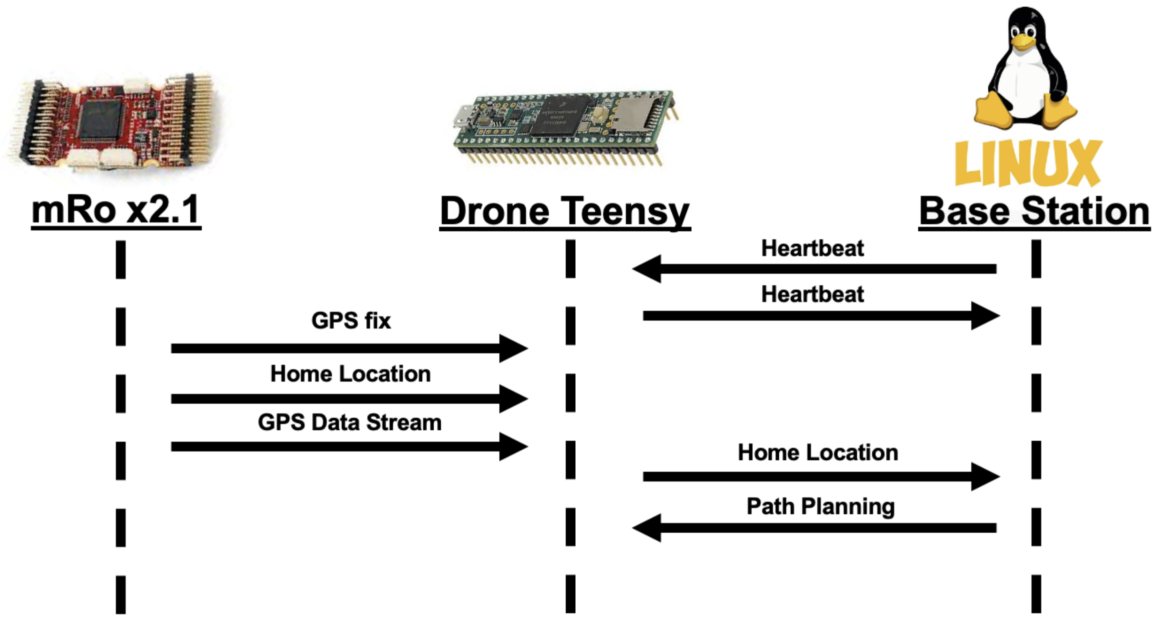


Figure 16: Drone System Setup Sequence Diagram

The sequence diagram in figure 16 above describes the information passed between the different subsystems, where the vertical axis progresses downwards in time.

3.3.3.2 Mission Control Logic Onboard the Drone

Once the setup process is complete and the mission planner application has sent the waypoints to the Teensy, the Teensy commences the mission. It controls the mission using a finite state machine, allowing interrupts from the flight controller and from the base station.

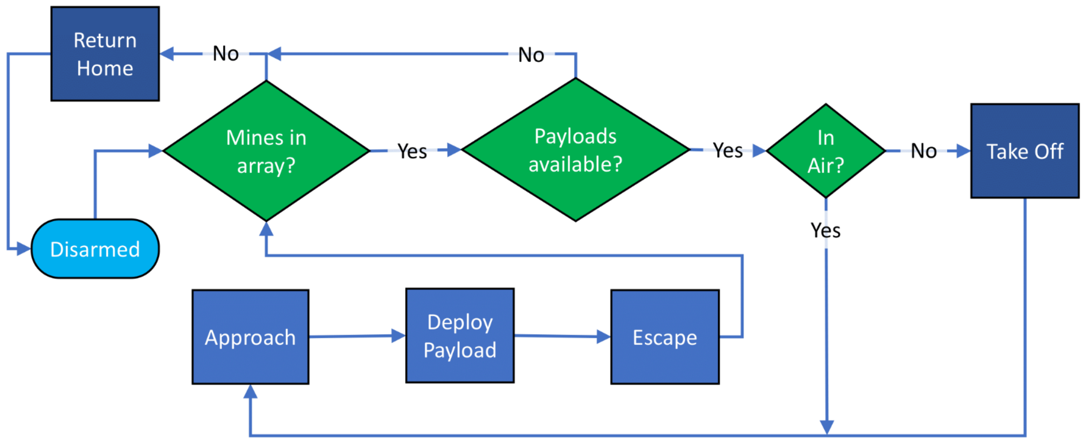


Figure 17: In-Flight Mission Control Flowchart

The flowchart in figure 17 above describes the finite state machine for the mission control onboard the drone. The Teensy maintains connection to the base station and confirms it with heartbeats throughout, and aborts the mission if communication is lost. This ensures constant user control over the active system in the high consequence environment.

3.3.3.3 Communication Between the Base Station and Onboard Computer

The base station and drone communicate via a pair of Holybro 500mW 915 MHz telemetry radios. The wireless communication established between the base station and the drone emulates a serial connection. The Holybro radios are running an open-source SiK firmware. Custom packet-based protocols allow for the information passed between the radios to be processed by the two systems and mitigate potential transmission errors.

3.3.3.4 Communication Between the Onboard Computer and the Flight Controller

The drone is controlled by a mRo X2.1 flight controller that handles the flight stability control loops allowing the drone to accurately perform flight maneuvers such as hovering, turning, and waypoint navigation. The mRo X2.1 is commanded directly from the Teensy 3.5 through a serial port using a MAVLink 1 protocol over the Serial2 input set in ArduPilot across the physical UART 3 port on the mRo.

3.3.4 Drone Mechanical Updates

3.3.4.1 Drone Dropper Attachment

Our team is borrowing a Tarot T-18 V2.1 drone from Professor Markus Nemitz. Due to the drone not belonging solely to this project, we decided that the new attachment would need to be easy to put on and take off the drone while still being able to support its payload. The attachment design clamps on to the carbon fiber rails at the bottom of the drone. The dropper mechanism is bolted through its lid to the bottom of the attachment frame. Four press-fitted hex nuts in the lid allow for secure fastening of the body of the dropping mechanism with the ability to refill the water balloon magazine by only removing four ¼”-20 bolts. Components of the dropping mechanism attachment design were 3D printed and assembled shown in figure 18.

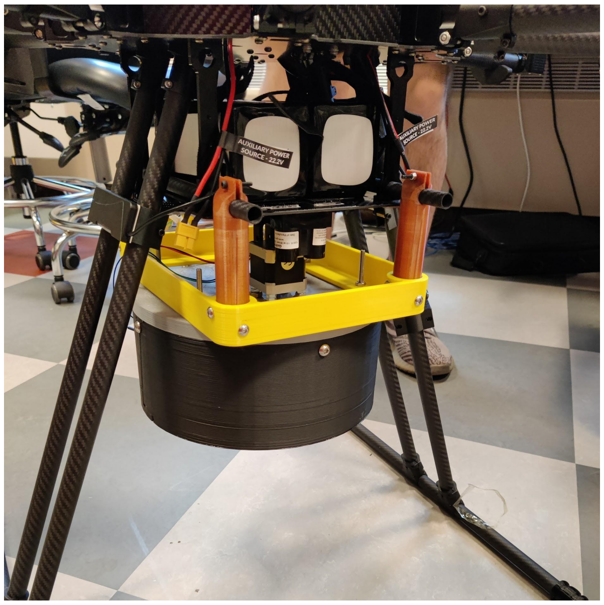

AUXILIARY POWER SOURCE - 22.2V
AUXILIARY POWER SOURCE - 22.2V

Figure 18: Drone Dropper Attachment Attached

3.3.4.2 Dropper Mechanism Redesign

When the drone crashed it highlighted structural flaws in the initial design. One of the main points of fracture was the ABS axle that was holding the balloon revolver in place. We decided to increase the diameter of the piece to strengthen the structure and prevent further breaks at that point.

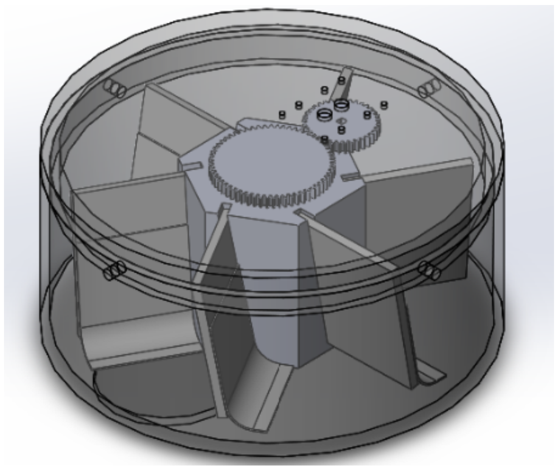


Figure 19: Drone Dropper Attachment CAD

Another shortcoming of the previously designed dropping mechanism was the fin design. It worked fine when the dropper mechanism was stable, but when it was in motion the balloons inside would get caught in the revolving mechanism causing a jam. We resolved this issue by inserting flexible extensions to the fins that allow the balloons to make more contact with the revolving mechanism reducing the chance of the balloons getting caught in the system. While this fix may reduce the chance of the balloons getting caught, the dropper mechanism can only spin in one direction so the flexible extensions wont get caught in the dropper mechanism.

3.3.4.3 Testing Criteria for the Dropper Mechanism

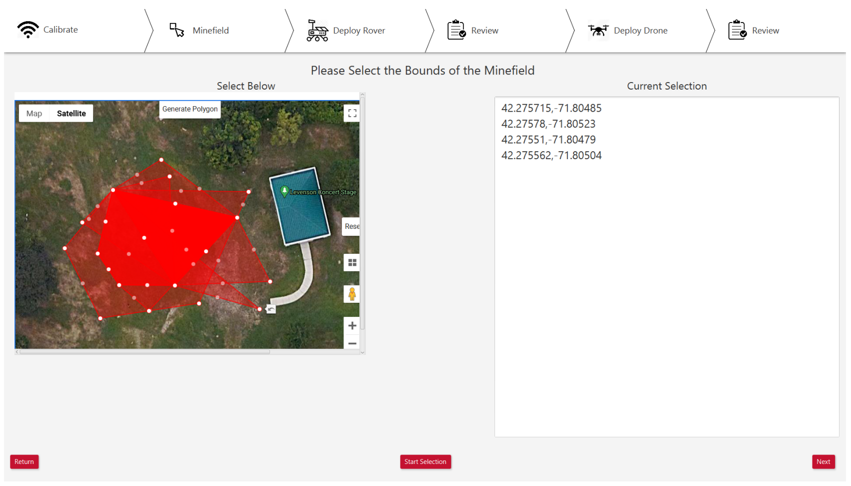
The dropper was tested by raising it to an elevation of 20' and actuating the revolving mechanism to drop on a pseudo landmine location. The exit hole was manually lined up above the center of the landmine, and each test was performed from the same position to determine precision. The payloads were water balloons, filled to the 1/2 lb metric identified to detonate the PMN-1 mines from 20'. The drops were filmed in slow motion to accurately identify impact locations, and the error was measured relative to the center of the test landmine.

3.4 Base Station Programming

3.4.1 User Interface

The User Interface (UI) was mostly designed and implemented during the previous year's project, with the Google Maps API having been implemented and integrated with the UI as well. However, it took some effort this year and contact between the current team and previous team in order to get the UI back to its previous functionality. A new Google Maps API account had to be set up, and the run configuration was extensively lengthened to properly load all needed modules for JavaFX and for the Google Maps API to load within the UI window.

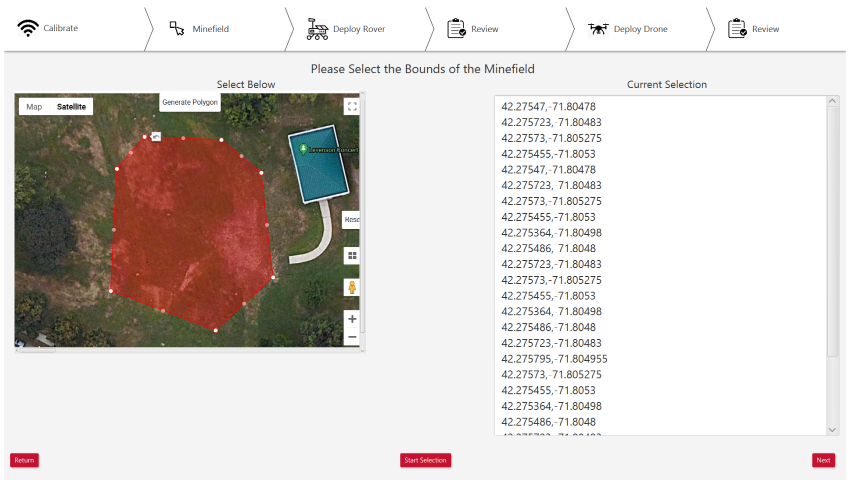
The next work that took place on the UI was checking for and fixing any bugs or errors observed. There were some minor bugs within the UI, and some larger ones within the Google Maps API. The UI had a few issues with unfinished buttons, for example, the "return" button after starting the UI to go back to the starting page didn't function correctly, and would open the window in the incorrect location without any navigation buttons. There was another button which simply hadn't had functionality added to it when clicked yet, the "Deploy" button on the "Deploy Drone" page. Then there were also larger issues with the Google Maps API, the two most notable being the ability to infinitely add polygons, and the duplication of vertices. When starting a selection of points to be the designated minefield area, the user drags a pointer to the various vertices, then hits a "Generate Polygon" button, which generates the vertices, lines between the vertices, and the enclosed area in a red color. The bug encountered was that once clicked, the button, as well as its functionality, remained on the page. This meant that the user could continuously click the "Generate Polygon" button, and multiple polygons would appear on top of each other, and when one was dragged, the others would not be moved. An example of this bug can be seen in figure 20. This would generally be confusing as it would: 1) cause the points in the "Current Selection" area to be incorrect, 2) make it very difficult to drag the correct points, and 3) cause significant problems when the data was fed to ROS for the rover pathfinding. To fix this issue, the functionality and the entire button are now removed as long as there is already a generated polygon in the Google Maps window.



42.275715,-71.80485
42.27578,-71.80523
42.27551,-71.80479
42.275562,-71.80504

Figure 20: Polygon Bug

The next bug was with the current selection area, when moving a point or adding a point, the selection would keep all previous points and add all of the points from the polygon again, creating many duplicate points as well as some erroneous points that had since been moved. An example of this bug occurring can be seen in figure 21, where there are seven vertices, but at least 22 points in the current selection area. This presented a major issue since the data in the current selection area is the same data that will then be sent to the ROS for rover pathfinding. Therefore, if this data is not correct, the rover will not be able to pathfind correctly, and the entire system fails before it begins. In order to fix this, the Google Maps API interface code and UI current selection code were extensively explored. The root of the issue ended up being the data sent from the Google Maps API to the UI, there were two separate arrays storing the vertices of the generated polygon being built simultaneously, however, one was building correctly, and the other was being built incorrectly, and would end up with many erroneous points in it. The second array was the one being sent as the point data to the UI, so this was changed to send the first array instead, since it contained the correct data. This solved the issue, and the second array was simply deleted from the code since it was only used for the purpose of sending the data back.



Please Select the Bounds of the Minefield

Current Selection

```
42.27547,-71.80478
42.275723,-71.80483
42.27573,-71.805275
42.275455,-71.8053
42.27547,-71.80478
42.275723,-71.80483
42.27573,-71.805275
42.275455,-71.8053
42.275364,-71.80498
42.275486,-71.8048
42.275723,-71.80483
42.27573,-71.805275
42.275455,-71.8053
42.275364,-71.80498
42.275486,-71.8048
42.275723,-71.80483
42.275795,-71.804955
42.27573,-71.805275
42.275455,-71.8053
42.275364,-71.80498
42.275486,-71.8048
```

Figure 21: Duplicate Vertex Bug

3.4.2 Rover Pathfinding

To allow for the rover to traverse the minefield designated in the Google Maps API, a pathfinding algorithm needed to be written. This algorithm was written to take in the ordered vertices from the API, then create a list of waypoints which the rover will travel to consecutively. The first version of this pathfinding algorithm was implemented to only process convex polygons. The algorithm for convex polygons uses two static values: the amount of overlap between rows to ensure all mines are caught and the width of the rover. The way that the algorithm works is by first finding the west-most and east-most points of the polygon. It establishes the line between these two points as the west-east line. It does this by finding the slope using $m = \frac{e_y - w_y}{e_x - w_x}$ and the y-intercept using $b = e_y - m * e_x$. Next, it finds all lines in the polygon, defined by two consecutive points in the array of vertices. It sorts these lines into upper and lower bound lines by checking whether either vertex connected to the line has a higher y value than the west-east line at the same x value. If either endpoint does, then the line is an upper bound, otherwise the line is a lower bound. However, this caused a problem whenever the west-east line happened to be an upper or lower bound: the west-east line would always be added to the lower bounds, even if it was an upper bound, leaving the upper bounds empty. This was fixed by adding a check after all lines had been added which adds the west-east line to whichever bound list (upper or lower) currently has no bounds within it, and makes sure the west-east line is not in the other list as well.

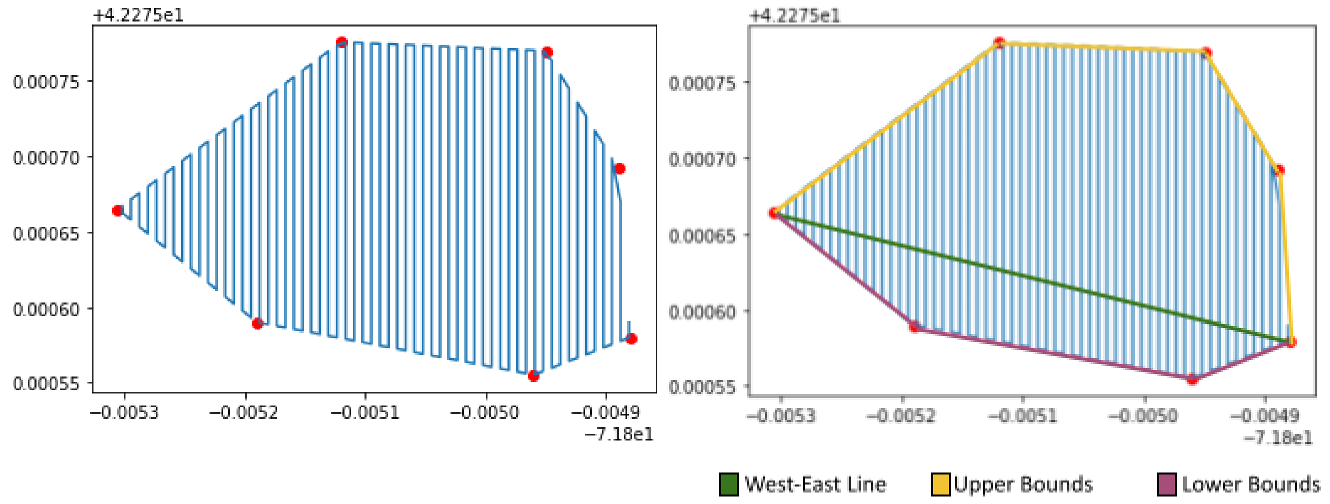


Figure 22: Visualization of Convex Rover Pathfinding Output (raw & with added visual aid)

Once the upper and lower bounds are established, the algorithm starts adding to the waypoints list. To do this, it starts just to the east of the west-most point, and while its x is smaller than that of the east-most point, it continues traversing the minefield. The algorithm has a boolean value "top" to keep track of whether the next waypoint should be at the top or the bottom of the polygon. For each row, the algorithm creates two waypoints, a top waypoint and a bottom waypoint. The top waypoint is set at the lowest y value of the upper bounds at the current x value, and the bottom waypoint is the highest y value of the lower bounds at the current x value. When "top" is true, the top waypoint is added first, then the bottom, and "top" is flipped to false, and when "top" is false, the bottom waypoint is added before the top, and "top" is flipped to true. After both waypoints are added, the width of the row $(width\ of\ rover\ *\ (1 - overlap))$ is added to the current x value, and the loop restarts. The output of this algorithm is then a list of waypoints, in order, which the rover will move to in order to search the entire field.

