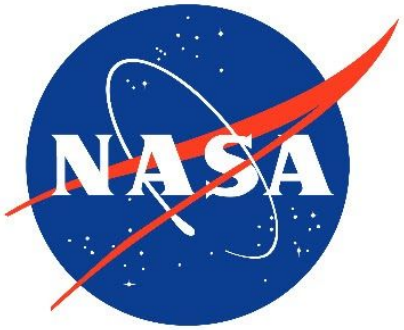

EZ-RASSOR

NASA EZ-RASSOR 2.0

UCF Fall 2019 Gold Team

Team Members:
Autumn Esponda
Martin Power
Daniel Silva
Daniel Simoes
Chin Winn

Project Sponsor:
Michael Conroy

Table of Contents

Introduction

Last year, UCF's original EZ-RASSOR Senior Design team developed software for a modular version of NASA's RASSOR, an autonomous regolith mining rover. The primary goal of this smaller scale rover, named the EZ-RASSOR, was to provide a functional demo for visitors at the Kennedy Space Center. Visitors are now able to interact with a smaller rover that demonstrates the RASSOR's capabilities. A key feature of this software is that it is entirely open source. This not only grants access to this work for educational purposes, but also allows for the development and iteration of this software by some of the best minds in the Robotics and AI industries.

In terms of functionality, the EZ-RASSOR is currently capable of the following, either autonomously or while being controlled via a mobile application:

- Roving across slightly treacherous terrain
- Mining and storing regolith with rotating drums
- Returning to base and dumping stored regolith
- Autonomously detecting and avoiding obstacles

There is also a simulation created to test the features of the EZ-RASSOR, along with a model mirroring the real RASSOR, as shown in Figure 1.

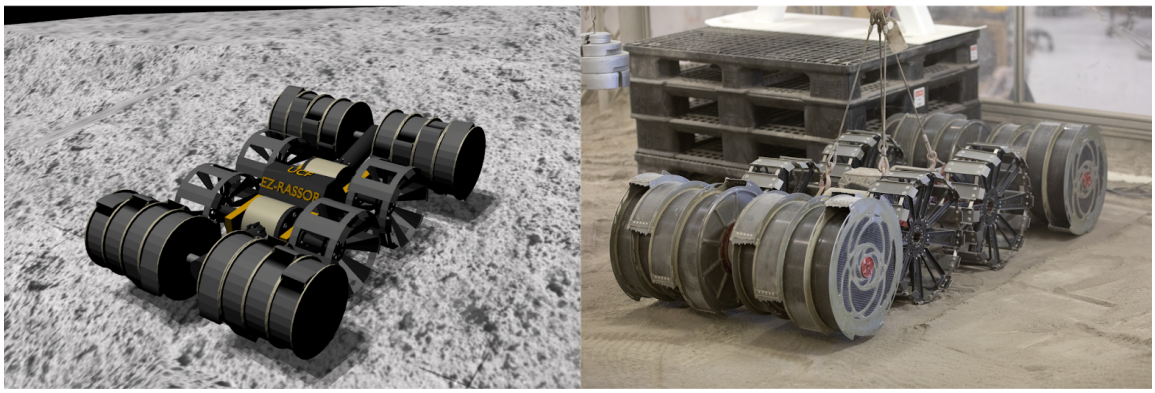

Figure 1 | Gazebo simulation model of EZ-RASSOR (left); NASA's RASSOR rover (right)

Project Narrative

The NASA EZ-RASSOR 2.0 project's purpose is to build upon the existing EZ-RASSOR project to develop a swarm control and task management system for large group of EZ-RASSOR robots. This project will entail the creation of an Artificially Intelligent system capable of breaking up high-level tasks into individual assignments and waypoints for each EZ-RASSOR rover. It will also involve enabling rovers to communicate and track data such as battery usage and encountered difficulties when completing the tasks. This will allow the system to make smarter and more efficient decisions in regards to routing and task divvying. The system must also avoid scenarios such as rover collision and dual tasking, while considering the possible effects rovers will have on their environment, such as deterioration of paths due to overuse. The system will maintain an obstacle and resource map of the surrounding environment. Overall, this system should be able to intelligently and efficiently route rovers, given knowledge of the environment and rover status.

Figure 2 | NASA's RASSOR in the large lunar regolith bin

Project Members

Project Sponsor:	Mike Conroy	mike.conroy@ucf.edu
Stakeholders/Mentor:	Kurt W. Leucht	kurt.leucht@nasa.gov
Development Team:	Autumn Esponda	autumnesponda@knights.ucf.edu
	Chin Winn	winnchintip@knights.ucf.edu
	Daniel Silva	danielzgsilva@knights.ucf.edu
	Daniel Simoes	ruptor@knights.ucf.edu
	Martin Power	power.martin@knights.ucf.edu
Senior Design Professors:	Dr. Mark Heinrich	heinrich@cs.ucf.edu
	Dr. Rick Leinecker	Richard.Leinecker@ucf.edu

Scope of Work

FSI and NASA determined that Senior Design work on EZ-RASSOR shall be delegated to two teams: Black Team and Gold Team. This is in accordance with the development of two major functions of this project, which is detailed in the table below.

Black Team	GPS-Denied Navigation and Localization
Gold Team	Artificial Intelligence and Swarm Control

Deliverables

By the end of the project timeframe, the EZ-RASSOR Gold Team shall have these deliverables:

- Github repository containing all code related to the development of this project which may include:
  - Gazebo simulations
  - ROS nodes
  - Machine Learning scripts
  - Swarm path planning and scheduling algorithms
  - And more, if applicable
- Final Design Document detailing the development of this project by the team, and all related information such as user documentation, and more
- PowerPoint presentation detailing the development of the project
- A NASA demonstration of the successful completion of the project requirement
- A UCF demonstration of the successful completion of the project requirement

Broader Impacts

The EZ-RASSOR project is meant to be an educational version of NASA's RASSOR that they developed for lunar excavation. On a local scale, NASA plans to send representatives to schools in order to give them presentations on what it is that they do. The EZ-RASSOR will be used at these presentations to give students a first hand example of the types of innovations that are created at NASA. Allowing younger students to be exposed to different projects like the EZ-RASSOR early on can help get them interested in robotics and computer science, which in turn can help inspire the innovators of tomorrow.

On a global scale, the EZ-RASSOR source code is open source and available to the public. There are already some companies using the source code to manage their own excavators and with EZ-RASSOR 2.0 there will potentially be even more. Additionally, the EZ-RASSOR project being open source gives NASA the ability to interface with the robotics community and use their ideas and implementations to improve their in-house projects. Although the EZ-RASSOR software itself will not be used in NASA's space exploration projects, this project serves as a means for cost efficient research and development for the original RASSOR on a smaller scale. Hence, it can be said that advancements made on the EZ-RASSOR software could very well impact the technologies developed by NASA to begin the colonization of outer celestial bodies.

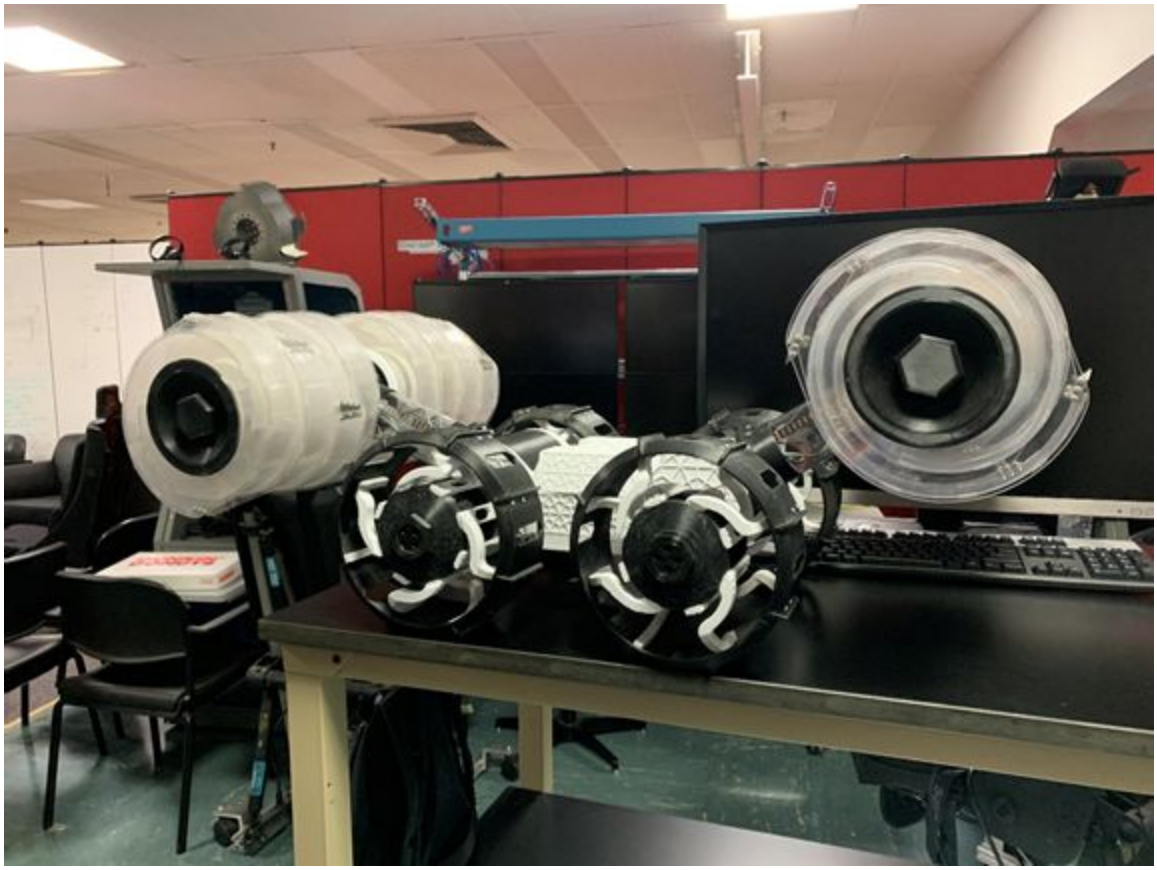

Figure 3 | A 3D printed EZ-RASSOR

The EZ-RASSOR project is also accessible to schools and universities. Almost all of the parts are 3D-printable and the software is completely open source. Schools, science centers, and universities can use their own 3D printers or pay to have the robot printed, then load and customize the software to their own needs and interests. Documentation is extremely important because of this, since there shouldn't be huge hurdles to using this technology in a research or education environment.

Motivations

Autumn Esponda

Figure 4 | A portrait of Autumn Esponda

For me, the EZ-RASSOR project really stood out amidst the multitude of other projects for a few reasons. First was that it is sponsored by NASA. I feel like that's a given, but I've personally always been fascinated by space travel and exploration. As a kid growing up in Florida, I grew accustomed to making trips out to Cape Canaveral for various space launches, visits to Kennedy Space Center

Visitor Center, and museums on the space coast. Being given the opportunity to work on a NASA project to help future exploration and colonization efforts on the moon and beyond made choosing this project a no-brainer.

I am beyond thankful for the opportunity to work alongside some amazingly talented engineers at NASA, Florida Space Institute, Florida SwampWorks, and those on the previous EZ-RASSOR team. Last years team really has set us up for success by giving us a well-documented piece of very functional software for us to expand upon. The opportunity to learn from the best on such an ambitious project was a huge draw for me as well. This project will help me through all my future endeavors and is going to be something I will be proud to talk about for the rest of my career.

Martin Power

Figure 5 | A portrait of Marty Power

I was motivated to work on the EZ-RASSOR project for many reasons. First was my interest in robotics, which I've had for a long time. I began doing robotics in middle school and continued through all of high school doing First Tech Challenge. However, I didn't work on any robotics projects during college and the EZ-RASSOR presentation made me realize how much I missed working with robots. Additionally, the area of robotics we will be working in, swarm control, has always been of interest to me and I am excited to learn more about it.

I’m very excited to work on a project with a team of experienced engineers at NASA who will be able to provide us guidance and mentorship. Because of this, the project will more closely mirror what working in the industry will be like, and help prepare me for my job after college.

Daniel Silva

Figure 6 | A portrait of Daniel Silva

The EZ-RASSOR project captured my attention in a number of ways, both in regards to my intrinsic motivations as well as career interests. Since a young age, topics such as autonomous vehicles, artificial intelligence and space exploration as a whole have captivated my mind and ultimately drove me to

pursue a career in computer science. This project offers not only an opportunity to gain experience in these fields, but to contribute to truly pushing the boundaries of our society. When speaking about the future of humanity, and more specifically the colonization of additional planets, a topic commonly on the forefront of such discussions is the feasibility and possible approaches to such a goal. NASA's RASSOR aims to tackle this by developing a fully autonomous hive of rovers whose purpose is to explore, analyze and terraform celestial bodies without the need for physical human interaction. This end goal is one that I find absolutely awe inspiring and is where I find my motivation. In addition to such a riveting and powerful mission, I am thrilled to work alongside and learn from engineers at the Kennedy Space Center, some of the best minds in these domains.

In terms of career advancement, working to enhance the autonomous system behind the EZ-RASSOR will allow me to leverage and further my knowledge in computer vision. On top of this, I will be given the opportunity to dive into reinforcement learning, swarm AI and robotics, topics which I have been longing to learn more about. These are the sorts of problems I hope to work on for the remainder of my career and I believe this once in a lifetime opportunity will propel my career in this direction. While this endeavor will be undoubtedly difficult, I am grateful for the opportunity to work on a project with such societal impact and personal significance.

Daniel Simoes

Figure 7 | A portrait of Daniel Simoes

When I was first deciding which project I wanted to do for Senior Design. I knew that I wanted to pursue a project in Artificial Intelligence or Machine Learning. Artificial Intelligence was always the career path I wanted to take when I began studying Computer Science, but I knew it'd be difficult to get experience in the field. Most internships only provide experience in software engineering, which can only help so much. I had also only worked in one internship previously with Lockheed Martin. So I didn't have many experiences to put on my resume. When I saw the NASA EZ-RASSOR project, I knew that I had to be a part of it.

Not only did this project provide the opportunity I needed to develop some real world experience with Artificial Intelligence, but it also granted me an internship with NASA. This kills two birds with one stone, so naturally I ranked this project as my first choice. Now that I managed to be selected as one of the members of

this team, I was to work as hard as possible to deliver results that will make me and me team members proud.

Chin Winn

Figure 8 | A portrait of Chin Winn. I hereby apologize for the lack of a better picture.

Picking this project as my first choice wasn't a tough decision for me, considering it was one of the only projects to be sponsored by NASA (and the NASA intern position was also a big plus). Coming into this year I have had two internships, one at a large, well-known company, but I've still never had any real-world experience with Artificial Intelligence and Machine Learning. As someone who is interested in ML and have been doing some personal projects related to ML for a while, the fact that this project heavily involves those topics really surprised me and piqued my interest even more than I was initially.

The opportunity to get to work directly with NASA and also hands-on experience with AI/ML is an experience that I would have never expected to get, but here I am. So given the opportunity, I will learn as much as I can and put in my share of effort to be a good teammate and deliver a functional project.

Initial Objectives

The initial outlined objective for the EZ-RASSOR 2.0 project is to enhance the autonomous functionality of the original EZ-RASSOR. On a broad spectrum, this will involve the improvement of the autonomous navigation, obstacle avoidance, swarm management and cosmic GPS systems.

The gold team specifically has been assigned to the design and development of an intelligent system, capable of managing a hive of individual EZ-RASSOR rovers. This swarm AI system will need to handle things such as rover task management and routing. To do so, the system must also be capable of maintaining a coordinate map of the surrounding environment to track points such as obstacles, hazards and resource locations. Using this, the system should be able to make truly intelligent rover tasking and routing decisions. In order to facilitate the creation of this map, individual rovers will need to be able to communicate amongst themselves, as well as with the central system. Specifically, rovers will send data such as current battery status, past usage, located obstacles or resources, and more. This information will be stored and used by the swarm AI system to add to the environment map and affect decisions.

Regarding tasking for a swarm of rovers, the scope of possible tasks is quite narrow for the time being. Currently, high-level tasks are limited to group mining of an area by creating a quarry or ditch at an angle which rovers are capable of navigating, as well as basic environment exploration.

Goals

Expanding on the objectives described above, our (gold team) main goal is to solve (or help solve) the problem of task management and efficiency. Rovers that can be deployed, communicate, navigate, and work together autonomously as a swarm would greatly reduce the need for human intervention, which means less resources are needed, and in turn translates to better efficiency.

The team has set the following goals in regard to developing an intelligent swarm management and tasking system for the EZ-RASSOR:

- Build and store a connected graph or grid map representation of the moon's surface
    - This map can be instantiated with NASA's Lunar Surface Model, which will be discussed in detail further along in this document.
    - This environment map should track large obstacles and terrain which the EZ-RASSOR would not be able to navigate
- Design and implement pathfinding and scheduling algorithms to handle the management and routing of multiple rovers
    - These algorithms will run on the above described environment map. Hence, the map representation should be suitable for these algorithms.
    - Calculated paths should not collide with known obstacles, treacherous terrain, or other rovers.
    - The starting point of such paths will be the rovers home base and the end point will be the current dig site
- Schedule rovers in an intelligent manner which takes into account each rover's individual battery life
    - Rover's should be sent back to the home base in a timely and safe manner for recharging
    - Scheduling should also ensure at least one rover is always currently mining at the dig site

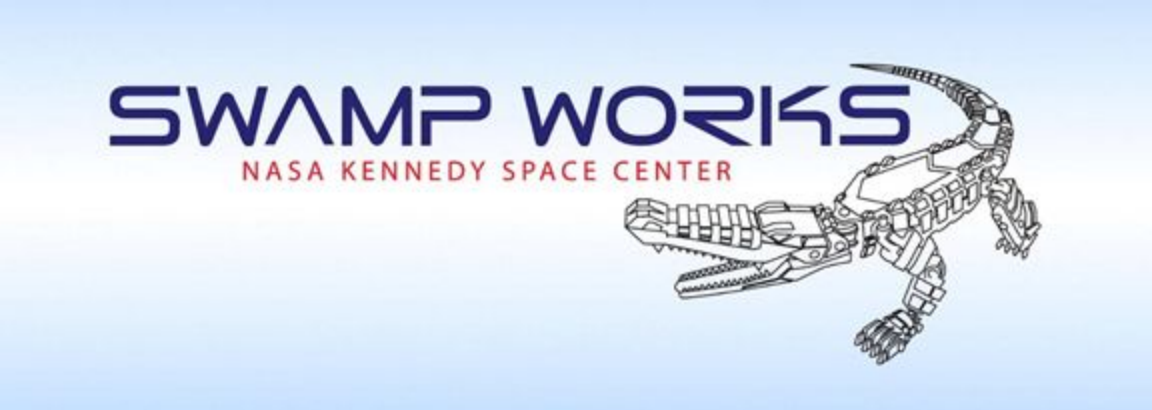

Figure 9 | Logo of Swamp Works

Function

Given a high level task, a swarm of individual EZ-RASSOR rovers should be able to divide work intelligently among themselves to efficiently complete the task. The artificially intelligent system will need to take into account previously received signals from individual rovers, as well as known information regarding the environment, in order to make truly intelligent tasking and routing decisions. To facilitate this functionality, the central AI system will support the creation of a coordinate map which will track several attributes of the surrounding environment. This map will be updated as rovers return information based on their experiences and encountered difficulties. A given swarm of rovers should be able to communicate to the central system in order to deliver their current status and location.

Legal, Ethical and Privacy Issues

Legal Issues

Due to the novel nature of the technologies that are being researched for this project, our team will have to rely quite heavily on the few bits of research that are already far and in between in order to come up with a solution that is anywhere close to being feasible for our project. Naturally one of the biggest issues from a legal issue comes from potential plagiarism of existing research if we are not careful. But this can be solved rather easily through proper citations. Another major issue that we come across is if we try to make use of a solution that is owned by an existing copyright holder. Our team will have to be careful to ensure that whichever solution we decide to employ for our swarm control will either be completely new, or at least ensure that is not already someone else's intellectual property.

Another potential legal issue that, while not directly related to the development of the project now, could come up at a later time is that every team member signed an NDA with NASA when we agreed to become unpaid interns for them. This means that any trade secrets or information we are told while on site during our trips to Kennedy Space Center's SwampWorks needs to remain tight lipped or legal action can be taken against us. It is hard to predict how this will impact our development of the project, but certainly, care should be taken so that no legal slip-ups are made.

Ethical Issues

Since our team is mainly in charge of incorporating AI into the EZ-RASSOR, it is important for us to address the ethical issues that come with AI on a prouder scale and how that affects the project we are working on. The first issue that needs to be addressed it the issue of possible unemployment when AI is introduced to a system. In our case, if there are 100 EZ-RASSORs being deployed on a planet with no AI for autonomy, then there would be 100 remote

pilots manning the EZ-RASSORs. Once AI and swarm functionality are added to these EZ-RASSORs, only 1 remote operator is needed for all 100 of them. This cuts out 99 potential employees that could be making a living off manning these excavators. A potential counter argument to this is the fact that if there was no autonomy in these excavators, then it wouldn't be profitable enough to deploy them anyways. So those 99 other potential employees would never have been hired for this job.

Privacy Issues

Regarding privacy issues, the EZ-RASSOR is known to be an open source alternative to NASA's own current RASSOR. From a legal standpoint, our team is unable to utilize NASA's code or hardware due to the fact that NASA is unwilling to share their code with others. This is mainly because NASA is a government owned property with sensitive information, while our EZ is entirely open source, designed to be used by anyone, including other countries. In turn, this means that NASA wouldn't want to give us information that other countries can utilize for their benefit. In relation to our entire project being open source, another issue can be seen where all of our code needs to be publicly available limiting us to the material that we can use and completely omitting the use of any copyrighted materials.

Another privacy issue comes with accidentally including personally identifiable information in either the GitHub codebase or the EZ-RASSOR wiki. Some of this information would potentially include full names or IP addresses depending on if there are networking related task, which there most likely will be.

Initial Thoughts on Implementation

As of Friday October 18, 2019 we got a lot more information from our meeting and discussion with the SwampWorks team, which provided more clarification as to what our short-term and long-term goals are, and how should we approach it. We also talked about what we should prioritize and complete before implementing other features.

As the main point of discussion for our meeting was about the main high-level task that the rover was designed to do— digging regolith, we discussed in more depth about what the task entails.

The “digging” process

The digging process, which sounds deceptively simple, turns out to be a much more complex and intricate process than what we initially thought. On the lunar surface, the rovers have to dig in a specific manner that will maximize the chance of the rovers getting in and out of dig sites or trenches safely and efficiently. For example, the slope into the trench or dig site has to be gentle enough for the rover to get in without falling or tripping, and get out without being stuck in the trench. Here, the most likely approach to digging that we are going to use is to let the rovers dig in layers with decrementing lengths. If we imagine a flat area of regolith, and we let the rover dig in a straight line (this will be one layer) of length N, and dig in a straight line in the opposite direction with length N-1, eventually we will have the slopes of the trench be gentle enough for the rovers to get in and out easily.

Why is it so important?

We also discussed more about why digging was important-- our main mission on the lunar surface is to gather usable resources, which means we have to dig for regolith. Regolith is very much abundant on the lunar surface, and contains oxygen, water, and other important substances that can be extracted for usage.

Short-term priorities

In terms of priority (in the short term at least), our discussion with the SwampWorks team suggests that we work on the simulation environment (Gazebo) to make it as complete as possible, with important features such as digging regolith and the processing of materials fully working and be simulated as accurately as possible. The simulation environment will be very critical to the development of our project since a bad simulation environment means we are unable to realistically and reliably test our implementation, which, in turn, will make us more susceptible to bad observations, and consequently, results.

Stretch suggestions from SwampWorks

A (stretch) good addition to the digging routine would be for the rover to be able to compute the amount of each substance we can process and extract from the regolith that has been dug based on assumptions about the composition of the regolith. (Density, less water on the surface, but more water under the surface, etc). This would be good for planning and optimization of the routine to improve efficiency (work less and get more).

Initial thoughts on rover communication schemes

All of those tasks mentioned above require a lot of computing power, of which the low-power Atom chip most likely will not be able to provide. So we assume that a lander (with much higher computing power) is in the area, processing and sending information to the rovers in its vicinity. This would help the rovers out in terms of processing data sent from the rovers into meaningful data and sending simple, usable bits of data that can be processed easily by the rovers, to them. This could be implemented using some kind of atomic, transactional queue. For example, consider this scenario: rover A takes many stereo pictures of the lunar landscape from its camera. Once rover A gets back in the area that is in the vicinity of the lander, the data in the queue is popped and sent to the lander, to be processed. To prevent corrupted data we implement the queue to be atomic, to account for interruptions, which is expected to be a problem on the lunar

surface. While rover A is offloading the data to the lander, rover B is busy gathering data a bit further away from the lander. Rover B will store the data in its internal memory, until rover B gets back to the area and starts pushing information to the lander. By this time, the lander has done processing rover A's data, thus, is ready to send the processed (ie. useful) data, to rovers A and B. Now rover B also has knowledge of the world around it that came from rover A. We can assume this behavior for rovers 1...N, as a swarm.

A need for scheduling algorithms

Considering that we know that all rovers will be dropped off from the lander and start at the same place, they will all have to return to the same lander. With this in mind and with the lander also being a charging station for the rovers (each full-sized RASSOR can work for 6 hours with no breaks until depleted, and charges for 2 hours until full), it would follow that we need to find a good scheduling algorithm in order to utilize all the charging stations present in order to get the most battery charging during the time allotted, and we also have to ensure that we have other rovers working out there while some are occupying all charging stations. A possible solution would be that we (if possible) could treat the scheduling problem as a constraint satisfaction problem, and perhaps try and use a constraint language to solve it. Here our constraints would be the minimum number of rovers that are required to be active at all times, the amount of rovers vs available charging stations, and task priority (?). Research on this approach is still pending, but it is a possible approach.

What is Swarm Computing and how does it relate to our project?

The idea is rather simple— we have relatively simple rovers that we want them to perform high-level tasks. We put them together with some kind of communication enabled for the rovers to talk to coordinate with each other to collectively perform a task, akin to a colony of ants building a fortress, or a group of beavers building a dam.

Although the concept sounds relatively simple, the technical implementation can be described as complex, and is a relatively new concept that is still being actively researched today.

Some thoughts on development / testing of ML model

Currently Gazebo is our only simulation environment for testing everything that has to do with the rovers. Though, in terms of Machine Learning testing, there is a good alternative in OpenAI Gym that we can use to test our models before we implement them in Gazebo. As of our current understanding, OpenAI Gym would provide a “playground” of sorts for us to test an even simpler version of our rovers to see how they respond to a simple environment, and to see if the model works at all.

Hardware Details

During our on-site meeting with the SwampWorks team at Kennedy Space Center, we discussed the hardware specifications of the EZ-RASSOR and this is what the RASSOR team at SwampWorks provided us with:

Intel Atom Z8350

Performance	
# of Cores	4
# of Threads	4
Processor Base Frequency	1.44 GHz
Burst Frequency	1.92 GHz
Cache	2 MB
Scenario Design Power (SDP)	2 W

Figure 10 | Specs for the Intel Atom Z8350

The processor, which is the main computing unit for each rover, is an Intel Atom Z8350 chip. The processor is a 4 cores, 4 threads processor with a clock speed of ~2GHz, and uses about 2W of power when running within its designed specifications. The processor is not particularly powerful in terms of computing power, but as we will detail later on in this document, this is a design choice that is reasonable and there are measures that we will take in order to make each rover work effectively within the bounds of its processing power provided by the Intel chip.

Intel RealSense T265 tracking camera

Operational Specifications	
Depth Resolution and FPS	848 X 800
Depth Field of View	D:163

Module Specifications	
Dimensions	108mmx24.5mmx12.5mm
Power	1.5 W
System Interface Type	USB 3

Figure 11 | Specs for the Intel RealSense T265 Camera

The camera that the SwampWorks team will be using with the EZ-RASSOR rover is the Intel RealSense T265 tracking camera, which will be connected to the rover over a USB 3.0 interface. The camera should give us a sufficiently detailed resolution of 848 X 800 pixels, with a field of view of 163 degrees which is almost hemispherical. We will expand on the importance of the camera and depth information later on in this document.