The second version of the rover pathfinding algorithm adds the capability to process concave polygons. It does this by splitting concave polygons into convex partitions and making use of the established algorithm to process convex polygons. There are three steps involved in processing the concave shapes: triangulation of the shape, partitioning of the shape into convex pieces, and finally taking advantage of the convex algorithm to process each of the partitions.

To triangulate the concave polygon quickly and effectively, a python library for triangulation was used called "mapbox-earcut". This library was developed by Samuel Kogler and is described by him as: "Python bindings for the C++ implementation of the Mapbox Earcut library, which provides very fast and quite robust triangulation of 2D polygons," (Kogler, 2020). This is the first step in generating waypoints for a concave shape since in order to make efficient partitions, the shape first has to be broken down into triangles since they are always convex.

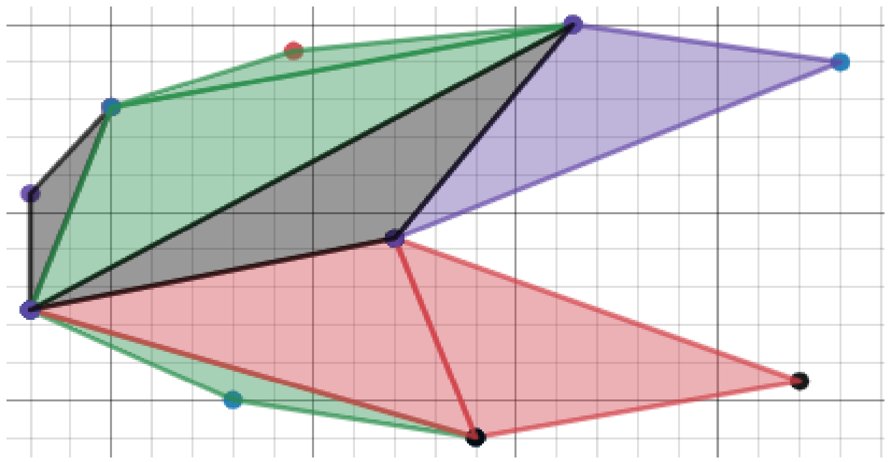


Figure 23: Visualization of a Triangulated Concave Polygon (using mapbox-earcut)

Once the shape is triangulated, the triangles can be combined to form larger convex shapes. This combination can greatly reduce the number of convex shapes that need to be traversed, which correlates to quicker and less tedious traversal of the field as a whole. This part of the algorithm was developed from scratch. To find the minimum number of partitions needed, an exhaustive tree of all possible convex partitions is made. This makes use of a recursive method and a node class, each node being one triangle in the fully triangulated polygon. For each node, this method checks the combination of itself and each adjacent node to check for convexity. If the combination of nodes is convex, it adds the combination to the possible partitions and recurses to try combining with the next set of adjacent nodes. It also simply recurses without combining the current node at all to make sure all cases are checked. Once there are no more nodes to check, it returns the smallest number of necessary partitions, and the previous recursion continues checking. It continues this until all possible combinations of convex shapes are searched and returns the set of partitions with the smallest number of polygons.

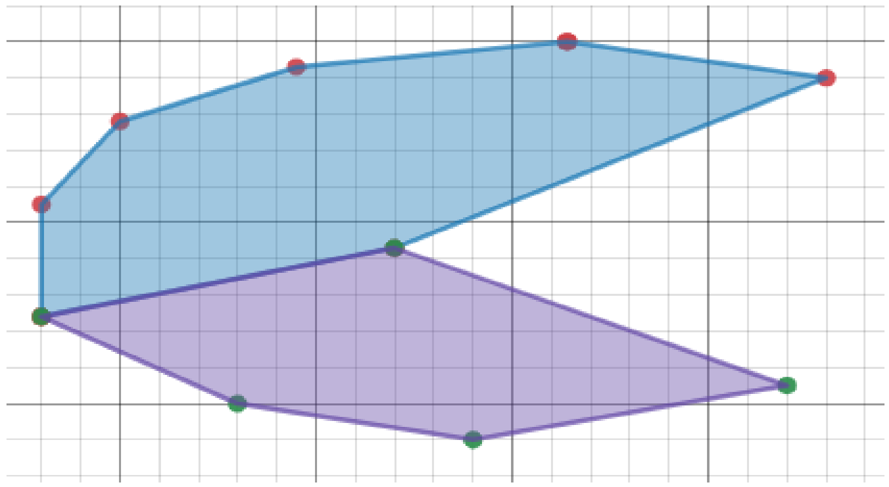


Figure 24: Visualization of a Concave Polygon Split into Minimum Convex Partitions

Finally, once the smallest number of convex partitions is found, each partition is put into the algorithm for generating waypoints for convex shapes one by one. This creates a set of waypoints for each convex partition which when put together traverse the entire designated concave field.

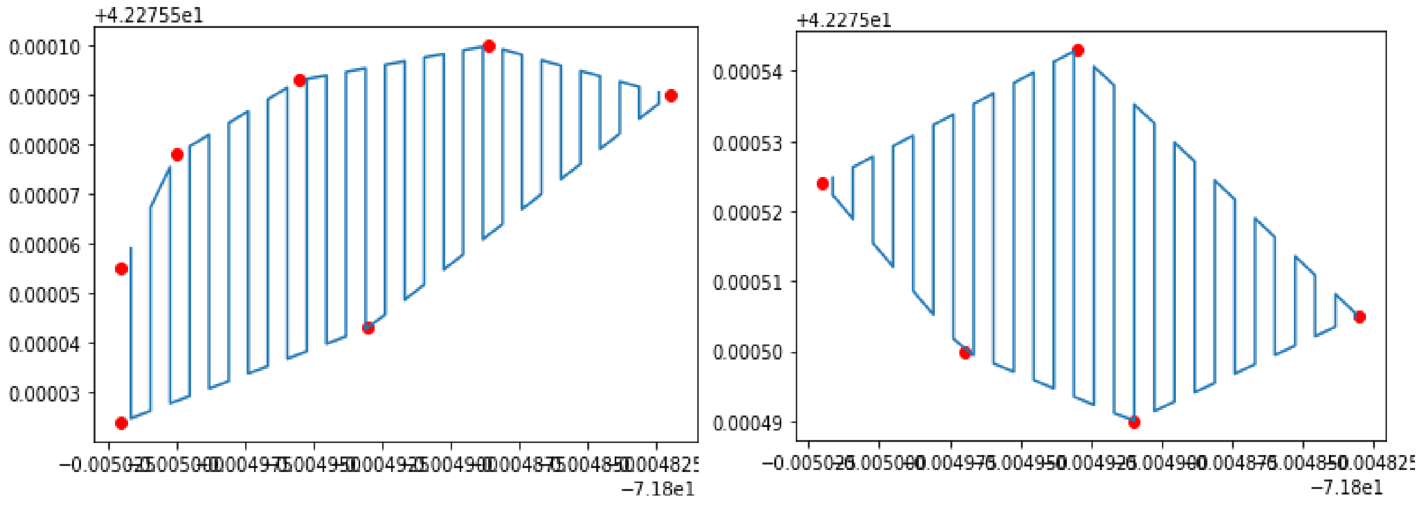


Figure 25: Visualization of Rover Pathfinding Output for a Concave Polygon Split into Minimum Convex Partitions

4.0 Results

4.1 Rover

4.1.1 3D Static, Motion, and Deflection Calculations

Our free body diagrams and results are denoted in Appendix D. From our analysis we have determined that the four-bar mechanism is more than capable in regards to enduring the forces it will be exposed to during operation. From the 3D static analysis the maximum internal force any of the members experience is 129.8 lbf. This takes place in member JL as shown in figure 26 when the angle of the link is 22.5 deg and the rover is rotating. When the system is not rotating the maximum internal force is 85.7 lbf, which takes place in members DF and JL. This is important because the linear actuators that we are using are rated up to 500N each. This means that if the rover is turning, the linear actuators would not be able to raise the metal detector, because the force on that member would be 577 N. If the rover is not turning though, it will be able to raise and lower the metal detector because it will only need to overcome an internal force of 381 N.

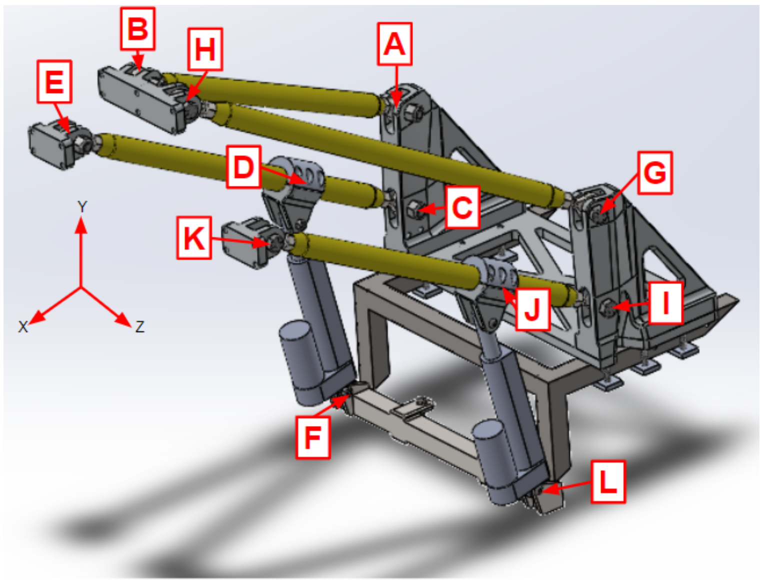


Figure 26: Joint Locations

The maximum reactionary force at any of the ball and socket pin joints is 129.8 lbf. This is the same as the maximum internal force because when the arm is set at 22.5 deg the linear actuator is pointed basically straight up. This is well within the capabilities of the steel structure. For the deflection calculation, we used the method of virtual work and found the internal forces and the unit forces in each of the members. The point that we were analyzing was the center of mass of metal detector linear slide. The deflection of the metal detector arm when the rover is not moving is 0.00075 in, and when it is turning it is 0.0018 in. This is a very small deflection and a great improvement over the previous design. In terms of attachment of the arm to the rover base, it is secure and well within the pull out strength of the bolts. The stress on the twelve bolts that attach the four-bar mechanism to the CERBERUS, shown in figure 27, is 391 psi, and the combined pull out strength of them is 110443 psi.

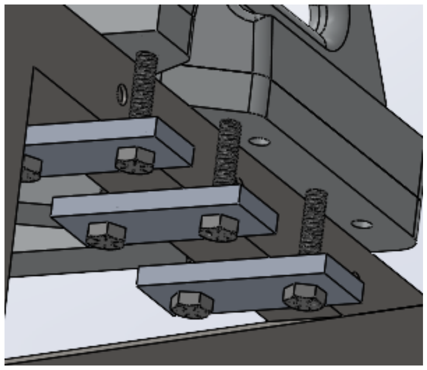


Figure 27: Attachment Bolts

We calculated the strain in each of the links, and the link with the highest strain still ended up being 0.000013. We calculated the strain by multiplying the internal force by the cross sectional area of the members, and then dividing it by the Young's modulus of elasticity for steel. Unfortunately, we did not have access to any strain gauges that were sensitive enough to give us an accurate enough reading with the strain being so low, so we were unable to test the strain with a strain gauge.

4.1.2 Electrical System Improvements

This year the primary goals for the rover electrical system focused around combining the electrical system from the Demining Phase IV MQP on the Husky base and the CERBERUS MQP electrical system on the Action Track base. This included both mixing the driving control systems and connecting the electrical components for the arm designed by the previous team into the new combined control system.

The drive system and power distribution system are the only parts of the rover electrical system still using components from the CERBERUS system. This is primarily because of the voltage requirements and the Talon SRXs used on the CERBERUS rover. On the Husky rover, last year's team was able to communicate directly from the Teensy board to the Jaguar motor controllers using PWM signaling. Since the Action Track rover has higher voltage motors and we are using Talon SRXs we needed to use the HERO board to communicate from the onboard computer to the drive motors instead of the Teensy. By combining the two systems we ended up with different power distribution requirements than either of the two previous electrical systems. And with the PD boards on the Husky base being part of their off-the-shelf hardware we had to modify the PD setup from the CERBERUS rover to fit the new requirements. This involved using two voltage converters to step down from the 24V battery supply to 12V and 5V, as well as using the 3.3V voltage output from the Teensy to supply some of the low voltage sensors.

The electrical wiring for the sweeper arm designed by last year's team was routed through a single bundle to bring the wires to the base of the arm structure and had no labeling to identify the purpose of each wire. We identified each wire already connected to a component of the arm and labeled them accordingly for easy identification. The wires intended for the electrical components of the metal detecting system are not currently connected and therefore have been left unlabeled until the metal detecting system is connected. We also used Anderson connectors at the base of the wire bundle leading to the main electrical box so that it will be easy to disconnect the arm from the rover base if needed (i.e. if the rover needs to be transported in two pieces).

Another goal for the rover electrical system was to improve durability and flexibility for future modification or maintenance. The flexibility was achieved by installing each electrical component with short lead wires with Anderson connectors on the ends. These are then connected to what are essentially extension cables, that can be easily made to different lengths, to reach the other end of the connection. This allows for the mounting location of a component to be easily changed without having to change the lead wires on the component itself.
The durability of the electrical system was improved in two ways. The first way was the replacement of the breadboards being used in the electrical system with soldered connections on perfboard. This makes the connections more secure and less at risk of coming loose due to vibrations from driving. Another way the durability of the electrical system was improved was in terms of the protection of the mounted electronics. The electrical system from the CERBERUS rover was mounted in an open top box and was not well protected from the elements which is important for this project since it is meant for outdoor use. The previous Demining team was also focused on protecting the electrical system from the elements and had purchased a Nanuk case to house the electronics. Unfortunately, when our team inherited the project, we were unable to locate this mounting case and decided to create our own mounting box for the Action Track base. The 3-D printed electrical mounting box shown in figure 28 will enclose all of the electrical components mounted in the base of the rover and has an external holding slot for the onboard computer so it does not overheat.

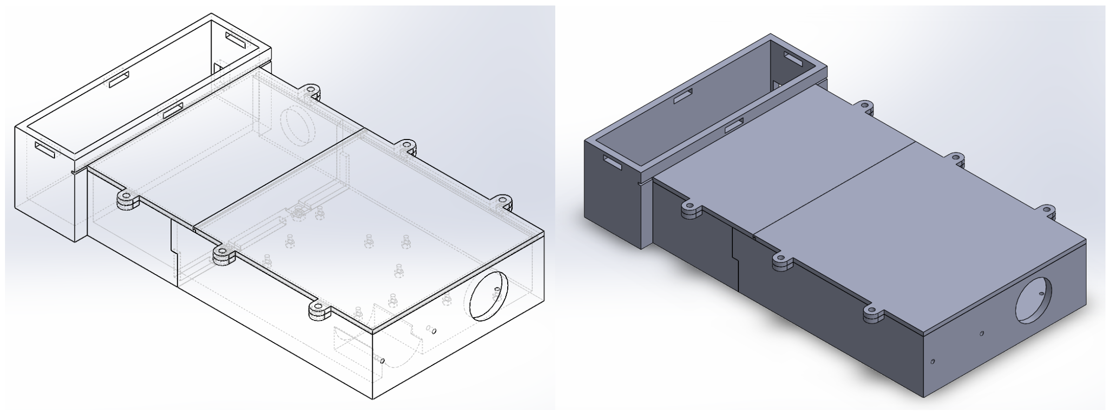


Figure 28: Electrical Mounting Box CAD

4.1.3 Rover Driving

We did not have the resources to complete the Rover ROS. A Git repository of drivetrain functions, map maintenance, and navigation written in ROS was added to the base station, based on RBE 3002 Turtlebot work. However, its adaptation to the Rover saw little progress. Lack of sensors were major barriers to its development. The CERBERUS base did not have encoders installed and solutions to this were mechanically demanding. Additionally, the RTK was not installed for precise GPS, and there was no IMU or LiDAR either.

LCD displays were added to the xBee controller and the XBee receiver to display the analog stick output. Initial intentions for establishing serial communication between the rover and a rover controller were going to be handled by a set of paired XBee controllers. When running the XBees between two arduinos they are able to establish a network, however there were limitations when trying to establish a network between an arduino and a teensy.

The XBee was updated to be commanded by a Teensy 3.5 opposed to the Arduino Uno used in previous tests cause some issues when trying to establish a stable network. Figure 29 shows a controller designed to house the rover controller electronics. This controller houses an XBee unit, analog stick, Arduino uno, two buttons and a LCD screen. However our resources were reallocated to prioritize other portions of the project so the controller was not printed.

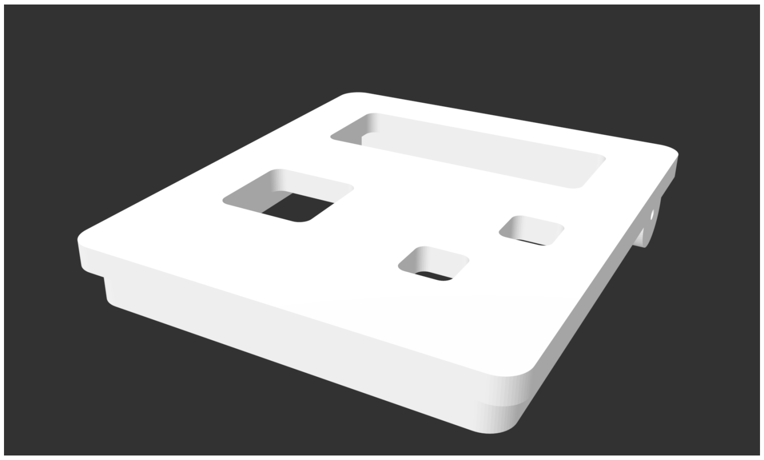


Figure 29: Controller Solidworks Model

The team successfully achieved manual control of the rover drivetrain through the Hero development board's NETMF capabilities. The Hero and Talons were successfully updated, and a simple one-stick drive program was written in C#. The motors speeds are given by the y-axis value of the stick +/- the x-axis for the left and right motors, respectively. The full code uploaded to the Hero can be found in appendix F. There is also an e-stop built into this control system. The communication mode of the Logitech controller can be switched from HID to Direct-X to immediately discontinue Hero board level function.

The drive train is extremely competent and is able to complete all necessary aspects of our mission from a driving standpoint. The drivetrain was tested using the manual control provided by the Hero. The team needed to put a cap on the velocity of the rover to maintain control and precision, but if the rover got itself in an unsafe position, the operator, with manual driving, would be able to very quickly navigate the rover out of harm's way.

With the arm attached and the metal detector installed, the metal detector remains stable while the rover is in motion. The rover has full mobility as it has no difficulties turning with the center of mass located only 2.5 inches in front of the center of rotation. In addition, the rover was tested on inclines and it did not show any difficulty in going up or down a hill, as well as turning on a hill. This is important because as the project continues on, this will allow the user to define a landmine field that contains a hill.

4.1.4 Testing Against Acceptance Criteria

Reflecting on the acceptance criteria for the rover understanding the project more fully, they were certainly ambitious. The rover did not fully accomplish any of the three main goals set for it. They were: for the rover to have the mine detecting apparatus attached and functioning on the new base, to have the low level C/C++ code completed for driving and sensor control, and for the ROS to communicate with the base station and manage the low level systems.

The rover does have the four-bar apparatus attached, but it is not electrically integrated. The rover does have manual drive capabilities, but it is not controlled through the originally planned system and the sensors for the system are either not integrated or non-existent. The ROS network has been outlined and starter code has been supplied, but the adaptation to the system has not been completed.

4.2 Drone

4.2.1 Flight Testing

4.2.1.1 Preliminary Flight

The Tarot T18 from UAV Systems was extremely delayed in its arrival. It had been scheduled for delivery in late September, the later half of A term. However, due to significant delays from UAV Systems, the drone only arrived in early December at the very end of B term. The team cooperated with two WPI students, Connor Miholovich and Jeremy Trilling, who have extensive autopilot and drone experience for the first flight test. The test was performed in Institute Park. The drone successfully took off and hovered, but an unknown error caused the loss of manual control and the drone safely autonomously landed. This error was later inferred to be a compass calibration issue that continued in our future testing. A rover unit using an autopilot board was used instead to learn MissionPlanner waypoint path planning, and successfully navigated around a simple course.

4.2.1.2 Autonomous Flight

The team developed a Pre-Flight Checklist for autonomous flight tests that we used to ensure the safety of the equipment and our team when flying. This checklist is located in Appendix E.

Connor Miholovich continued to work with the team, and assisted in the FAA clearances and provided his expertise. Flight tests were performed at Green Hill Park. The team also worked with Meredith Merchant in the WPI athletics department and Deputy Marsh from the WPI Police to gain permission to fly the drone on WPI's athletic fields, though no tests ended up being performed on them.

The continued tests yielded similar results to the preliminary flight test. Manual control was used to lift off, but as soon as the drone began to move away from its take-off location, manual control was lost and the drone autonomously landed. When attempting autonomous flights, the drone took off, hovered, then landed. The compass was identified as the issue, but the team was unable to complete a full calibration and continued to receive the error "Compass calibration error -1". The drone was initiating this autonomous safety protocol because its safety faults recognized that it would not be able to navigate back home in an emergency.

The team successfully calibrated the compass once during a test on March 7th, but because of time constraints was only able to perform manual flight tests. The drone performed well on basic maneuvering.

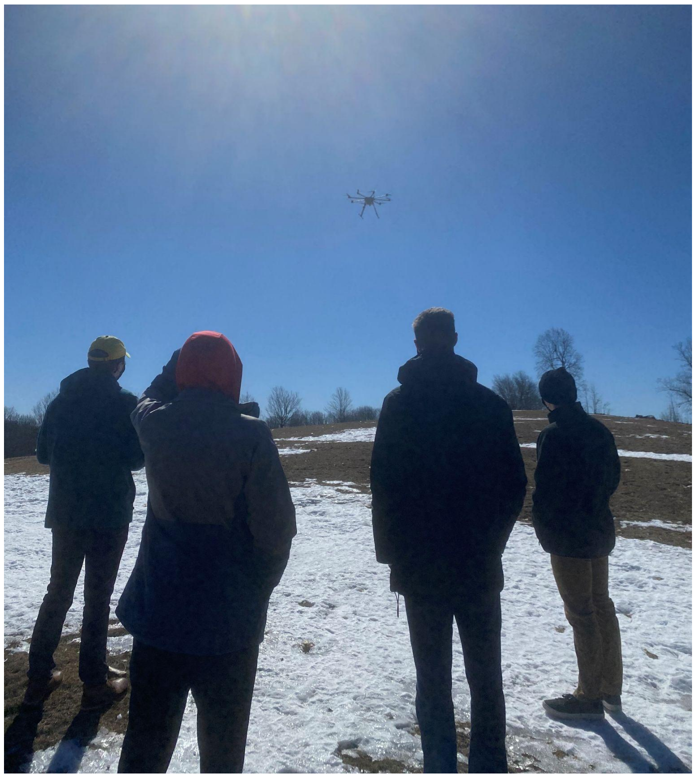


Figure 30: Successful Compass Calibration Drone Flight at Green Hill Park

The Tarot T-18's compass needed to be recalibrated for the next flight however, and the team was not able to replicate our success in the next and final pre-reconfiguration flight we attempted. Reflecting on the successful compass calibration, we are unsure as to whether it was actually accomplished or if the error was somehow avoided, since the compass should not need to be recalibrated before every flight.

4.2.1.3 Post-Reconfiguration Flight

After the team attempted to install the Pixhawk 4 and then reinstalled and reconfigured the mRo X2.1, tests to integrate the mission control system were attempted. However, the compass calibration issue continued and an additional error "need mavlink2 for item transfer." was shown as well.

Final testing attempts resulted in resounding success! After contacting UAV Systems for assistance, the team determined the mavlink2 item transfer error was non-critical. The team additionally solved the compass calibration issue by plugging a laptop into the mRo and calibrating the compass through MissionPlaner over a wired connection. The drone was put through much more rigorous maneuvering tests, and fared mildly well. When doing some tighter turns, it did seem slightly unstable, but never dangerously so. The only challenging and critical maneuver necessary for the mission is the prompt escape of the drone to a safe location after a drop. It was able to complete this well under manual control.

The drone also successfully lifted and dropped water balloons. The weights of the payloads were not an issue for the drone's flight, nor were the shifting of weights due to their deployment.

4.2.2 Electrical Systems Update

A major portion of the drone's upgrades was the integration of the drone's onboard microprocessor, allowing us to actuate the dropping mechanism and establish communication between the base station and mRo X2.1. Figure 31 shows the drone with the configuration we got out of the box from UAV Systems.

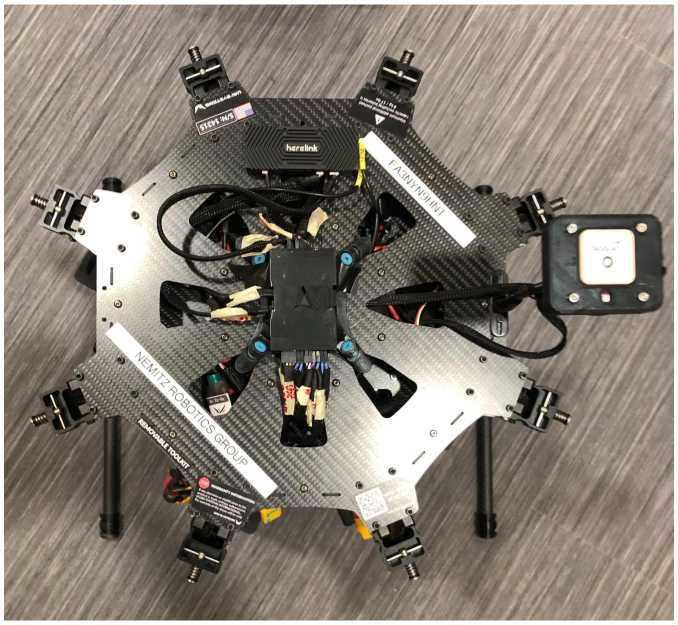


Figure 31: Drone Electrical System from UAV Systems

After getting the drone, the team decided to map out the voltage requirements for each system and then draft the diagram in figure 32 to account for all necessary components. One 11.1V power supply using a 5V voltage converter was able to power the Teensy 3.5 microcontroller, dropper mechanism and telemetry radio.

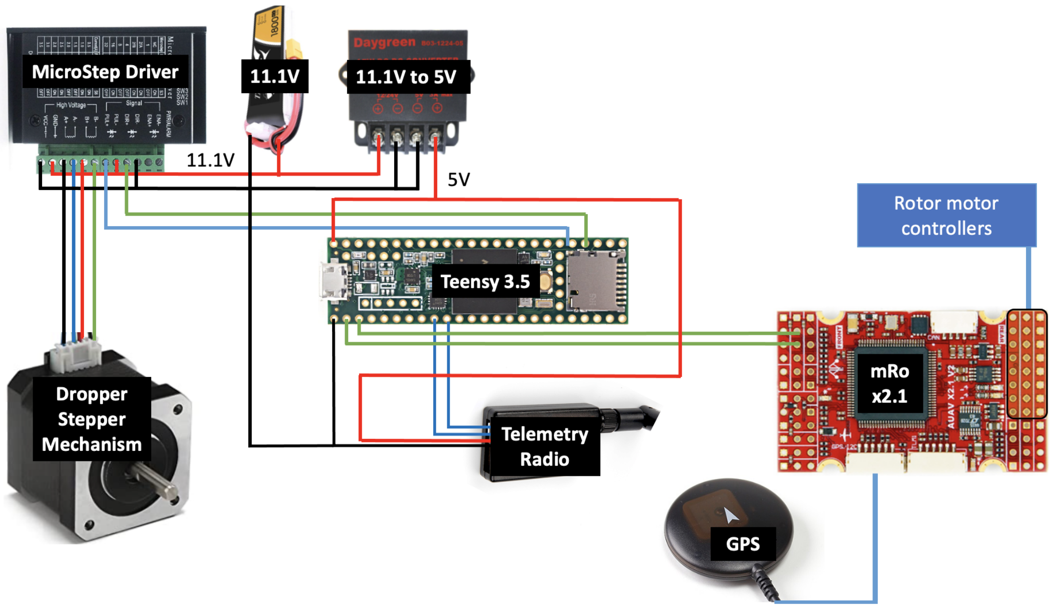


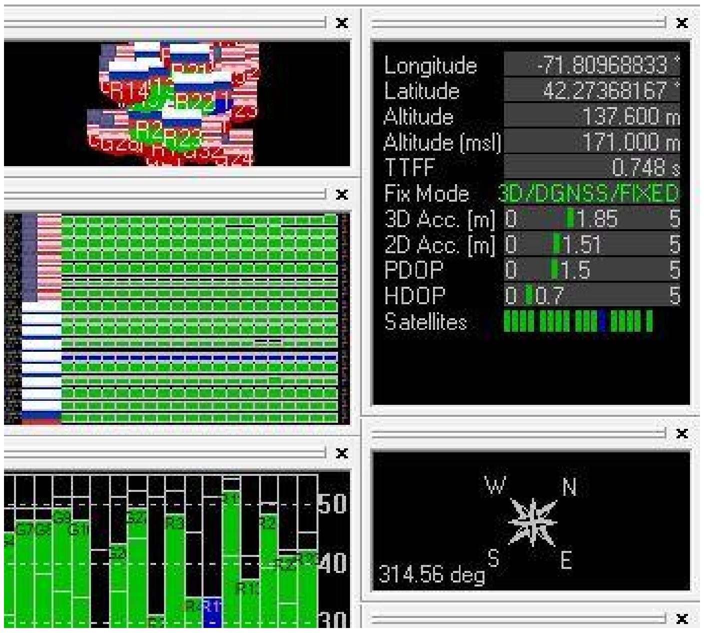
Figure 32: Drone Electronics System

After testing each of the modules individually to ensure they were getting the correct voltage, the team created perf boards for permanent mounting of the electrical systems. FThe drone electronics are housed inside of the mounting structure that attaches the dropper mechanism to the drone.

4.2.2.1 RTK Unit Installation and Snow Drone Team Assistance

The RTK installation on our drone was not completed. The RTK modules were configured, and the base station successfully found a TIME/FIX position estimation. However, we never got a rover module to connect to it. Since any board can be configured as a base station or a rover, we tested multiple combinations of configurations, all to no success.

Working with the Snow Drone team, their fresh C94-M8P boards successfully achieved a TIME/FIX lock as a base station and connected to it as a rover with a 3D/DGNSS/FIXED connection.



```
Longitude       -71.80968833 °
Latitude         42.27368167 °
Altitude            137.600 m
Altitude (msl)      171.000 m
TTFF                  0.748 s
Fix Mode     3D/DGNSS/FIXED
3D Acc. [m]  0    1.85      5
2D Acc. [m]  0   1.51       5
PDOP         0   1.5        5
HDOP         0 0.7          5
Satellites
```

Figure 33: Screenshot of the u-center interface from the RTK fix achieved by the Snow Drone modules

The connection took about 15 minutes to achieve after one of our reconfigurations, and was completed in the middle of the Quad on the Worcester Polytechnic Campus on a clear day.

Our team ended our efforts on the RTK boards when we discovered that they would not be able to interface with the mRo X2.1 flight controller on our drone without more significant effort. The mRo board only has three UART ports, as compared to the original design which accounted for the Pixhawk 4's eight UART ports. All of the UART ports on the mRo board were occupied with other peripherals. When the implementation of the RTK boards concluded, the next steps were to investigate possible discrepancies in the radio modules on the boards and their baud rates through the 9-pin serial connection, since they may have been altered by last year's team.

The Snow Drone MQP team was using a Pixhawk 4, and so was able to continue their implementation. We gave them the USB to 9-Pin cable, and instructed them on the process of using AT commands to update the C94 module with it. Their efforts concluded with successful field tests of the precision of the units, but they did not finish integrating it with their Pixhawk 4. We are cooperating with them to provide a guide for the continued integration of this unit for both our projects.

4.2.3 Dropper Tests Comparison

4.2.3.1 Initial Dropping Mechanism Tests

The accuracy of the drops were measured by taking slow motion footage of the water balloons as it impacted the pseudo landmine as seen in Figure 34. Distance from the target landmine was recorded by reviewing the footage and using a tape measure.

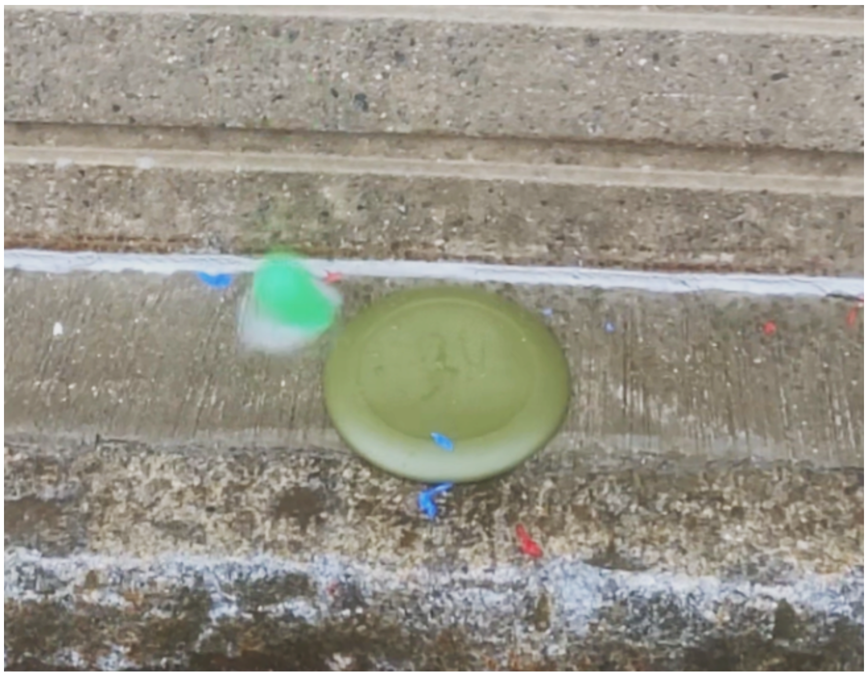


Figure 34: Slow-Motion Frame of Payload Impact (Updated Dropper, balloon #8)

The first dropper tests were completed in Early December. They were to gauge the accuracy of the dropper when a water balloon was dropped from 20 ft. Since the electronics were unfinished, the revolver was actuated manually as consistently as possible. Figure 35 shows the locations of eight water balloons that were dropped and how far away they were from the target location.

Test #:	Distance from Mine (in):
1	7
2	5
3*	0
4	5.5
5	6.5
6	0
7	3
8*	2.5

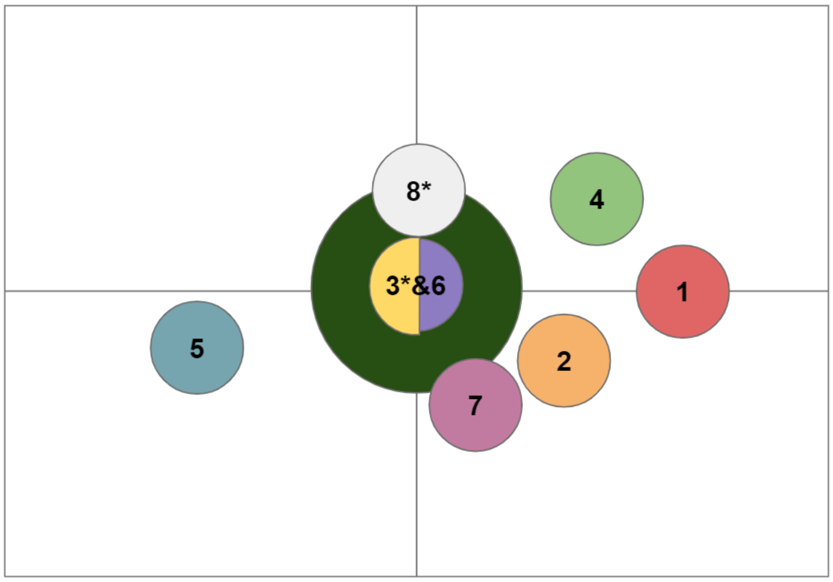


Figure 35: Initial Dropper Test Landing Spread

Water balloons 3 and 8 have an asterisk next to them because each had complications when being deployed. During test 3 the water balloon got jammed between the rotating fins and the exit hole of the dropper and test 8 hit the wall of the garage before reaching the ground.