802.11ac Wi-Fi

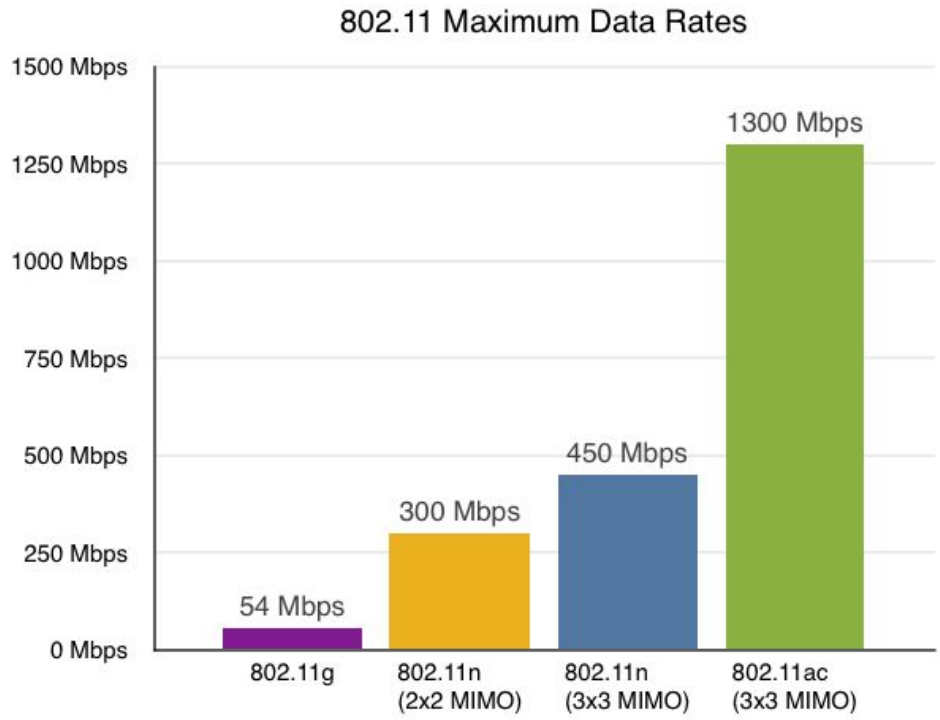

Figure 12 | Max data rates for different 802.11 standards

The rovers will be equipped with 802.11ac Wi-Fi for wireless communications. According to specs, the 802.11ac interface provides a bandwidth of up to 1300 Mbps, which should be sufficient for communication purposes in the context of this project. Our main concern here is not bandwidth, as we can see that the 802.11ac hardware should be able to easily provide, but rather, the operational range is of a larger concern. The 802.11ac standard hardware generally has an operational range of ~170ft at most, and since our rovers are limited in power we should not expect our rovers to be able to transmit signals in that maximum range at all times, seeing that we are operating within power limits. This constraint is to be further researched and tested.

Initial Specifications & Requirements

At the point in time of this set of requirements being drafted, the gold team had yet to hold a preliminary meeting with NASA's SwampWorks team. It was believed that the objectives and requirements for this project were likely to be narrowed or transformed following this meeting. It should also be noted that a number of the requirements below depend on the work completed by previous EZ-RASSOR teams, as well as the requirements defined for this year's black team. Hence, it could be said that the comprehensive set of requirements for the EZ-RASSOR 2.0 includes those of these other ventures.

After two initial meetings with our project sponsor, Mike Conroy of the Florida Space Institute, the following specifications and requirements were defined for the gold team specifically :

Req. #	Description
1	The EZ-RASSOR software shall remain open source
2	The EZ-RASSOR software shall continue to be well documented in order to facilitate further development and maintenance
3	The EZ-RASSOR software shall provide a solution that is presentable for demos around the Kennedy Space Center
4	The EZ-RASSOR software shall be designed to facilitate transfer between other similar robotic systems or simulation environments
5	The EZ-RASSOR software shall be compatible with and maintain consistent performance in a Gazebo simulation. ○ The simulation shall contain tests of the desired EZ-RASSOR features ○ Namely, the simulation should be able to handle the testing of rover swarm intelligence with up to 100 individuals units ○ The simulation shall be to scale in terms of measurements and physics of the environment

6	The EZ-RASSOR software shall utilize ROS as a central middleware system to run the robot's external hardware. ○ The central middleware system shall manage all relay between the simulation, autonomous control, and remote control modules
7	The EZ-RASSOR software should utilize swarm AI technology to enable a hive of rovers to handle tasks collaboratively and efficiently
8	Possible tasks at the moment are limited to the mining/return of materials and environment exploration
9	Individual rovers should communicate with the central swarm system in order to send signals and data such as battery consumption, wheel slip rate, lost rovers and found obstacles
10	The EZ-RASSOR system should build a 3d map of the surrounding area which tracks things such as known obstacles, hazards and resources
11	The swarm AI system should take into account previously received data and map information in order to avoid rover collisions, dual tasking, and creating routes which lead to known obstacles or hazards
12	Individual EZ-RASSORs should be capable of autonomous navigation
13	Individual EZ-RASSORs should exhibit obstacle detection and avoidance
14	Individual EZ-RASSORs should be able to detect nearby EZ-RASSORs and communicate with them
15	Individual EZ-RASSORs are responsible for the autonomous gathering and depositing of materials into the processing unit of the lander

16	Individual EZ-RASSORs should be capable of self localization and have the ability to communicate their location to the central swarm system
17	The coordinate grid and rover exploration capabilities shall be within a limited range
18	The lander should have a good scheduling scheme to dispatch EZ-RASSORs to work and recharge their batteries at the charging station

Table ${#} | Key Project Requirements

Current Specifications & Requirements

Following the team's meeting with NASA's SwampWorks research team, many of the above requirements were altered, dropped, or defined as stretch goals. The team also held biweekly meetings with our mentor, Kurt W. Leucht. As the semester progressed, these meetings provided useful discussions and insights into the feasibility and possible solutions to these requirements. These meetings also resulted in significant developments in our requirement list, leading to the updated requirement list below. Note that this requirement listing is as of December 2019.

Req. #	Description
1	The EZ-RASSOR software shall remain open source
2	The EZ-RASSOR software shall continue to be well documented in order to facilitate further development and maintenance
3	The EZ-RASSOR software shall provide a solution that is presentable for demos around the Kennedy Space Center
4	The EZ-RASSOR software shall be designed to facilitate transfer between other similar robotic systems or simulation environments
5	The EZ-RASSOR software shall be compatible with and maintain consistent performance in a Gazebo simulation. ○ The simulation shall contain tests of the desired EZ-RASSOR features ○ Namely, the simulation should be able to handle the testing of rover swarm intelligence with up to 20 individuals units ○ The simulation shall be to scale in terms of measurements and physics of the environment
6	The EZ-RASSOR software shall utilize ROS as a central middleware system to run the robot's external hardware.

	○ The central middleware system shall manage all relay between the simulation, autonomous control, and remote control modules
7	The EZ-RASSOR software should utilize swarm AI technology to enable a hive of rovers to handle tasks collaboratively and efficiently
8	Tasks are limited to the mining and return of materials/regolith
9	Given the coordinates of a dig site or area, the Swarm AI system should generate paths and schedules for each individual rover. Paths will be returned in the form of waypoints for each rover to navigate towards.
10	The EZ-RASSOR system should maintain a coordinate map of moon environment surrounding a lander. This map should be instantiated with NASA's Lunar Surface Model and hence, will only be as accurate as NASA's data permits.
11	The environment map should track large obstacles and terrain which the EZ-RASSOR would not be able to navigate.
12	The coordinate map should be stored in such a way that allows for path planning algorithms to be ran. Likely in the form of a connected graph.
13	Generated paths should not collide with known obstacles, treacherous terrain, or other rovers.
14	Rover paths and scheduling should be constrained by a rovers potential battery life. The central system is responsible for ensuring rovers are brought back to the lander for charging in a timely manner.
15	Rover scheduling shall be done in such a way that ensures at least one rover is always mining at the dig site.
16	Individual EZ-RASSORs should be capable of autonomous navigation

17	Individual EZ-RASSORs should exhibit obstacle detection and avoidance
18	Individual EZ-RASSORs are responsible for the autonomous gathering and depositing of materials into the processing unit of the lander
19	The coordinate grid and rover exploration capabilities shall be within a limited range

Table 1: Key Project Requirements

Stretch Goals and Nice to Haves

Req. #	Description
1	The EZ-RASSOR system should store a 3d representation of the surrounding area to tracks things such as known obstacles, hazards, resources, and slope of terrain.
2	Individual EZ-RASSORs should be capable of self localization
3	Individual rovers should communicate with the central swarm system in order to send signals and metrics such as battery consumption, wheel slip rate, and lost rovers at specific locations.
4	As rovers navigate the area, they should create 3d point cloud representations of the environment and send this data, as well as locations where obstacles were found, back to the central system
5	The swarm AI system should be capable of updating its environment map, based on data and signals received from exploring rovers. This would ultimately lead to an increasingly accurate and expanding representation of the moon environment.

Table 1: Project Stretch Goals

Additional Ideas from the Team

Autumn Esponda

- Division of singular large task into subtasks
    - The user should only have to input a single and very high level command (“Build a road that goes from here to there”)
    - System intelligence should be able to adequately divide up work between each rover
    - Potentially assign certain rovers to be stationary and act as relay stations for mesh network
        - Take into account statistics on individual rovers and their past performance, current battery levels, etc.
- Load balancing between rovers assigned to a task
    - Not sending any individual rovers to perform a task they don’t have power for
    - When reading sensor data, system needs to track which rovers provide inaccurate or erroneous data
        - Such a rover could be always be used as a relay in the mesh network
        - Sensor data can be corrected by sending another known working rover to measure the same data points and apply a curve to inaccurate rover’s data
    - System should take into account the slope of terrain as it takes more power to move uphill and less downhill when compared to level surface
- Mesh network
    - This will be key to enabling the system to work over long distances
        - Will be required on the Moon and beyond as there isn’t an established satellite system to keep base up to date with all of the rovers spread around
    - Certain rovers can be stationed as relay points
    - Rovers working on task can provide status updates on a regular interval to the relay (and thus to the base station)
    - Network will need to take into account if a rover goes dead or missing
        - Perhaps implement redundancy to protect the network if a relay goes offline

Martin Power

- Data-driven range map
    - Begin with short-range map and update based on rover sensor and battery data
    - Slowly send rovers out on longer missions to get a better representation of the terrain
    - Maintain attributes of individual rovers
        - Know which rovers tend to consume more power/have less battery life
        - Know which rovers have inaccurate sensors
- Automate task assignment
    - AI which can be given high-level list of tasks and divide work among individual rovers
    - Given parameters such as:
        - Amount of regolith to collect
        - Area to create range map of
        - How many rovers are available to work with
- Distributed multi-agent system
    - Each rover has an idea of its capabilities/current tasks
    - Rovers communicate between each other and divide tasks among themselves
    - Each one has an idea of where it is in relation to other robots
    - Decide where to go and where to mine based on the location of others
    - Be able to detect when a rover has become non-functional so the task can be reassigned and the rover can possibly be recovered
- Communication framework
    - Use WiFi to transmit commands/create mesh network, depending on what we decide upon
    - WiFi is natively supported by ROS
    - WiFi is available in all educational contexts without any need for additional devices/setup, increasing accessibility
- Recommendations and predictions
    - Collect data about individual rovers to alert that one may soon malfunction

    - Be able to give time estimates for how long a task will take based on the yield of a dig site and how efficiently the EZ-RASSORs are able to dig.

Daniel Silva

- Master node architecture vs distributed intelligence
    - Master node may be more efficient and plausible given limited computing power on rovers. The master node itself could have a larger computing architecture, allowing minions to to be weaker
    - Distributed intelligence would protect against system failure in the case of master node loss
    - Nodes will need to communicate with one another in order to send signals based on experience
- Using a grid system to represent the area surrounding rover charging stations
    - Individual coordinates may require several data points such as slope, obstacles, surface conditions,
    - Coordinate system is updated as rovers explore the surrounding area and signals are sent back from exploring rovers
    - Data could include rover slip rate, battery consumption, lost rovers, stagnant periods, loss of control, etc. at specific coordinate areas.
- Reinforcement learning for swarm control
    - Algorithm outputs efficient routes and handles task management for each individual rover
    - Learning signals could involve rover collision, dual tasking, known obstacles, surface conditions, rover positioning, rover battery life, etc
    - Additional, more specific tasks may be assigned following the initial meeting with NASA's SwampWorks team
    - RL has been making great advancements in swarm control and agent learning systems. Could prove to be a great solution to the bulk of our assignment

Daniel Simoes

- We should build our own small remote controlled vehicle to test our implementations
- Our implementation should be at a higher level than the current functions

    - Our “director” should make use of the current API implemented to call the provided functions.
- Make use of black team’s GPS denied navigation system to receive location information for all of the rovers.
    - We must agree on how the API endpoints will be presented for the locations.
    - Our implementation needs to not rely heavily on black team’s implementation in case they don’t finish.

Chin Winn

- Pipeline improvements and software engineering nice-to-haves :
    - Implement a continuous integration/continuous delivery pipeline (if possible) to simplify the build and deployment process end-to-end
    - Project should be test-driven; write test suites with as much coverage as possible, with unit tests and integration tests being functions that will be used in deployment in order to enforce good programming practices and minimize defects in production
    - Ensure that the simulation suite has a one-to-one (or as close as possible) relationship with the real RASSOR project, since simulation is cheaper and can be utilized as often as required to be, knowing that the simulation reflects the real behavior of the rover would be important

- Hardware simulation :
    - We can assume that the rover which uses an Intel Atom processor, will not be able to process sophisticated data (i.e., depth map, stereo image from the camera) from the RealSense camera at a satisfactory amount of frames per second
    - We will need to emulate the computing power of the rover by limiting our systems’ CPU core counts and clock speeds
    - This enables us to be able to profile the performance of our code
    - Push the rover chip as far as we can, and offload the complicated tasks that require a lot of computing power to the base station.

Research and Investigation

The following sections will discuss the research that was done in order to understand the current design and structure of the project as well as what the team needed to research in order to meet the requirements agreed upon with the team at the SwampWorks lab. Not all of the technologies discussed below are to be incorporated into the final product, but were researched in order to find the most optimal solution for each problem or task.

The problem of swarm management can be broken into several distinct problems, each of which depend on the solution of the others to succeed. Firstly, the central management system must maintain a map of the surrounding environment to track points of interest such as obstacles and resources. Given a high-level task, the system could then leverage this environment map to assign target locations and generate paths for each individual rover. These paths must avoid collisions with known obstacles, as well as collisions with other EZ-RASSOR rovers.

Lastly, rovers must be scheduled in an efficient and intelligent manner. Potential solutions to each of these problems (mapping, pathing, and scheduling) are discussed at length below.

Machine Learning

Our intelligence may use some type of machine learning to accomplish its task of dividing up the tasks among a swarm of EZ-RASSOR rovers efficiently and effectively. Machine learning is typically broken up into four categories: Supervised Learning, Unsupervised Learning, Semi-Supervised Learning, and Reinforcement Learning.

Supervised Learning

Supervised learning involves training a machine learning model on previously tagged data, and using this tagged data to train the model to recognize certain characteristics. This type of machine learning is typically used in domains such as image recognition and making predictions. While our intelligence will be

making predictions about when rovers will fail, that won't be the main focus and will likely be handled more simply.

Unsupervised Learning

Unsupervised learning is where the data is untagged and the machine learning model extracts its own "features", which are just patterns in the data. This is useful for learning about and analyzing data, and can reveal a structure to data that seems completely unstructured. Unsupervised learning likely won't be useful to our group, since we have a very clear goal in mind, to optimize the efficiency of resource gathering.

Semi-Supervised Learning

Semi-supervised learning is using a combination of supervised and unsupervised learning, typically using the tagged data to begin training the model and then continuing to train it on the untagged data and allowing it more freedom in picking out important features. Semi-supervised learning won't be very useful to us, for the reasons that supervised and unsupervised learning weren't useful to use either.

Reinforcement Learning

Reinforcement learning more closely models how humans learn. The model is given a task and repeatedly executes the task, all the while changing its strategy to see what works. The model is given rewards for completing the task, with more rewards given for accomplishing the desired task "better". Better could mean a multitude of things, but in our case better is more efficiently using resources such as time and energy. Reinforcement Learning seems like the best paradigm for our group to use to train our model, as we are trying to optimize a task and won't have access to tagged data.

Environment Mapping

The first step in designing the swarm AI system will be to build a map or representation of the moon environment. Nearly all of the functionality of the swarm AI system will be dependent on this map. Therefore, it is of utmost importance that the map is both accurate and reliable. Another key consideration

will be the storage efficiency of such a map. While the exact specifications of the lunar base are not set, NASA SwampWorks team has made it clear that we are expected to keep memory consumption and computation as low as possible. However, the proposed plan is to keep this map and the majority of computation on a machine located on the moon base. This allows our team to not necessarily be constrained by computational power, and rather focus on strong implementation of these technologies.

The primary focus of this project will be to get a swarm AI system up and running, complete with rover path planning and scheduling based off of this environment map. Additional stretch goals will include allowing this environment to be updated and grow as rovers explore the surrounding environment. Therefore, beginning with a static environment would allow our team to focus on the base requirements before delving into our stretch goals.

However, if we will not be leveraging rovers in order to learn and build an environment map, the question which remains is how could we still create a map which represents the moon's environment. NASA has previously used the Lunar Reconnaissance Orbiter to capture a high-resolution topographic map of the moon’s surface. Thankfully, NASA’s SwampWorks team has communicated to us that they would be able to supply us with this map of the moons elevation. This topographic map looks a bit like Figure 13.

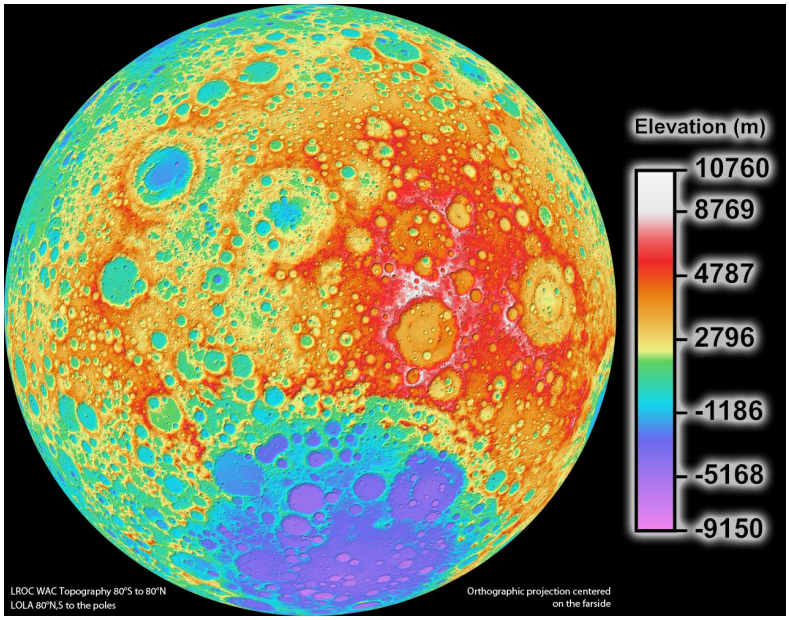

Figure 13 | Elevation map of the moon from NASAs LOLA instrument array, part of the LRO

Map Representation

The primary requirement of the environment map within the swarm AI system is that it must be of a form which allows path finding and scheduling algorithms to run on it. Several complex options exist which satisfy this requirement including navigation meshes [19], visibility points [20] and quadtrees [21]. However, a graph abstraction that generates too few nodes, such as a visibility graph, may render a multi-agent pathfinding problem unsolvable, even though it works for the single agent case. On the other hand, a graph obtained from imposing a regular grid contains more nodes, covering all locations of the traversable space, and offers more path options to avoid collisions between units. Hence, grid maps, besides being very popular and easy to implement, are more suitable to multi-agent problems. Below is a visualization of an example grid map, and how this would translate to a connected, undirected graph.

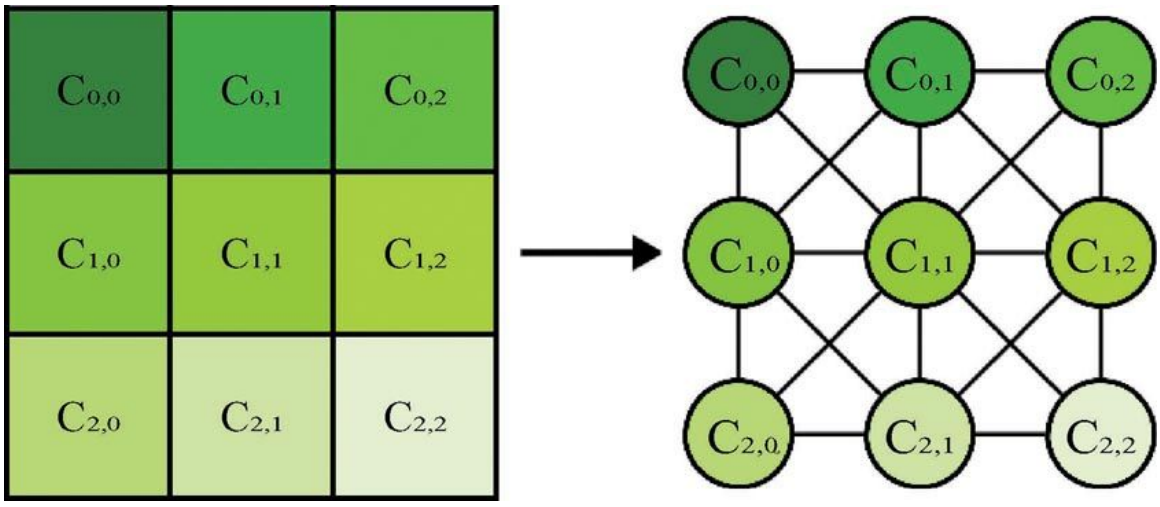

Figure 14 | (Left) An environment grid; (Right) The corresponding connected graph representation of the environment grid

Due to its ease of implementation and practicality, this grid map approach will likely be the preferred method for our team to transform NASA's topographic map into a data structure suitable for path finding and scheduling algorithms.

OctoMap

One of the stretch goals we've defined for this project is to enable the map to be dynamically updated as the surrounding environment is explored by individual rovers. This would allow the system to learn more about its world and make

decisions which leverage this knowledge gained over time. With a now ever-changing environment, we believe there are three main requirements which must be upheld in order to maintain a dynamic map.

- Probabilistic: This map is directly dependent on data received from individual rovers, as well as the accuracy of rover localization. For example, if a rover finds an obstacle and attempts to update the map, it must localize itself to get the coordinates at which the obstacle is at. This localization will likely be done in a GPS-denied scenario, and hence, will have a large degree of error. Due to this, our map should represent the environment in a probabilistic fashion. The probability associated with an area can then adapt each time a rover reports on given area. This probabilistic representation will account for the underlying uncertainty in received data and allow for multiple rovers to work together in building our environment map.
- Model un-explored areas: Our system can only plan obstacle-free paths for areas which have already been explored. Hence, unmapped areas must also be represented so that the system can avoid such areas. The system can also use this information to ensure individuals rovers act accordingly when exploring unmapped areas.
- Efficient: With a map as large as one of the moon, memory consumption will likely be a major battle. Our mapping method must be efficient enough to be stored in the systems main memory and be readily available for all EZ-RASSOR rovers.

Several approaches, such as methods which rely on point clouds, multi-level surface maps, or volumetric representations, have been proposed to model 3D spaces, however none meet all our requirements simultaneously.
An Octree is a hierarchical data structure designed for 3D spatial subdivision. Each node in an octree represents a smaller 3D volume, or voxel, and this voxel is recursively subdivided until a minimum voxel size is reached. [1] uses this data structure to offer a probabilistic and memory efficient method of mapping 3d space. In this approach, each cell is represented as a Boolean value representing whether that space is occupied. When a cell is noted as occupied, then that space is initialized in memory. This greatly reduces memory consumption. Of course, unoccupied space essentially represents un-explored space. In order to deal with rover inaccuracy, [1] represents the occupancy of a

cell with a probability, and these probabilities are thresholded when being used for navigation. A voxel is only considered to be occupied when the threshold is reached and assumed to be free otherwise. Probabilities themselves are calculated with what is essentially a Bayesian probability, where the current confidence of that voxel and the confidence of a new measurement are both considered before updating the voxel.

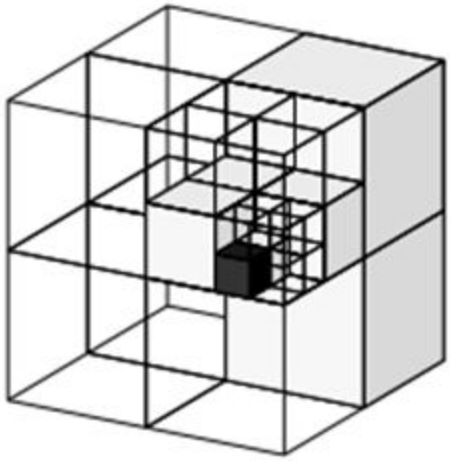

Figure 15 | An example of an octree storing free (white) and occupied (black) space

The last introduction made by this paper is to represent the probability of each voxel with a log-odds value. Here, a probability can be converted into a log-odds value, and back into a probability if necessary.

$$\mathrm{L}(n \mid z_{1:t}) = \mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t)$$

where

$$\mathrm{L}(n) = \log\left[\frac{P(n)}{1 - P(n)}\right]$$

Finally, these log-odds values are then clamped using upper and lower bounds. Intuitively, this will limit the number of map updates needed to change the state of a voxel. Once a voxel reaches the lower or upper bound, we can say multiple rovers have confirmed this voxel's state with high confidence and we consider this cell to be stable. In a static environment such as the moon, all voxels should

eventually reach a stable state after enough measurements have been received from individual rovers. When generating maps, if all children of an inner tree node are stable nodes with the same occupancy state, then we say the children can be pruned.

This leads us to the final benefit of the approach proposed in this OctoMaps [1] paper. By using the clamping method, which results in large groups of stable nodes, we can effectively compress the map and reduce redundant information. This is done with the pruning technique described above, which essentially groups stable voxels into larger cells. This does an excellent job of reducing the memory required to store explored and stable maps. This seems to suit the task of autonomous rover mining perfectly. In this scenario, rovers will likely be returning to the same dig site for long periods of time, and of course, the moon environment will be nearly perfectly stable. This should result in most of our map being either: entirely stable and able to be compressed (areas where rovers have been mining), or largely uninitialized (areas which rovers haven't explored yet). Both scenarios lead to efficient memory usage and should allow for our map to be easily stored in the central system's memory.

The OctoMap method described above would allow us to efficiently represent the moon environment in a probabilistic manner, accounting for the error in rover observations. Another key benefit of this approach is that OctoMaps are natively supported by ROS, the robot operating system used to control the EZ-RASSOR. This is a strong benefit for our team, signifying that we can use an open sourced implementation of this approach, rather than having to implement from scratch.

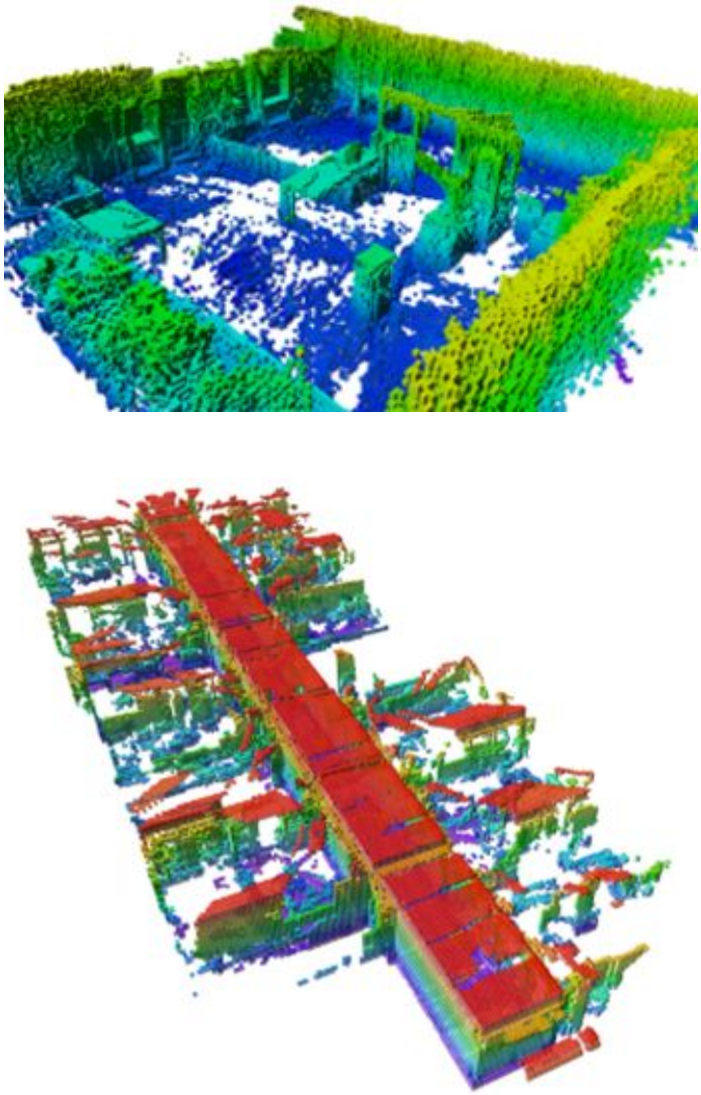

Figure 16 | Maps generated using the OctoMap approach

Path Finding

Multi-agent path finding (MAPF) is well-studied in both AI and robotics. At a glance, the problem may sound trivial. Given that we know the rovers' start position, in our case the charging station or moon base, and a goal location, a possible dig site for our EZ-RASSORs, we simply need to generate non-colliding paths for each individual rover. However, there will be multiple things to consider when deciding the best approach to such a problem.

An important consideration is whether to approach path planning in a traditional *algorithmic* way, or to have the system essentially *learn* by trial and error. The

latter describes what is known as Reinforcement Learning, which has been successfully utilized to create systems capable of outperforming humans in several complex problems. The advantages and disadvantages of these approaches will be described in the following sections.

Algorithmic Path Finding

A number of path-finding algorithms are discussed below, as well as each of their shortcomings. An explanation of how these algorithms can be expanded to a multi-agent setting will then be provided, as well as recent advancements made in this domain.

Dijkstra's Algorithm

Perhaps the greatest benefit of algorithmic path planning is that many of these algorithms guarantee a shortest path from the starting point to end point. A popular algorithm which guarantees this is Dijkstra's algorithm. Dijkstra's works by repeatedly visiting the closest not yet visited node in a graph until the goal is reached. Each time we reach a new node, we calculate the distance from the starting point for each of the neighboring nodes by summing the distances of the edges that led to the current node. If the distance to a neighboring node is less than its currently known distance from the starting point, we'd update that known distance. This leads to finding the shortest path from the starting point to each node in a graph. The below diagram visualizes Dijkstra's result on a simple case. The teal areas represent those which the algorithm scanned, while the pink square is the starting point and purple is the goal.

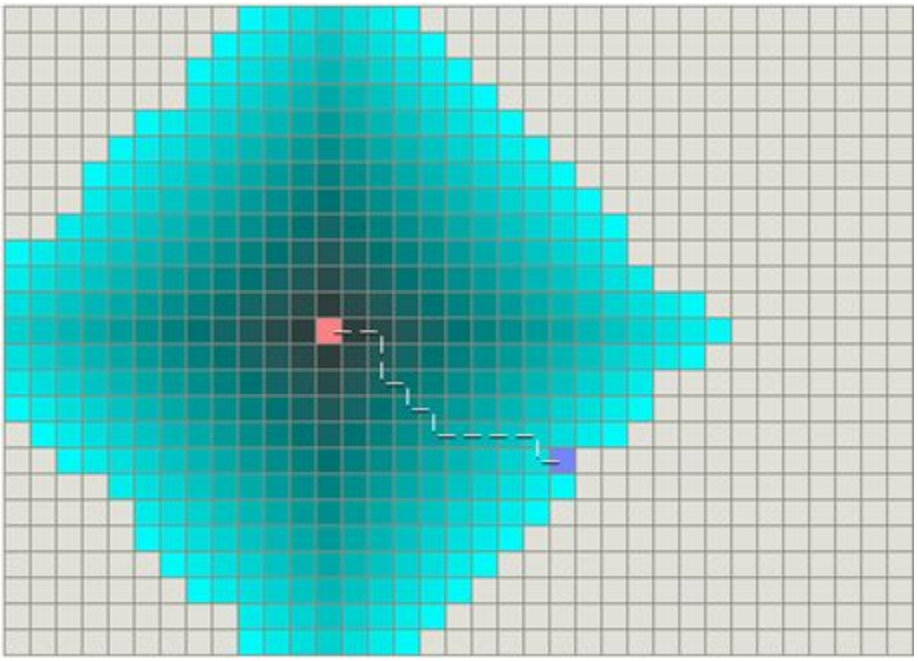

Figure 17 | Djikstra's result on a simple grid

Due to Dijkstra's nature, it requires visiting many nodes before finding the shortest path. Hence, the algorithm runs relatively slowly. In practice, this would be a major shortcoming due to the expected size of the moon environment.

Greedy Best-First Search

A significantly faster approach is known as Greedy Best-First-Search. This algorithm works similarly, although it utilizes a heuristic which estimates a nodes distance to the goal. As it explores the environment, rather than visiting the node closest to the starting point, Greedy BFS decides to visit the node closest to the goal. As you can see below, Greedy Best-First-Search finds the shortest path significantly faster than Dijkstra's.

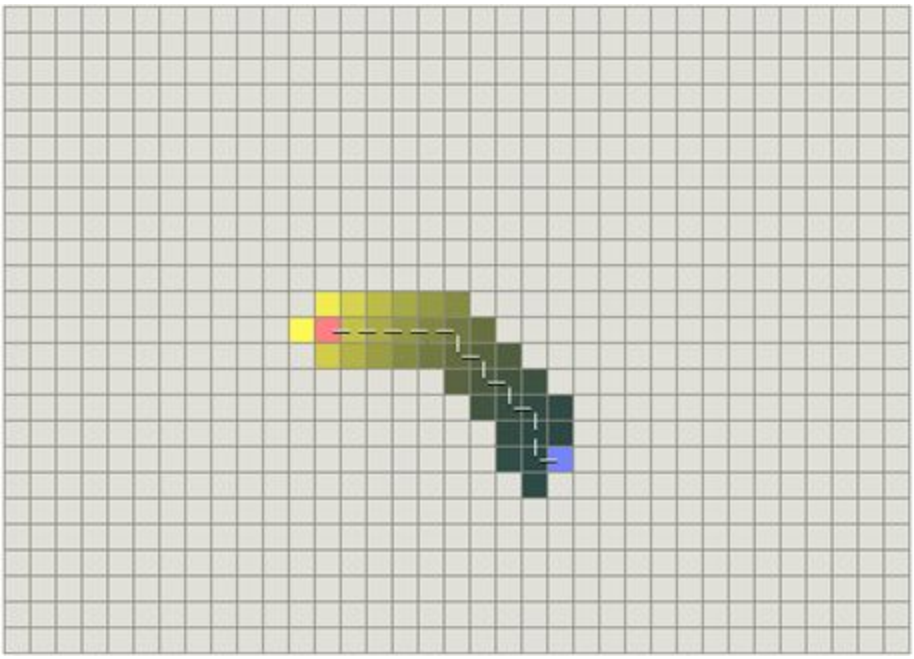

Figure 18 | Greedy Best-First-Search's result on a simple grid

However, running these algorithms on a slightly more complex graph demonstrates a significant disadvantage of Greedy Best-First-Search. Below is a visualization of Dijkstra's run, and then Greedy BFS will follow.

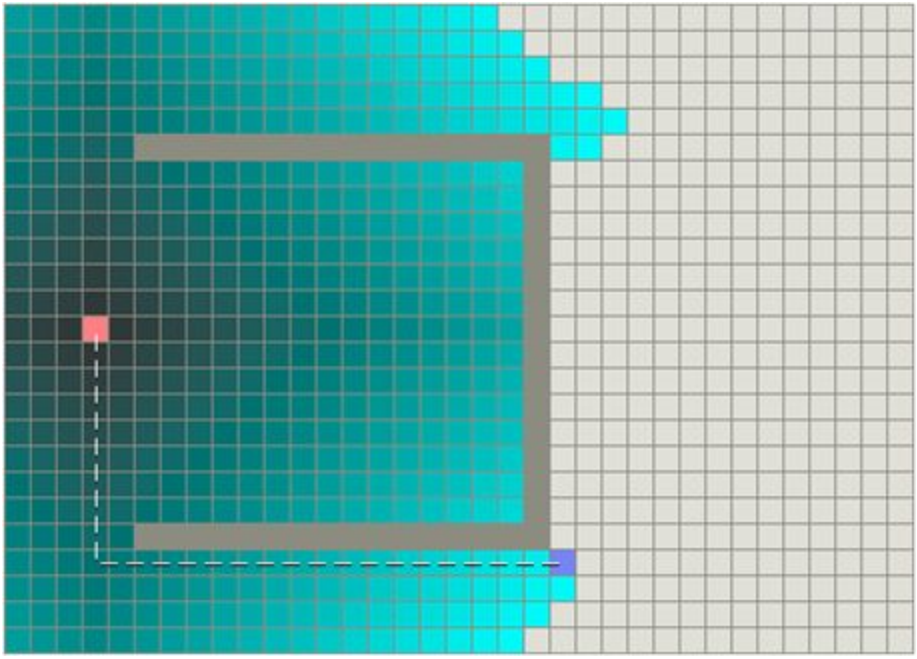

Figure 19 | Dijkstra's result on a grid with an obstacle

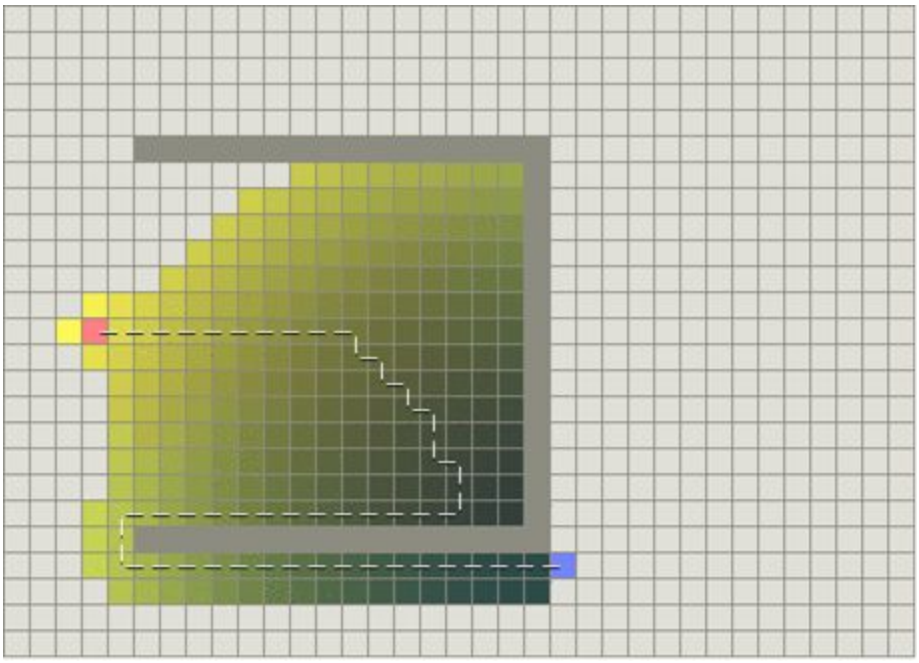

Figure 20 | Greedy Best-First-Search's result on a grid with an obstacle

For Dijkstra's run, the results are as expected. Although it explores a large portion of the environment, we do end up with the shortest path possible. On the other hand, Greedy Best-First-Search runs much faster, although the path is sub-optimal. This is due to its nature of moving in the direction of the goal, no matter the cost of the path so far. This introduces a key shortcoming, which is that Greedy Best-First-Search does not guarantee finding the shortest path. It seems that the perfect approach lies somewhere in the middle of the approaches described thus far.

The A* Algorithm

The A* algorithm is arguably the most common path-finding algorithm, due to its ability to solve the issues described above, and is the basis for a majority of multi-agent path finding algorithms. Like Dijkstra's, A* is guaranteed to find the shortest path, and like Greedy Best-First-Search, it leverages a heuristic to intelligently guide itself towards the goal. Running A* on the above problem demonstrates why it is so popular.

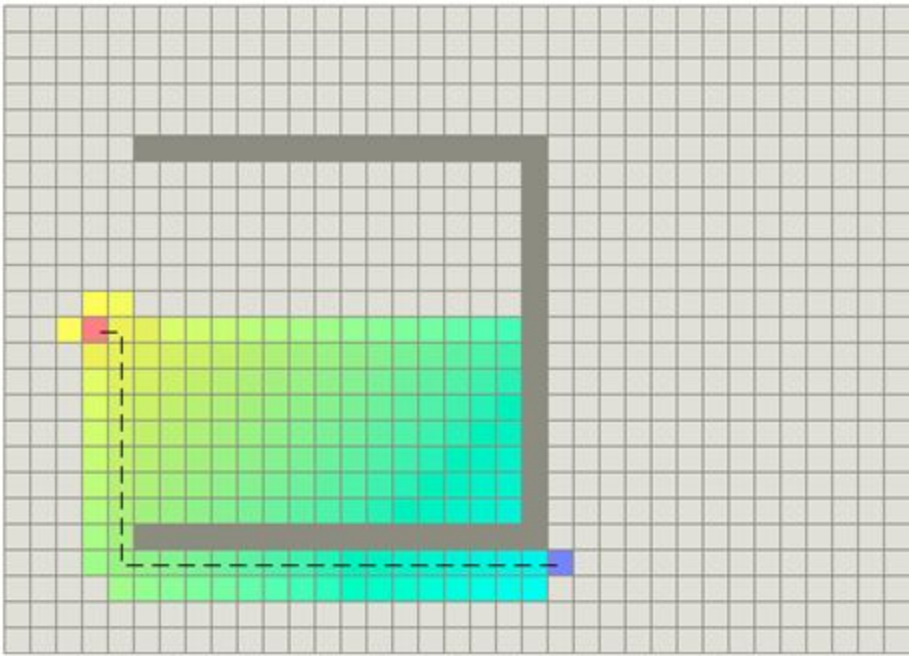

Figure 21 | The A* algorithm ran on the grid with an obstacle

A* is able to find the same route as Dijkstra's, while having to explore even less space than Greedy Best-First-Search. To understand how A* works, we'll start off with the formula used to calculate and update the cost of nodes in a graph.

$$f(x) = g(x) + h(x)$$

Here, g(x) represents the shortest distance between the current node and the start node. This is the only metric tracked by Dijkstra's. However, like Greedy BFS, A* also utilizes a heuristic. This heuristic, h(x), is the estimated distance from the current node to the goal. In practice, this heuristic is usually precomputed and simply ignores obstacles. Example heuristics could simply be the Euclidean or Manhattan distance from a node to the goal.

$$Euclidean = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
$$Manhattan = |x_1 - x_2| + |y_1 - y_2|$$

Figure 22 | Example heuristics

F(n) is then the total cost associated with visiting a given node, x. The A* algorithm works by traversing the nodes in a graph, repeatedly choosing to visit the neighboring node with the lowest f value. After moving to a new node, we calculate a new f value for each of its neighboring nodes and update the neighbor's f value if we've found a more optimal path. By leveraging the heuristic, A* is nudged in the correct general direction to the goal while minimizing the cost of the path to get there. The exact steps of the A* algorithm are described below.

1. Instantiate the OPEN and CLOSED lists. These are used to keep track of nodes which have been fully examined or not. Note that each node also keeps a pointer to its parent node, which allows us to track how that optimal path was found.
    a. OPEN begins with only the starting node
    b. CLOSED begins as empty
2. Repeat the following until we reach the goal or OPEN is empty:
    a. Move to the node in the OPEN list with the lowest f value
    b. STOP if the current node is the goal
    c. Add the current node to the CLOSED list
    d. For each node, x, adjacent to the current node:
        i. Calculate a new distance from the starting point to the neighbor, based on the g value of the current node and the distance between the current and neighbor
            1. New_g = g(current) + distance(current, x)
        ii. If neighbor in OPEN and New_g less than g(x):
            1. Remove neighbor from OPEN
        iii. If neighbor in CLOSED and New_g less than g(x):
            1. Remove neighbor from CLOSED
        iv. If neighbor not in both CLOSED and OPEN:
            1. Overwrite g(neighbor) to New_g
            2. Overwrite f(neighbor) to g(neighbor) + h(neighbor)
            3. Add neighbor to OPEN
            4. Set neighbor's parent to the current node
3. After completion, we use the parent pointers to backtrack from the goal node to the starting node, resulting in the final shortest path.

Multi-Agent Path Finding (MAPF)

Thus far, we've discussed path finding for a single agent, which can be efficiently solved using the A* algorithm. However, when we have several agents moving simultaneously in a shared space, the problem becomes significantly harder. The system must now navigate agents around obstacles, while ensuring individual agents do not collide. In the case of a single agent, the state space of the problem is bounded by the size of the map. However, in the case of multiple agents, the number of possible states grows exponentially as the number of agents increases. This introduces significant complexity and computational bottlenecks, which research in this domain has yet to efficiently solve. The MAPF problem is notoriously NP-Hard to solve optimally. Yet, researchers within this field have developed algorithms which can compute optimal collision-free paths for any arbitrary number of agents. At a high level, there are two common approaches for solving the multi agent scenario.

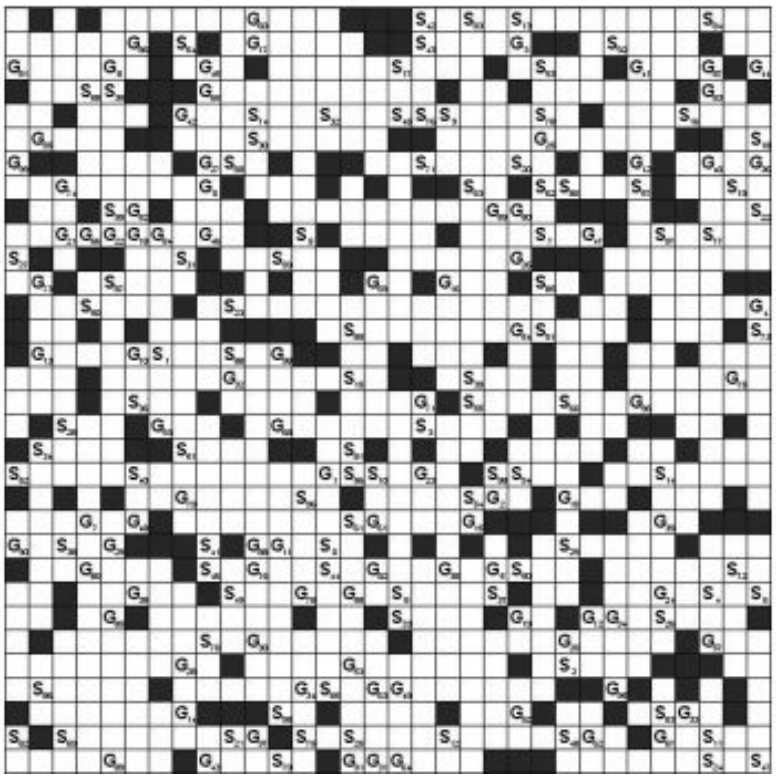

Figure 23 | An example environment to solve. Agents must navigate from S_i to G_i

Centralized A* Search

The first of which is known as a centralized A* search, which leverages a global decision maker to plan paths for all units simultaneously. In a centralized search, each agent moves synchronously to an adjacent unoccupied node in a single time step. This results in a joint plan containing all agent's actions, which includes wait steps along with movements.

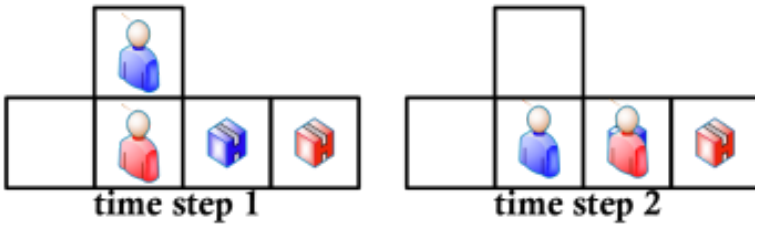

Figure 24 | This figure shows how a wait action might be used by a pathing algorithm to navigate a crowded or narrow environment

Existing research has shown that with this approach, it is NP-Complete to decide if a solution of at most *n* moves exists. This approach is theoretically optimal, however, it yields an exponentially large state space of $O(m^n)$, for *n* units on a graph with *m* nodes. Despite its completeness and optimality guarantees, centralized A* search has a prohibitive complexity and does not scale well to a large number of agents. This results in the method being quite impractical for a majority of multi-agent path finding problems, being intractable even for relatively small maps with several agents.

Decentralized Path Finding

On the other hand, scalability to larger problems can be achieved with a decentralized approach, which decomposes the global search into a series of smaller searches. A standard decentralized search begins by running the single agent A* algorithm for each individual agent, ignoring all other agents. Following this, the generated paths are inspected for collisions with one another. If a collision is found, a constraint (essentially a temporary obstacle) is added to the grid and the colliding paths are re-calculated. This process is repeated until all paths are found to be collision-free. On average, this approach significantly reduces the amount of computation needed and yields results much faster than its centralized counterpart. However, methods such as this are incomplete and are unable to distinguish between problem instances which can or cannot be

solved. This can often lead to long run times of this algorithm if no time or computation constraints are applied. Moreover, decentralized searches are free to return sub-optimal paths, which may or may not be a deal-breaker depending on the application.

In summary, centralized on-line searches guarantee completeness and optimal paths, but result in a prohibitive complexity and do not scale to large numbers of agents. In contrast, decentralized methods give up optimality in order to reduce computation significantly. These algorithms are often much faster and are able to scale up to larger instances, well beyond the capabilities of a centralized search. Hence, decentralized searches are often applied to problems with sizeable maps, large numbers of agents, or situations where optimal paths are not required. This leads the team to believe a decentralized approach may better suit this project, where the moon environment will be quite large and the number of EZ-RASSOR rovers will not necessarily be bounded. However, as mentioned earlier, these methods have no known formal characterizations of their running time, memory requirements, and the quality of their solutions in the worst case. Additionally, lack the ability to answer in a reasonable bounded time whether a given problem can be successfully solved.

In practice, both centralized and decentralized approaches to multi-agent pathfinding have quite severe disadvantages. With our current state of knowledge, there seems to be an inherent and unavoidable trade-off between scalability, optimality, and completeness. Nonetheless, recent research within this domain have begun to bridge the gap between the two with what's known as a bounded suboptimal approach. These methods are able to address both completeness and tractability simultaneously. One of the most powerful techniques in this category is the MAPP algorithm. In order to understand this approach, however, we'll need to cover a few decentralized approaches which have been developed over the years.

Local Repair A*

Over the past few decades, Local Repair A* has arguably been the baseline decentralized approach for multi-agent path finding. Most notably, this method has been extensively utilized in the video game industry with great success. At a high level, Local Repair A* is quite similar to the standard decentralized search described above. The algorithm begins by searching for paths around obstacles

to a goal for each individual agent, ignoring all other agent's actions. A key differentiator between this algorithm and the others we'll discuss throughout this section is that Local Repair A* runs online. What this means is the agents will actually begin following these individual routes, and are aware of their current neighbors. When a collision is imminent, meaning an agent is about to move into a currently occupied position, the algorithm replans the remainder of this agent's pathing with an additional A* search. During this replanning, the algorithm will treat this collision location as a new obstacle. This process would then continue until all agents reached their goals. Unfortunately, this constant replanning of paths will prove to be quite costly. In the best case, Local Repair A* will reduce the runtime of multi-agent pathfinding to *O(m*n)*, for *n* units on a graph with *m* nodes.

Compared to the runtime of a centralized search, $O(m^n)$, this is a significant improvement. However, Local Repair A* has been shown to frequently generate cycles and is unable to prevent bottlenecks. Various extensions of this algorithm have been developed with attempt to escape such cycles. One possibility is to add what's known as an *agitation level* to each agent, which increases each time an agent is forced to replan. Random noise is then added to the distance heuristic from A* in proportion to an agent's current agitation level. This results in agent's behaving increasing randomly as they run into more collisions, which may allow an agent to escape cycles and bottlenecks. Although these additions aid in minimizing cycles, there still remains a number of situations which can result in a worst case runtime. For example, if a bottleneck of agents occurs in a crowded location within the map, these cycles may take arbitrarily long to be resolved. In such a situation, Local Repair A* would continuously reroute to in an attempt to escape. Because each replaning is made independently of the other, this may lead to cycles in which the same location is repeatedly visited by the same agent.

Due to these previously mentioned problems, Local Repair A* can often results in quite large runtimes, and in the worst case, may never terminate. The approaches described below attempt to overcome these shortcomings.

Cooperative A*

For several years, Local Repair A* remained the standard approach for solving the problem of multi-agent pathfinding. However, recent advances have begun to outperform this approach. The basis of these high performing algorithms is what's known as Cooperative A*. Like a typical decentralized search, Cooperative A* also beings with a series of individual path searches.Unlike previous approaches, however, these individual searches occur sequentially, rather than simultaneously. This allows the algorithm to take into account the planned routes of all agents. To enable this, a *wait* move is included in each agent's action set, which enables it to remain stationary. Once an agent's path is found, the states along the route are stored in a data structure known as a *reservation table.* This reservation table essentially represents the agent's shared knowledge of one another's paths. Each entry within the reservation table is impassable and avoided by following paths, allowing the algorithm to calculate non-colliding paths without any additional replanning.

The specific implementation of this reservation table is separated from the map which this algorithm is run on. So long as the data structure is able store reservations in two spatial and one time dimension, it will function for the purpose of this algorithm. A practice and popular implementation of a reservation table treats it as a three-dimensional grid, again, two spatial dimensions and one for time. Cells which correspond to an agent's planned route will be marked as impassable for the specific duration which the path remains in that location. Due to the additional time dimension, this 3-D representation would end up quite large and sparsely populated, especially for a reasonably sized spatial grid. Hence, this 3-D grid representing a reservation table is often implemented as a hash table, with hashing being done on a (*x, y, t)* key.

Although Cooperative A* has significantly improved runtimes and results in less cycles when compared to Local Repair A*, this approach is not without weaknesses. Due to its use of a time dimension, the success of this method in practice is heavily reliant on the ability of physical agent to replicate their paths exactly. This means that in the case of our EZ-RASSOR rovers, if a single agent differs from its path in the time dimension, perhaps due to getting stuck or slowed, unexpected collisions are now a possibility and out of the control of the

algorithm. Moreover, Cooperative A* is simply unable to solve certain classes of difficult problems. An example of such a situation is visualized below.

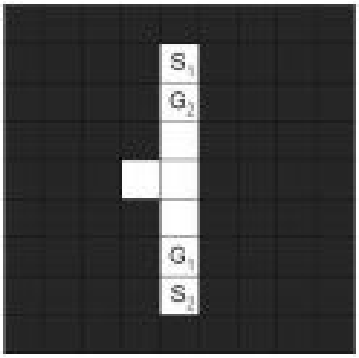

Figure 25 | Cooperative A* is unable to solve such a deceptively difficult map. Here, one agent is tasked to navigate from S_1 to G_1, while the other must get from S_2 to G_2

In the environment above, the solution for one of the agents directly prevents any possible solution for the other agent. This is due to Cooperative A*'s inherent sensitivity to the ordering of agents. More concretely, this algorithm would first compute the path for the first agent and mark this path as occupied in the reservation table. When the algorithm then attempts to find a path for the second agent, there are no available paths. On top of this, this nature to solve paths sequentially results in increasingly less optimal paths for lower priority agents. This challenge has led to some research in how to prioritize certain agents based on their end goal. However, this method is less applicable to our problem due to the fact that each EZ-RASSOR will be navigating to and mining from the same trench.

Hierarchical Cooperative A*

Extensions of Cooperative A* allowed for improvements in both computation needed and solvability of challenging environments. The most successful of these is Hierarchical Cooperative A*, which aims to improve the heuristic behind each individual A* search. This extension uses a simple hierarchy containing a single state space or grid map abstraction, which ignores both the time dimension and the reservation table. Abstractions are created using the STAR abstraction technique described in [11]. The technique involves grouping states, or nodes on a graph, or points on a grid, within a specified distance of one another. One abstraction level is made of a number of states, all assigned to some abstract state grouping. This process is then repeated to create multiple

abstraction levels until a level is created containing a single abstract state. This forms an abstraction hierarchy whose top level is the trivial search space, and the bottom level is the original search space. In contrast to the previous Cooperative A* algorithm, the Hierarchical approach calculates the distance heuristic in A* on this abstracted state space, ignoring any potential interactions with other agents.

The process of A* search on this hierarchical abstract search space is relatively straightforward. Like standard A*, at each step of the search a state is removed from the OPEN list and each of its successors are added to the OPEN list if it has not yet been previously visited. Before adding a state to the OPEN list, we'll calculate the heuristic value between that state, *S,* and the goal, *G*. This is computed by searching at the next higher level of abstraction using the abstract state corresponding to *S*, $\phi(S)$, as the start state and the abstract state corresponding to *G*, $\phi(G)$, as the goal. Once the abstract path between $\phi(S)$ and $\phi(G)$ is found, the distance of this path is used as the heuristic between *S* and *G*. Outside of recursively searching each next abstract state to compute the heuristic, the process of individual searches remains the same as a standard A* search.

While this process may seem computationally expensive, it should be noted that when an abstract path is found between $\phi(S)$ and $\phi(G)$, the abstraction distance to goal information is now known for all abstract states on this path. Additionally, each of these abstract states will correspond to several states on the lower level of the hierarchical space. Thus, a single abstract search will produce heuristic values for many states or nodes. In order to reduce computation, all of this information will be cached in order to prevent unnecessary computation. However, despite this caching technique, it remains true that to solve a single base level problem, the algorithm will need to search each level of abstraction at least once.

Windowed Hierarchical Cooperative A*

The algorithms we've discussed thus far have made significant improvements from a baseline decentralized search. Nonetheless, there remains a number of unsolved challenges to tackle.

A major issue with the decentralized algorithms we've discussed is their sensitivity to agent ordering. Due to the nature of these methods, agents whose paths are computed first will always receive the short, more efficient path. One possible solution to this is to dynamically vary the agents' orders, so that every agent will have the highest priority at some point in time. Another issue is the tendency of these algorithms to terminate once the agents reach their destination. In specific environments, possibly including the theoretical moon environment, a halted agent at its terminating state may block off parts of the map from other agents. In an ideal scenario, agent's would have the ability to continue organizing themselves in order to reduce the probability of such a scenario. Lastly, these previously discussed approaches calculate a complete path from the starting position to the goal for each agent, and only once these paths are found do agent's begin to execute the plan. It many scenarios, it is safer and more efficient to interweave path finding and execution. This would avoid the need to plan for long-term scenarios and collisions which may or may not occur.

An effective and simple solution to all of these issues is to window the search. In order words, each individual search is limited to a given depth. In this approach, the algorithm would search for partial routes for each agent, then they begin following these routes. During the execution of these paths, this window would be moved forward and a new partial route would be computed. This sliding of the window is usually done at regular intervals, typically when an agent is half-way through its partial route. To ensure the agents move toward the goal, only the search at the base level is windowed, while the abstract searches are executed to full depth.

Additional extensions to Windowed Hierarchical Cooperative A* allow the search to continue once the agent has reached its destination. In this scenario, the agent's goal is no longer to reach the goal, but to complete the window via a terminal edge. Theoretically, any sequence of *w* moves would reach the goal, however, this algorithm is capable of finding the lowest cost sequence. This optimal sequence is the partial route which takes the agent closest to its destination, and enables the agent to stay there for as long as possible. Once other agent's arrive, this approach will allow each agent to stay in the vicinity of the goal, without colliding with other agents. This functionality could prove to be especially useful in the context of our project, where multiple rovers may be at

the goal, e.g. the dig site, simultaneously. It may even be perhaps possible to modify this stage in order to create paths around the dig site which would result in efficient and clean trenches being dug.