These tests indicated that the rotational velocity of the revolving mechanism imparted a horizontal error onto their descent. Additionally, the stored asymmetric water balloons had difficulty rolling as the mechanism rotated, and were in varying orientations upon their release from the dropper. Their aerodynamics are inconsistent as well. To improve upon the initial design we aim to compensate for the variable rotational velocity by using a stepper motor to have a more consistent initial placement of balloons to improve consistency.

The initial tests for the dropper mechanism showed that 3 of the 8 water balloons dropped were on target giving an accuracy of 37.5%.

4.2.3.2 Updated Dropper Tests

Figure 36 shows the locations of eight water balloons that were dropped during the second dropper test and their spread from the desired target location. The updates were to the mechanical structure for the revolver and lid, and this time the actuation was controlled by the stepper motor.

Test #:	Distance from Mine (in):
1	4.5
2	3
3	1
4	4
5	4
6	4
7	3.5
8	5

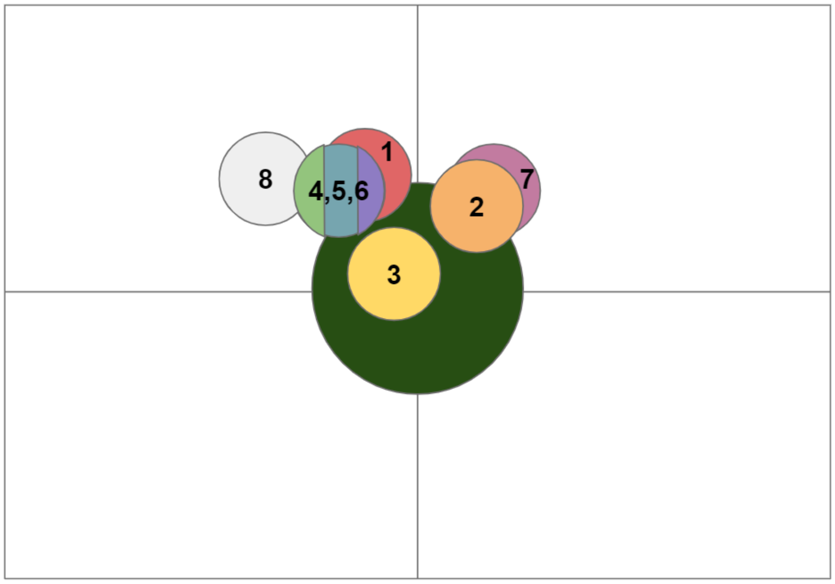


Figure 36: Updated Dropper Test Landing Spread

These tests encountered two jams where the water balloon got stuck part way out of the exit hole. In these instances, we reset the test. None of the balloons got stuck in the revolver as they had previously.

The second test of the dropper mechanism showed that 6 of the 8 water balloons dropped were on target giving an accuracy of 75%. This is double the accuracy of the previous test. The second test shows that the implementation of the microstepper driver allows the stepper motor to have a more consistent dropping spread.

4.2.4 Testing Against Acceptance Criteria

The drone system almost meets the acceptance criteria. The drone is able to communicate with the base station application, but did not progress past the setup of the mission control system. The dropper is run off of a simple application uploaded onto the onboard Teensy microcontroller. The control code for the dropper mechanism was not integrated into the flight control program on the drone. Once this issue can be resolved and the dropper is integrated into the flight control program, the drone should meet the mission control goal. The dropper can deploy up to six payloads, with an accuracy of 75%, This means that the dropper subsystem does meet its goal requirement. The drone achieved successful manual flight and payload deployment from as seen in Figure 37.

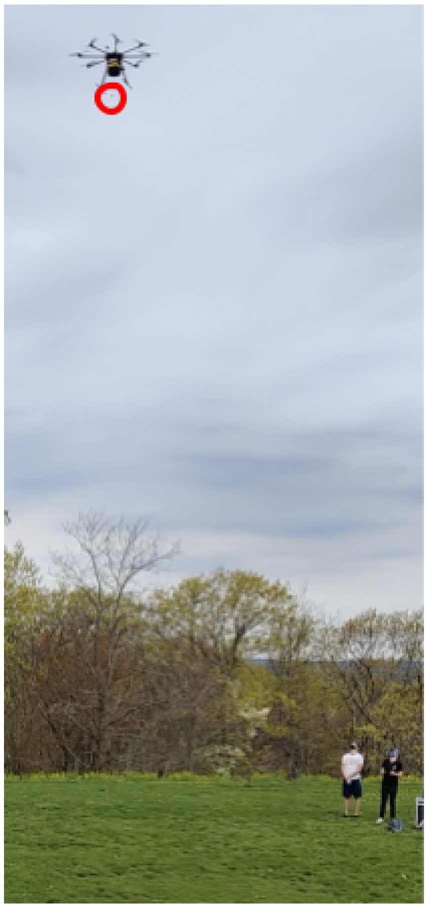


Figure 37: Drone Deploying the Payload Mid Flight

4.3 Base Station

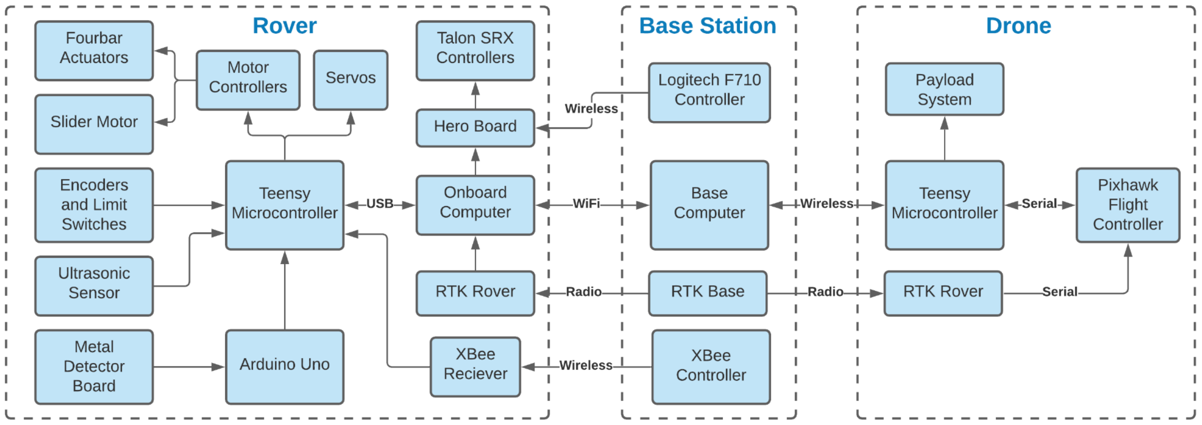


Figure 38: System Communications Diagram

4.3.1 User Interface

Once the User Interface and Google Maps API were configured and running completely as they were after the previous teams work, testing of the User Interface started. During the testing of each element of the interface, it was discovered that the UI had a few bugs, that some buttons on the interface were missing any basic functionality despite them being simple to implement, and that some pages were not displaying as they should. On the minefield designation page, there was one error with the data being sent back from the Google Maps API which made the window display duplicate vertices over and over, and another error which allowed the user to generate multiple polygons in the Google Maps API which would overlap and cause issues.

After discovering and fixing the bugs within the User Interface and Google Maps API, there were no future issues when testing the User Interface. All pieces of the User Interface which have coded functionality currently work as intended. The rover “Deploy” button functions as intended, but does so slowly, the reason for this is currently unknown.

In an attempt to integrate the User Interface with the Base Station after testing it’s functionality, methods for allowing Java with JavaFX to communicate with ROS nodes were researched. There were two promising options which the team attempted to implement. The first tool was rosjava, which allows Java executables to be run as ROS nodes when normally this only works with C/C++ and Python executables. Ultimately, issues with JavaFX compatibility with rosjava and the Java version needed for the Google Maps API prevented this from being possible (JavaFX is only compatible until Java 8, however, the Google Maps API requires a later version to load). There is very little documentation for rosjava, and none for using JavaFX with rosjava, so this made any progress guess-work. Similar issues persisted with jrosbridge, lack of documentation and compatibility with JavaFX caused the team to move on to focus on more essential project work despite being close to getting it functional.

4.3.2 Rover Pathfinding Algorithm

The original implementation of the Rover Pathfinding Algorithm resulted in erroneous waypoints whenever the west-east line belonged to the set of upper or lower bounds, and when the polygon contained a perfectly vertical (north-south) line. The bug with the west-east line was due to the fact that no matter how it was checked, if the west-east line belonged to one of the bounds, it would always be added to the upper or lower, despite the possibility for it to be either. This was because of how the upper and lower bounds were populated, checking whether either vertex was above the west-east line, since the west-east line has no vertices above itself, it would always be added to the lower bounds. The bug with vertical bounds was caused by trying to calculate the slope of the line by computing $m = \frac{e_y - w_y}{e_x - w_x}$ when $e_x - w_x$ is equal to zero, this would cause an error due to trying to divide by zero.

After these bugs were fixed, multiple convex and concave shapes were tested. However, these testing efforts did not end up being very extensive due to other deliverables being prioritized since the Rover Pathfinding Algorithm was not planned to be integrated until future years work. It was found that all of the polygons entered successfully generated non-erroneous waypoints for the rover, however, the number of polygons tested was only 15. During these tests, it was found that the concave algorithm performs poorly with increased numbers of boundary points, so they should be limited whenever possible. This is due to the partitioning algorithm needing to create an exhaustive tree of all possible partitions, which grows exponentially as the number of boundary points increases. Despite the low number, the team believes that all potentially problematic conditions have been tested and were successful after bug fixes. Since the Google Maps API does not allow for creating polygons which contain holes, this does not pose an issue for the Rover Pathfinding unless future teams decide to implement the ability to incorporate holes into designated minefields for geographical features such as a rock formation or a small body of water.

4.3.3 Testing Against Acceptance Criteria

Reviewing the acceptance criteria reveals that we wished to have the system much more integrated than it ultimately was. In part, this was due to shortcomings with the base station. We were not able to implement the rover communication code, and we were also not able to set up the User Interface to communicate with the drone and rover comms. The rover communications code was not realistic to develop until the rover could be driven autonomously, so this portion was stalled before any work could be done. After spending much time attempting to get the User Interface to communicate through ROS, we eventually had to cut our losses to focus on other more pressing deliverables that were not roadblocked. We did, however, meet the criteria for the

Rover Pathfinding Algorithm; we were just unable to implement it into the system yet because of complications with the autonomous driving of the rover and lack of rover communications code.

5.0 Conclusion/Discussion

5.1 Rover Discussion

The transition to the CERBERUS MQP's Action Track rover base from the Clearpath Husky A100 sacrificed short term functionality for the long-term robustness of the system. The mechanical and electrical work involved in moving the critical DAS system was vast, and deprecated many efforts from previous MQPs. We anticipate a significant challenge will be the ROS control for the new rover. The Husky platform had a pre-written ROS library developed for it by Clearpath. Not only will future teams have to write a new one from scratch, but the CERBERUS rover had been originally designed to be driven manually, and does not have encoders on the drive motors, LiDAR, or any other sensors needed for autonomous navigation yet.

Our team additionally faced electrical challenges because of the structural changes being made to the rover base. Many electrical systems' installations were delayed because of welding that needed to be done on the rover chassis that would have damaged them had they been installed.

Because of our relatively smaller team size compared to previous DAS MQP phases, we did not have the resources to test the sensor systems involved in identifying mines. Some resources that may be useful for future teams in this endeavor are the ADI-MassRobotics Sensor Fusion Challenge, which this year's team has already created a substantial body of work for, and a possible collaboration with Cornerstone Research Group which is working on a related landmine searching project.

5.2 Drone Discussion

Upon initial purchase, the drone was supposed to arrive near the end of A term. However, after complications with UAV Systems the drone did not arrive until the end of B term. After arrival, the team had trouble interfacing with the flight controller board, and could not find the model of Pixhawk that was installed on the drone. After reaching back out to UAV Systems about this, we learned they had not installed the Pixhawk 4, and had instead used the mRo X2.1. The limitations of the mRo board are discussed in section 3.3.2.2. The board has basic functionality, but struggles to support the integrated system of this project.

We additionally had significant struggles with the mission control system between the base station application and the Teensy. After cooperation with the original author of the code, Andy VanOsten, the issue was finally identified, but there still remained no solution. The base station had been receiving its own communications, and the Teensy became stuck early on in its setup loop. We suspect this has to do with the Teensy not getting the information it needs from the mRo board, and may be specifically tied to the setting of the home location which requires compass calibration. This issue was only solved at the very end of the project and the comms were not tested again. The serial MAVLink communication with the mRo must also be confirmed, though the mavlink2 item transfer error may not be critical, it may impede Teensy-mRo communications. However, after these few linchpin issues are resolved, the system should be completely functional.

Finally, the dropper mechanism is not a robust solution. It works basically, but the high precision of the RTK is made irrelevant by the high inaccuracies of the dropper. Additionally, a payload capacity of six is relatively low. Possible redesigns could incorporate a "tray" of dartlike payloads dropped from a higher altitude. However, the decision to pursue this must be made while balancing the value of this improvement against the time and resources it would take to redesign this subsystem, especially when the current dropper is functional, even if it is not elegant.

5.3 Base Station Discussion

The choice of Java with JavaFX for the Base Station User Interface made any attempts to integrate it with the ROS communication nodes very difficult. JavaFX is no longer natively supported by Java, which means that Java 8 is preferred due to its native support for JavaFX. The issue with Java 8 later became evident though, as the team discovered that a newer version of Java is now required to load the WebEngine needed for the Google Maps API. Until the Java version was updated, the Google Maps API would simply error. This made the UI require a complex run configuration to function. It also meant that integrating the UI was much more complicated since we had to investigate solutions like rosjava, which would allow a .java file to run as a ROS node, or jrosbridge, which would allow a .java file to publish and listen to ROS topics. There was no documentation for using JavaFX with either rosjava or jrosbridge, meaning that once the team exhausted reasonable ways to attempt communication, we were forced to move on. These attempts ended up costing the team a lot of time with no real results to show for them.

5.4 Future Work for the MQP

5.4.1 Drone

5.4.1.1 Communication between Drone Electronics

Future work for the drone should start with the replacement of the mRo X2.1 board with a Pixhawk 4. UAV Systems has personnel who can help create a guide to install the Pixhawk 4. This will allow the drone to have access to a port for the RTK integration. The baud rates of the radio modules on the C94 M8Ps should be confirmed and updated as necessary using a USB to 9-pin serial cable and AT commands. UAV Systems shared a brief list of references that may aid with the integration of the Teensy microcontroller, but were unable to provide further assistance. The mission control system between the base station application, Teensy, and flight controller also needs to finish being debugged. The code had previously managed the flight of a drone in operation, so once the few errors existing are fixed it should be almost fully operational.

5.4.1.2 Dropper Mechanism

The current method for actuating the dropper mechanism uses a 12 tooth gear attached directly to the stepper motor, however this driving gear is only supported on one side of the axle causing the revolving mechanism to occasionally skip when dispensing water balloons. This can be fixed by redesigning how the revolving mechanism is driven.

The dropper actuation code needs to be implemented into the main flight code. Future teams should test the accuracy of the dropping mechanism when in flight and see how the constantly changing weight affects both the drone flight and dropper accuracy, and look into possible changes to the payloads and the deployment system.

Currently the payloads that are used to detonate the mines are water balloons, however they tend to cause jams in the dropper mechanism due to their malleability and high friction of the rubber. Future teams should look into payload deployment of "stress ball" like sand bags to reduce the chance of jamming.

5.4.2 Rover

5.4.2.1 Rover Mobility

To achieve fully autonomous driving capabilities of the CERBERUS Rover, encoders must be installed. With the encoders on the drivetrain axles, information on the speed and direction of the motors will be available for the program to use to guide the rover along its designated path. To bring the center of mass of the rover closer to its center of rotation, the weight of the linear slide system at the end of the four-bar mechanism can be reduced. The weight can be reduced by removing more of the material from the aluminum plate that supports the linear slide rails. With the center of mass moved closer to the center of rotation, the rover will have less resistance and improved capabilities when turning.

5.4.2.2 Rover Electrical Systems

Due to time constraints and minimal documentation from previous MQPs we were unable to finish installing the electrical components of the metal detector as well as the ultrasonic sensors that will be used to keep the metal detector level relative to the ground. These systems are included in the rover electrical schematic and will need to be installed accordingly by a future MQP. Another electrical component that a future team should consider is encoders for the drivetrain, as these will be necessary for implementing autonomous driving of the rover. Part of this challenge will be determining a mounting solution for the encoders since the drive motors' transmissions are in a fully enclosed housing.

5.4.3 Base Station

5.4.3.1 User Interface

The 2019-20 and 2020-21 teams both used IntelliJ to run the User Interface (which is in Java). To run the User Interface with JavaFX, since compatibility for JavaFX was removed after Java version 8, the 2020-21 team used these VM options in the run configuration to access all necessary JavaFX modules:

```
--module-path "C:\Users\mlsha\Documents\GitHub\MQP_DAS_20-21\javafx-sdk-11.0.2\lib"
--add-modules javafx.graphics,javafx.controls,javafx.fxml
--add-exports javafx.graphics/com.sun.javafx.util=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.sg.prism=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.scene=ALL-UNNAMED
--add-exports javafx.base/com.sun.javafx.logging=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.prism=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.glass.ui=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.geom.transform=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.tk=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.glass.utils=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.font=ALL-UNNAMED
--add-exports javafx.controls/com.sun.javafx.scene.control=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.scene.input=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.geom=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.prism.paint=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.scenario.effect=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.application=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.text=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.scenario.effect.impl.prism=ALL-UNNAMED
--add-exports javafx.graphics/com.sun.javafx.iio=ALL-UNNAMED
```

Figure 39: VM Options for UI Run Configuration

This should be added to the VM options section of the run configuration after changing the module path to the appropriate location. The JavaFX SDK is located in the github repository, so it will be there when cloned. There is also a hardcoded file path on line 129 in MineFieldController.java that will need to be amended (this also leads to a file that is in the repository). Most likely, a new Google Maps API key will have to be generated, which will replace the previous key on line 232 of GoogleMapSelect.html.

After set-up is complete and the User Interface and Google Maps API are running, the next steps for the User Interface are to integrate it with drone and rover communications through ROS and then add functionality to all buttons which should send data to the drone and rover communications. There is currently some starter code in the UI from the 2019-20 which attempts to communicate with ROS but did not end up helping the 2020-21 team, however, it may be useful to look into first. Attempts were made to establish communication during this project using rosjava and jrosbridge. Jrosbridge ended up being the more promising of the two, but still suffered from minimal documentation. The team encountered issues with compatibility of JavaFX with Maven, which would make implementation of jrosbridge much simpler. Ultimately, the team moved on before fully exploring jrosbridge, so this would be a good avenue to explore next year. If this also proves fruitless in a similar vein to rosjava, the nuclear option of rebuilding the User Interface without JavaFX or potentially even in a ROS supported language (C++/Python) becomes more viable. A much simpler potential work-around to having to rebuild it, however, would be to have the UI write to a file which the ROS node then reads from and vice versa.