In short, Local Repair A* may be a decent algorithm for more simple maps, those with few bottlenecks and agents. However, in larger environments with more agents, Local Repair A* is significantly outperformed by Cooperative A* approaches. Moreover, Hierarchical A* improves the heuristic of A*, resulting in less cycles and bottlenecks. Finally, a windowed search such as Windowed Hierarchical Cooperative A* is much more robust and efficient, finding short and more successful paths. The below charts were retrieved from [6], and do an excellent job of visualizing the benefits of each algorithm. Note that the number in parenthesis next to WHCA represents the size of the window used.

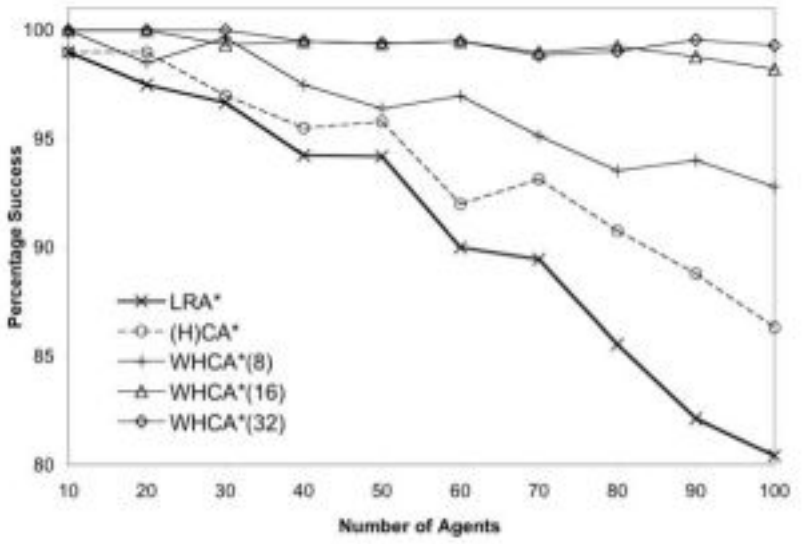

Figure 26 | Percentage of agents that successfully reach the goal

Here, Local Repair A* begins to fail due to bottlenecks in the environment. However, the three Cooperative A* algorithms enable agents to step aside and allow others to pass, efficiently solving these bottlenecks.

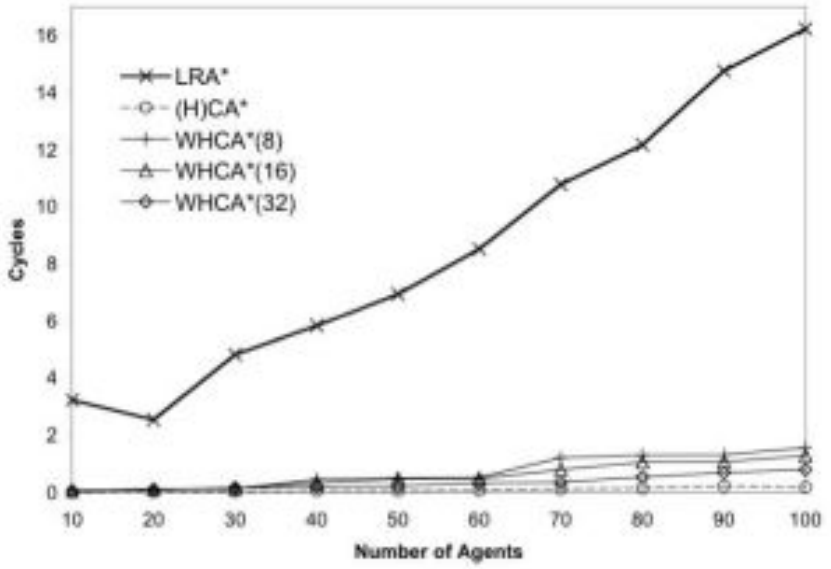

Figure 27 | Average number of cycles in each agent's route

In terms of cycles produced, the Cooperative A* and Hierarchical A* algorithms produce very few, due to the fact that they precompute the entire path for each agent. The nature of Windowed A* rerouting at regular intervals results in slightly more cycles, however the paths maintain high consistency due to the hierarchical heuristic. Local Repair A* of course results in a large number of cycles, due to agents repeatedly attempting to escape bottlenecks.

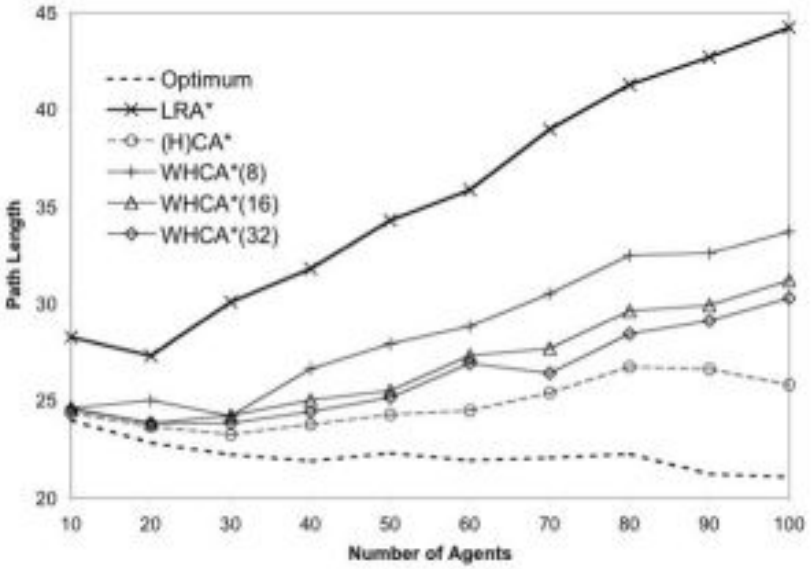

Figure 28 | Average path length for each agent

Figure 28 does an excellent job of visualizing the effectiveness of Hierarchical Cooperative A*, which results in shorter paths than its counterparts. This is due to its ability to compute paths in one fell swoop, with a sophisticated heuristic. It also displays the effect of window size on the Windowed Hierarchical Cooperative A*'s performance. With a large window, the algorithm tends to behave more like Hierarchical Cooperative A*, but with a smaller window it behaves similarly to Local Repair A*. This is due to the constant rerouting of paths within the window.

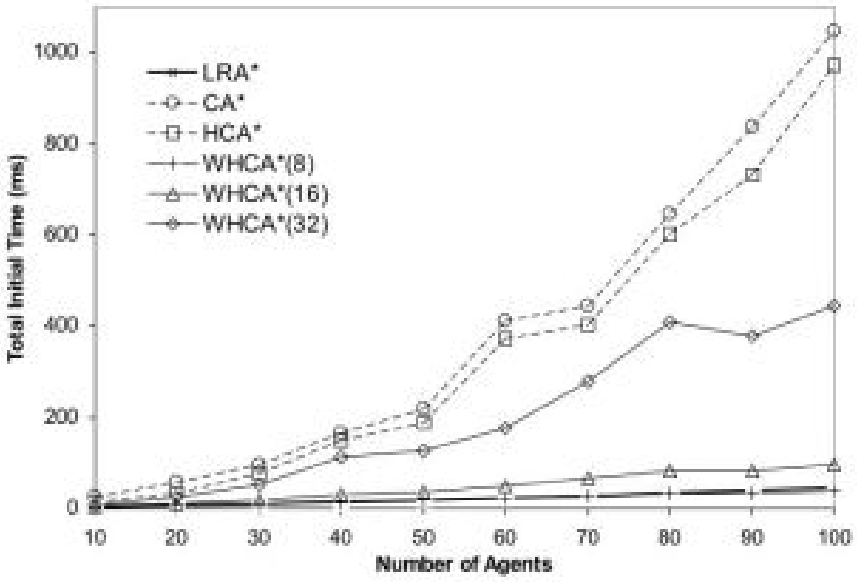

Figure 29 | Total calculation time for all agents

Above, Local Repair A* is shown to be quite fast in flat out runtime when compared to cooperative algorithms. This is due to cooperative algorithms having to perform full depth searches in multiple abstraction level. However, here is where Windowed Hierarchical Cooperative A* truly shines, having similar runtimes to Local Repair A* while performing much better in the metrics from above.

In all cases, however, the strongest bottleneck with regard to performance is clearly the number of agents within the environment.

The FAR Algorithm

Similar to Hierarchical A*, FAR starts with a preprocessing step where a grid map is abstracted. However, the graph is now abstracted into what's known as a flow-annotated search graph. This is a structure enhanced with flow restrictions to avoid head-on collisions and to reduce the branching factor in search. Also similar to previous decentralized algorithms, FAR begins with a complete A* search is run for each unit independently, computing a path that ignores the other agents on the map. Plan execution starts as soon as a path is computed for each agent. Again, paths are then stored in a reservation table and a wait move ins included in the agents action set when they must wait for another agent to pass. This helps to avoid deadlocks, which is when two or more agents are forced to wait for each other in a cycle. When deadlocks cannot be avoided, a deadlock breaking procedure attempts to repair plans locally, rather than a larger-scale replanning step. This local replanning is what differentiates FAR from previous algorithms, which replan the entire route. However, FAR is quite similar to Windowed Hierarchical Cooperative A* search in this regard, although it uses this specialized flow-annotated search graph.

The MAPP Algorithm

The MAPP algorithm is arguably the strongest performing and most advanced multi-agent pathfinding algorithm discussed in research today. It does an excellent job of combining the strengths of centralized and decentralized approaches, featuring tractability and partial completeness guarantees. Given a problem with *m* graph nodes and *n* agents, MAPP boasts a worst case runtime of $O(m^2n^2)$ and a worst case space complexity of $O(m^2n)$. It should be noted that this is slower than the best case for Hierarchical Cooperative A* and the FAR algorithm, however, these were built for fast runtimes and are able to solve significantly less complex environments than MAPP. As we'll discuss in the coming section, MAPP also guarantees absolutely no cycles or deadlocks, something that no other algorithm thus far is able to accomplish.

MAPP keeps its running costs low by removing the need for any replanning of paths. A path for each unit will be computed and will require no additional replanning during execution. This is an important note and will come to be quite valuable for this EZ-RASSOR project. The MAPP algorithm is centered around the idea of a *blank travel*, meaning a unit can only travel from its current location

to an adjacent node if a *blank* is located there. Intuitively, if the adjacent node is currently occupied by another agent, MAPP attempted to bring a blank along an alternate path. This alternate path scenario is displayed in the figure below, and is highlighted in bold. The alternate path below connects two adjacent nodes, without passing through the occupied node. When possible, the blank is brought to the next node by shifting all units along this alternate path. This process lies the basis for all path planning by the MAPP algorithm.

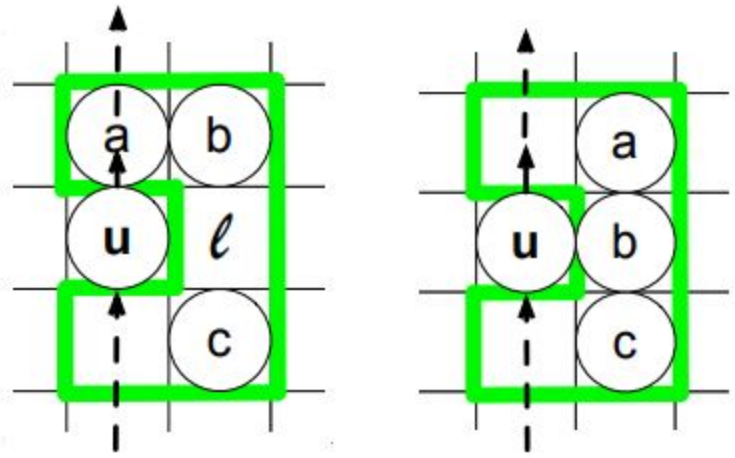

Figure 30 | The At the left, unit *u* is blocked by *a,* and a blank is found at location *l* along the alternate path marked in bold. At the right, by sliding *b* and then *a* along the alternate path, the blank is brought to the next node.

The below figure shows a complete example of how MAPP deals with collisions. There are two initial units, *a* and *b.* MAPP uses a total ordering of the active units in each progression step towards calculating a route. This allows each unit to share priority, averaging out to equally optimal paths for each unit. Here, *a currently* has a higher priority than *b.* The targets of a and b are drawn in stars. In frame *i*, as the two agents move towards the target, *a* becomes blocked by *b.* Then, in frame *ii,* a blank is brought in front of *a* by sliding *b* down. As a side affect, *b* will get pushed off its path. However, by frame *iii,, a* has reached its goal. The algorithm then takes a repositioning step, undoing *b's* moves until *b* is back on its path and has a blank in front of it. Finally, *b's* is able to advance and reach its target. Essentially, this is a more sophisticated A* with maneuvers built to deal with collisions. There are several additional extensions to this algorithm, however, this base understanding is sufficient to understand the benefits and inner workings of the MAPP algorithm.

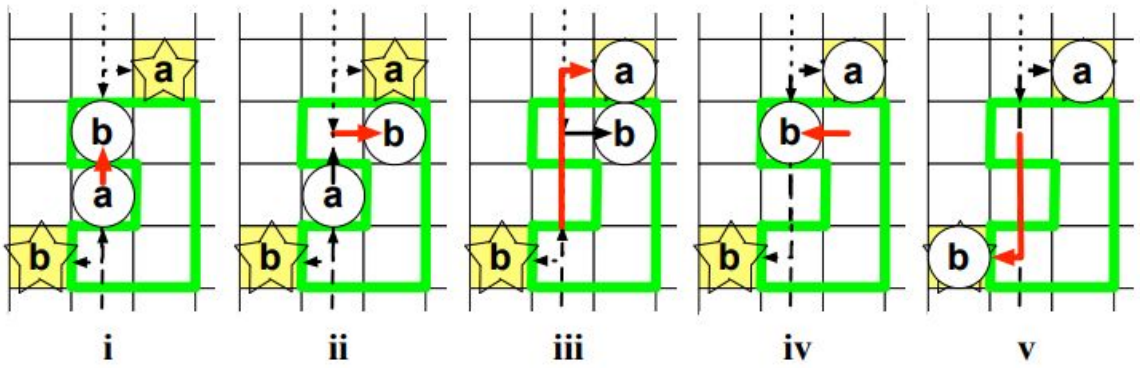

Figure 31 | Example of how MAPP works by sliding blank states

In conclusion, MAPP has significantly better success ratios, solving many more difficult environments, and scalability than previously discussed decentralized algorithms. An additional note is that it requires no replanning during the execution of paths. This will be discussed in detail in the design section of the document, however, it is for this reason that MAPP will likely be the optimal choice for solving the problem of pathfinding for our EZ-RASSOR rovers.

Swarm Intelligence

There are many facets to the concept of swarm intelligence, as it is a constantly-evolving field in computer science which takes many forms. The goal with swarm systems is usually to take ideas directly from nature. Bees, for instance, are a common example of swarm behavior. While each individual bee may not be particularly intelligent, nor can one direct the whole colony. They are, to put it frankly, rather stupid and can only complete very simple tasks. A single worker bee more or less gathers honey and helps to feed the queen bee. However, when you put hundreds of them together, something very unique happens; they form an intelligent system capable of so much more than the sum of its parts. They can create large and complex structures to support the hive, expand their swarm over vast distances, find resources as a large group, and immensely surpass what would be expected from a single unit. All of this using simple chemical signals which each member of the colony can recognize. How can we leverage this knowledge to help us colonize other planets?

First, we must define the problem at hand. Suppose we've just landed on the lunar surface; how do we gather resources quickly and effectively so we can sustain ourselves? How can we refuel our ride home using the resources all around us? This is where a swarm intelligence steps in. Just as a colony of bees can gather resources and create shelter, resources, and long-term storage space for themselves, we can use a swarm of small and relatively simple robots to gather resources for us. The Swarm Intelligence could be given a task, for instance, to collect 50 kilograms of oxygen and 50 kilograms of hydrogen to start making rocket fuel and air for a return trip from the moon. It then becomes the job of the artificial intelligence to leverage all of the information it has at hand to figure out where all of the required resources are, which rovers should it send to which sites, how much is each rover responsible for excavating? Are there any obstacles on the way that could potentially pose a threat to any one of the rovers not coming back?

All of these decisions will be handled by the central intelligence, and then instructions given out to each individual rover. They will then go about their task, helping others who get stuck or lost as necessary, to complete the task. What might take one large, complex, and expensive rover many days to complete, might only take a couple hours for tens of hundreds of smaller inexpensive rovers and a central intelligence. This means that progress could be made exponentially quicker as more units get delivered to the surface. The system could easily integrate new rovers into the swarm, as they're each very simple and perform simple tasks. This is cheaper in the long run, as it costs thousands of dollars per kilogram of mass to get to the moon and beyond, so making things smaller and more efficient is paramount to the success of In Situ Resource Utilization, which NASA is pushing for as the future of space exploration.

Figure 32 | A few Swarmie robots

SwampWorks has experience already with swarm intelligence systems. Swarmies, as named by the team at SwampWorks, are designed to have a rudimentary sensor system and a wifi connection to talk to the rest of the swarm. Modeled after ants, their purpose is to find resources (in the case of these robots, in the form of QR codes on the ground) in an efficient manner and as a swarm. We hope to discuss in detail what SwampWorks has learned in regards to swarm intelligence and best practices or techniques to achieve swarm behaviors, as they have much more direct experience. They have been very receptive to the idea of discussing high level details of past and current projects despite not being able to share code or models with us.

Additional Path Planning Details

The path planning we intend to implement will need to take in many variables for many rovers and create an ideal path for them all which will accomplish the task given. Such variables about each rover include:

- Current location
- Battery levels
- Battery efficiency
- Amount of wear on treads
- Accuracy of sensors
- Time for the batteries to recharge
- Overall number of uses

More variables regarding rover status may come into play as the hardware and software develops, but that covers most of the important data that needs to be taken into account by the Artificial Intelligence when designating rovers to go to certain destinations. Certain conditions about the terrain also need to be included in the model. These factors include:

- Slope at a given point on the surface and how that affects the amount of power each rover will use to traverse it
- Remaining duration of sunlight (the rovers will be recharged with solar power)
- Wheel slip based in a given area on estimations and data from rovers that have already traversed that area
- Unexpected obstacles or holes based on data from rovers (or rovers that didn't come back)
- Adding some variance into the path so rovers don't create a rut from going over the same exact path many times

The path planning we do should produce a series of waypoints which will guide the EZ-RASSOR rovers around any environmental obstructions, with the onboard navigation handling the navigation to those waypoints and micro-level obstacle avoidance.

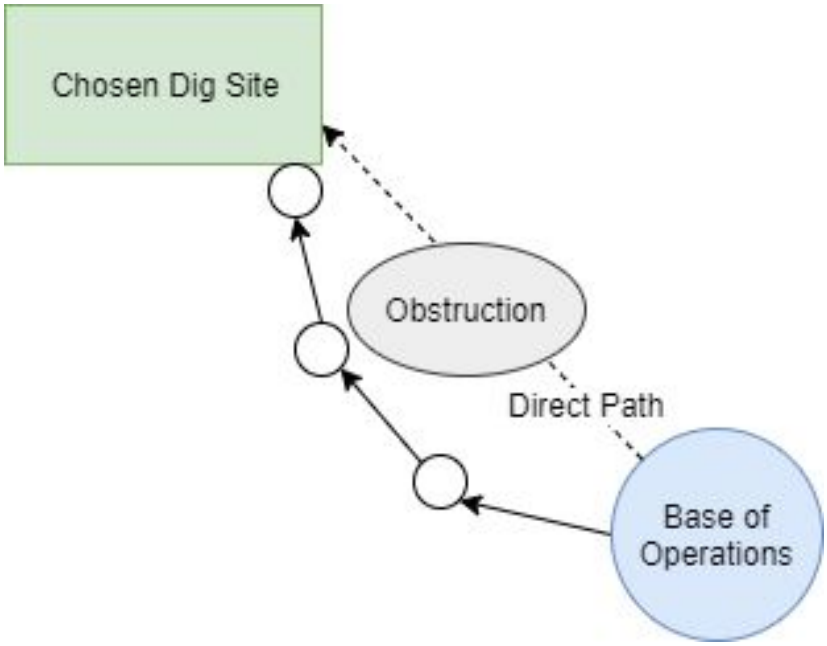

Figure 33 | Diagram demonstrating the waypoints necessary avoid a large obstruction in the path from the base of operations to the dig site

There are a lot of unknowns for the path planning model when we first begin in a given area, but the system should be able to learn and adapt as rovers begin exploring and reporting key data points about their travels. This may be done with reinforcement learning so the system will be able to give better and better directions and create more efficient paths as more data comes in.

During our initial visit with at SwampWorks lab one of the engineers mentioned it might be a wise decision to try and model the actual soil displacement from EZ-RASSOR digging so we could take the trenches they make into account in the path planning algorithms. This would be quite a challenge to do natively in Gazebo, which may require either a different tool to visualize the specific behavior of digging, or may just require a work-around to be created in Gazebo. One of the proposed methods of doing that would be to alter the world mesh on the fly, as the trenches are made. This may not be needed, however, as that may not necessarily hinder the ability for the Artificial Intelligence system to plan a navigable path for each rover.

Hardware-Software Interface

When the team spoke with the engineers at SwampWorks that currently are working on the RASSOR and MINI-RASSOR, they had mentioned the very real issue of interfacing with the actual hardware of the rovers. This may require us to either build drivers from the ground up, or to find a way to make existing drivers work for whatever the hardware ends up being. As this is a rapidly developing project, the specifics for the final hardware of the MINI-RASSOR is not set in stone, nor is it certain at the moment. We may opt to instead keep it in the simulation, and leave creation of drivers for the actual hardware as work for a future team once the final hardware has been decided upon. This may end up being the most ideal approach, as the main task we've been given is to create an Artificially Intelligent swarm system, and specific hardware implementation can be integrated at a later point in time. Our mentor at NASA has mentioned to the team that EZ-RASSOR will be a multi-year endeavor that this year alone has 5 separate teams from two schools.

Figure 34 | The most recent iteration of the MINI-RASSOR hardware

The reason for hardware-software integration is even an issue is due to the rover having an x86 chipset. Interfacing with the hardware isn't quite as easy when compared to a Raspberry Pi, for instance. The high level operating system on the x86 board needs to know how to talk to the hardware, including motors for the wheels and barrel drums, the actuators for the arms, the sensor arrays, and every other part on the rover that needs to receive commands from the computer. This requires a system driver so the high level commands from ROS can be turned into something the x86 system can actually use to send commands directly to all of the motors and actuators.

Lunar Surface Data

NASA, being a government agency, has lots of the data it gathers available to the public. One of the main sources of lunar data for NASA is their Lunar

Reconnaissance Orbiter (LRO). It was launched from NASA's Launch Complex 41 on June 18, 2009 and entered lunar orbit on just five days later. NASA made available in September of this year a displacement map created from data captured of the lunar surface in spring 2019 by the Lunar Orbiter Laser Altimeter instrument team. The maximum resolution available from this data set is 64 pixels per degree.

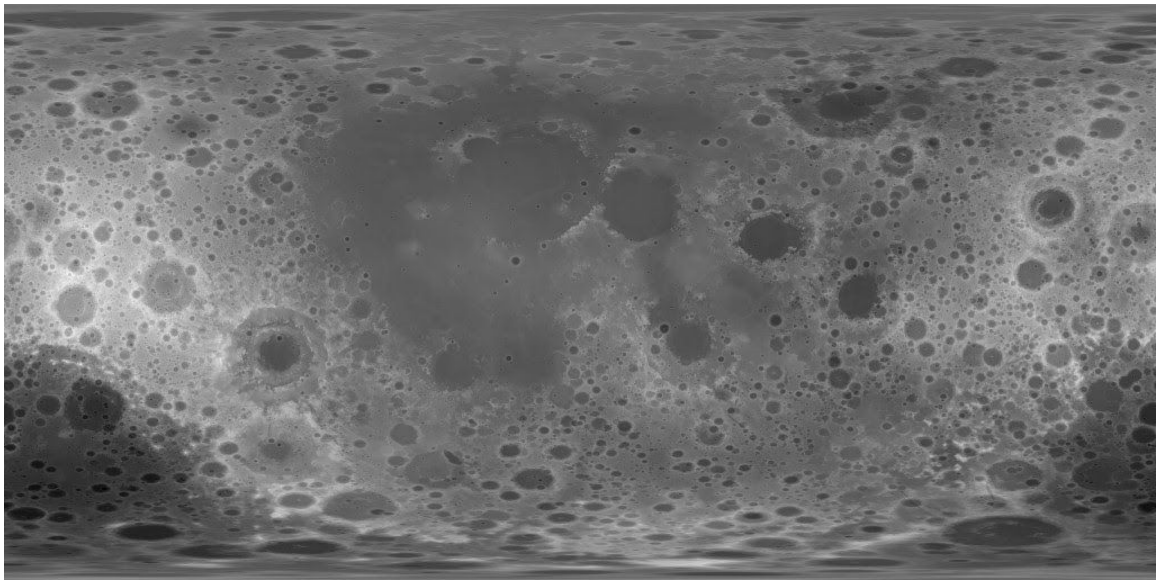

Figure 35 | The displacement map generated from LOLA data

The entirety of orbital data captured from the moon is stored in one central database [25]. From here, one can access all of the raw data collected from various missions and instrument sets, including the LRO. The highest resolution of displacement maps (the ones we're focused on) is the Selene/LRO Digital Elevation Model (SLDEM). This data set is generated from Depth Estimation Maps from both the SELENE orbiter (operated by Japanese Aerospace eXploration Agency) and point surface height data from the LRO. This data proves very useful in the context of our project. We can use this data to both improve our simulation as well as initialize a cost map for our path planning algorithm.

Firstly, what even is a displacement map? A displacement map, also known as a height map, is a grayscale image which is used to displace the surface of a model according to the value of each pixel. This is done by the process of displacement mapping, where the displacement map is wrapped around a model

or mesh. Then, the surface geometry of the model is moved either up or down according to the displacement map. The amount which the point is displaced is calculated by checking the pixel value at each point. If the pixel's value is 0% (white), then the point is moved inwards like a crater. If the value is 100% (black), then the point is raised outward like a mountain. Finally, if the value is 50% (grey), then it is not displaced at all. This method of deforming a mesh is computationally rather expensive, as it creates a model with a very large number of polygons and complex geometries, but will create a high-fidelity representation of a 3 dimensional surface.

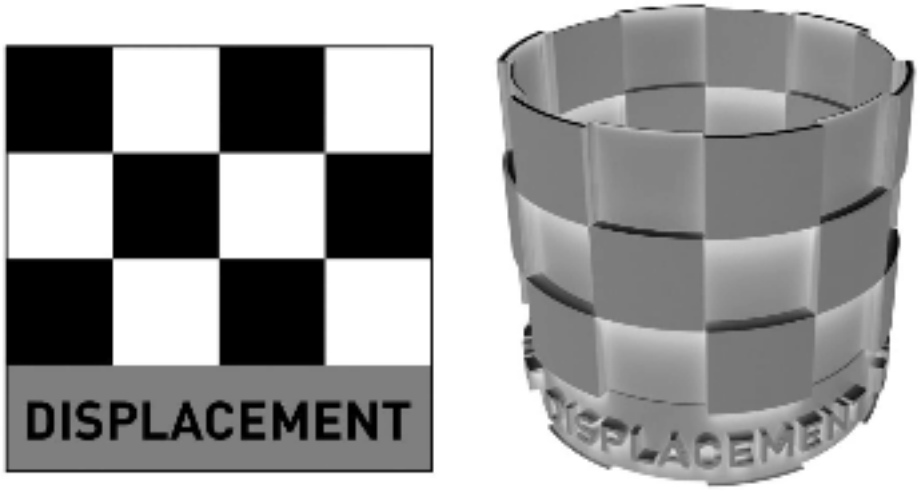

Figure 36 | An example of how a displacement map can affect the surface geometry of a model

The first use of this in the context of our project is going to be for the Gazebo simulation. We can take the displacement map from NASA and import it into Gazebo to make a more varied lunar surface. The current map in place is only of a 514m by 514m square of lunar surface around the Apollo 15 landing site. This limits both the amount of area that can be covered by the rover, as well as the amount of area we can use for our swarm AI system. With a displacement of the entire lunar surface, we are no longer limited in area to use for testing in Gazebo.

Secondly, we can use the NASA displacement maps to initialize an environment map for the swarm system. This will be a crucial step in creating the Path

Planning System for the swarm. The displacement map will be turned into a connected graph or perhaps a grid map where a higher incline would make for a higher cost, similarly to figure 37 below.

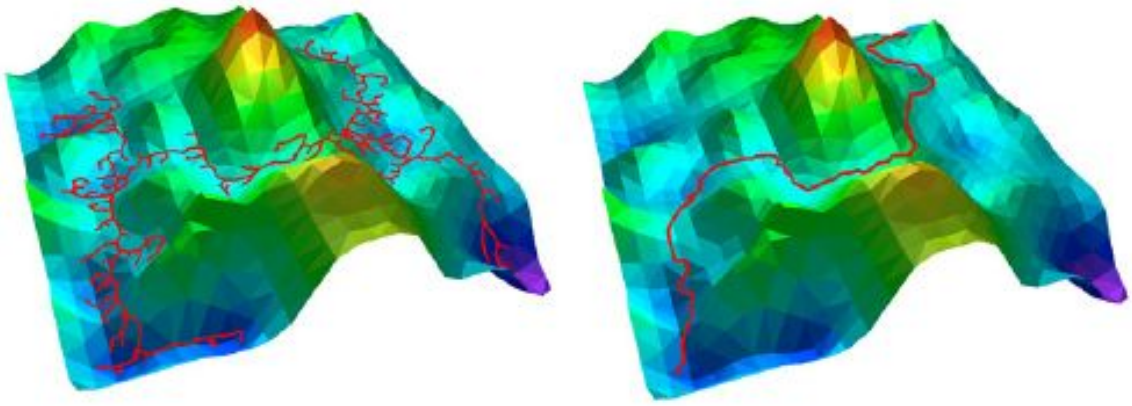

Figure 37 | 3D cost map showing how the ideal route is chosen from all of the potential ones [32]

Once we have generated this most basic cost map, we could determine high-level hazards and places to avoid. If the slope of a given node is above or below certain thresholds, it could be marked as a hazard and be disconnected from the graph. From there, we could run our path-planning algorithms to generate our waypoints for the EZ-RASSOR rovers to follow.

Regolith

Regolith is essentially a catch-all term for space-dirt, but it has a general makeup which is extremely useful for space travel/colonization. Regolith can be collected and many elements be extracted from it. This includes Oxygen and Hydrogen, which can be used for multitudes of things. Oxygen can be used to make breathable air, Hydrogen can be combined with the Oxygen to create water for astronauts to drink and to use to grow plants. Hydrogen is also used for rocket fuel. It costs approximately $1000 per kilogram to ship things into space. Therefore, cutting down on weight is extremely important to success in space travel.

In terms of qualities, regolith is lightly packed and in a shard-like form. This is due to the conditions of the moon itself. With the only disturbances being asteroids and other space debris, the surface of the moon isn't weathered like Earth is. The

lack of gravity and disturbances also means that the regolith isn't very tightly packed, as seen in Figure 38.

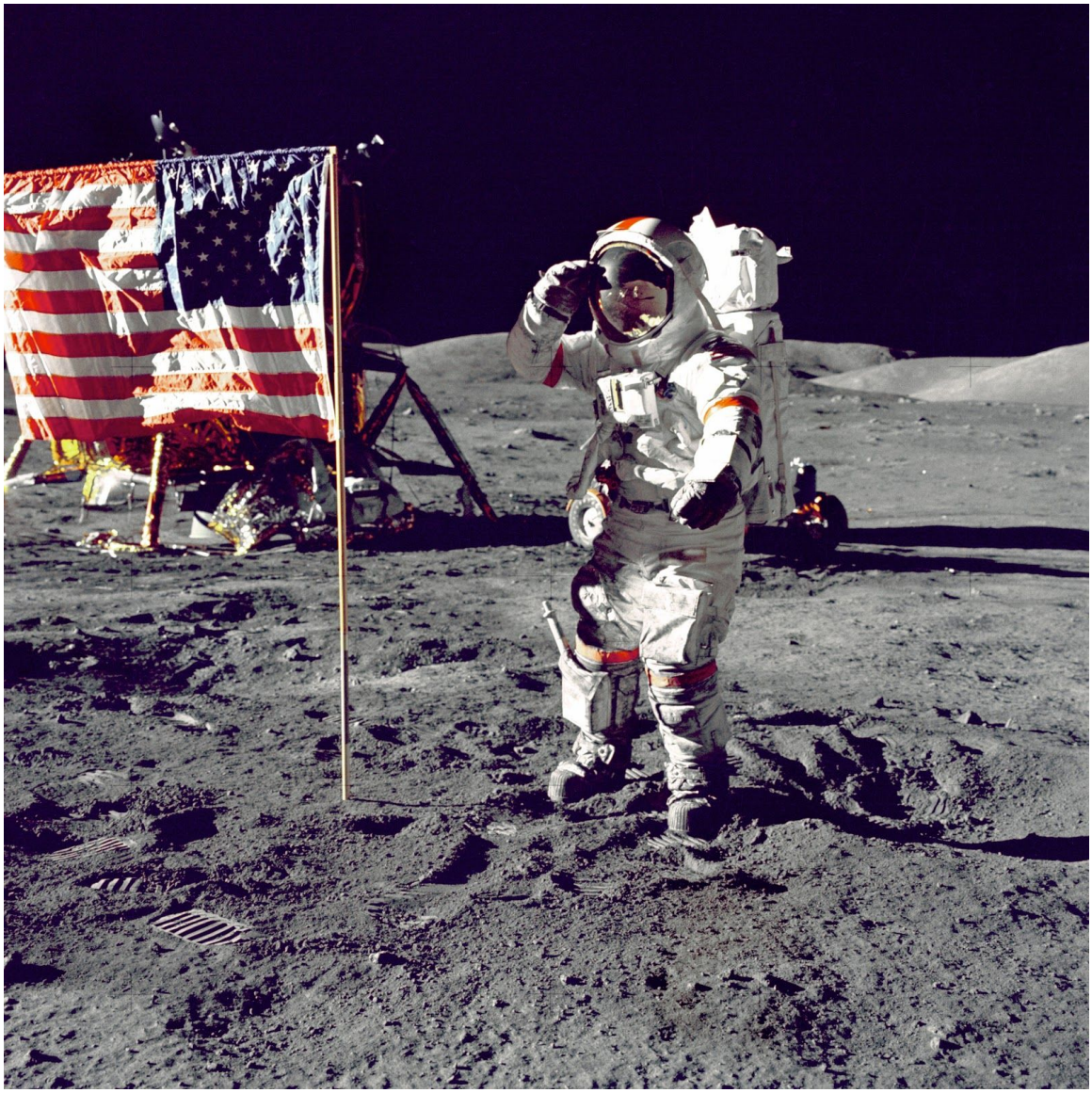

Figure 38 | Surface of the moon. Photo by NASA on Unsplash

The astronaut's footprints are easily seen in the regolith, even though the astronaut's weight is so low due to lack of gravity.

NASA has also developed a cement-like mixture of regolith with a polymer that can be used in 3D-printing. This can be used to build structures and tools for Astronauts to use. The EZ-RASSOR is made almost completely out of 3D-printed parts, everything except the motors' gearboxes and the electronics. This means

that the swarm of EZ-RASSORs don’t have to be brought on the rocket, reducing weight dramatically.

Scheduling

To most efficiently gather resources, our swarm control system must effectively divide up tasks which most fully use our limiting resources. Our constraints include limited number of EZ-RASSOR rovers, limited space for recharging EZ-RASSOR rovers, and possibly limited space at a dig site. Kurt, our mentor, has also mentioned that it would be possible that we would only be able to have one EZ-RASSOR at the dig site at any given time.

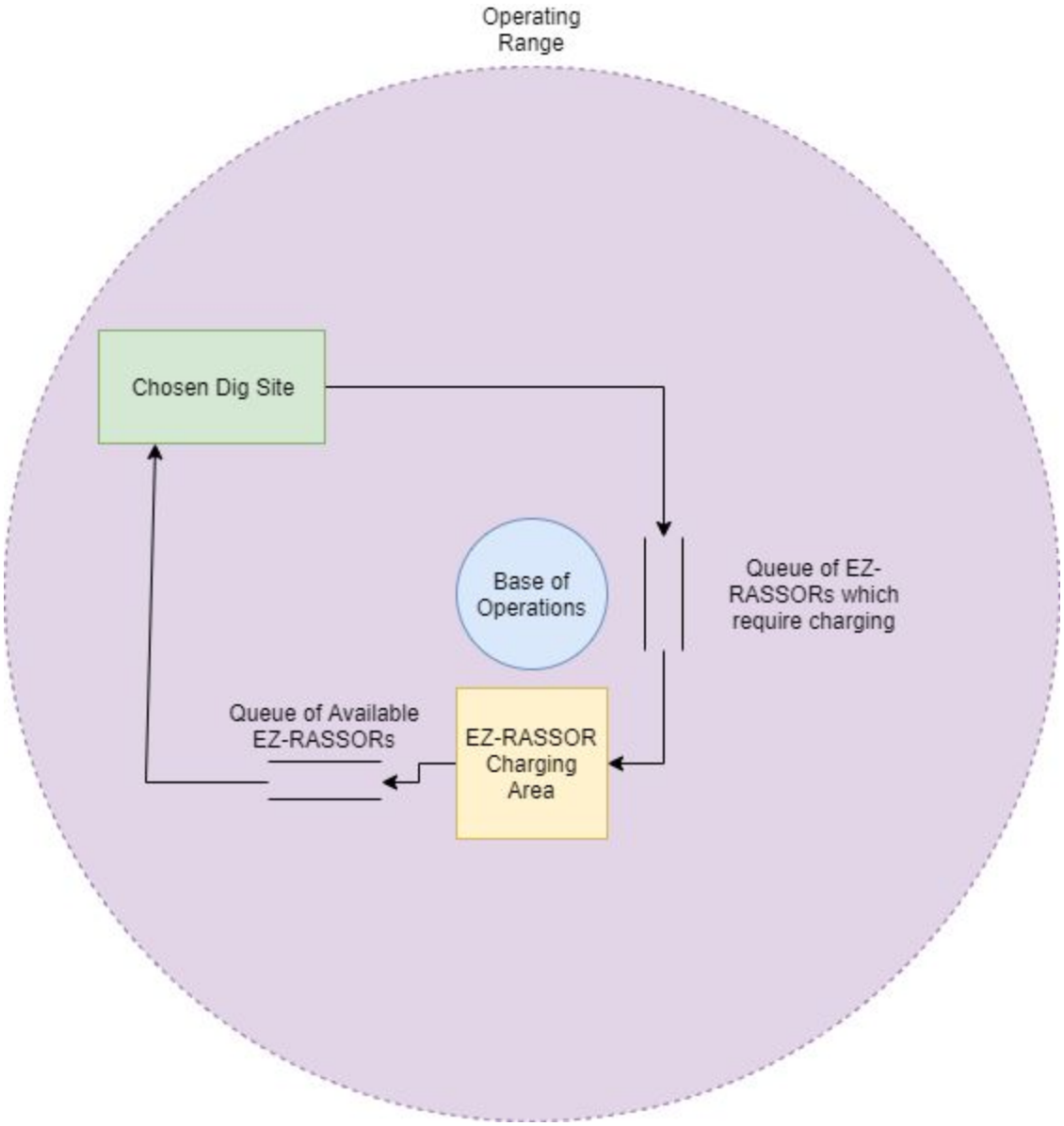

Figure 39 | A diagram depicting the high-level flow of EZ-RASSORs between the different important areas

Each EZ-RASSOR has an estimated working time to charging time ratio of three to one. This was based on the data SwampWorks has collected about the full sized RASSOR.

The scheduler may also make decisions based on the individual characteristics of each EZ-RASSOR. For instance, the scheduler should prioritize the rovers which have the lowest amount of time spent in the field, which will more evenly spread the amount of wear across the swarm of rovers.

Simulating Swarms in Gazebo

One of our objectives we hope to accomplish with this project is determining a sweet spot for the number of EZ-RASSOR rovers to deploy to most efficiently mine regolith. We will accomplish this goal by running many experiments in the Gazebo environment and controlling variables such as the number of EZ-RASSOR rovers which can charge at a time and the amount of regolith which is mined by each EZ-RASSOR. Then, we can run a sufficient amount of simulations using our swarm intelligence with different numbers of EZ-RASSOR agents and determine what the most optimal number of EZ-RASSORs working at once is.

These tests will also help us to identify the limiting factors for a swarm of EZ-RASSOR rovers. We suspect the main bottleneck which determines the efficiency of the swarm to be the amount of EZ-RASSORs which can charge at a time. However, further testing may lead to additional limiting factors being revealed.

Digging Routine

To prevent the EZ-RASSOR from getting stuck, the rover needs to follow specific optimal patterns for digging. This will involve creating a trench which the EZ-RASSOR can drive into and out of. Figure 40 shows the approximate dig pattern which we will use.

Figure 40 | The dig pattern which we will use to prevent the MINI-RASSOR from getting stuck

To accomplish this pattern, the EZ-RASSOR will start digging the trench, and will progressively lessen the length of it's dig, until it reaches a point where either it's collection drums are filled or it has hit a rock which it cannot dig through.

The EZ-RASSOR would be unable to exit the trench it digs for one of two reasons. The first is that the angle of the side is steep enough to cause the EZ-RASSOR to flip over when it attempts to climb it. The second is that the grade of the side is too steep for the EZ-RASSOR to drive up when it's regolith collection barrels are full.

Simulating Dirt

We can classify Lunar Regolith (read: dirt) as a group of particles. Since we are simulating digging functionality with our EZ-RASSOR platform, one might expect to also simulate dirt in the Gazebo simulator. Ideally that would be the case, but due to constraints in Gazebo itself, and computing power constraints, simulating interactions of fine particles in this context may not be possible. As part of our research, we looked into the general methods of simulating particles (and as an extension, dirt) in order to learn more about the workings of it.

Discrete Element Method

A discrete element method (also known as DEM), is a family of numerical methods for computing the motion and effect of a large number of fine particles with uses of rotational degrees-of-freedom, stateful contact, and some complicated geometries (i.e., polyhedra).

DEMs are relatively expensive in terms of computational power, which means we are limited by it in many situations in either the number of particles, or the length of the simulation.

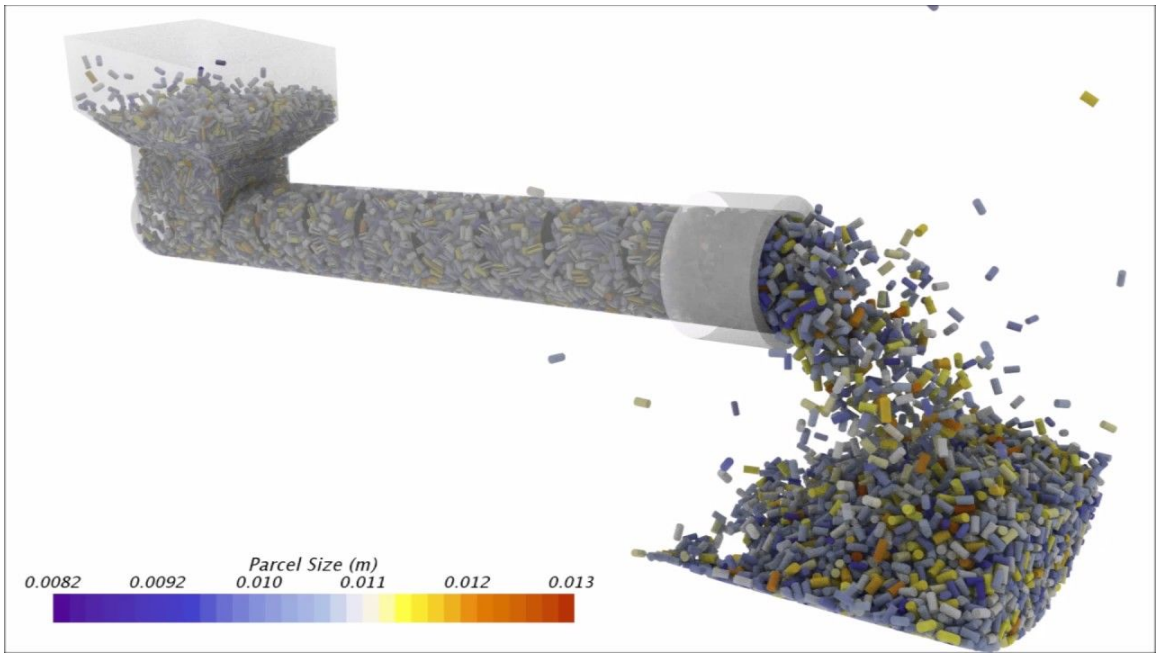

Figure 41 | Example of a DEM simulation of parcels being carried upwards by a rotating drum

Parallel Computing of DEMs

A solution to the computing power problem of Discrete Element Methods would be to build code that takes advantage of parallel and multiprocessor computing capabilities plus using distributed systems to scale up the number of particles or length of simulation, or both.

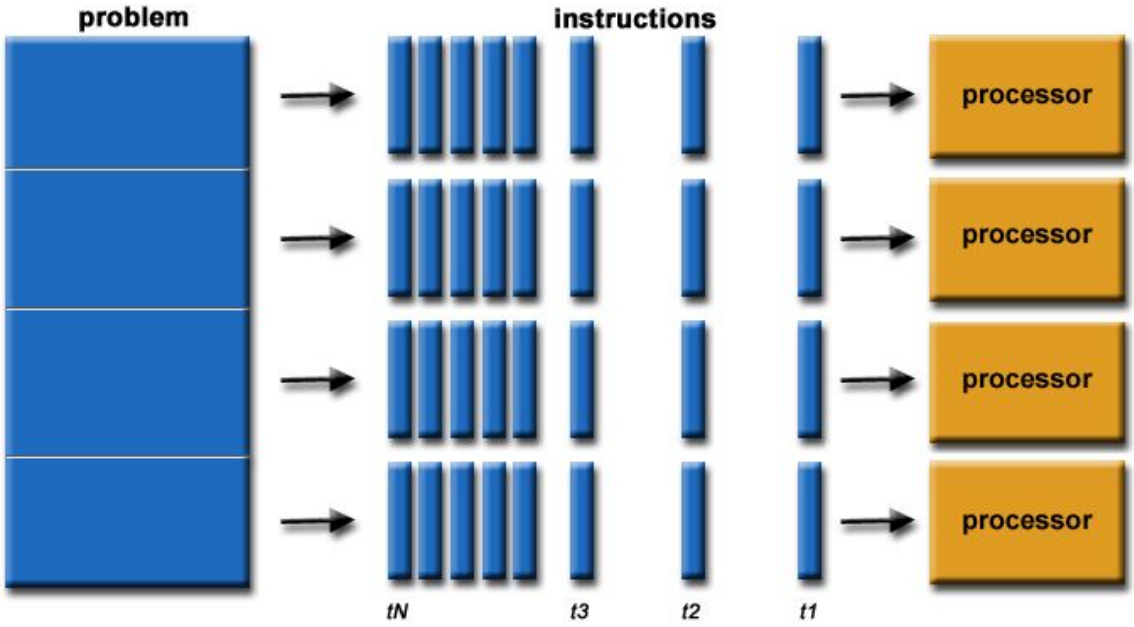

Figure 42 | Parallel execution of a problem

Although the approach sounds very good on paper, there are some drawbacks that we have to consider when writing multithreaded code. One of the biggest problems in multithreaded programming is that not all tasks will scale perfectly between threads. In fact, it is very hard to write code that scales anywhere close to perfectly across all threads.

Consider a multithreaded program that has some intermediate computation in a way that threads will have to wait for results from some thread in order to move forward with the execution. Although not as much of a bottleneck as it is in a single-threaded program, we can still see that the performance of a multithreaded program is still highly dependent on single-threaded computation. Amdahl's Law demonstrates this.

Amdahl's Law is a formula which gives the theoretical speedup of a task at a fixed workload that can be expected of a system whose resources are improved. It is often used in parallel computing to predict the theoretical speedup when using multiple processors.

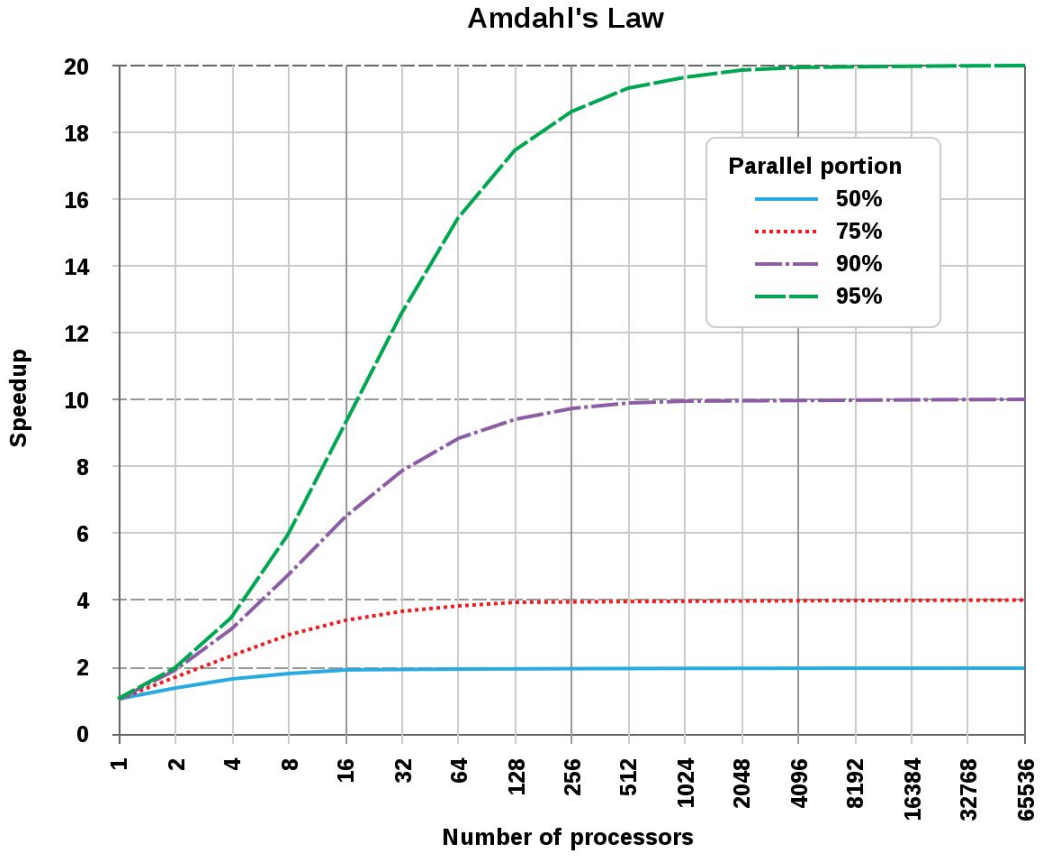

Figure 43 | Scaling of a program across multiple threads as predicted by Amdahl's Law

Amdahl's law can be formulated in the following way:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

Where

- S_{latency} is the theoretical speedup of the execution of the whole task;
- S is the speedup of the part of the task that benefits from improved system resources;
- P is the proportion of execution time that the part benefiting from improved resources originally occupied.

And,

$$\begin{cases} S_{\text{latency}}(s) \leq \dfrac{1}{1-p} \\ \lim\limits_{s \to \infty} S_{\text{latency}}(s) = \dfrac{1}{1-p}. \end{cases}$$

This shows that the theoretical speedup of the execution time increases with the improvement of the resources of the system (read: more threads), and that regardless of the magnitude of the improvement, the theoretical speedup is still always limited by the part of the task that cannot benefit from the improvement (read: cannot be parallelized).

Homogenization

Due to computing the physics of each individual particle being so expensive, as an alternative we can average the physics across many particles, and as a result, treat the material as a continuum. For example, for solid-like granular behavior that we see in soil mechanics, this approach would usually treat the material as elastic or elasto-plastic, and models it with the finite element method or a mesh-free method.

Finite Element Method

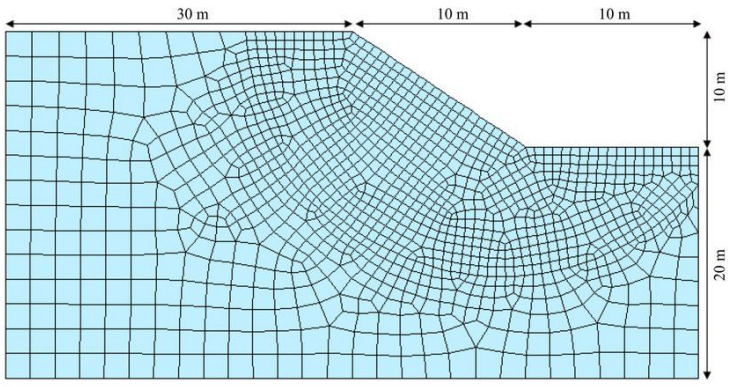

Figure 44 | FEM Mesh that is created to be used with FEM computing software to find a solution for a fluid dynamics problem.

The Finite Element Method is a numerical method for solving partial differential equations in two or three space variables. Its approach is to subdivide a large system into smaller, simpler parts that are called finite elements. This would result in a construction of a mesh of the object, which is the numerical domain for the solution, which is comprised of a finite number of points. The Finite Element Method turns it into a system of algebraic equations, and approximates the unknown function over the domain. These finite elements are then assembled into a larger system of equations that models the entire problem.

To simulate or not to simulate?

As we can see, simulating fine particles like dirt is computationally expensive, and although there are solutions that we could employ to model the particles, they are very much complex and difficult to implement. And even then, the results are not guaranteed, as we can see from the parallel simulation demonstration.

Add to the fact that Gazebo also does not natively support particle simulation, we will not simulate the digging and interactions with the regolith part of the process and assume that the functionality of basic digging is well-implemented by the EZ-RASSOR team at SwampWorks. In our case, we will focus on simulating the swarm computing environment instead.

Understanding the Existing EZ-RASSOR ROS Codebase

The work from last year's team will prove to be a very important asset for the team, as their code is clean, well documented, and modular in nature. This means that we can simply swap out nodes or change their functionality without having to change everything else above or below it in the pipeline.

The code snippet shown below in figure 45 demonstrates the modularity of the codebase. Everything is neatly placed in their own separate functions

```
# Main loop until location is reached
while at_target(world_state, ros_util):

    if uf.self_check(world_state, ros_util) != 1:
        rospy.logdebug('Status check failed.')
        return

    # Get new heading angle relative to current heading as (0,0)
    new_heading = nf.calculate_heading(world_state, ros_util)
    angle_difference = nf.adjust_angle(world_state.heading, new_heading)
    if angle_difference < 0:
        direction = 'right'
    else:
        direction = 'left'

    # Adjust heading until it matches new heading
    while not ((new_heading - 5) < world_state.heading < (new_heading + 5)):
        ros_util.publish_actions(direction, 0, 0, 0, 0)
        ros_util.rate.sleep()

    # Avoid obstacles by turning left or right if warning flag is raised
    if world_state.warning_flag == 1:
        uf.dodge_right(world_state, ros_util)
    if world_state.warning_flag == 2:
        uf.dodge_left(world_state, ros_util)
    if world_state.warning_flag == 3:
        uf.reverse_turn(world_state, ros_util)
        rospy.loginfo('Avoiding detected obstacle...')

    # Otherwise go forward
    ros_util.publish_actions('forward', 0, 0, 0, 0)

    ros_util.rate.sleep()
```

Figure 45 | Code snippet from the ROS function that navigates the rover to a given location while avoiding obstacles.

A lot of high-level functionality was implemented, which for the team means we won't have to spend too much time figuring out the architecture for the main functionality. Instead, we can focus on the implementation of the scheduling systems and enhancements to the existing code.

Another useful function we have to work with is the Auto Dig function. This function has the rover *simulate* digging by having it move forward at a slower speed as well as rotate the drums. This works well within the Gazebo simulation environment, as it doesn't have the capability to simulate the actual particle interactions between the lunar regolith and the rotating drums. This will also prove immensely helpful in a demonstration situation, as well as for our scheduling algorithms.

The Auto Dig function benefits the scheduling algorithm in that it simulates the time it would take and the path the rover would take to dig, even though the Gazebo simulation is unable to properly simulate digging into regolith. This means that for a final demonstration, we could still use Gazebo despite its shortcomings. Using Gazebo, we can use a lot of the previous teams setup for running the simulation, thus saving us the time of creating a whole new environment from scratch to make up for the missing functionality. However, we need to keep in mind to openly share that the simulation isn't actually *digging down*, but simulating the motions and time taken by the digging.

```
def auto_dig(world_state, ros_util, duration):
    """ Rotate both drums inward and drive forward
        for duration time in seconds.
    """
    rospy.loginfo('Auto-digging for %d seconds...', duration)

    uf.set_front_arm_angle(world_state, ros_util, -0.1)
    uf.set_back_arm_angle(world_state, ros_util, -0.1)

    # Perform Auto Dig for the desired Duration
    t = 0
    while t < duration*40:
        if uf.self_check(world_state, ros_util) != 1:
            return
        ros_util.publish_actions('forward', 0, 0, 1, 1)
        t+=1
        ros_util.rate.sleep()

    ros_util.publish_actions('stop', 0, 0, 0, 0)
```

Figure 46 | Code snippet of the Auto Dig function

Initial Design Summary and Diagram

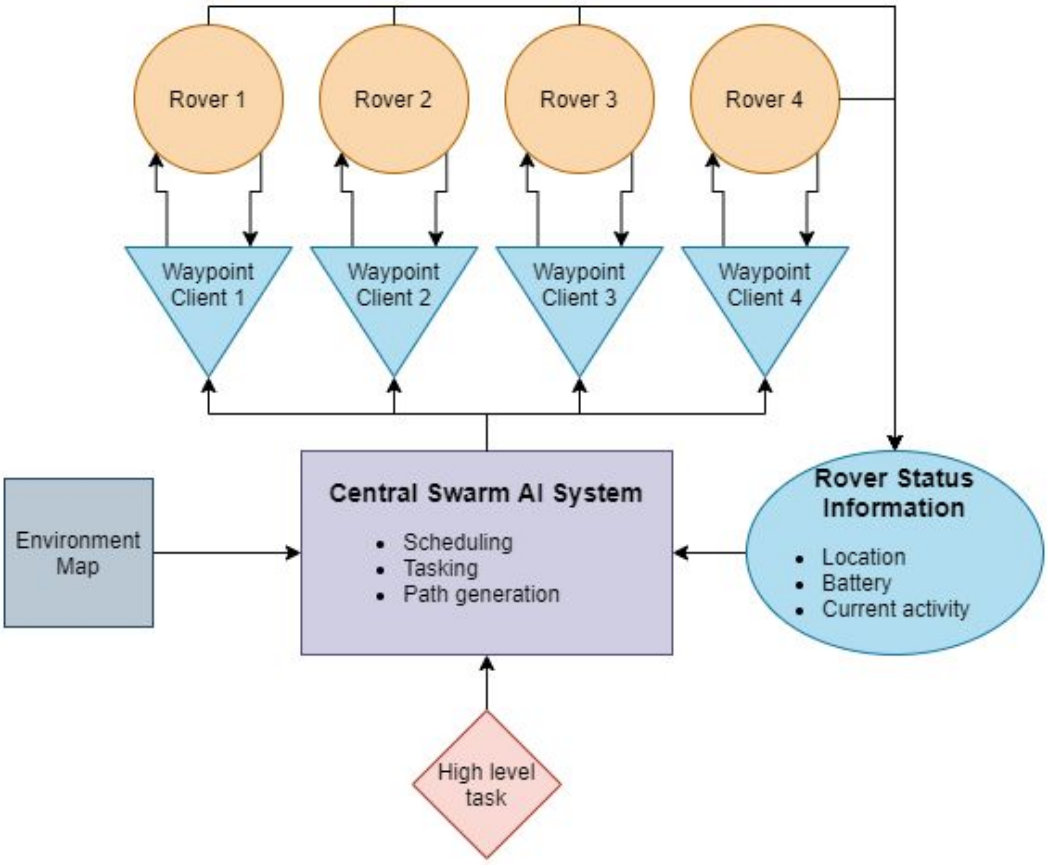

Figure 47 | High level diagram of how each system in our project will communicate with each other

This diagram is built to portray the flow of data between several interconnected systems, which allow the swarm AI system to manage the tasking and routing of individual EZ-RASSOR rovers. This constant flow of data between a mesh of rovers allows the system to continuously update its knowledge of the environment as the landscape is explored, as well as track the status of individual rovers. It should be noted that this diagram attempts to visualize only the swarm based decision making process. Each individual rover is responsible for its own autonomous navigation and localization functionality. The majority of this, however, was developed in the previous years endeavor and will be iterated upon by this year's black team.