Once communication through ROS topics is established, functionality can be added to the "Deploy" and "Abort" buttons. The deploy button for the rover should send the boundaries of the minefield as well as the deploy message for the rover. The deploy button for the drone should just send the deploy message for the drone as the mine locations should already be known by the base station. The abort buttons should simply send an abort message which the drone/rover communications will pass along to the drone/rover.

5.4.3.2 Rover Pathfinding

The Rover Pathfinding code can be found in Appendix D, and is in Python. There should be no issues running the code as is after the required libraries are installed. Once it runs, it should be further tested to verify success, especially in any boundary cases. To test a polygon, all that needs to be done is have a defined polygon (2D array of shape [n,2] where n is the number of vertices), and pass the polygon into the wpoints(points) method as the "points" argument. However, the points should be ordered (y,x) since the algorithm uses the format (lat,lng).

In its current state, the Rover Pathfinding Algorithm should be sufficient for any reasonable minefield designation. However, the concave performance rapidly decreases with an increasing number of boundary points. This is due to the partitioning of the concave polygons, which has exponential growth due to the exhaustive tree of all partitions. This could be solved by utilizing a spiral traversal instead of row traversal if necessary. In a spiral traversal, the rover would traverse each boundary line, then increment inward toward the center of the shape and traverse just inside of each boundary line, continuing to spiral inward until the whole field had been checked for mines. This would inevitably come with its own set of challenges to overcome, but would improve performance of the Rover Pathfinding Algorithm. It would also be a single solution to both convex and concave polygons.

5.4.3.3 Rover System

The rover communications need to be developed to take in messages from the UI and pass them to both the rover and the Rover Pathfinding Algorithm, as well as take in messages from the rover and pass them to the UI and the drone communications. The rover communications should be written in a ROS supported language (C++/Python) so that they can be run as a ROS node in order to communicate with the rover. Ideally, the rover communications will take in the deploy message and boundaries from the UI, process the boundaries into waypoints using the Rover Pathfinding Algorithm, then pass the deploy message and initial waypoint to the rover. Then it should wait for the rover to request the next waypoint and send it once requested as well as send the status update to the UI. If the Teensy 3.5 identifies a potential mine signature from the Uno, it should send this information to the onboard computer and halt the field search. The maintainMap ROS node should update the mine list and dilate the map so that the rover maintains a safe distance. The navigation node will plan around this when continuing to traverse the field. It should also send the information that a mine has been detected back to the UI. Once the rover has finished traversing the field, the missionControl node should tell navigation to guide it back home using waypoints, and send the mine data collected to the drone communications.

Since ROS abstracts node functionality, any node in the network can be run on either the rover onboard computer or the base station laptop. Our team recommends that the node containing the code state machine that stops the rover upon receiving a mine signal should be run on the rover in case of WiFi failure, but that as many nodes as possible be off-loaded onto the base station. Map maintenance, navigation, and drivetrain sample code will be provided, as will DAS specific ROS node outlines, but the implementation remains unfinished.

5.5 Conclusion

Abandoned landmines are a major inhibitor to the development of countries recovering from conflict. They prevent land development, stunt economic growth, and have a severe cost to human life - affecting 20,000 victims every year, 70-85% of whom are civilians (Landminefree.org, 2017) (Humanity & Inclusion, 2020). Landmines are widespread, affecting 58 nations across the globe (The HALO Trust, 2015). The diversity of environments that need to be demined calls for a similar diversity of solutions. Popular contemporary demining methods all have strengths and weaknesses. Machine demining is simple and effective, but it is also expensive, destructive, and limited by topography and environment. Animal assisted demining is relatively fast, but requires much more resources in training handlers and raising animals, and can be limited by how long and in what environments the animals can operate. The animals can also become confused in mine dense areas. Manual demining is thorough and versatile, but requires rigorous training and is slow and dangerous.

The purpose of the Demining Autonomous System MQP since its inception has been to create a relatively inexpensive demining alternative that is easily deployed by any non-technical user. Even though there were limitations due to the global COVID-19 pandemic slowing down process development on the project, our team was able to provide a stable baseline for the Demining Phase VI group to build upon. While we did not meet the goals we set out to accomplish, they were set without fully understanding either the state of the project we inherited or the scope of the work to be done. The team this year started with a new rover, a new base station, and a new drone, but left the project in a good state to see major progress in the years to come, and almost all of the necessary preliminary work for integrating the system as a whole was completed.

Though the project faced adversity, it did ultimately see great success. The four-bar of the rover was installed onto the new CERBERUS base. The new drone was outfitted with a new electronic system and dropper mechanism and is flight ready. The base station's UI was updated and it's pathfinding algorithm was built from scratch.

Up to this point, the DAS has been mostly theoretical in its design. As the DAS approaches completion, the project should find a sponsor who would employ the system, allowing future design decisions to be tailored to their individual situations and needs.

References

1. Aziz, Mahmoud. "Facts about Landmines." *Home - Minesweepers*, 2017, www.landminefree.org/2017/index.php/support/facts-about-landmines
2. Habib, Maki. (2002) Mine Clearance and Technologies for Effective Humanitarian Demining. *The Journal of Mine Action* 6.1.17. Retrieved from https://commons.lib.jmu.edu/cgi/viewcontent.cgi?article=2259&context=cisr-journal
3. Hemapala, K T M U & Razzoli, Roberto. (2012). Design and Development of a Landmines Removal Robot. International Journal of Advanced Robotic Systems. 9. 10.5772/50907.
4. Humanity & Inclusion. (2020). *Landmines - a deadly, disabling threat to humanity.* Retrieved from https://www.hi-us.org/landmines#:~:text=71%25%20of%20landmine%20victims%20are,are%20no%20longer%20at%20war.
5. International Campaign to Ban Landmines (ICBL). (2019). *Landmine Monitor 2019*. Landmine & Cluster Munition Monitor.
6. Kogler, Samuel. (2020). mapbox-earcut [Computer Software]. Retrieved from https://pypi.org/project/mapbox-earcut/
7. Mikulic, Dinko. (2013) Humanitarian Demining Techniques. In: Design of Demining Machines. Springer, London. https://doi.org/10.1007/978-1-4471-4504-2_1
8. Santos, A. R., Vanosten, A. D., ElShakhs, B. A., Foltan, E. D., McKenna, J. C., Niski, J. A., Ehlers, K. C., & Schmitt, M. T. (2020). *Demining Autonomous System*. Retrieved from https://digitalcommons.wpi.edu/cgi/viewcontent.cgi?article=8540&context=mqp-all
9. Smith, Andy. (2019). *PMN Anti-personnel blast mine.* Humanitarian Mine Action. Retrieved from: https://www.nolandmines.com/explosive_hazards/minesPMN.htm
10. The HALO Trust. (2015). *Landmines Still Exist In 58 Countries and Four States*. Retrieved from https://www.halotrust.org/latest/halo-updates/news/landmines-still-exist-in-58-countries-and-four-states/
11. Trevelyan, James. (2000) Reducing accidents in Demining: Achievements in Afghanistan. *Journal of Mine Action* 4.2.3. Retrieved from https://commons.lib.jmu.edu/cisr-journal/vol4/iss2/3

Appendices

Appendix A: Authorship

Section	**Author(s)**	**Editor(s)**
Abstract	**Troy Howlett**	**Eamon Oldridge**
1.0 Introduction	**-**	**-**
1.1 Problem Statement	Alex Hagedorn, Eamon Oldridge	Maggie Raque, Eamon Oldridge, Jeremy Wong
1.2 Project Description	Alex Hagedorn, Eamon Oldridge	Malek ElShakhs, Eamon Oldridge, Jeremy Wong
2.0 Background	**-**	**-**
2.1 Related Work	Eamon Oldridge, Maggie Raque	Malek ElShakhs, Eamon Oldridge, Jeremy Wong
2.2 Previous Work	All	Malek ElShakhs, Eamon Oldridge, Jeremy Wong
3.0 Methodology	**-**	**-**
3.1 Problem Formation	Eamon Oldridge	Eamon Oldridge, Jeremy Wong
3.2 Rover System Updates	Malek ElShakhs, Alex Hagedorn, Troy Howlett, Eamon Oldridge, Maggie Raque	Eamon Oldridge, Jeremy Wong
3.3 Systems Integration of New Drone	Eamon Oldridge, Jeremy Wong	Maggie Raque
3.4 Base Station Programming	Malek ElShakhs	Eamon Oldridge, Jeremy Wong
4.0 Results	**-**	**-**

4.1 Rover	Alex Hagedorn, Troy Howlett, Eamon Oldridge, Maggie Raque, Jeremy Wong	Eamon Oldridge, Jeremy Wong
4.2 Drone	Eamon Oldridge, Jeremy Wong	Maggie Raque
4.3 Base Station	Malek ElShakhs	Eamon Oldridge
5.0 Conclusion/Discussion	**-**	**-**
5.1 Rover Discussion	Alexander Hagedorn, Eamon Oldridge, Maggie Raque, Troy Howlett	Malek ElShakhs
5.2 Drone Discussion	Eamon Oldridge, Jeremy Wong	Maggie Raque
5.3 Base Station Discussion	Malek ElShakhs	Eamon Oldridge
5.4 Future Work for the MQP	All	Malek ElShakhs, Eamon Oldridge, Maggie Raque
5.5 Conclusion	Malek ElShakhs, Eamon Oldridge, Jeremy Wong	Maggie Raque