Central Swarm AI System (Daniel Silva, Chin Winn, Autumn Esponda, Daniel Simoes, Martin Power)

This is where the bulk of the hive management and collaboration decisions are made. Either using an algorithmic approach or being trained with reinforcement learning, this system will output optimal tasks and routes for each individual rover in a swarm, given a high level task or goal. When called, this module will first make calls to the environment grid system and rover status database. This allows the system to make informed decisions based on what the environment is like surrounding the high level task and the current status of the swarms rovers.

Pathfinding System (Daniel Silva, Chin Winn)

The pathfinding system will be tasked with the creation of a sufficient set of waypoints which will provide a high-level path for the EZ-RASSOR rover to follow. Due to the computational restraints of the EZ-RASSOR the path must be predetermined and sent to individual rovers by the central swarm system. This functionality will require the implementation of state of the art pathfinding algorithms for multiple agents. These algorithms will run on the environment map described above, and must not collide with known obstacles or with other EZ-RASSOR rovers.

Environment Grid System (Daniel Silva, Daniel Simoes)

This system is responsible for building and maintaining a coordinate map of the surrounding area to track points such as obstacles, hazards and resource locations. As rovers explore the environment, they will be expected to send signals back to this module in order to update the map, based on what they encounter. This will likely involve sending signals over WiFi, which is natively supported by ROS.

Rover Status Information (Martin Power, Chin Winn, Autumn Esponda)

This module ensures the central system is able to monitor metrics such as remaining battery life, current rover locations, and even rogue or offline rovers. Being able to access data points such as these will allow the central system to make decisions based on the status of individual rovers, on top of known environment information. This will create a robust swarm AI system, capable of making truly intelligent and adaptive decisions given a high level task or goal.

Rover Scheduling Subsystem (Martin Power, Autumn Esponda)

The scheduling subsystem will be in control of the rovers when they are at the base of operations. It's main focus will be on maintaining a queue of rovers which require charging and making decisions based on the current need the swarm intelligence system has for more rovers. This system also has the responsibility of ensuring at least one rover is always working at the dig site. In order to facilitate this, this subsystem will need to balance rovers based on their battery, and distance from the digsite and home base, in order to efficiently cycle rovers between the dig site and home base.

Work Breakup

Member	Central Swarm AI System	Pathfinding System	Environment Grid System	Rover Status Database	Rover Scheduling
Danny	X	X	X		
Autumn	X			X	X

Chin	X	X		X	
Daniel	X		X		
Marty	X			X	X

Design Details

The following detailed design of the swarm management and tasking system we intend to implement will focus on the base requirements our team has outlined for this project. Once these are accomplished, the team's focus will shift to some of the stretch goals we've discussed. The base requirements for this project are outlined below:

- Given the coordinates of a dig site or area, the Swarm AI system should generate paths and schedules for each individual rover. Paths will be returned in the form of waypoints for each rover to navigate towards.
- The EZ-RASSOR system should maintain a coordinate map of the moon environment surrounding a lander. This map should be instantiated with NASA's Lunar Surface Model.
- The environment map should track large obstacles and terrain which the EZ-RASSOR would not be able to navigate.
- The coordinate map should be stored in such a way that allows for path planning algorithms to be ran. Likely in the form of a connected graph.
- Generated paths should not collide with known obstacles, treacherous terrain, or other rovers.
- Rover paths and scheduling should be constrained by a rovers potential battery life. The central system is responsible for ensuring rovers are brought back to the lander for charging in a timely manner.
- Rover scheduling shall be done in such a way that ensures at least one rover is always mining at the dig site.

In order to accomplish these goals and requirements, the team will need to build and integrate the following subsystems. These subsystems will interact and communicate in the ways described in the above diagram.

Environment Mapping

The primary objective of the environment mapping system will be to represent the moon environment, as well as track obstacles and hazards on the surface. In order to do this, the team will leverage NASA's Lunar Surface Model to instantiate our environment map. This surface model comes in the form of what's

known as a displacement map, however, we want to represent our environment with a weighted connected graph. In order to facilitate the transformation of this displacement map into a graph representation, we'll need to leverage the properties of this displacement map. At each pixel, a displacement map has a value which represents the elevation at that coordinate. A visualization of this is provided below.

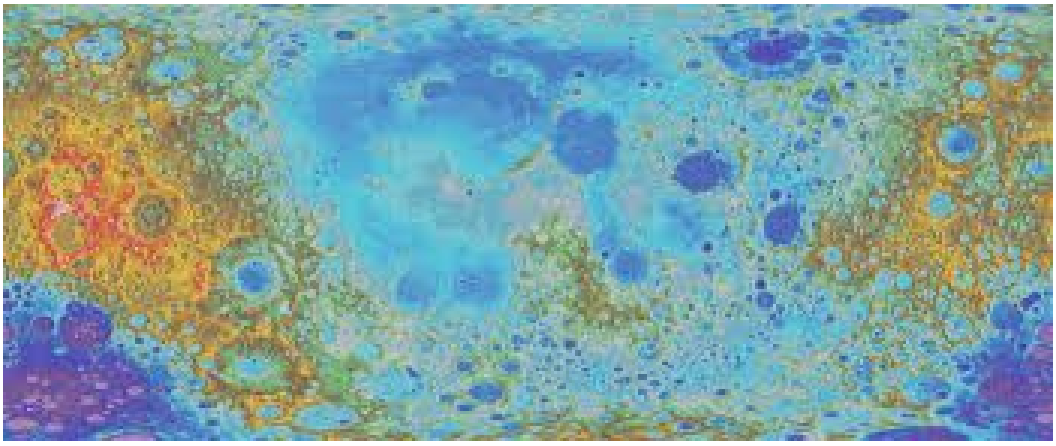

Figure 48 | Lunar Topological Map.

With this information, we could calculate the slope between each coordinate of this map. It is this slope value that we will use as the cost between nodes in our connected graph.

However, the current EZ-RASSOR, or even RASSOR itself, has limited navigation capabilities. Specifically, the rover is unable to climb slopes which are too steep. This exact slope will be provided by Swamp Works in the following weeks. For practicality, areas or edges in the graph which the physical EZ-RASSOR cannot climb will simply be treated as obstacles. In order to represent this, we can recur through our connected graph, disconnecting edges which have a cost or slope that is too great. This will result in an accurate representation of the moon surface, which accounts for high slopes and obstacles.

It should be noted that in our research, the team found that the highest resolution topological map of the moon is represented roughly in 328 feet per pixel. Hence, each edge in our connected graph will represent roughly 300 feet in practice. This signifies that any paths generated from this graph will likely be considered to be at a high level, and there will surely be small obstacles along the way which

were not captured by this elevation map. However, the 2019 EZ-RASSOR teams have discussed this and came to the conclusion that short range obstacle detection and avoidance is the task of the black team. This means the black team will be responsible for storing this map and generating what are essentially way points on the moon surface for individual rovers to follow.

Multi-Agent Path Finding

Arguably the most important task of the overall swarm management and tasking system we plan to implement is to find optimal paths for each individual rover, to and back from some target location. We are assuming these target locations will be known beforehand and communicated to this central system. Target locations for the moment are currently limited to dig sites. The idea and end goal is to generate paths for each individual rover to follow as they cycle between the dig site and charging station. These paths must not collide with any tracked obstacles on the map, as well as with other rovers or paths. In order to achieve this, we will implement a multi-agent pathfinding algorithm

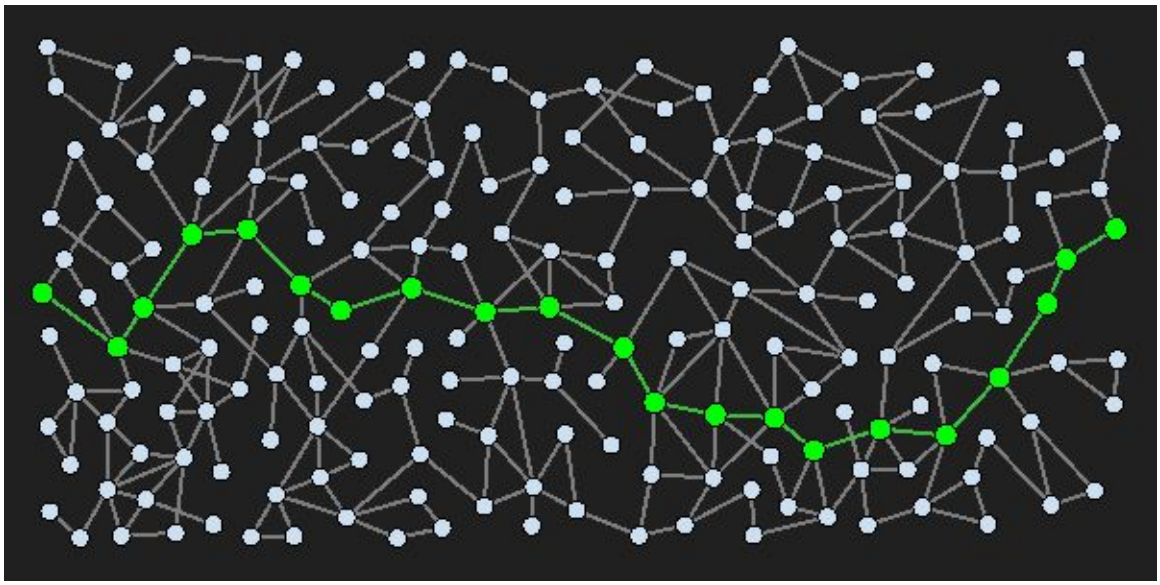

Figure 49 | Visualization of pathfinding on a connected graph representation

After conducting extensive research into this domain, the team has come to the conclusion that the MAPP algorithm would best suit this task. Reasons for this are outlined below:

1. Decently low run times in the worst case
2. Guarantees absolutely no cycles or deadlocks
    - In practice, creating a path with a cycle or deadlock would be fatal for a system such as a swarm of EZ-RASSORs. If any number of rovers were to get caught in such a scenario, they would likely be lost forever unless we apply some other intervention techniques.
3. Tractability and completeness guarantees
    - MAPPs low runtime and completeness guarantees will ensure that even with a map as large as a portion of the moon's surface, the algorithm will always terminate in some non-exponential time.
4. Requires no replanning during execution.
    - This is quite important when theorizing about the application of this system to the real world.
    - With this approach, the central system will need to compute these optimal and non-colliding paths just once for each digsite.
    - Once sent to the rovers, so long as they are able to truly follow these paths, the central system will not have the need to communicate with rovers until a new digsite is determined.
    - This is quite useful being that communication between rovers and the home base is not guaranteed once rovers get too far away.

In practice, this MAPP algorithm will run directly on the environment map we plan to build and store within the central management system. This will allow the system to generate optimal paths to and from dig sites around the moon. Once this base level of functionality is reached, the team can begin to experiment with multi-dig site scheduling. In other words, the system would now efficiently distribute rovers across multiple dig sites. There is a large extent of research within the field of multiple goal multi-agent path finding, in which the team has much to learn. However, the primary goal in the preliminary stages of development will be to implement the MAPP algorithm for the case of a single dig site, using a static map of the moon's surface.

Final Implementation Details

The path planner is implemented using the Local Repair A* algorithm, which is a decentralized path planning algorithm that is highly-scalable to large multi-agent

systems. The algorithm works by running an A* search for each individual rover, then fixes conflicts at a local level.

Decentralized searches have downsides such as suboptimal paths and cannot distinguish between problem instances that can and cannot be solved. These downsides must be considered in some applications, however our system isn't affected much by them. Our rovers will mostly be working in large, open spaces, so an unsolvable problem is highly unlikely to occur. Additionally, slightly suboptimal paths will have very little effect on the system overall. Using a centralized search would solve these problems, however they are much more computationally expensive, which grows exponentially with how much you scale the system.

Local Repair A* allows for rovers to have their paths replanned if they have veered too far off course while attempting to avoid collisions with other rovers and any other previously unknown obstacles. The Local Repair A* algorithm requires active tracking of each rover, which is why our system has a rover status tracking node, discussed in Section IV. The swarm system will actively track each of the rover's positions and update paths whenever the system determines that it has gone too far off path. This way, a rover which has gone off course can efficiently return to being on the correct path. The rovers themselves are also able to detect when they are about to collide with something in front of them, which acts as another failsafe in case an unexpected event occurs.

The Local Repair A* algorithm is very efficient due to the decentralized nature of it. The computation doesn't all happen at once, and the system can begin running before all of the computation has been finished. This means that each rover is able to begin executing it's task as soon as it's path has been generated. Any replanning will be handled by a secondary ROS node, the waypoint_client that the swarm controller generated for that rover.

Rover Scheduling

The rover scheduling subsystem will be made up of three subsystems, one which will handle the dispatching of rovers to the dig site (Dispatch Subsystem), one which will handle efficiently queueing rovers for charging once they return (Charging Subsystem), and one which will track real-time data about each rover, including the rover's battery level, location, whether the rover is stuck

somewhere, and the amount of wear and tear the rover has experienced, which will likely be tied to the number of dig missions it has been sent on (Rover Status Subsystem).

Rover Scheduling Subsystems

- Dispatch Subsystem
    - Primary Responsibility
        - Control flow of rovers to the dig site, based on the number of rovers which can be accommodated at the dig site
    - Other Responsibilities
        - Track the current dig site and dig site capacity
        - Transfer coordinates generated by path finding system to EZ-RASSORs
    - Relevant Data
        - Current dig site
        - Locations of all dig sites
        - Number of EZ-RASSORs available to dispatch
        - Number of EZ-RASSORs at each available dig site
        - Capacity of each dig site
- Charging Subsystem
    - Primary Responsibility
        - Decide which EZ-RASSORs require charging and what order they should charge in
    - Other Responsibilities
        - None
    - Relevant Data
        - List of each EZ-RASSOR available for charging
        - Current battery level of each EZ-RASSOR
        - Approximate “safe” charge level for a rover to be sent out
        - Current demand for more rovers
- Rover Status Subsystem
    - Primary Responsibility
        - Track current status of each EZ-RASSOR
    - Other Responsibilities
        - Provide information to the other two subsystems when they require it, and act as a middleperson between the two subsystems

- Relevant Data
  - List of each EZ-RASSOR being used
  - Status of each EZ-RASSOR, including:
    - Battery Level
    - Location
    - Whether it is en route to dig site, returning from dig site, digging, queued for charging, charging, queued for deployment, or broken down
    - Amount of dig missions it has completed
    - Average battery usage coefficient

Final Implementation Details

Our implementation of the scheduling system has it implanted within the swarm_control node. The scheduler reads the information that is being tracked by the rover status system and uses it to determine if it should send it to charge. The scheduler is constantly tracking the battery levels of the rovers and will call them back before they've completed their task if the system thinks it won't be able to return to home base if it loses more battery.

Software

Gazebo Simulator

Gazebo is a graphical robotics simulator which the previous team had used for the purpose of virtually testing code written for the rover. It is a very robust program that simulates all aspects of the robot. One of the key features that Gazebo has is the ability to gather various sensor data from the robot as if it were a physical robot moving in a real terrain. In the case of EZ-RASSOR, the rover has a set of forward-facing stereo cameras. Gazebo has the ability to get exactly what the cameras would see if they were real, which allows one to, for example, train models on the virtual environment using the virtual cameras and learn what may or may not work on the physical robot. This is a very useful feature for the purposes of this project specifically as we won't be able to access the rover particularly often to figure out how to train a model on the real cameras. We can find out what kind of machine learning models work based on the simulated sensor data and that will apply almost exactly to the actual rover.

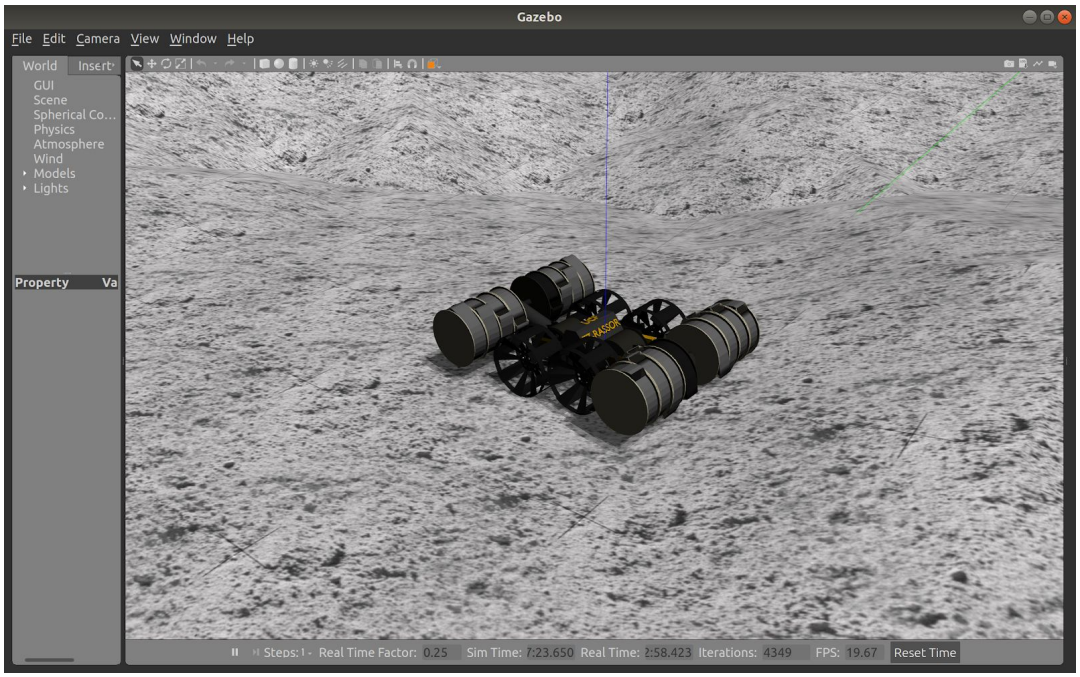

Figure 50 | EZ-RASSOR Model in the Gazebo simulator

The simulator also allows for multiples of the same model to be loaded into one world at a time, which will be useful when it comes to the swarm intelligence, which will be discussed in more detail below. Gazebo is a robust system which allows for easy ways to visualize and place several rovers, which in conjunction with ROS, allows us to have dozens of rovers going at the same time which is just not feasible to have that many real rovers for testing purposes.

The previous EZ-RASSOR Senior Design team created models of the EZ-RASSOR robot and rigged them. Gazebo supports ROS and the team loaded their code onto the simulated EZ-RASSOR to do most of their testing. Our team plans to use a similar approach, using simulation to do most of our testing since we don't have access to a swarm of EZ-RASSOR rovers. Additionally, this will prevent the overhead of building and maintaining a swarm of robots and creating an accurate real-world environment for testing. Figure 51 shows the current status of the EZ-RASSOR model in the Gazebo simulation.

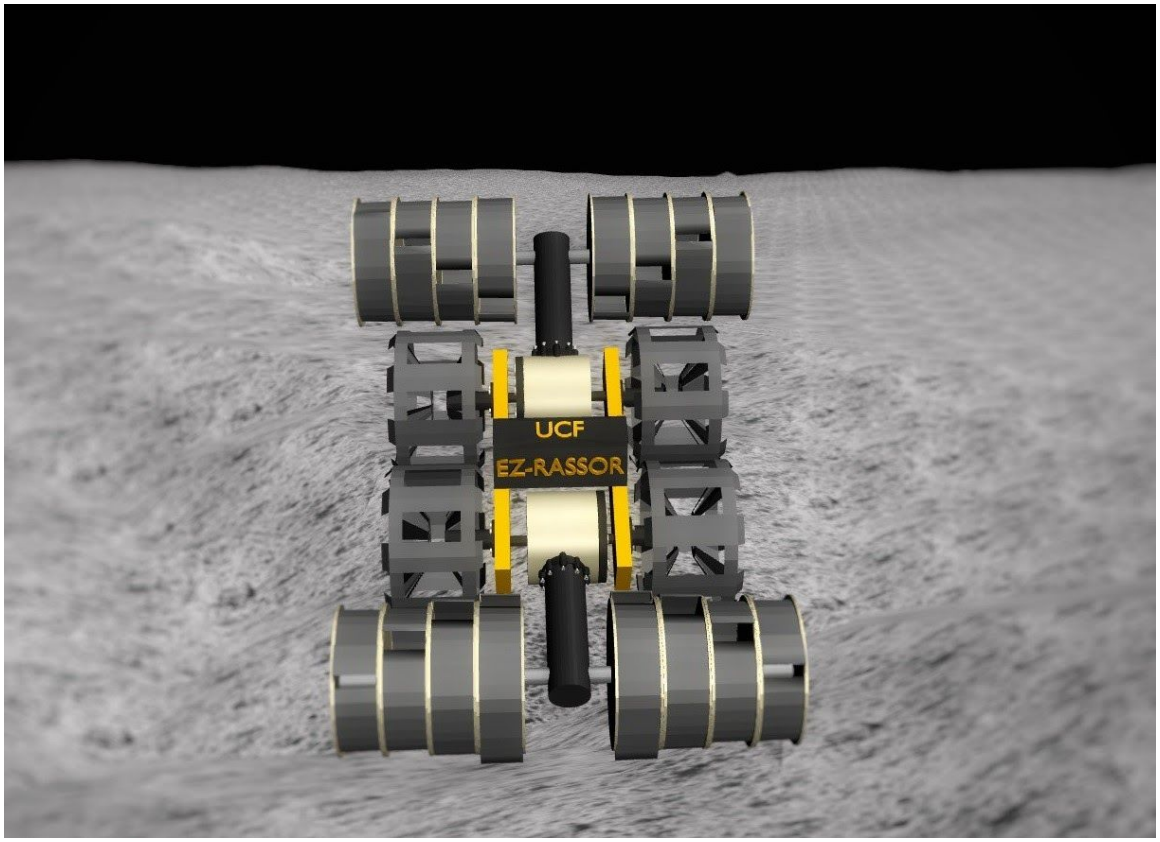

Figure 51 | Underside of the EZ-RASSOR model in the Gazebo simulation. The rover is able To drive around, as well as move its collection buckets in simulation.

Currently, the Gazebo simulation supports all of the functionality of the EZ-RASSOR, but the environment itself doesn't support functionality we will need to test some of the functionalities we plan to implement. The simulation currently doesn't support digging, which will need to be implemented for us to test our trench digging subroutine.

OpenAI Gym

Figure 52 | The OpenAI logo

OpenAI Gym is an open source toolkit designed to make simulating and testing reinforcement learning algorithms easier than alternative solutions. It was made by the for profit company OpenAI and is easy to set up and get working simulations quickly. While OpenAI Gym isn't as powerful and feature complete as Gazebo, it will prove to be a useful stepping stone before we test our full fledged models in Gazebo.

While Gazebo is a 3D simulator, OpenAI is a 2D playground and because of this, there will be certain limitations as to what we can actually simulate on it. Our plan will be to simulate a top down view of the moon surface and create some simple two dimensional rovers to test our models with. We won't be able to test our models against edge cases like steep hills or valleys, but we can use rocks and pits to at least simulate obvious obstacles. The goal with using OpenAI Gym first before plunging straight into Gazebo is that we can focus more on the model and less on learning Gazebo's complicated interface. The hope is to get it working in OpenAI Gym and then fine tune it in Gazebo.

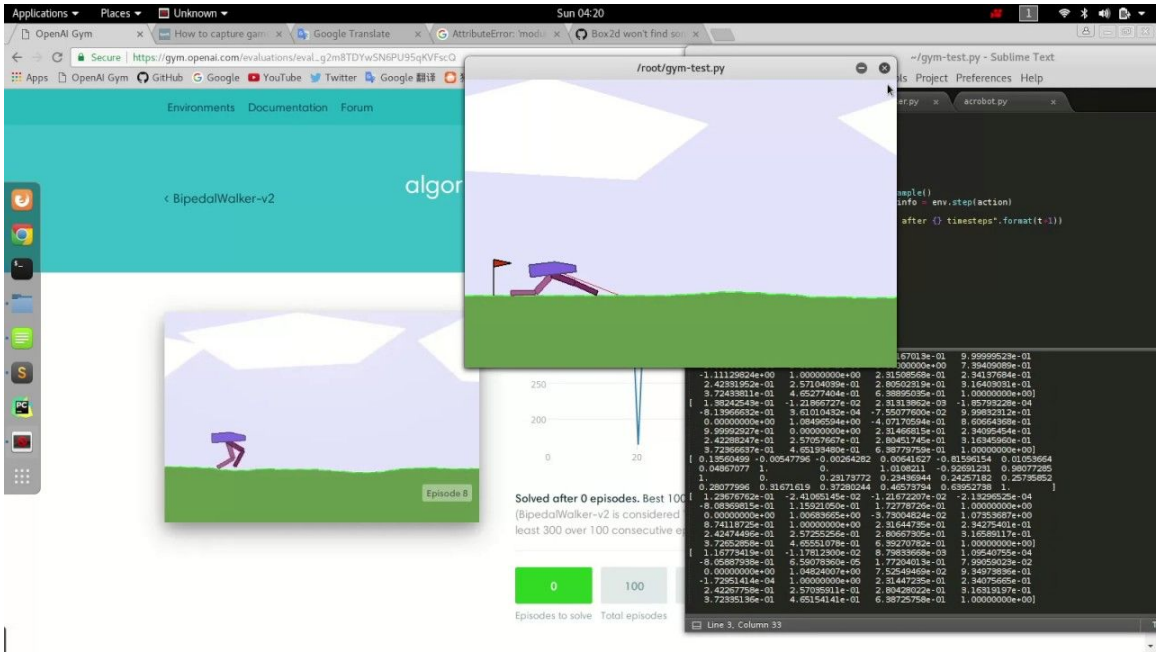

Figure 53 | The OpenAI Gym interface

Python

Figure 54 | The Python logo

In the context of this project, Python is the main programming language used due to its simplicity and popular user base. From a performance standpoint, one would think C++ should be the way to go to maximize performance since the rovers are relatively weak in power, but given the short time frame of the project and the relatively complex goal that we are trying to achieve, Python would be able to get us there in a more efficient manner. Python's large user base in Artificial Intelligence and Machine Learning also means a lot of software and

library support for those tasks such as PyTorch, Tensorflow, and OpenCV. The previous team's base is also written in Python, so with that in mind we decided to go with Python as well.

Robotic Operating System (ROS)

Figure 55 | The ROS logo

Robotics control software is *hard*. There are so many individual moving parts involved in getting even the simplest of robotics applications up and running. If we had to write such a program or software suite from scratch to operate the RASSOR robots, it would be a tremendously difficult task in and of itself. This is where ROS comes in.

Started in 2007 as the product of the research performed at two separate robotics programs and Stanford University, ROS is a collection of frameworks which provides hardware abstraction and robust message-passing between separate process. This all among many other features to aid in the creation of robotics systems. It is open-source and provides hardware abstraction, low-level device control, and package management. There is a large community of software developers creating new packages, usually open-source as well, for ROS. This allows for creation of very complex functionality in a significantly shorter time then writing it all from scratch, as well as having codebases be easier to maintain because of having many dependencies.

At its core, ROS is a publish/subscribe message-passing middleware, where all communications are asynchronous. This means that some modules will publish a set of Topics and others to subscribe to that topic. Modules can communicate with each other through these clearly-defined protocols which almost forces modular and easy to read functions which lends well to modularity and easy to

maintain code. In addition to that, ROS supports a string file format called a Bag, and uses them to store data for future use or even use during robotic operations.

We are using ROS for the operation of the EZ-RASSOR system because the previous team had implemented basic functionality of the rover using it. ROS is used on both the real EZ-RASSOR robots, as well as used to control each individual rover in the Gazebo simulation environment.

PyTorch

Figure 56 | The PyTorch logo

PyTorch is an open-source machine learning library backed by Facebook's artificial intelligence research group. PyTorch is popular and well-supported, as we can see from its very active Github repository. PyTorch has a well-polished Python interface, and provides tensor computing with acceleration from the GPU (Nvidia CUDA). Over the past few years, this framework has taken the research world by storm, and is now the most used machine learning framework on research projects. This popularity means we are likely to find answers in case we come across issues. Continuing with this trend, PyTorch will be the EZ-RASSOR gold team's go-to machine learning framework throughout the development of this project.

OpenCV

Figure 57 | The OpenCV logo

Since a lot of the rover's functions rely on looking at the surroundings and making sense of them, we will need a library to process visual data in real-time. OpenCV is a library of programming functions mainly aimed at real-time computer vision, originally developed by Intel. OpenCV support deep learning frameworks such as PyTorch, which we will be using.

React Native

Figure 58 | The React Native logo

The EZ-RASSOR, both simulated and real, can be controlled through a mobile application. This application is written in React Native, meaning it can be deployed to both Android and iOS from one code base which gets compiled down into native code for their respective devices. The app is currently hosted on the Google Play Store and could be put on the Apple App Store if funding was provided, but is unavailable at the time. It can, however, still be manually installed onto an iOS device using what is called Free Provisioning. This means

that app can be installed manually to an iOS device from a machine running MacOS, but the app will only work for seven days without reinstalling it.

Figure 59 | User interface for the EZ-RASSOR Controller mobile application

Rationale for Chosen Technologies

Operating Systems

Due to the nature of the project, it is not intended to be cross platform. It is being developed on a very specific stack of software that was decided on by the previous team. One of the most important parts of this stack is the operating system. The operating system for this project should be fast, efficient, reliable, and easily usable. The Operating System also needs to be able to run on less powerful hardware such as a Raspberry PI since that is what our EZ-RASSOR uses. It also had to be easy for future contributors to pick up. This is why last year's team decided on using Linux and ROS to develop and run the EZ-RASSOR.

Why Linux?

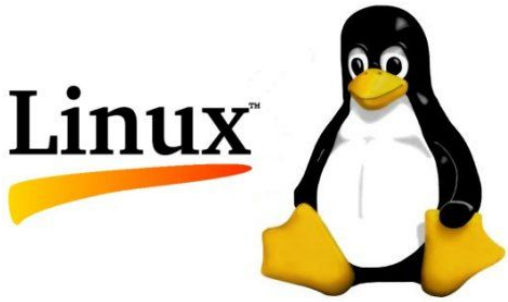

Figure 60 | The Linux logo

Linux is the operating system of choice for the EZ-RASSOR project, both for development and also running onboard the EZ-RASSOR. Our team will be using Ubuntu Xenial (Ubuntu 16.04) as our development operating system and Ubuntu Server 18.04 onboard the EZ-RASSOR.

LInux is an extremely versatile development environment and many of our team members already use it as their environment of choice. Other members of our team use Mac OS as their main operating system, and the transition from Mac OS to Linux is generally considered pretty easy. Linux also allows our team to easily make use of technologies such as Git and Docker and is highly customizable.

Having a common operating system between the development environment and the onboard operating system will make it easier between transitioning between the two systems. Ubuntu Xenial is the main operating system which ROS supports, so it makes sense to use for a primarily ROS-based project. Our team has attempted to install ROS on other Linux distributions and even other versions of Ubuntu and it was excessively tedious and very buggy.

Linux is a very commonly used operating system across the industry and especially in the robotics field. Since this project is open source, we want to encourage easy collaboration and part of encouraging that is by making the project easy to set up and well supported. Ubuntu itself is also free and open source, increasing accessibility and allowing us to stay free of tricky legal problems.

One of the goals of the EZ-RASSOR project is to provide a learning platform for students to experiment with a full robotics platform. Using open source technologies lowers the cost for schools and lower-income students to work with our platform. This is an important aspect of the EZ-RASSOR project and further solidifies our choice of Ubuntu as our platform.

Additionally, Linux is commonly used in industry and a student who decides to learn about the EZ-RASSOR project can learn about Linux at the same time. Linux isn't commonly used in K-12 schools, even technology focused ones, and this project provides an avenue for educators to introduce Linux into their curriculum. Familiarity with Linux is a very marketable skill, especially for a very young software engineer.

Why ROS?

Figure 61 | The ROS logo

ROS is an open source robotics operating system designed with the goal of making robotics programming more standardized. When you're building a robot, you are usually integrating many different components together, made by different people and companies, and there are usually issues with getting all the components to communicate with each other and accomplish the task which they've been tasked with. ROS attempts to solve this by allowing the creation of ROS nodes, which are discrete programs run on each component. These nodes communicate with each other through the ROS graph, which is the collection of all the nodes created. All of the nodes function independently, but communicate with all of the other nodes with commands to move, sensor data, and many other things.

Using ROS furthers our project's goal of remaining open source and accessible to students and schools. ROS is a free, open source platform and is the standard platform in the robotics industry today. This will further expose students who are working with the EZ-RASSOR platform to real world technologies and excellent resume boosters. Since ROS is very common in the industry, using ROS makes it easier for industry professionals to contribute to the project as well. One of the goals for the EZ-RASSOR project is to give a platform for NASA to learn from colleges outside of their typical bubble.

ROS is widely used, widely talked about, and therefore widely supported. There are troves of forum posts and discussions about how to accomplish things when working with ROS. This will greatly benefit our team when we encounter problems working with this technology, especially since nobody on our team has worked with ROS before.

ROS is fully supported by Gazebo, our chosen simulation software. ROS is the only robotics middleware which is supported by Gazebo, which certainly helped cement the decision to use ROS for this project. Being able to accurately simulate the whole EZ-RASSOR system is extremely important for ease of testing. This is especially important for our portion of the project, since we cannot feasibly create for example one hundred EZ-RASSORs to correctly test our swarm control with. However, with Gazebo, we are able to test with varying amounts of rovers and on a simulated moon surface.

The final advantage for ROS is that ROS nodes can be written in Python. Python is an extremely versatile language and will be used for almost all aspects of our project. Keeping the codebase mostly in a single language has many benefits. First of all, it increases our team's efficiency since less time needs to be spent learning programming languages and we can redistribute that time to learning about technologies such as path planning and artificial intelligence. Additionally, a uniform codebase increases accessibility for others to learn from and contribute to the project. Python is such a popular language and is very accessible to new programmers, so students have many resources for learning Python quickly and easily, and are highly likely to already know it if they've programmed before.

Simulation Software

When last year's team was deciding on which simulation software they wanted to use, it was important to look for one that was open source, but was still fully capable. There actually aren't that many fully featured and open source simulation software to choose from, but here are the main 3 that were decided between.

Why Gazebo?

Figure 62 | The Gazebo logo

Gazebo is a robotics simulation environment that was released in 2003 and has been in development ever since. It was made in C++ and has full support for ROS. Gazebo will prove very useful to simulate the lunar surface and test and change many different parts of our robots without actually having to swap out any hardware or worry about damaging the expensive parts. More importantly, with

the use of Gazebo we won't need access to a physical robot at all through the course of our development until the final testing phase. It is expected that we will be able to implement all of the robot actions in the simulated environment. Since last year's team took care of the user controlled actions of our robot, we will focus on using Gazebo to refine the autonomous and swarm AI operations. By using Gazebo, our team can test all of the aspects of the ROS architecture to decide on how data should be passed between nodes. We will be able to test out all of our autonomous functions and see how well the perform and see which functions should be used on the physical robot once we get access to it at the end of the semester. Since there is no realistic way to get access to 100 different robots in the real world, implementing and testing swarm robotics will be where Gazebo will truly shine. We will be able to develop and test many different ways for a group of EZ-RASSORS to interact together without having to either acquire the expensive robots or risk damaging them. Naturally, using a simulation can't account for all possible real world problems, and it is never perfectly accurate to how the real world will be. But despite this, by at least having a starting point to test in, we will have much more progress done by the time we had to fine-tune it for the real world. Since our project revolves specifically around swarm AI, it is unlikely that we'll ever be able to test all of the functions in the real world anyways.

The file type that Gazebo uses to create ROS compatible robot models is URDF, with URDF meaning "Unified Robot Description Format." The file is actually a type of XML file and contains information about the robot and its components, such as defining the size of specific components, deciding how the component connects to other components, and giving that component a unique name. Some examples of parts are the wheels, the chassis, and the different sensors that will be simulated in the Gazebo. XML isn't actually a programming language, and it should not be mistaken as such. XML is actually a markup language. Since XML is not a programming language, you can actually define any real functionality in the file. The file is simply used for letting Gazebo know how the parts look, the size of the parts, and how the parts come together to make the actual robot.

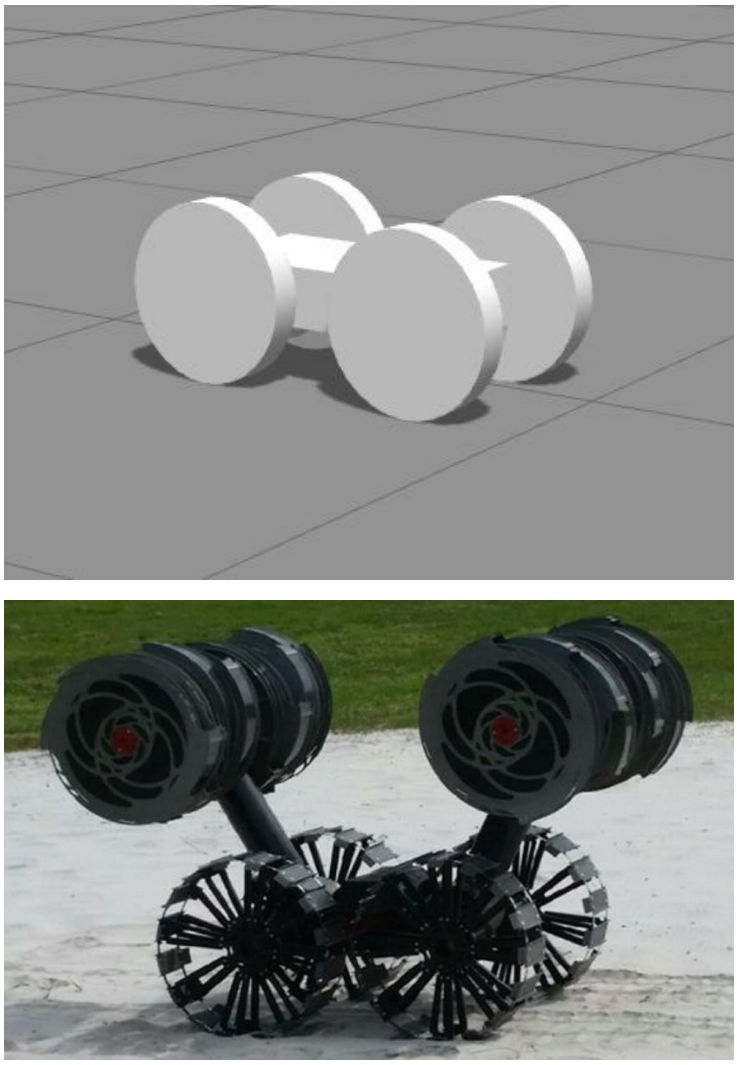

Figure 63 | Comparison of a prototype model and the RASSOR

Aside from being able to simulate the functionality of the robots, Gazebo also gives us a way to make different kinds of environments to test our EZ-RASSOR in. This means we can test our EZ-RASSOR in, say, a simulated moon. We'd never have access to the real thing but our robots are meants to work on foreign planets, so this is extremely useful. We can test our robots on hilly terrains and flat lands, but in order to do so we will have to make what are called "height maps" which Gazebo will then interpret to make the terrain. Height maps are black and white images where the darker areas are lower in altitude while lighter areas are higher.

One of the great things about the Gazebo community is that you can share models and simulation backdrops for others to use in their own projects. Because

of this, the background used in our simulation are used from models shared by other community members. Gazebo actually allow you to be very intricate with your designs, to the point of being able to define how your object reacts to sunlight. By using existing environments instead of making our own, we can focus more on actually coding th AI of our rovers.

When the previous team designed the EZ-RASSOR rover for the Gazebo simulator, they made use of open source models and built it up from there. This made having functional tires and propers weights much easier. The rest of the rover is designed to match the actual hardware that our real robot uses and includes the sensors such as the camera used to look at the robot's surroundings.

Why Not V-REP?

Figure 64 | The V-REP logo

One of the most popular robotics simulations solutions on the market today is V-REP. It was made by a Swiss company known as Coppelia Robotics and is still maintained today. Some of the things this V-REP has going for it include a great array of physics engines, a sizable selection of premade robots, and that the software is very well documented because of the company that maintains the product. The robotics community generally regards V-REP to be a complete simulations environment, since it gives the user a huge selection of resources that already come with the software that gives developers quite a bit of control of the simulation.

The problem with this program is that it isn't actually open source. While the software can be used for free by educational institutes under the GNU GPL, in all other cases it requires a commercial license. Since the program is not free, it isn't

a great choice for this project. Also, since the EZ-RASSOR project is open source, it would be ideal for the simulations software to have an active community to help contribute assets to the project.

Why Not ARGoS?

Figure 65 | The ARGoS logo

Like Gazebo, ARGoS is completely open source and is still being actively worked on by the community on GitHub. The ARGoS simulator is multiplatform and is readily available on Windows, MacOS, and Linux. It is made to be used for large-scale robotics simulations. ARGoS uses C++ to allow users to add code in their simulations to give them more control than using the UI normally would. Based on what other researchers were able to achieve in ARGoS, it seemed like a very promising solution; by browsing online you can find many projects that make use of huge scenes and a bunch of robots that move through the massive environments.

Even though ARGoS seemed like the perfect program for the EZ-RASSOR project given its use with massive swarms of robotics, which is exactly the case for our project, last year's team decided that it wasn't the right program to use due to its lack of support. ARGoS was originally created by one person named Carlo Pinciroli and even though there have been others that have contributed to the project since it was originally made, it is only a measly 11 contributors. Due to this project having to be worked on over the span of two semesters, it is important that the simulator is well-documented and established. Not only that, but the UI for the program does not have many powerful features and as such, we would have had to write C++ code for the emulator to accomplish things that would otherwise be a straightforward process on other simulators.

Machine Learning Software

One of the main goals of the EZ-RASSOR Gold team this year is to replicate the automation functionalities of NASA's own RASSOR robot. Since our system needs to work with little to no human input, we need to employ some form of artificial intelligence and machine learning. Our plan is to make use of reinforcement learning and neural networks to train our EZ-RASSOR's to be fully autonomous in forign environments. The training process should ideally be straightforward and flexible as we hope to focus our efforts in other aspects of the process.

Why PyTorch?

Figure 66 | The PyTorch logo

PyTorch is a relatively new machine learning framework that was developed by the AI research team at facebook. It was released in 2018 and is an open-source library for dataflow programming across many different tasks. Even being as new as it is, PyTorch is very well documented and their platform is stable. While PyTorch is not as straightforward to use as Keras, it has superior debugging capabilities, flexibility, and customizability when compared to both Keras, and has a very high performance like TensorFlow. Moreover, a number of members on the EZ-RASSOR gold team have prior experience with this framework. Therefore, the team has decided that PyTorch will allow us to efficiently accomplish our goals relating to machine learning, without the steep learning curve or rigid API constraints some of the other popular frameworks have.

Why Not Caffe?

Figure 67 | The Caffe logo

Caffe was developed at the University of California Berkeley at the Berkeley Artificial Intelligence Research Lab. It is meant to be easy and straightforward to use so that newer developers can get something working as quickly as possible. In 2017 Caffe2 was announced by Facebook which would have more tools including Recurrent Neural Networks. Since Caffe2 was merged into PyTorch on March 2018, there is no real reason to consider Caffe for our project anymore.

Why Not TensorFlow?

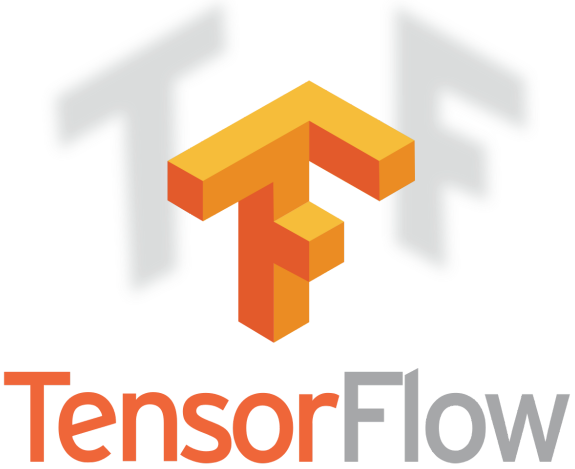

Figure 68 | The TensorFlow logo

TensorFlow was originally developed by the Google Brain and was meant to only be used internally at Google. It was released to the public in 2015 and is an open-source library for dataflow that can be used for machine learning and neural networks. While TensorFlow is extremely powerful and widely used, it has a steep learning curve when compared to Keras and PyTorch and our team do not have the time to invest in learning the ins and outs of it within these two semesters.

Why Not Keras?

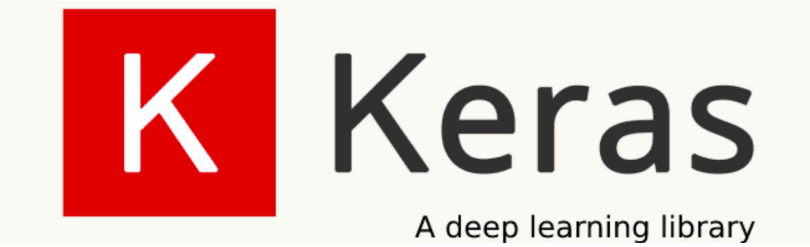

Figure 69 | The Keras logo

Keras is a neural-network library that was developed in Python and released in 2015. It is capable of being run on top of other libraries such as TensorFlow and is meant to make experimenting and getting results up and running faster than using TensorFlow. Keras is the most straightforward solution out of all the ones presented, however, it serves as more of an abstraction layer on top of Tensorflow. This severely reduces the flexibility of the API, and makes it quite convoluted to create truly custom models or algorithms. PyTorch, on the other hand, finds a nice balance between abstraction and customizability, falling in the middle of the spectrum between Tensorflow and Keras

Computer Vision Software

Why OpenCV?

Figure 70 | The OpenCV logo

Interacting with a real environment, rather than a simulation, can lead to a lot of potential problems when developing an intelligent system. A large part of the input data is supplied by the cameras for our project. Our autonomous control system needs to develop a comprehension of its environment mostly from 2d image data. OpenCV is an open source computer vision library that is able to work with tensorflow effectively. OpenCV will provide state of the art vision algorithms to detect / classify objects as well as object tracking and distance measures to assist in building an environment model.

Mesh Network Connectivity

Many issues need to be addressed before deployment when implementing a mesh network. What are the total distances between the robots we want to permit, how many devices we want to connect? Do we want to communicate to many robots with one host or do we want a group of robots to all be able to communicate with each other? Various technologies have various strengths and weaknesses, but none of them would be the ideal candidate. This section describes in detail the advantages and disadvantages of these technologies and identifies which one is the best for this application.

Why Wi-Fi?

Figure 71 | The Wi-Fi icon

Wireless Internet connection, which has been widely available since the turn of the century, has been a reliable, safe, and fast way to communicate with paired devices time and time again. For the EZ-RASSOR project, Wi-Fi is very relevant

since ROS already fully supports Wi-Fi connectivity. This simplifies the implementation of our mesh network and greatly decreases our workload.

Wi-Fi enables connectivity almost everywhere we go these days, from your house, to coffee shops. Even public places offer Wi-Fi through hotspots that can either be free or paid.

> The 802.11 standard provides several distinct radio frequencies to use in Wi-Fi communications. Each range is divided into multiple channels and regions provide rules and regulations on channels which set maximum power levels within these frequency ranges. We are lucky enough in Orlando to have premium frequency ranges which would offer us higher-than-average Wifi speeds. [32]

Wi-Fi offers a reliable connection between every member of the swarm and enables robots to interconnect amongst each other. Unfortunately, WiFi could pose challenges in the future due to some notable drawbacks. Wi-Fi has the possibility of consuming an excessive amount of battery when used as transceivers on board the EZ-RASSOR. Also, microcontrollers in charge of processing the logic of the robots are not very fast, and as such could not handle the high data rate and cause errors that we may not be able to predict. The team has plans in place that may eliminate the on-board processing issue, but we still have yet to find out the exact battery and capacity that will be used on the final robot.

Why Not Bluetooth?

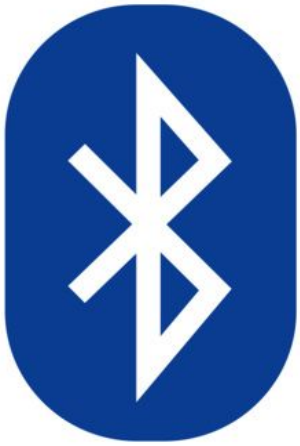

Figure 72 | The Bluetooth logo

Similar to Wi-Fi, Bluetooth has been a reliable and consistent solution for wirelessly connecting devices for quite some time. Most notable, it has been used for connecting wireless earphones, speakers, and more recently it has been used for connecting smart watches. While Bluetooth has many upsides to it, there are a couple issues that come up when considering it for our mesh network.

In general, Bluetooth hardware is made up of two main elements. The first component is the radio device. This device is in charge of sending and modulating the Bluetooth signal to and from itself to other radio devices. The second element is the digital controller. The digital controller is a custom CPU that is mainly in charge or running what is called a Link Controller. This Link Controller processes the signal and deals with FEC protocols. The Link Controller also deals with both synchronous and asynchronous transfer functions. The CPU is also in charge of any instructions the main device has.

The first main issue with Bluetooth is that it is not very secure.

> Bluetooth 2.1 and earlier did not require any form of data encryption. Additionally, if encryption was implemented, the encryption key is valid for only about twenty-four hours. This is because if more time is given, an attacker could apply an XOR attack to retrieve the encryption key. Later versions of Bluetooth address this security flaw, but even in recent years,

> Bluetooth has been criticized for severe security flaws. In April 2017 a set of Bluetooth vulnerabilities called “BlueBorne” have arose. These vulnerabilities would allow attackers to connect to devices without authentication and have full control over the device. [33]

The second issue with Bluetooth, and probably the more notable one, is its lack of support for mesh networks. Since the conception of Bluetooth, it has been meant to be used for one-on-one communication. This means there is one device that sends out a signal, and only one other device that receives it. Since our project needs to incorporate multiple devices all communicating with each other or with on decided on host device, this removes Bluetooth as a viable option for this project. Since the release of Bluetooth 3.0, Bluetooth has incorporated some limited support for mesh networking. But since it is in such early stages and only has support for a small range of devices, it would not be a good idea for use to incorporate Bluetooth into our project since we know Wi-Fi can get the job done much more reliably.

Programming Languages for On-Board Processes

Given that our project is running on an embedded system, one of the main aspects that affect how efficiently and practically the software runs is the means in which the software is implemented. Keeping this in mind, the obvious solution seems to be to stick to simple, bare-bones languages and toolings for all of the assets that are being stored and run locally on the EZ-RASSOR’s chip. This is why when deciding on the proper tools to use on the system with dedicated, domain-specific uses, the best course of action is to keep the use of heavy runtimes, libraries, and computationally expensive process to a minimum when possible.

The ROS architecture has support for hardware programming in two programming languages, C++ and Python. While keeping in mind the idea of focusing on being more efficient on the lower level, it makes sense to initially lean towards having the on-board EZ-RASSOR software be written in comparatively lower level and compiled languages. This made it seem like C++ was the better choice for the ROS module. However, while doing more research into the things that could bottleneck a system that already has a high complexity that needs to

make use of multiple toolings, and also considering the goal of the project and the nature of how long Senior Design students have to develop their projects, the choice to use Python instead of C++ made much more sense to make for this project.

Why Python?

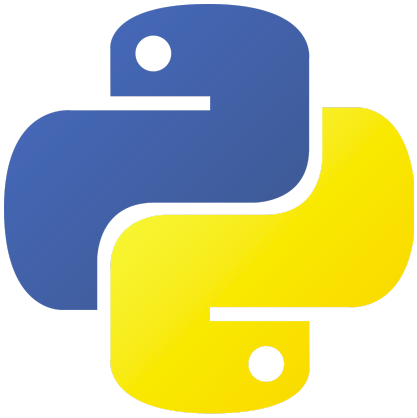

Figure 73 | The Python logo

A good portion of the code is going to be developed in Python and this decision has less to do with language's mechanics, such as the grammar and available tools, and more with the facility for designing an entire system using a wide variety of complex modules offered by the many resources and components that developer groups from many different fields can easily use. In other words, if additional complex modules that can be considered an entire project on their own needs are to be used to fulfill a specific requirement, it is likely we can find a well-packed open-source Python library. Probably due to the nature of Python as a language and its past, its ease of use and its flexible learning curve have made it easy for professionals as well as hobbyists to create a host of accessible plug and play components as the programming language has become more popular over the years.

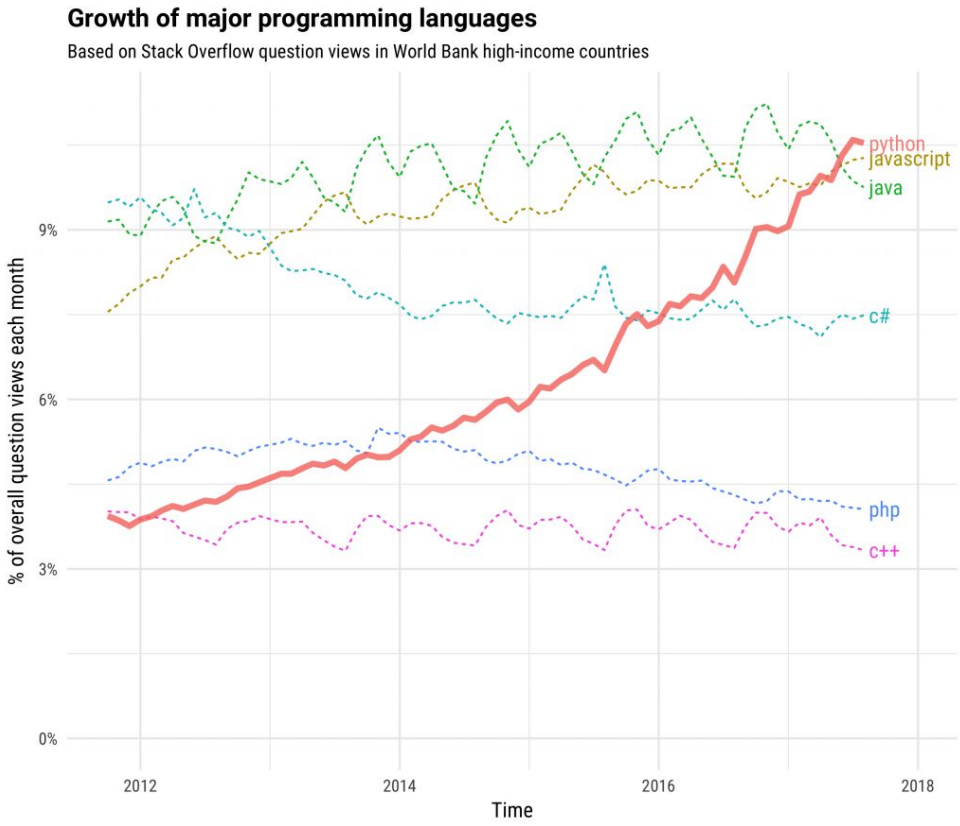

Figure 74 | Python's growth as shown via views on Stack Overflow questions, compared to other major languages

Given that Python is meant to be a general purpose programming language, it can be used in many different kinds of fields, which is what has helped cause the language to grow to much over the recent years. Since the EZ-RASSOR project is a project that has components that make use of different fields in computer science, making use of a general purpose programming language with a breadth of support for dedicated uses allows for groups to work closer together and makes it so there is less of a need for middlemen to communicate between components.

Having a strong community of developers that use and work with a programming language definitely help speed up the development of our project, as there is a wide range of resources that can be easily and freely accessed from a quickly growing and passionate community to significantly lower the time spent dealing with issues in development, creating custom modules from scratch, and deciding on the best courses of action to take when having to decide on a solution for a particular problem. The Python community is growing faster than almost any

other programming language in history, which is most likely due to how easy it is to use and how accessible it is. Based on data from the TIOBE Index, which looks at and analyzes the usage of programming languages and their popularity, Python beat out C++ and rose to the top three languages in popularity in their report released in September, 2018. The idea that Python might be considered one of, if not the, fastest growing programming language is also supported by Stack Overflow.

Even though there are a multitude of new technologies constantly being introduced to the world of software development, Python has shown extremely promising strength in what it is capable of by raising the bar for what it means for a programming language to grow. When comparing Python to other technologies that have been introduced in the recent years, Python's growth is much more than theirs.

Another thing to keep in mind is the fact that the development life-cycle of the EZ-RASSOR project is one that requires rapid prototyping, and given our teams familiarity with Python from previous projects, and development methods that work in perfect unison with how the language works enforces the idea that our team will have a much easier and more straightforward time using this language over another. There will be less time being spent taking glances at the documentation of the language, questioning why things are done a specific way with a given language, and fixing code, given that our team is much more familiar with Python than C++. On top of that, having a team-wide agreement on which methodology to use is much easier when using a programming language that more of the team is familiar with.