Appendix B: CAD Models

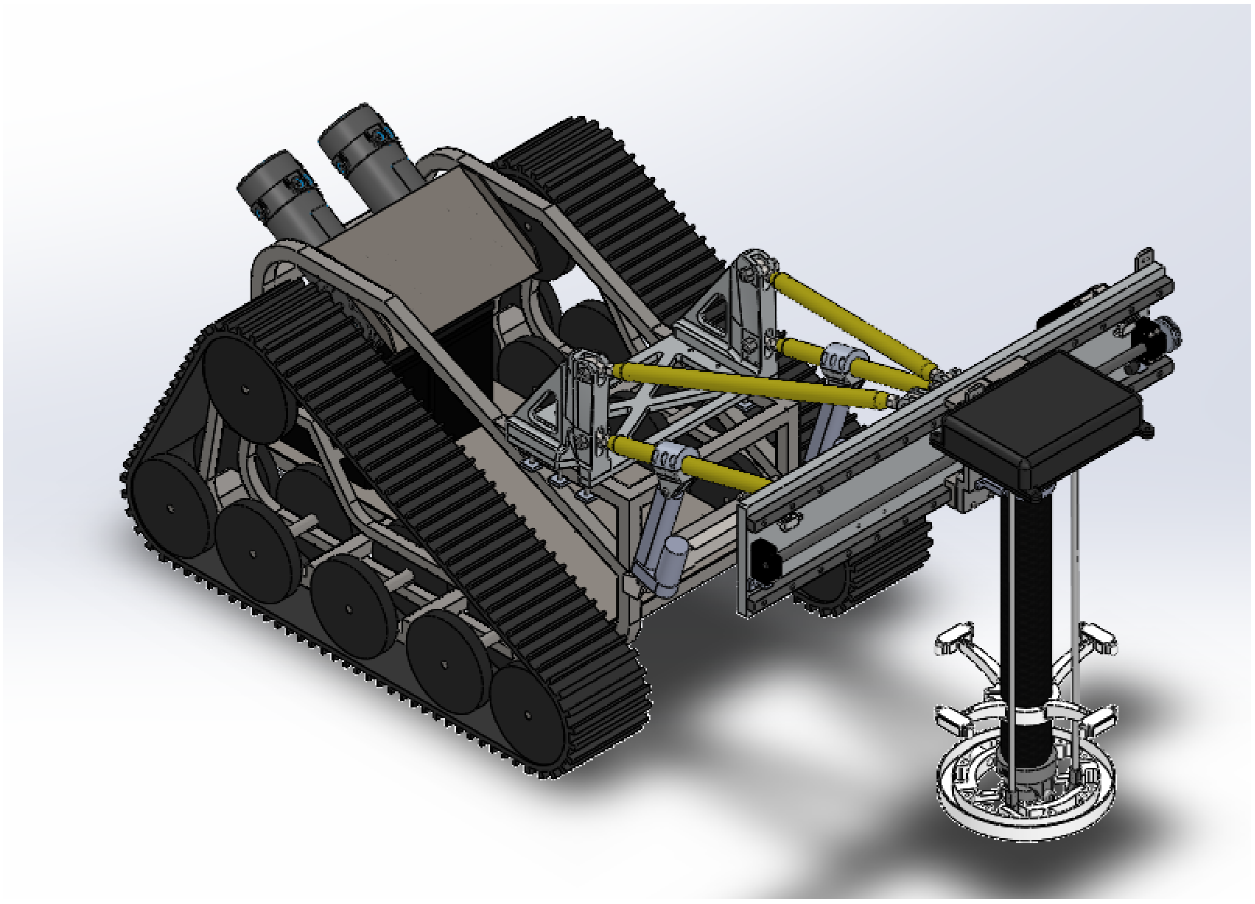


Figure 40: Isometric View of Rover

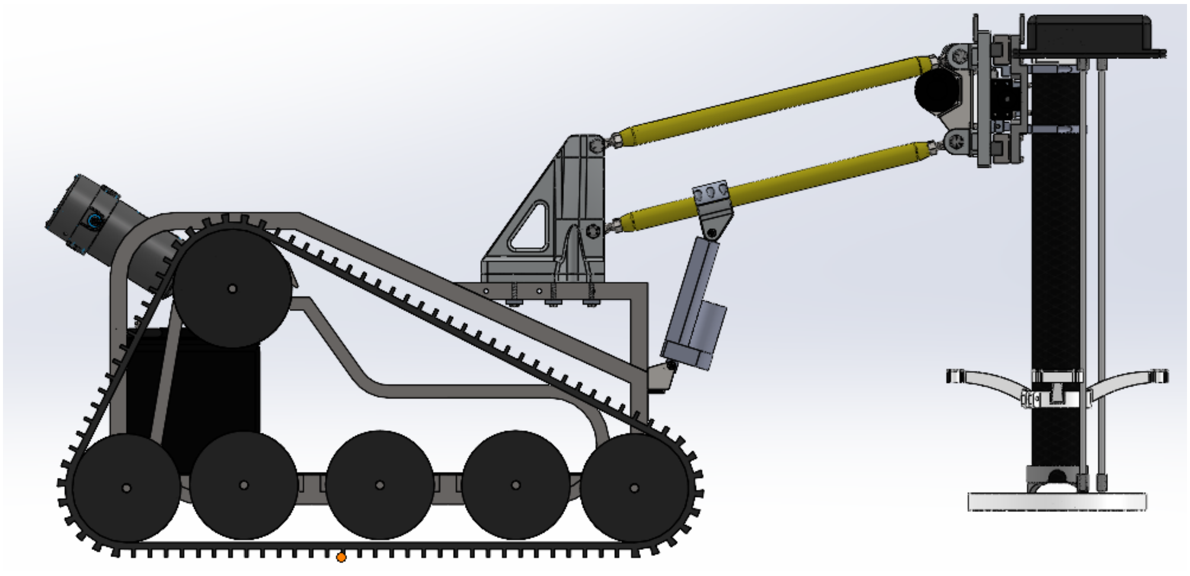


Figure 41: Side View of Rover

Appendix C: Rover Pathfinding Algorithm

```
import matplotlib.pyplot as plt
import mapbox_earcut
import numpy as np

WIDTH = 0.00000891 # width of the rover in lat/lng !TODO THIS SHOULD BE VERIFIED!
OVERLAP = 0.2 # how much overlap between lines (1 being 100% overlap, 0 being 0% overlap)

# Calculate the latitude of a line with slope m and y-intercept b at the given longitude
def calc_lat(lng, m, b):
    return m * lng + b;

def wpoints_convex(points):
    if(len(points) > 2):
        lupper, llower = [], [] # upper bound lines and lower bound lines
        w, e = points[0], points[0] # west and eastmost points

        wpoints = [] # waypoints to send to the rover (will be populated in order)

        # find west and eastmost points
        for p in points:
            if(p[1] < w[1]):
                w = p
            elif(p[1] > e[1]):
                e = p
        #print(w[1], e[1])

        welineM = (e[0]-w[0])/(e[1]-w[1]) # slope of the line between w and e
        welineB = e[0] - welineM * e[1] # y-intercept of the line between w and e
        #print(welineM)
        #print(welineB)

        # add all lines to upper or lower bound arrays
        for i in range(len(points)):
            j = 0 # next point after i

            if(i < len(points)-1): # if i is the last point, keep j as the first point
                j = i+1

            m = 0 # slope of i-j line
            b = 0 # y-intercept of i-j line
            if(points[i][1] == points[j][1]): # if i & j have the same longitude, the line is vertical
                m = float('inf')
                b = float('-inf')
            else:
                m = (points[i][0] - points[j][0]) / (points[i][1] - points[j][1]) # equation for slope
                b = points[i][0] - m * points[i][1] # equation for y-intercept

                # add line to upper or lower bound
                if(points[i][0] > round(calc_lat(points[i][1], welineM, welineB), 6) or points[j][0] >
round(calc_lat(points[j][1], welineM, welineB), 6)):
                    lupper.append((m, b))
                else:
                    llower.append((m, b))


        # check for if the west-east line is an upper or lower bound
        if(len(lupper) == 0):
            if (welineM, welineB) in llower:
                llower.remove((welineM, welineB))
            lupper.append((welineM, welineB))

        elif(len(llower) == 0):
            if (welineM, welineB) in lupper:
                lupper.remove((welineM, welineB))
            llower.append((welineM, welineB))

        posLat = w[0] # current position (latitude) (UNUSED)
        posLng = w[1] + (1.-OVERLAP)/2. * WIDTH # current position (longitude)

        top = True # boolean value, whether the rover is at a top boundary

        # add waypoints to wpoints
        while(posLng > w[1] and posLng < e[1]):
            tLat = calc_lat(posLng, lupper[0][0], lupper[0][1])
            bLat = calc_lat(posLng, llower[0][0], llower[0][1])

            for l in lupper: # get the lowest latitude of the upper bounds at the current rows longitude
                newLat = calc_lat(posLng, l[0], l[1])
                if(newLat < tLat):
                    tLat = newLat

            for l in llower: # get the highest latitude of the lower bounds at the current rows
longitude
                newLat = calc_lat(posLng, l[0], l[1])
                if(newLat > bLat):
                    bLat = newLat

            if(top): # if at the top, add the top waypoint first, then the bottom
                wpoints.append((tLat, posLng))
                wpoints.append((bLat, posLng))
                top = False
            else: # if at the bottom, add the bottom waypoint first, then the top
                wpoints.append((bLat, posLng))
                wpoints.append((tLat, posLng))
                top = True

            posLng += (1.-OVERLAP) * WIDTH # move to next row while accounting for overlap

        #print(wpoints)

        # PLOTTING POINTS AND GRAPHING ------------------------------------------------

        xpts = []
        ypts = []

        xwpts = []
        ywpts = []

        for p in points:
            xpts.append(p[1])
            ypts.append(p[0])

        for p in wpoints:
            xwpts.append(p[1])
            ywpts.append(p[0])
        plt.plot(xpts, ypts, 'ro', xwpts, ywpts)
        plt.show()

        return wpoints # return the full list of waypoints for the convex polygon

# get circular range of array (return values in the array wrapping from the end of the array back to the
beginning)
# array is the array to return, start is the starting index, stop is the ending index, mod is the size
of the array
def crange(array, start, stop, mod):
    result = []
    index = start
    while(index != stop):
        result.append(array[index])
        index = (index + 1) % mod
    result.append(array[index])
    return result

# returns whether the angle between three consecutive points is clockwise
def cw(a,b,c):
    return float(b[0] - a[0]) * (c[1] - b[1]) >= float(b[1] - a[1]) * (c[0] - b[0])

# returns whether the angle between three consecutive points is counterclockwise
def ccw(a,b,c):
    return float(b[0] - a[0]) * (c[1] - b[1]) <= float(b[1] - a[1]) * (c[0] - b[0])

# returns whether the given array of points forms a convex shape
def convex(points):
    allcw = True
    allccw = True

    # tests whether all angles are clockwise (takes every consecutive 3 points and finds if the angle
between any is not clockwise)
    for i in range(len(points)):
        a = points[i % len(points)]
        b = points[(i + 1) % len(points)]
        c = points[(i + 2) % len(points)]
        if not cw(a,b,c):
            #print(a,b,c)
            allcw = False

    # tests whether all angles are counterclockwise (takes every consecutive 3 points and finds if the
angle between any is not counterclockwise)
    for i in range(len(points)):
        a = points[i % len(points)]
        b = points[(i + 1) % len(points)]
        c = points[(i + 2) % len(points)]
        if not ccw(a,b,c):
            #print(a,b,c)
            allccw = False

    # if all angles are of the same orientation, the polygon is convex
    return allcw or allccw

# split an array into equally sized partitions (needed to split the triangulated shape into triplets of
three points each)
def split(arr, n):
    ret = []
    for i in range(0, len(arr), n):
         ret.append(arr[i:i + n])

    return ret

# return whether one convex shape is adjacent to another (if they share 2 points)
def adjacent(poly1, poly2):
    adj = 0

    for i in poly1:
        for j in poly2:
            if(i[0] == j[0] and i[1] == j[1]):
                adj += 1

    return adj == 2


# Node is the basis for partitioning, polys is the list of all polygons in the field, source is the
Nodes own triangle/polygon, nodes is the list of nodes not yet searched, possible is a list of all
potential convex partitions found thus far
class Node:
    def __init__(self, polys, source, nodes, possible):
        self.polys = polys
        self.source = source
        self.nodes = nodes
        self.possible = possible

# finds the starting index for a combined polygon on the first shared point (if the second shared point
is not the next point) otherwise, the second shared point
def fstartind(poly1, shared, missing):
    for i in range(len(poly1)):
        for j in range(len(shared)):
            if(poly1[i][0] == shared[j][0] and poly1[i][1] == shared[j][1]): # if i is a shared point
                for j in range(len(shared)):
                    if(len(poly1) > i+1 and poly1[i+1][0] == shared[j][0] and poly1[i+1][1] ==
shared[j][1]): # if i+1 is a shared point
                        return i+1 # returns i+1 if i and i+1 are both shared
                return i # returns i if i is shared and i+1 is not

# combine two polygons into one polygon in the correct point order without duplicating shared points
# works assuming poly2 is a triangle (always should be)
def combine(poly1, poly2):

    shared = [] # points shared between poly1 and poly2 (will always have a length of 2 since only
adjacent polygons are combined)
    missing = [] # points present in poly2 and not present in poly1

    # populate shared and missing
    for j in poly2:
        present = False
        for i in poly1:
            if(i[0] == j[0] and i[1] == j[1]):
                shared.append(i)
                present = True

        if(present == False):
            missing = j
    # get the starting index for poly1
    startind = fstartind(poly1, shared, missing)

    # using the starting index, get the circular range of poly1
    # this combined with the starting index function guarantees that one shared point will be the first
element while the other will be the last
    poly_comb = crange(poly1, startind, (startind-1)%len(poly1), len(poly1))

    # since the first and last elements are the shared points, the missing point (only ever adds
triangles to poly1) can be added to the end without messing up the order of points
    poly_comb.append(missing)

    return poly_comb

# remove the given array from an array of arrays (.remove() will throw an error)
def remove_array(to_remove, r_from):
    ind = 0
    for i in range(len(r_from)):
        if np.array_equal(to_remove, r_from[i]):
            ind = i

    r_from.pop(ind)

    return r_from

# recursive method which creates an exhaustive tree of all possible combinations of convex shapes and
returns the least number of convex shapes possible
def partition(node):
    # base case: if there are no more nodes to check, just return the poly list that the node has
    if(len(node.nodes) == 0):
        return node.polys

    # for all nodes to still check
    for i in node.nodes:
        if(adjacent(node.source, i)): # if the node is adjacent to the current polygon
            new_poly = combine(node.source, i) # try adding the node to the polygon

            if(convex(new_poly)): # if the resulting polygon is still convex, add this as a possible
partition and recurse
                temp_polys = node.polys.copy()
                temp_nodes = node.nodes.copy()

                temp_nodes = remove_array(i, temp_nodes) # remove the added node from the list of nodes
to check
                temp_polys = remove_array(node.source, temp_polys) # remove the initial polygon from the
polygon list
                temp_polys = remove_array(i, temp_polys) # remove the added node from the polygon list

                temp_polys.append(new_poly) # add the combined polygon back into the polygon list

                node.possible.append(partition(Node(temp_polys, new_poly, temp_nodes, [])))

    # also add the case where the current polygon is complete and moving onto the next node is the best
option
    node.possible.append(partition(Node(node.polys, node.nodes[0], node.nodes[1:], [])))

    minimum = node.polys # list of the minimum number of polygons in the possible list

    # find mindex and minimum selection
    for i in node.possible:
        if(len(i) < len(minimum)):
            minimum = i

    #print("MINIMUM: " + str(minimum))
    return minimum # returns the minimum selection in all possible partitions


# get waypoints for any shape
def wpoints(points):
    allwpoints = []

    if(convex(points)): # if already convex, just use the convex function
        allwpoints.append(wpoints_convex(points))
        return allwpoints
    else:
        res = np.array(points).reshape(-1, 2) # reshape the array in order to use the library
        result = mapbox_earcut.triangulate_float32(points, [len(points)]) # library function to
triangulate the polygon using earcutting
        print("TRIANGULATING...")
        triangulated = split(res[result], 3) # split the result into arrays of three (triangles)
        print("PARTITIONING...")
        allpolys = partition(Node(triangulated, triangulated[0], triangulated[1:], [])) # produces list
of minimum partitioned convex polygons
        for i in allpolys:
            print("GENERATING POLYGON " + str(i) + " WAYPOINTS...")
            allwpoints.append(wpoints_convex(i)) # generate waypoints for each convex polygon and add
each set to the list
        return allwpoints
```

Appendix D: Four-bar Analysis

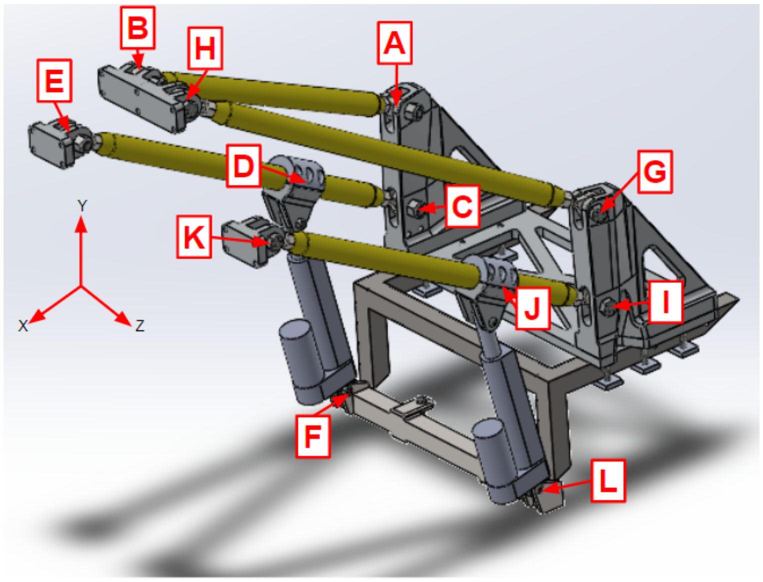


Figure 42: Joint Locations

Static Analysis

Determined the 3D forces within the four-bar mechanism as well as the deflection due to the weight of the linear slide.

Assumptions due to symmetry:

$A_x = G_x$

$A_y = G_y$

$|A_z| = |G_z|$

$B_x = H_x$

$B_y = H_y$

$|B_z| = |H_z|$

$C_x = I_x$

$C_y = I_y$

$D_x = J_x$

$D_y = J_y$

$E_x = K_x$

$E_y = K_y$

$F_x = L_x$
$F_y = L_y$

Equilibrium equations for the linear slide:

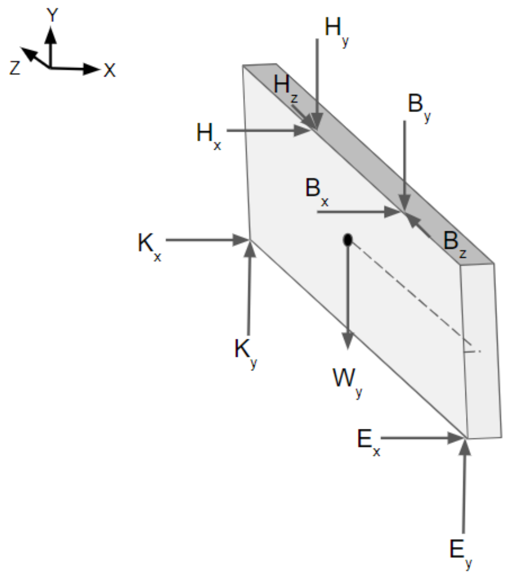


Figure 43: Free Body Diagram of the Linear Slide

$$\Sigma M_E = r_{EB} \times F_B + r_{EH} \times F_H + r_{EK} \times F_K + r_{EW} \times F_W = 0$$
$$\Sigma F_y = E_y - B_y - H_y + K_y - W_y = 0$$
$$\Sigma F_x = E_x + B_x + H_x + K_x = 0$$
$$\Sigma F_z = B_z - H_z = 0$$

Equilibrium equations for link AB:

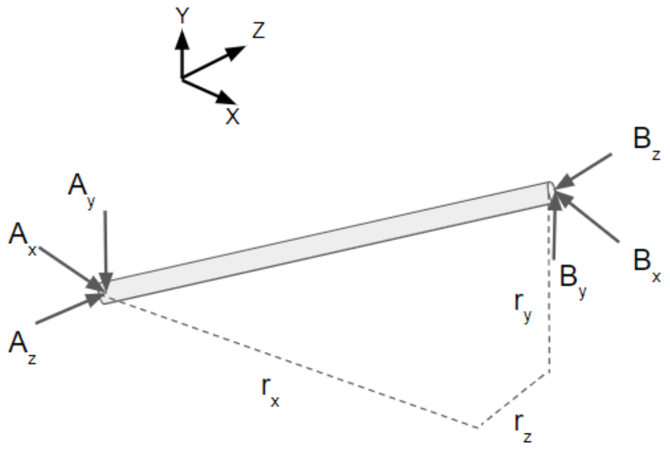


Figure 44: Free Body Diagram of Link AB

$$\Sigma M_A = r_{AB} \times F_B = 0$$
$$\Sigma F_y = A_y - B_y = 0$$
$$\Sigma F_x = A_x - B_x = 0$$

Equilibrium equations for link CE:

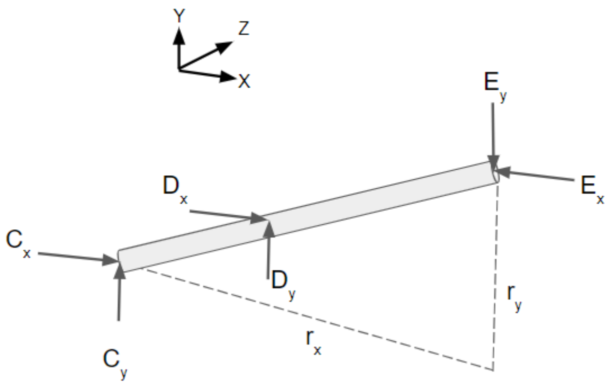


Figure 45: Free Body Diagram of Link CE

$$\Sigma M_C = r_{CD} \times F_D + r_{CE} \times F_E = 0$$
$$\Sigma F_y = C_y + D_y - E_y = 0$$
$$\Sigma F_x = C_x + D_x - E_x = 0$$

Equilibrium equations for linear actuator FD:

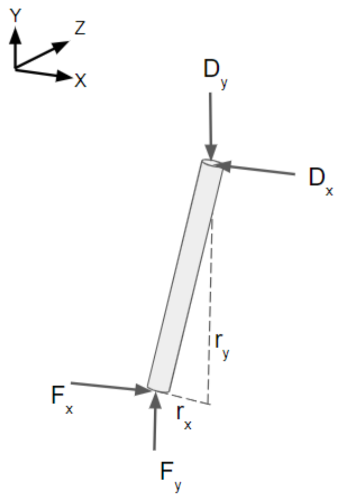


Figure 46: Free Body Diagram of Linear Actuator FD

$\Sigma M_F = r_{FD} \times F_D = 0$
$\Sigma F_y = F_y - D_y = 0$
$\Sigma F_x = F_x - D_x = 0$
Equilibrium equations for forces acting on the four-bar mechanism:

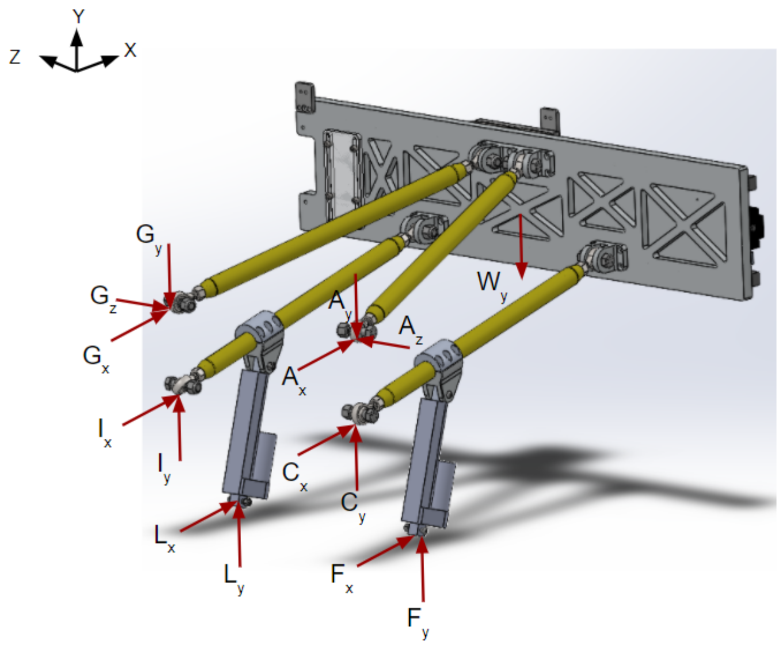
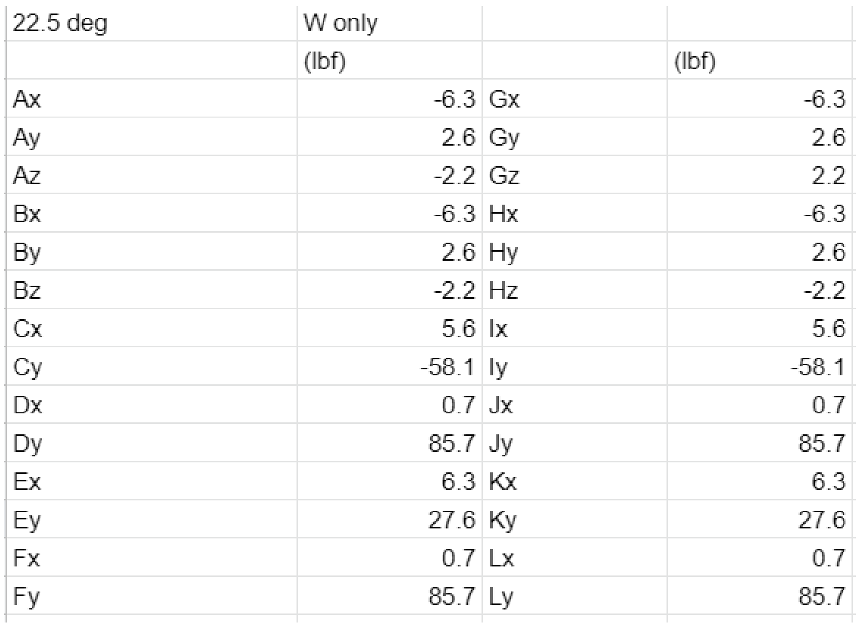


Figure 47: Free Body Diagram of External Forces on the Four-bar Mechanism

$\Sigma M_C = r_{CW} \times F_W + r_{CA} \times F_A + r_{CF} \times F_F + r_{CG} \times F_G + r_{CI} \times F_I + r_{CL} \times F_L = 0$

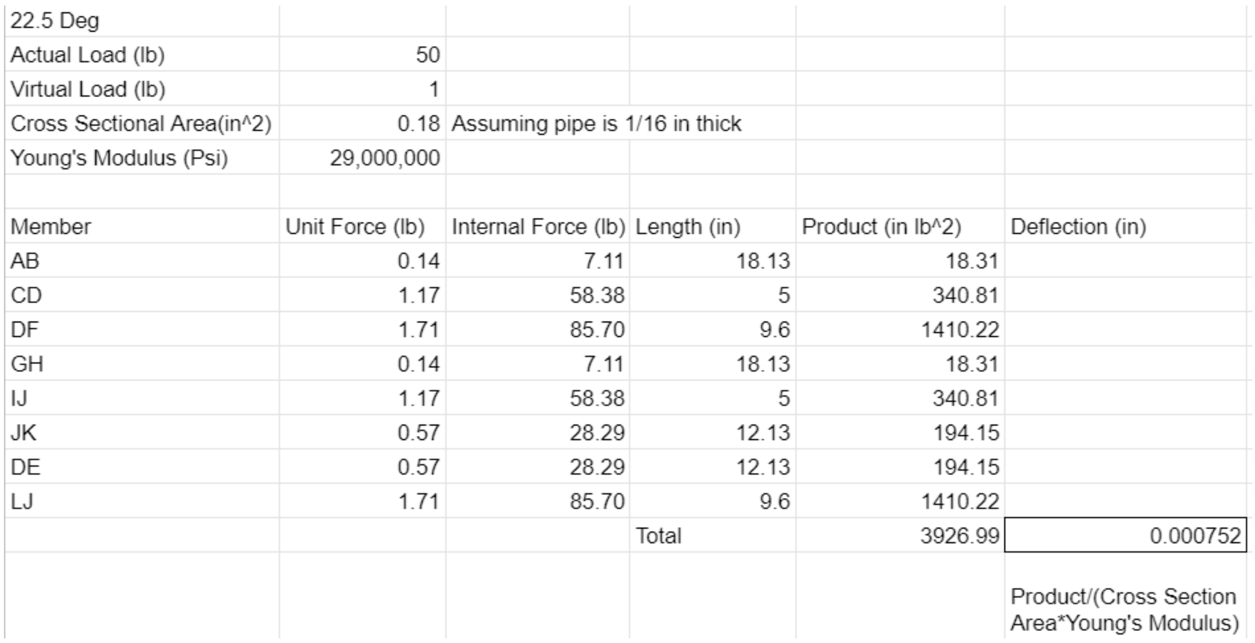
$$\Sigma F_y = A_y + C_y + F_y + G_y + I_y + L_y - W_y = 0$$
$$\Sigma F_x = A_x + C_x + F_x + G_x + I_x + L_x = 0$$
$$\Sigma F_z = A_z + G_x = 0$$

Analyzed forces and deflection at an angle of inclination for the CE member at 22.5 degrees and 60 degrees.