Why Bash Shell?

Figure 75 | The Bash Shell logo

Our team will use the Bourne Again Shell (Bash) as the primary shell for this project. Bash has a select few features that make it ideal for use in our Linux environments on both our development environment, as well as the environment running all the software on the physical EZ-RASSOR.

Firstly, what does a shell even do? A shell is essentially middleware to interface with the operating system of a computer. It provides access to many system calls, such as printing to stdout, writing to files, creating new files, etc. One of the most powerful features of the shell is that it can be used to write scripts, which simply stated is a collection of commands.

For this project specifically, we need the ability to tie together all of the elements of EZ-RASSOR together, such as ROS and any other Python programs that are required for operation. One of the most important features we need is for the shell to be easily installable and configurable by the end user. Given that this is an open source project, we'd like to keep everything as user-friendly as possible so others can freely expand upon the work done on EZ-RASSOR.

Bash is the ideal candidate for the aforementioned requirements for this project. Using Bash scripts, we can automate the installation process, operation of the EZ-RASSOR software, and any other system level task required. This means that our software will be both user friendly *and* consistent, as running one master

install script will be far better than having to install and run each component individually.

```
# Install ROS automatically with APT.
install_ros() {
    require "sudo" "apt" "apt-key"
    os_version="$1"
    ros_version="$2"
    key_server="$3"

    # Add the correct repository key to APT for ROS.
    echo_command="echo \"deb http://packages.ros.org/ros/ubuntu $os_version main\""
    ros_latest_dir="/etc/apt/sources.list.d/ros-latest.list"
    sudo sh -c "$echo_command > $ros_latest_dir"
    sudo apt-key adv --keyserver "$key_server" --recv-key "$RECV_KEY"
    sudo apt update

    # Install ROS and initialize rosdep.
    sudo apt install -y "ros-${ros_version}-ros-base" python-rosdep
    set +e
    sudo rosdep init
    set -e
    rosdep update

    # Source the ROS installation.
    source_setups_in_directory "$ROS_INSTALL_PARTIAL_DIR/$ros_version"
}
```

Figure 76 | A code snippet from the Bash script used to install all the software required for the EZ-RASSOR

Using these scripts will speed up the development process and help keep our installations consistent so the software runs as expected on any supported device.

Testing Plan

Testing the Path Planning

Given a premade, correct mapping of the area around our swarm intelligence, it should be able to produce a series of coordinates, which when followed in order, will lead to the pre-chosen dig site. To test this functionality, we will provide a map to our system and a series of dig sites and evaluate whether the sets of coordinates our system returns form a valid path, accounting for any obstacles.

Path Planning Test Cases

- Test Case 1
    - Purpose
        - Ensure the system produces a valid, mostly straight path for a dig site with no significant obstacles between it and the base of operations.
    - Test Description
        - Provide the path planning system with a map without any obstacles present.
    - Expected Result
        - Any path generated by the path planning system should lead directly to the dig site.
- Test Case 2
    - Purpose
        - Ensure the system is able to account for a large obstruction in the path to the dig site.
    - Test Description
        - Provide a map which contains one large obstruction between the base of operations and the dig site.
    - Expected Result
        - The path generated should guide the EZ-RASSOR completely around the obstruction and finish at the dig site.
- Test Case 3
    - Purpose

        - Ensure the system is able to path through an obstruction-dense area.
    - Test Description
        - Provide a map with a very narrow valid path through a very obstacle-dense area where it is more optimal to travel through the obstacles than around them.
    - Expected Result
        - The system should generate a path through the obstacle field, and create many waypoints for the EZ-RASSOR to follow.
- Test Case 4
    - Purpose
        - Ensure the system is able to path around an obstruction-dense area when it is more optimal to travel around it.
    - Test Description
        - Provide a map with a very narrow valid path through a very obstacle-dense area where it is more optimal to travel around obstacles than through them.
    - Expected Result
        - The system should generate a path around the obstacle field
- Test Case 5
    - Purpose
        - Ensure the system produces accurate short-length paths
    - Test Description
        - Provide a series of maps which test the same things as Test Cases 1-4, specifically over a very short range.
    - Expected Result
        - The system should generate paths which are described in the Expected Result section of Test Cases 1-4
- Test Case 6
    - Purpose
        - Ensure the system produces accurate medium-length paths
    - Test Description
        - Provide a series of maps which test the same things as Test Cases 1-4, specifically over a very short range.
    - Expected Result

        - The system should generate paths which are described in the Expected Result section of Test Cases 1-4
- Test Case 7
    - Purpose
        - Ensure the system produces accurate long-length paths
    - Test Description
        - Provide a series of maps which test the same things as Test Cases 1-4, specifically over a very short range.
    - Expected Result
        - The system should generate paths which are described in the Expected Result section of Test Cases 1-4

Testing the Scheduling

Given a set of information about the EZ-RASSOR rovers available to the scheduler, the scheduler should coordinate an efficient execution of the task provided to the swarm intelligence. This includes deciding which dig site(s) to send EZ-RASSORs to, tracking relevant information about the status of each EZ-RASSOR, and properly allotting charging time based on the current demand for EZ-RASSOR rovers and the batter level of each EZ-RASSOR which has returned to the base of operations.

Scheduling Test Cases

- Test Case 1
    - Purpose
        - Test whether the dispatch subsystem correctly dispatches rovers that are available to be dispatched to a single dig site
    - Test Description
        - Provide a simulated rover status database and allow the dispatch subsystem to run
    - Expected Result
        - The system shouldn’t choose to dispatch rovers unless there are available spots at the current dig site
- Test Case 2
    - Purpose

        - Test whether the dispatch subsystem correctly dispatches rovers that are available to be dispatched to a multiple dig sites
    - Test Description
        - Provide a simulated rover status database and multiple dig site locations and capacities, then allow the dispatch subsystem to run
    - Expected Result
        - The system shouldn't choose to dispatch rovers unless there are available spots at any dig site, and should be checking all dig sites for available spots
- Test Case 3
    - Purpose
        - Test whether the dispatch subsystem correctly receives lists of waypoints from the path planning system and transfers them to the relevant EZ-RASSOR
    - Test Description
        - Provide a simulated set of waypoints and a queue of rovers to be dispatched
    - Expected Result
        - The system should transfer the set of coordinates to the next rover in the queue
- Test Case 4
    - Purpose
        - Test whether the charging subsystem utilizes all of the charging stations available to it whenever possible
    - Test Description
        - Provide a list of rovers with relevant data for each one and run the charging subsystem
    - Expected Result
        - The system should assign rovers to each charging station, then track when stations will be opening up and assign new rovers to them
- Test Case 5
    - Purpose

        - Test whether the charging subsystem correctly prioritizes rovers to with high battery levels to charge in times of high demand
    - Test Description
        - Provide a list of rovers with varying current battery levels and set the high demand flag from the dispatch subsystem
    - Expected Result
        - The system should prioritize charging the highest battery level EZ-RASSORs to meet the current demands of the dispatch subsystem
- Test Case 6
    - Purpose
        - Test whether the charging subsystem correctly prioritizes rovers to with low battery levels to charge in times of low demand
    - Test Description
        - Provide a list of rovers with varying current battery levels and set the low demand flag from the dispatch subsystem
    - Expected Result
        - The system should prioritize charging the lowest battery level EZ-RASSORs since there is a surplus of rovers available to the dispatch subsystem
- Test Case 7
    - Purpose
        - Test whether the charging subsystem correctly allows rovers to skip charging if they have a battery level deemed safe to send on a mission
    - Test Description
        - Provide a simulated safe battery level and a EZ-RASSOR which has that or higher battery level
    - Expected Result
        - The system should send the EZ-RASSOR directly to the dispatch system
- Test Case 8
    - Purpose
        - Test whether the rover status subsystem accurately receives and updates all of the data which it is tracking

- Test Description
  - Provide an initial list of EZ-RASSOR rovers and preset data, then simulate updates to that data
- Expected Result
  - The data tracked by the rover status subsystem should accurately represent the updated data provided to it

Testing the Environment Mapping

The environment mapping system is tasked with creating, maintaining, and updating an environment map which will be utilized by the path planning system. Given a collection of elevation data, the environment mapping system should produce a new environment map. While the swarm control system is running, the environment mapping system should monitor the rover status subsystem for reports from the rovers about obstacles they encounter and update the environment map based on that.

Environment Mapping Test Cases

- Test Case 1
  - Purpose
    - Given elevation data about the moon, the environment mapping system should create a cost map in a form which can be utilized by the path planning system
  - Test Description
    - Provide the map generation functionality of the environment mapping system with elevation data from the moon
  - Expected Result
    - A connected graph with cost values representing the slope of the surface between those nodes
- Test Case 2
  - Purpose
    - Given elevation data about the moon, the environment mapping system should create a cost map and remove edges between nodes which have too high elevation change
  - Test Description

        - Provide the map generation functionality of the environment mapping system with elevation data from the moon
    - Expected Result
        - A connected graph with cost values representing the slope of the surface between two nodes with edges that represent too large of changes in slope removed from the graph
- Test Case 3
    - Purpose
        - To test whether the environment mapping system is able to update it's map based upon updates supplied to it by the rover status database
    - Test Description
        - Provide the map generation functionality of the environment mapping system with elevation data from the moon, then provide updates from the rover status database to the environment mapping system
    - Expected Result
        - A connected graph with cost values representing the slope of the surface between two nodes with edges that represent too large of changes in slope removed from the graph which is different from the graph originally created by the system, and accurately reflects the updates given by the rover status database

Testing the Efficiency of Different Quantities of EZ-RASSORs

One of the tasks for our project is to conduct tests to give NASA more insight into how many EZ-RASSOR rovers would be required to most effectively mine resources on the moon. So, once our swarm intelligence system is created and functioning, we need to simulate different amounts of EZ-RASSORs and analyze the data we collect from these tests.

Efficiency of Different Quantities of EZ-RASSORs Test Cases

- Test Case 1
    - Purpose
        - Test the effectiveness of a small amount of available EZ-RASSORs
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a small amount of rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation
- Test Case 2
    - Purpose
        - Test the effectiveness of a medium amount of available EZ-RASSORs
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a medium amount of rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation
- Test Case 3
    - Purpose
        - Test the effectiveness of a large amount of available EZ-RASSORs
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a large amount of rovers for the swarm intelligence to control

    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation
- Test Case 4
    - Purpose
        - Test the effectiveness of a small amount of available EZ-RASSORs and a larger amount of available charging space than Test Case 1
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a small amount of rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation and will be compared to the data collected in Test Case 1
- Test Case 5
    - Purpose
        - Test the effectiveness of a medium amount of available EZ-RASSORs and a larger amount of available charging space than Test Case 2
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a medium amount of rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation and will be compared to the data collected in Test Case 2
- Test Case 6
    - Purpose

        - Test the effectiveness of a large amount of available EZ-RASSORs and a larger amount of available charging space than Test Case 3
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate a large amount of rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith and terminating when the regolith is collected. Data about efficiency will be collected for further evaluation and will be compared to the data collected in Test Case 3

Testing the Overall Swarm Intelligence

Our final product we are creating for this project should function as a full swarm intelligence system, and should take as input an amount of regolith to collect and a list of EZ-RASSOR rovers available to it and efficiently and effectively divide up the work, accounting for many factors such as amount of charging stations, the environment between the base of operations and the dig site(s), and any differences between individual EZ-RASSOR rovers.

Overall Swarm Intelligence Test Cases

- Test Case 1
    - Purpose
        - Ensure the swarm intelligence collects the given amount of resources
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers in an intelligent way, efficiently collecting regolith up to the specified amount

- Test Case 2
    - Purpose
        - Ensure the swarm intelligence successfully terminates upon completion of the task
    - Test Description
        - Provide the swarm intelligence with an amount of resources to gather and simulate rovers for the swarm intelligence to control
    - Expected Result
        - The swarm intelligence should divide up the tasks among rovers to collect the regolith, then terminate when the regolith has been collected

Project Budget & Financing

Software

The majority of the project will likely be done using open source/free software such as PyTorch, Python, ROS, and OpenCV. With our current plans, developing the swarm AI system will require the use of machine learning methods such as reinforcement learning and other computer vision tasks (object detection, vehicle ego-motion, depth estimation, etc). This will likely introduce the need for powerful hardware in order to train algorithms capable of performing such tasks. The senior design labs provide machines with GPUs free of charge, which should suffice for this project. In the case where these are not available, there are several cloud-based GPU services we could utilize. Many of these provide free credits after creating an account and would likely provide enough computing power for the length of this project. Specifically, Microsoft Azure awards $200 worth of free credits after creating an account under an outlook email, and Google Compute provides $300 with a gmail. On top of this, several group members have GPU enabled machines which could function as a last resort. This project will also involve the implementation and use of a simulation capable of testing large hives of rovers. However, this simulation could be hosted on any of the options listed above.

Lastly, the requirements of last year's EZ-RASSOR project involved the creation and deployment of an iOS/Android app to serve as a manual controller for the system. For the previous team, this introduced a fee in order to publish the app on app stores. The EZ-RASSOR 2.0 gold team is under the impression that this was a one time thing and the app is available for use without any additional fees. In conclusion, as it stands, the software development portion of this project should not require a budget.

Hardware

Similar to last year's team, our team will be using an RC car kit from Sunfounder, with pre-existing Raspberry Pi, to create a sample RC car which will simulate the EZ-RASSOR's functionality. The kit is priced around $90, and some additional required parts will bring the total cost to around $130. We will make use of the Innovation Lab on campus for free 3D printing and acrylic cuts.

Item	**Price**
Sunfounder RC Car Kit	$90
Power Supply	$10
SD Card	$10
Batteries/misc.	$20
Total	**$130**

Facilities & Equipment

Orientation at Kennedy Space Center and SwampWorks

On October 18th 2019, both the Black and Gold teams made our first trip to Kennedy Space Center. We arrived around 8:30AM, and started our day there at the badging center. We got our temporary badges after some waiting, and we drove straight into Kennedy, to the SwampWorks building.

We were greeted by Mike Conroy, and some NASA engineers working at SwampWorks as we walked into the building and the lab. We were given an amazing opportunity to walk through the building and see everything at SwampWorks. SwampWorks is known to be an innovative working space that encourages collaboration and fast-paced research development, as we can see from the layout of the building with no walls and cubicles, and a lot of standing desks akin to many fast-growing start-ups that we see today-- something you don't usually expect from a government building.

We were shown the RASSOR, of which pictures were included earlier in this document. We were also blown away by the large glass box that was dubbed the "test bin". The test bin is a 625 sq. ft. testing bin that the RASSOR can be placed in and tasked with different tests with regolith. The test bin contained several tons of dust that is engineered to mimic lunar regolith. The team started with samples of lunar regolith brought by previous lunar exploration missions, and engineered the "artificial" version of the regolith to be similar to the real regolith samples, and recreated them using local components mixed together.

The morning continued with several NASA engineers at SwampWorks giving presentations about what they were doing, how they do things, and the impact of their work (hint: they are doing impactful work.). The presentations were informative and rather funny at times, which we all appreciated it. We proceeded to sign some documents and gave them to our sponsor Mike Conroy.

After the presentations, we were given the opportunity to see the [something] bay, which assembled several parts of the Apollo lunar missions. We then proceeded to see the famous Vehicle Assembly Building (VAB). We took some pictures to remind ourselves the massive scale of the building, and how awesome it looked. After taking pictures, we all went to lunch at one of Kennedy's cafeterias.

Figure 77 | The Vehicle Assembly Building (VAB) at NASA's Kennedy Space Center

In the afternoon, we returned to SwampWorks and had a chance to ask questions and discuss things with the engineers including our sponsor, Kurt Leucht. Kurt and the engineers provided us with a lot of crucial information pertaining to our project, and we wrote down all that information as much as possible. We then decided to hold subsequent weekly or bi-weekly meetings over teleconference before our trip came to an end.

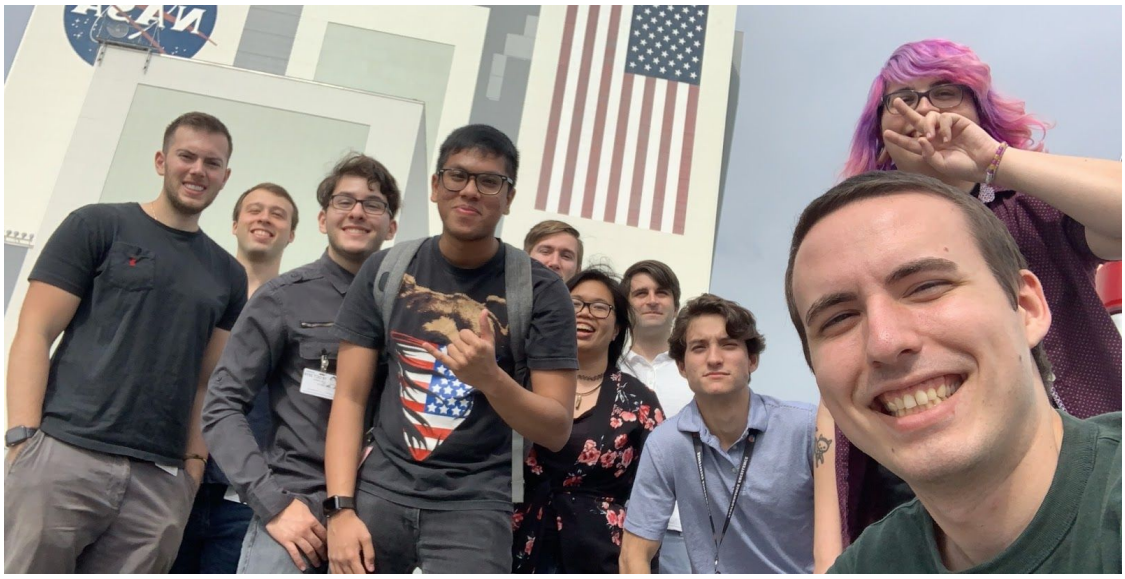

Figure 78 | A group picture of the Black and Gold teams

Teleconference over Microsoft Teams

After our initial Kennedy and SwampWorks visit late October, we decided to hold subsequent meetings with SwampWorks through teleconference, namely over Microsoft Teams.

Figure 79 | General information about Microsoft Teams

Microsoft Teams is a popular teleconference and meeting software that is used by various organizations around the world. Our meetings were scheduled weekly or bi-weekly depending on the availability of all parties involved, and the software proved to be reliable and presented no issues whatsoever.

Project Milestones

Date	Task	Comments
September 18th	Senior design teams assigned	
September 23rd	First meeting with Mike Conroy	Discussed project topics and opportunities
September 24th	Swapped a teammates based on project preference	Decided upon swarm control for our topic
September 27th	Second meeting with Mike Conroy	Discussed specific requirements and goals for the project
October 2nd	Project requirements document due	
October 4th	TA Check In	
October 18th	Initial meeting with NASA SwampWorks team	Refine objectives and focus of project
October 21st	Begin research in path finding and scheduling for multiple agents, as well as Gazebo simulations and environment mapping	
November 1st	Bi-Weekly tag up with Kurt Leucht	Discussed scheduling constraints surrounding dig site and rover battery capacity. Updated list of

		requirements to reflect Kurts input.
November 22nd	Second bi-weekly tag up with Kurt Leucht	
November 15th	Project outline and approach should be well defined	
November 27th	Each current member is finished with Onboarding document provided by previous team	
November 28th	Finalize research and refine and agree upon final approach	
December 5th	Final design document due	
December 16th	Begin work on improving the Gazebo simulation	Includes support for multiple agents, simulation of path finding and autonomous navigation, as well as the mining of regolith
December 16th	Begin work in creating an environment map based off NASA's topological map of the moon	Environment map will start off as a connected graph or grid map representation
December 30	Begin implementing path finding algorithms on arbitrary grid maps	
February 7th	Simulation should be	

	updated to support swarm AI and mining testing	
February 12th	Critical Design Review	
February 15th	Prototype of environment map should be complete	At this point, the map should be suitable for path finding and scheduling algorithms
February 30th	Complete prototype of multi agent path planning and scheduling algorithms	Algorithms should function properly on arbitrary, simple grid maps
March 1st	Unit testing of path finding, scheduling, and mapping systems	
March 5th	Begin integration of path finding systems with moon based environment map in the Gazebo simulation	Goal is to have functional path finding and scheduling for multiple rovers in an environment which simulates the moon's surface
March 27th	Full implementation completed.	Includes multi agent path finding and scheduling on a moon based environment map
March 27th	Integration testing of swarm AI system and environment map in the Gazebo simulation	
April 3rd	Finished cleaning up	

	code and adding final touches	
April 5th	Add completed code to EZ-RASSOR repo, along with accompanying documentation	
April 13th	Final presentation	

Conclusion

Overall, this project was an awesome undertaking for the whole team, and an opportunity for us to work alongside some of the most creative minds at NASA with the Swamp Works team. Swarm robotics is a fast-paced and quickly growing field in both research and application. In many ways, the problem our team is facing has never been tackled before. Related subtasks, sure, but the specific task of a swarm of rovers digging for resources on an extraterrestrial planet is unprecedented and poses a unique set of tasks, requirements, limitations, and goals. The work we put into this may not end up outside of Swamp Works labs, but the potential impacts it may have are limitless. With NASA's upcoming manned missions, the aptly named Artemis program, set to return to the moon by 2024, our ideas may live on the moon in less than a decade. EZ-RASSOR is almost a sort of pathfinder for the technology that will end up one day supporting a lunar colony. NASA's continual push for humans to not only travel to, but to thrive on other planets is an inspiration for all of humanity. Their dedication has also inspired the team to deliver the best product possible, not just for *the grade*, but also to make an impact on the future of space exploration.

Next Steps

Our team was able to accomplish all of our major goals and then some. The main stretch goal we didn't accomplish was having the map maintained by the pathfinding system updated by rover data. Implementing this would require updates to the rover status tracking system, specifically adding functionality where a rover could signal that it has encountered an obstacle. The pathfinding system could then update the map and pre-plan paths that avoid that obstacle.

To make the system more robust, more additions to the rover status tracking system could be implemented. Statistics such as battery efficiency specific to each rover, average speed, and average time to fill regolith drums could be tracked, then data analytics could be run to extract useful information. Possible insights include predicting part failure and making smarter decisions about when a rover is able to successfully complete a mission.

Project Terms and Definitions

Term	Definition
Environment Map	An environment map is a representation of the surface of a planet, in our case the moon, with respect to the surface elevation. An environment map can be 2D or 3D, and our initial environment map will likely be stored as a graph.
EZ-RASSOR	The EZ-RASSOR is the software which controls the MINI-RASSOR, which is the open source version of NASAs RASSOR rover. The RASSOR is a mining rover designed to mine regolith in low-gravity situations.
Gazebo	Gazebo is a simulation software which the EZ-RASSOR project uses to test the functionality of the EZ-RASSOR rover. It has full ROS support and the Senior Design team from Fall 2018 - Spring 2019 created a fully rigged model of the EZ-RASSOR for the Gazebo simulation environment.
Lunar Base	The Lunar Base is where the swarm control system our team is developing will be housed. This will be the central location and may be the home to astronauts, other rovers, along with charging and repair stations for EZ-RASSOR rovers.
Regolith	The terra on the surface of the moon. Regolith contains an abundance of useful materials and will be mined and harvested by the EZ-RASSOR rover. Oxygen, water, and material which can be used to 3D print can all be extracted from Regolith.
SwampWorks	Kennedy Space Center’s research and development team. This team is responsible for the creation of the RASSOR, as well as a number of additional significant

	innovations within the realm of aerospace.
Swarm Robotics	True swarm robotics is a coordinated group of robots, all in communication with each other, which divide up work to accomplish an overarching goal.

References

1. Mueller, Robert P., et al. “Regolith Advanced Surface Systems Operations Robot (RASSOR): Semantic Scholar.”, 1 Jan. 1970, https://www.semanticscholar.org/paper/Regolith-Advanced-Surface-Systems-Operations-Robot-Mueller-Cox/ad3e19575eda9ae498921fa49406c17d757b0c16
2. A. Hornung,. K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees" in *Autonomous Robots*, 2013; DOI: 10.1007/s10514-012-9321-0. Software available at http://octomap.github.com
3. Hart, P., Nilsson, N. and Raphael, B. (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions Systems Science and Cybernetics, 4, 100-107. http://dx.doi.org/10.1109/TSSC.1968.300136
4. N. Sturtevant. (2012) "Benchmarks for grid-based path finding" Transactions on Computational Intelligence and AI in Games
5. A. Patel. (2015) "Amit Patel's path finding notes" http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
6. Silver, D. (2005). Cooperative Pathfinding. In Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE), pp. 117–122.
7. Silver, D. (2006). Cooperative pathfinding. AI Programming Wisdom, 3, 99–111.
8. Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In AAAI/IAAI, Vol. 1, 530–535.
9. Zelinsky, A. 1992. A mobile robot exploration algorithm. IEEE Transactions on Robotics and Automation 8(6).
10. Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In AAAI/IAAI, Vol. 1, 530–535
11. Holte, R.C., T. Mkadmi, R.M. Zimmer, and A.J. MacDonald (1996), “Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach”. to appear in Artificial Intelligence.

12. Wang, et al. “MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees.” *ArXiv.org*, 16 Jan. 2014, arxiv.org/abs/1401.3905.
13. Wang, K.-H. C., & Botea, A. (2008). Fast and Memory-Efficient Multi-Agent Pathfinding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pp. 380–387.
14. Wang, K.-H. C., & Botea, A. (2009). Tractable Multi-Agent Path Planning on Grid Maps. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1870– 1875.
15. Wang, K.-H. C., & Botea, A. (2010). Scalable Multi-Agent Pathfinding on Grid Maps with Tractability and Completeness Guarantees. In Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 977–978.
16. Stern, Roni. “Multi-Agent Path Finding – An Overview.” *SpringerLink*, Springer, Cham, 1 Jan. 1970, link.springer.com/chapter/10.1007/978-3-030-33274-7_6.
17. Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided Deep Reinforcement Learning for Swarm Systems. *arXiv: 1709.06011*, 2017.
18. Wolfgang, et al. “Overview: A Hierarchical Framework for Plan Generation and Execution in Multi-Robot Systems.” *ArXiv.org*, 30 Mar. 2018, arxiv.org/abs/1804.00038.
19. H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hoenig, S. Kumar, T. Uras, H. Xu, C. Tovey and G. Sharon. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *Proceedings of the IJCAI-16 Workshop on Multi-Agent Path Finding,* 2016.
20. Multi-Agent Path Finding with Kinematic Constraints∗ Wolfgang Honig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian and Sven Koenig
21. M. Liu, H. Ma, J. Li and S. Koenig. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS),* pages (in print), 2019.
22. Tozour, P. (2002). Building a Near-Optimal Navigation Mesh. In Rabin, S. (Ed.), AI Game Programming Wisdom, pp. 171–185. Charles River Media
23. Rabin, S. (2000). A* Speed Optimizations. In Deloura, M. (Ed.), Game Programming Gems, pp. 272–287. Charles River Media.

24. Samet, H. (1988). An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures. NATO ASI Series, Vol. F40.
25. “SVS: CGI Moon Kit.” *NASA*, NASA, https://svs.gsfc.nasa.gov/4720
26. *Orbital Data Explorer User's Manual*, https://ode.rsl.wustl.edu/moon/pagehelp/quickstartguide/index.html
27. “High Resolution Topographic Map of the Moon – Moon: NASA Science.” *NASA*, NASA, https://moon.nasa.gov/resources/87/high-resolution-topographic-map-of-the-moon/
28. *PDS Geosciences Node, Washington University, St. Louis, Missouri*, https://pds-geosciences.wustl.edu/lro/lro-l-lola-3-rdr-v1/lrolol_1xxx/document/sldem2015.txt
29. *PDS Geosciences Node, Washington University, St. Louis, Missouri*, pds-geosciences.wustl.edu/lro/lro-l-lola-3-rdr-v1/lrolol_1xxx/document/sldem2015.txt.
30. Deb, Sayantini. “Keras vs TensorFlow vs PyTorch: Deep Learning Frameworks.” *Edureka*, 27 Nov. 2019, www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/
31. Jain, Yashwardhan. “Tensorflow or PyTorch : The Force Is Strong with Which One?” *Medium*, Udacity India, 14 Nov. 2018, medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4
32. Shi, G. & Keqiu. (2017). Signal interference in WiFi and ZigBee networks. Cham, Switzerland: Springer.
33. Labiod, H., Afifi, H. & Santis, C. (2007). Wi-Fi, Bluetooth, Zigbee and WiMAX. Dordrecht London: Springer.
34. Prakash, Abhishek, and Abhishek PrakashI. “How to Install Ubuntu Linux on VirtualBox on Windows 10 [Step by Step Guide].” *It's FOSS*, 14 Aug. 2019, https://itsfoss.com/install-linux-in-virtualbox/
35. Evangelho, Jason. “Beginner's Guide: How To Install Ubuntu Linux 18.04 LTS.” *Forbes*, Forbes Magazine, 25 July 2019,

https://www.forbes.com/sites/jasonevangelho/2018/08/29/beginners-guide-how-to-install-ubuntu-linux/#788b9ebd951c

36. Mainprice, Jim & Sisbot, Emrah & Jaillet, Léonard & Cortés, Juan & Alami, Rachid & Siméon, Thierry. (2011). Planning human-aware motions using a sampling-based costmap planner. Proceedings - IEEE International Conference on Robotics and Automation. 5012 - 5017. 10.1109/ICRA.2011.5980048.