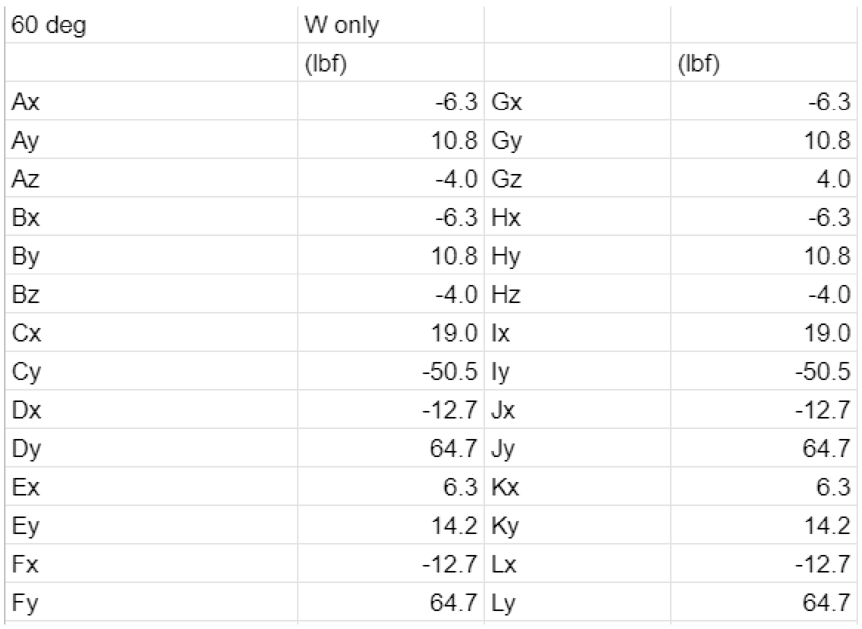
22.5 deg	W only		
	(lbf)		(lbf)
Ax	-6.3	Gx	-6.3
Ay	2.6	Gy	2.6
Az	-2.2	Gz	2.2
Bx	-6.3	Hx	-6.3
By	2.6	Hy	2.6
Bz	-2.2	Hz	-2.2
Cx	5.6	Ix	5.6
Cy	-58.1	Iy	-58.1
Dx	0.7	Jx	0.7
Dy	85.7	Jy	85.7
Ex	6.3	Kx	6.3
Ey	27.6	Ky	27.6
Fx	0.7	Lx	0.7
Fy	85.7	Ly	85.7

Figure 48: Force Results at 22.5 degrees



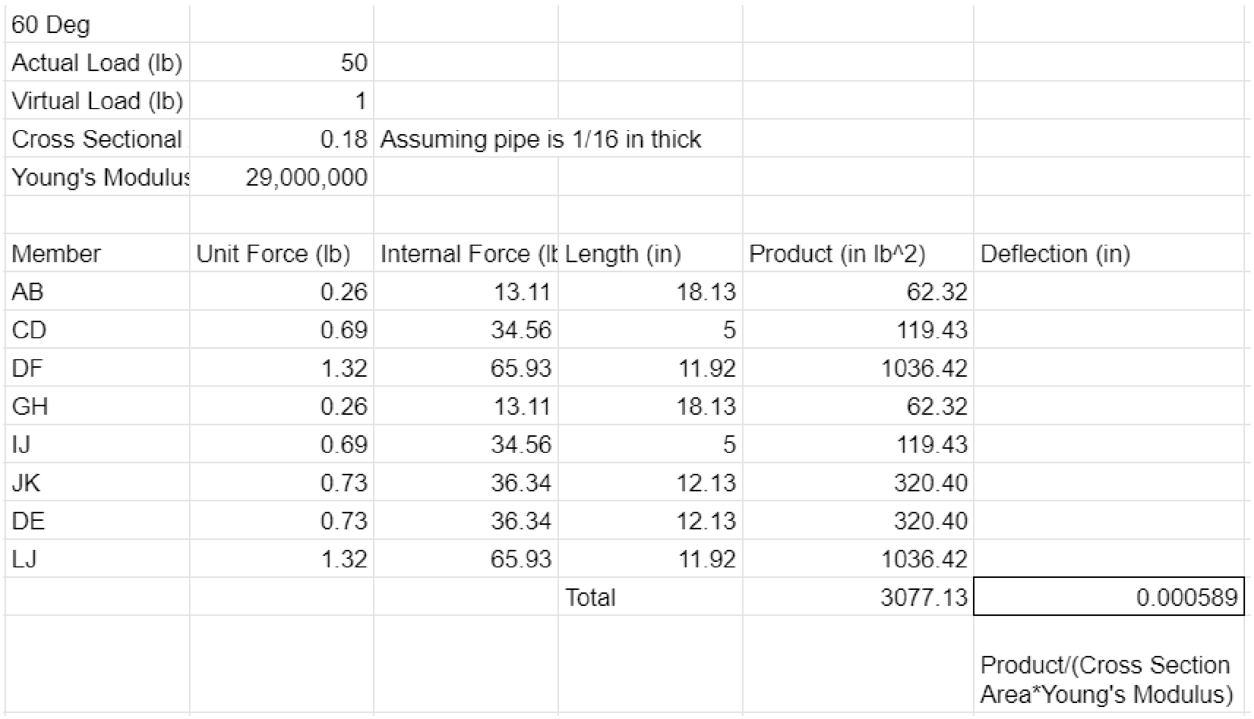
22.5 Deg					
Actual Load (lb)	50				
Virtual Load (lb)	1				
Cross Sectional Area(in^2)	0.18	Assuming pipe is 1/16 in thick			
Young's Modulus (Psi)	29,000,000				
Member	Unit Force (lb)	Internal Force (lb)	Length (in)	Product (in lb^2)	Deflection (in)
AB	0.14	7.11	18.13	18.31	
CD	1.17	58.38	5	340.81	
DF	1.71	85.70	9.6	1410.22	
GH	0.14	7.11	18.13	18.31	
IJ	1.17	58.38	5	340.81	
JK	0.57	28.29	12.13	194.15	
DE	0.57	28.29	12.13	194.15	
LJ	1.71	85.70	9.6	1410.22	
			Total	3926.99	0.000752
					Product/(Cross Section Area*Young's Modulus)

Figure 49: Deflection Results at 22.5 degrees



60 deg	W only		
	(lbf)		(lbf)
Ax	-6.3	Gx	-6.3
Ay	10.8	Gy	10.8
Az	-4.0	Gz	4.0
Bx	-6.3	Hx	-6.3
By	10.8	Hy	10.8
Bz	-4.0	Hz	-4.0
Cx	19.0	Ix	19.0
Cy	-50.5	Iy	-50.5
Dx	-12.7	Jx	-12.7
Dy	64.7	Jy	64.7
Ex	6.3	Kx	6.3
Ey	14.2	Ky	14.2
Fx	-12.7	Lx	-12.7
Fy	64.7	Ly	64.7

Figure 50: Force Results at 60 degrees



60 Deg					
Actual Load (lb)	50				
Virtual Load (lb)	1				
Cross Sectional	0.18	Assuming pipe is 1/16 in thick			
Young's Modulu	29,000,000				
Member	Unit Force (lb)	Internal Force (lt	Length (in)	Product (in lb^2)	Deflection (in)
AB	0.26	13.11	18.13	62.32	
CD	0.69	34.56	5	119.43	
DF	1.32	65.93	11.92	1036.42	
GH	0.26	13.11	18.13	62.32	
IJ	0.69	34.56	5	119.43	
JK	0.73	36.34	12.13	320.40	
DE	0.73	36.34	12.13	320.40	
LJ	1.32	65.93	11.92	1036.42	
			Total	3077.13	0.000589
					Product/(Cross Section Area*Young's Modulus)

Figure 51: Deflection Results at 60 degrees

Rotational Analysis

Determined the 3D forces within the four-bar mechanism as well as the deflection due to the weight of the linear slide and the force on the linear slide due to the rotation of the rover.

Equilibrium equations for the linear slide:

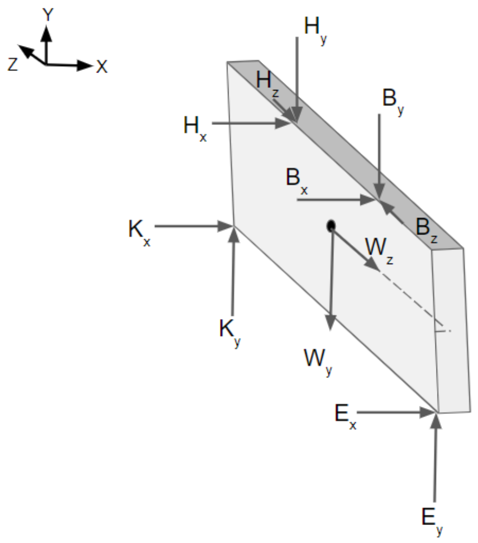


Figure 52: Free Body Diagram of Linear Slide

$$\Sigma M_E = r_{EB} \times F_B + r_{EH} \times F_H + r_{EK} \times F_K + r_{EW} \times F_W = 0$$
$$\Sigma F_y = E_y - B_y - H_y + K_y - W_y = 0$$
$$\Sigma F_x = E_x + B_x + H_x + K_x = 0$$
$$\Sigma F_z = B_z - H_z - W_z = 0$$

Equilibrium equations for link AB:

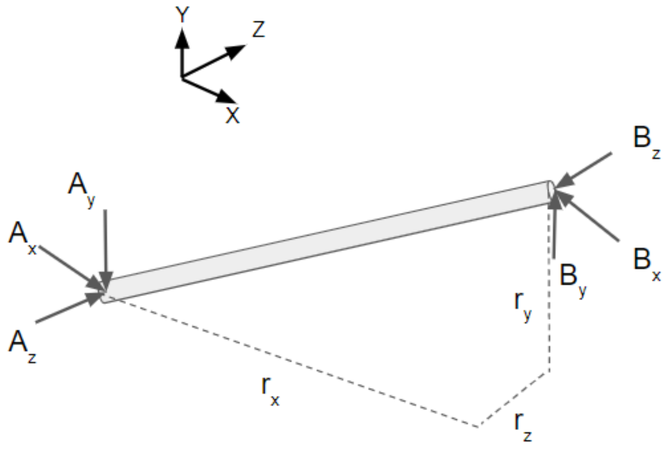


Figure 53: Free Body Diagram of Link AB

$\Sigma M_A = r_{AB} \times F_B = 0$

$\Sigma F_y = A_y - B_y = 0$

$\Sigma F_x = A_x - B_x = 0$

$\Sigma F_z = A_z - B_z = 0$

Equilibrium equations for link GH:

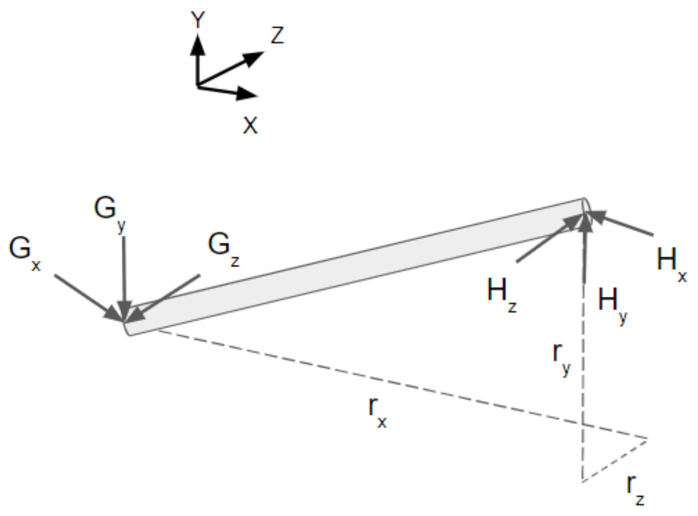


Figure 54: Free Body Diagram of Link GH

$\Sigma M_G = r_{GH} \times F_H = 0$

$\Sigma F_y = G_y - H_y = 0$

$\Sigma F_x = G_x - H_x = 0$

$\Sigma F_z = G_z - H_z = 0$

Equilibrium equations for link CE:

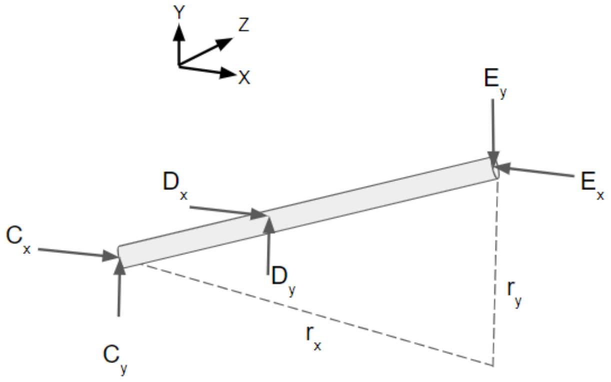


Figure 55: Free Body Diagram of Link CE

$$\Sigma M_C = r_{CD} \times F_D + r_{CE} \times F_E = 0$$
$$\Sigma F_y = C_y + D_y - E_y = 0$$
$$\Sigma F_x = C_x + D_x - E_x = 0$$

Equilibrium equations for link IK:

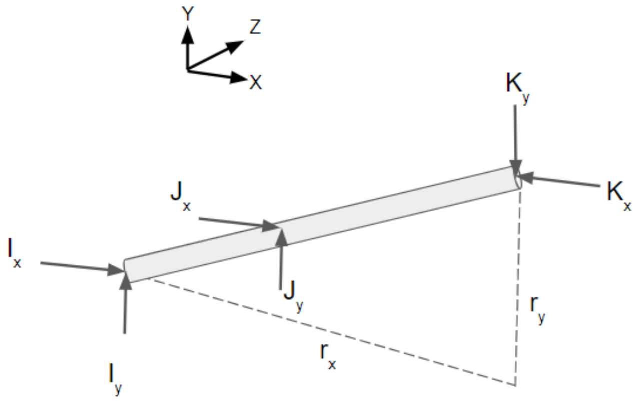


Figure 56: Free Body Diagram of Link IK

$$\Sigma M_I = r_{IJ} \times F_J + r_{IK} \times F_K = 0$$
$$\Sigma F_y = I_y + J_y - K_y = 0$$
$$\Sigma F_x = I_x + J_x - K_x = 0$$

Equilibrium equations for linear actuator FD:

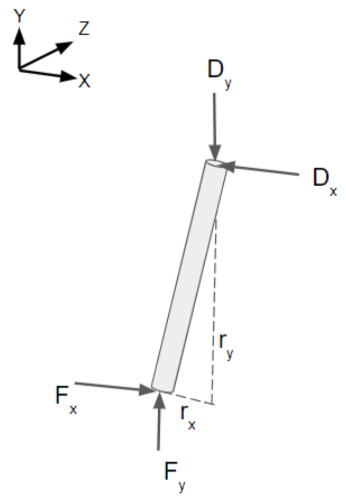


Figure 57: Free Body Diagram of Linear Actuator FD

$\Sigma M_F = r_{FD} \times F_D = 0$
$\Sigma F_y = F_y - D_y = 0$
$\Sigma F_x = F_x - D_x = 0$

Equilibrium equations for linear actuator LJ:

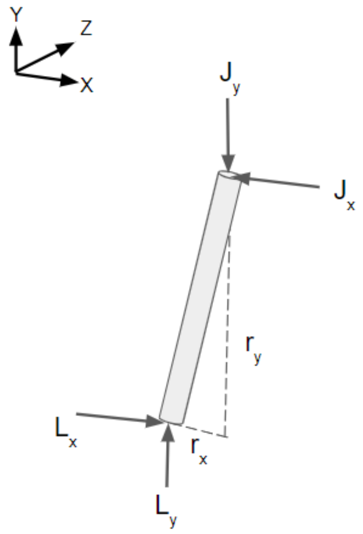


Figure 58: Free Body Diagram of Linear Actuator JL

$\Sigma M_L = r_{LJ} \times F_J = 0$
$\Sigma F_y = L_y - J_y = 0$
$\Sigma F_x = L_x - J_x = 0$

Equilibrium equations for forces acting on the four-bar mechanism:

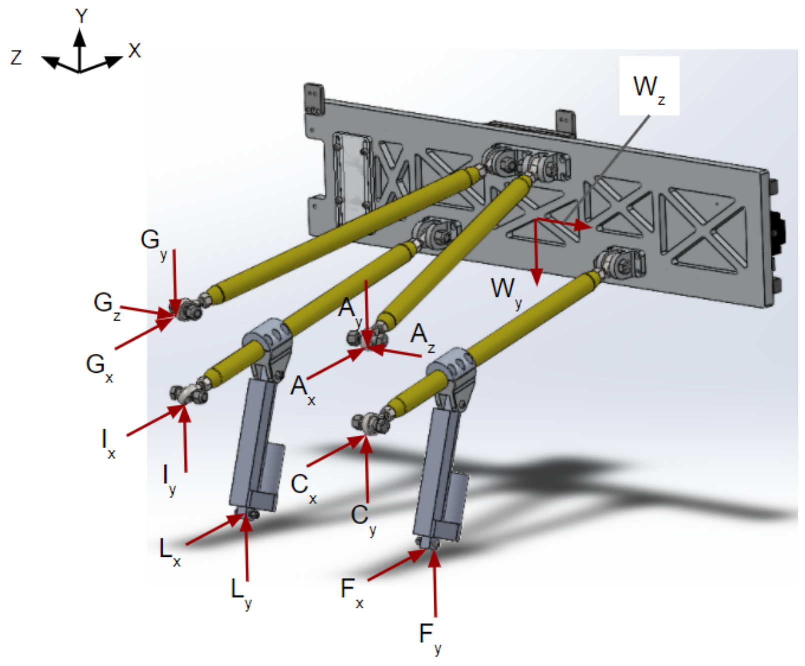
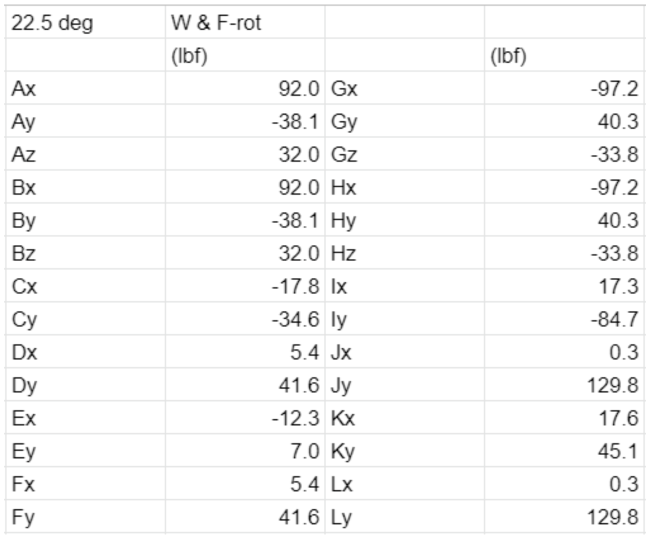


Figure 59: Free Body Diagram of External Forces on the Four-bar Mechanism

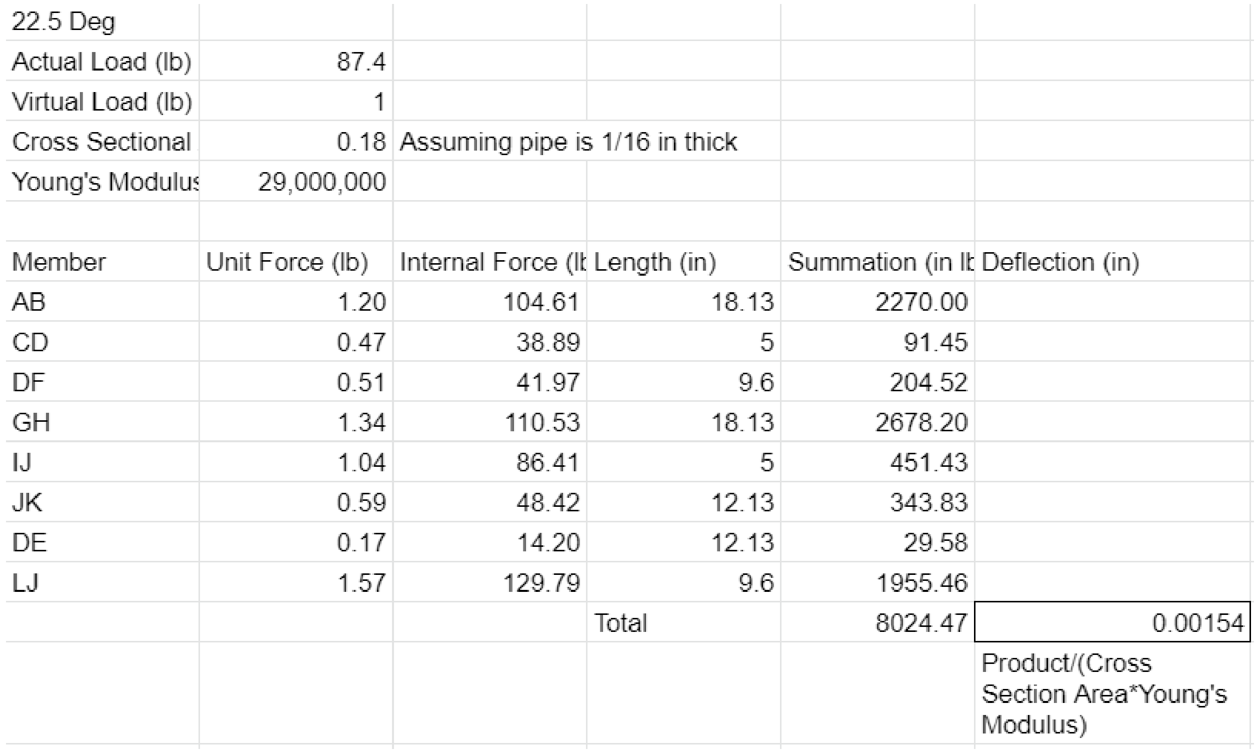
$$\Sigma M_C = r_{CW} \times F_W + r_{CA} \times F_A + r_{CF} \times F_F + r_{CG} \times F_G + r_{CI} \times F_I + r_{CL} \times F_L = 0$$
$$\Sigma F_y = A_y + C_y + F_y + G_y + I_y + L_y - W_y = 0$$
$$\Sigma F_x = A_x + C_x + F_x + G_x + I_x + L_x = 0$$
$$\Sigma F_z = A_z + G_x - W_z = 0$$

Analyzed forces and deflection at an angle of inclination for the CE member at 22.5 degrees and 60 degrees.



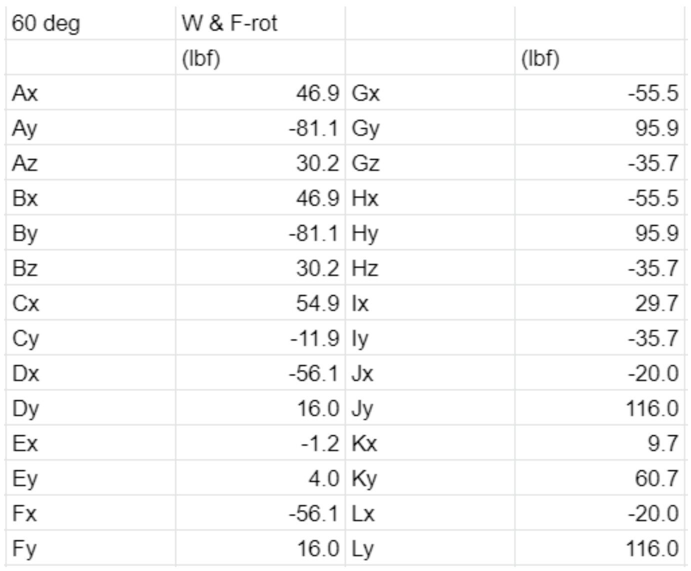
22.5 deg	W & F-rot		
	(lbf)		(lbf)
Ax	92.0	Gx	-97.2
Ay	-38.1	Gy	40.3
Az	32.0	Gz	-33.8
Bx	92.0	Hx	-97.2
By	-38.1	Hy	40.3
Bz	32.0	Hz	-33.8
Cx	-17.8	Ix	17.3
Cy	-34.6	Iy	-84.7
Dx	5.4	Jx	0.3
Dy	41.6	Jy	129.8
Ex	-12.3	Kx	17.6
Ey	7.0	Ky	45.1
Fx	5.4	Lx	0.3
Fy	41.6	Ly	129.8

Figure 60: Force Results at 22.5 Degrees



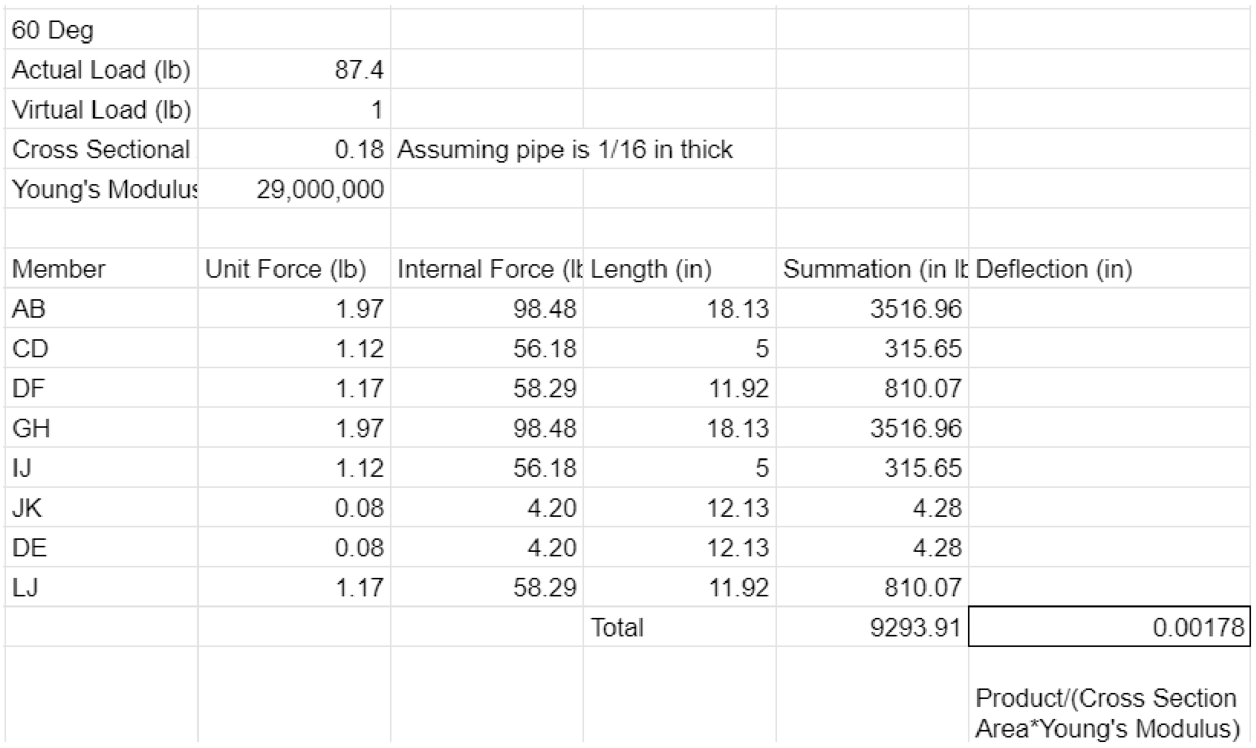
22.5 Deg					
Actual Load (lb)	87.4				
Virtual Load (lb)	1				
Cross Sectional	0.18	Assuming pipe is 1/16 in thick			
Young's Modulu	29,000,000				
Member	Unit Force (lb)	Internal Force (lt	Length (in)	Summation (in lt	Deflection (in)
AB	1.20	104.61	18.13	2270.00	
CD	0.47	38.89	5	91.45	
DF	0.51	41.97	9.6	204.52	
GH	1.34	110.53	18.13	2678.20	
IJ	1.04	86.41	5	451.43	
JK	0.59	48.42	12.13	343.83	
DE	0.17	14.20	12.13	29.58	
LJ	1.57	129.79	9.6	1955.46	
			Total	8024.47	0.00154
					Product/(Cross Section Area*Young's Modulus)

Figure 61: Deflection Results at 22.5 degrees



60 deg	W & F-rot		
	(lbf)		(lbf)
Ax	46.9	Gx	-55.5
Ay	-81.1	Gy	95.9
Az	30.2	Gz	-35.7
Bx	46.9	Hx	-55.5
By	-81.1	Hy	95.9
Bz	30.2	Hz	-35.7
Cx	54.9	Ix	29.7
Cy	-11.9	Iy	-35.7
Dx	-56.1	Jx	-20.0
Dy	16.0	Jy	116.0
Ex	-1.2	Kx	9.7
Ey	4.0	Ky	60.7
Fx	-56.1	Lx	-20.0
Fy	16.0	Ly	116.0

Figure 62: Force Results at 60 Degrees



60 Deg					
Actual Load (lb)	87.4				
Virtual Load (lb)	1				
Cross Sectional	0.18	Assuming pipe is 1/16 in thick			
Young's Modulus	29,000,000				
Member	Unit Force (lb)	Internal Force (lb	Length (in)	Summation (in lb	Deflection (in)
AB	1.97	98.48	18.13	3516.96	
CD	1.12	56.18	5	315.65	
DF	1.17	58.29	11.92	810.07	
GH	1.97	98.48	18.13	3516.96	
IJ	1.12	56.18	5	315.65	
JK	0.08	4.20	12.13	4.28	
DE	0.08	4.20	12.13	4.28	
LJ	1.17	58.29	11.92	810.07	
			Total	9293.91	0.00178
					Product/(Cross Section Area*Young's Modulus)

Figure 63: Deflection Results at 60 degrees

Appendix E: Preflight Checklist:

Preflight Checklists

1. Correct PPE
2. Batteries Charged & Secured
3. Aircraft Hardware OK
4. Equipment & Gear OK
5. Transmitter Controls OK
6. Props OK & Tight
7. Software/Firmware Update
8. Transmitter Control Power ON
9. Aircraft Power ON
10. Compass Calibration
11. Camera ON
12. Satellite Connection

Appendix F: Rover Hero .NETMF Drive Code

```
using System;
using System.Threading;
using Microsoft.SPOT;
using System.Text;

using CTRE.Phoenix;
using CTRE.Phoenix.Controller;
using CTRE.Phoenix.MotorControl;
using CTRE.Phoenix.MotorControl.CAN;

namespace HERO_MQP_Arcade_Drive_Testing
{
    public class Program
    {
        /* create a talon */
        static TalonSRX right = new TalonSRX(0);
        static TalonSRX left = new TalonSRX(1);

        static StringBuilder stringBuilder = new StringBuilder();

        static CTRE.Phoenix.Controller.GameController _gamepad = null;

        public static void Main()
        {
            /* loop forever */
            while (true)
            {
                /* drive robot using gamepad */
                Drive();
                /* print whatever is in our string builder */
                Debug.Print(stringBuilder.ToString());
                stringBuilder.Clear();
                /* feed watchdog to keep Talon's enabled */
                CTRE.Phoenix.Watchdog.Feed();
                /* run this task every 20ms */
                Thread.Sleep(20);
            }
        }
        /**
         * If value is within 10% of center, clear it.
         * @param value [out] floating point value to deadband.
         */
        static void Deadband(ref float value)
        {
            if (value < -0.10)
            {
                /* outside of deadband */
            }
            else if (value > +0.10)
            {
                /* outside of deadband */
            }
            else
            {
                /* within 10% so zero it */
                value = 0;
            }
        }
        static void Drive()
        {
            if (null == _gamepad)
            {
                _gamepad = new GameController(UsbHostDevice.GetInstance());
                Debug.Print("Controller object initialized");
            }

            float y = _gamepad.GetAxis(2); // analog right (1) to left (-1) on right joystick
            float twist = _gamepad.GetAxis(5); // analog forward (-1) and back (1) on right joystick

            Deadband(ref y);
            Deadband(ref twist);

            float leftThrot = y + twist;
            float rightThrot = y - twist;

            left.Set(ControlMode.PercentOutput, leftThrot);
            right.Set(ControlMode.PercentOutput, -rightThrot);

            stringBuilder.Append("\t");
            stringBuilder.Append(y);
            stringBuilder.Append("\t");
            stringBuilder.Append(twist);
        }
    }
}
```

Appendix G: Attachment Part Analysis

CAD stress analysis with the applied loads on the components.

Material: Plain Carbon Steel
Max Stress = 9,370 psi
Yield Strength = 53,700 psi
FOS = 5.73

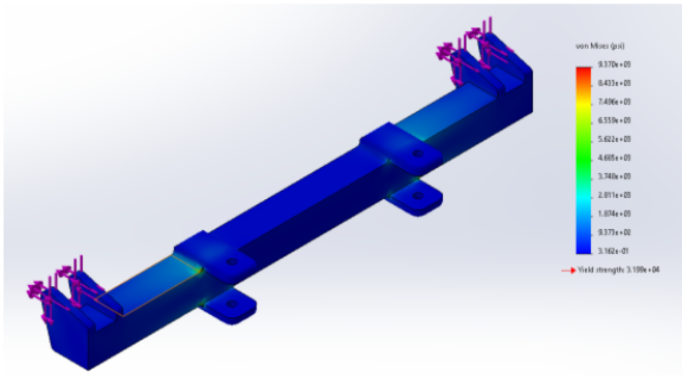


Figure 64: ANSYS Analysis of Linear Actuator Attachment Bar

Material: Plain Carbon Steel
Max Stress = 2,164 psi
Yield Strength = 53,700 psi
FOS = 24.1

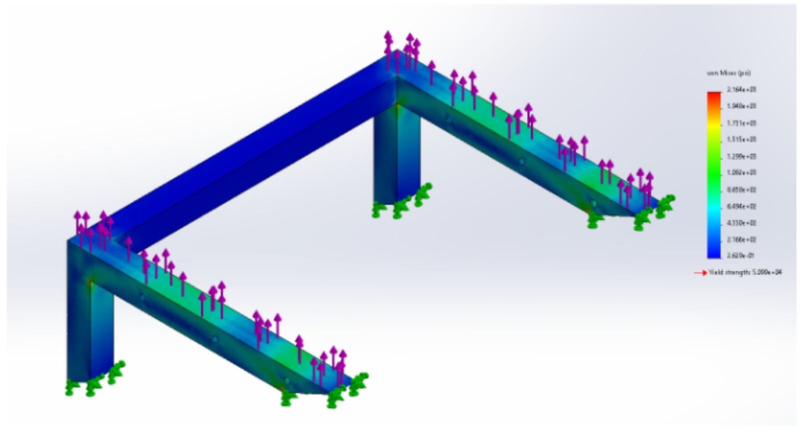


Figure 65: ANSYS Analysis of Four-bar Attachment Frame

Material: 6061-T6 Aluminum

Max Stress = 83.7 Psi

Yield Strength = 35,000 Psi

FOS = 418

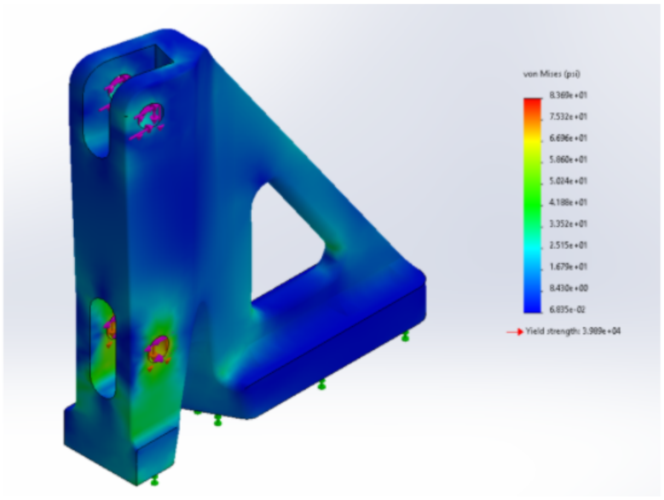


Figure 66: ANSYS Analysis of Left Angle Bracket