University of Arkansas – CSCE Department
Capstone II – Final Report – Spring 2023

NASA Robot Mining Competition

Ahmed Moustafa, Rohit Kala, Justin Kilgo, Jackson Newman, and Jackson Burger

Abstract

At the current growth in population, we will soon exceed the planet's non-renewable resource availability. This vast difference between demand and supply necessitates a new source for scarce minerals: space. As we explore the final frontier, we face a critical challenge: how to extract the minerals we need to sustain our civilization without destroying our planet. For these extraterrestrial mining expeditions, mankind must apply advanced technology to develop lunar robots to kickstart the era of space mining. The objective of this project is to engineer and program a robotic lunar excavator for the NASA Lunabotics competition that will maneuver through complex, rough terrain and mine regolith simulants. As the CSCE section of the NASA Robot Mining Competition Capstone team, our focus is on the software and computer system aspects of the project through simulation and physical testing to achieve semi-autonomous function. Our team will take a collaborative approach, leveraging the diverse skills and expertise of team members from multiple departments. We will use an agile development methodology to iteratively design, build, and test our excavator, and will use ROS 2 to integrate our software and hardware components. By using a combination of simulation and physical testing, we will optimize our robot's performance and ensure that it meets the demands of the Lunabotics competition. A key aspect of the maneuverability of the robot is position-tracking which will use a combination of accelerometer measurements and camera object detection. With a dataset of possible field elements and regoliths, we may train a model using modern object detection libraries to facilitate autonomous navigation so the robot can safely mine. This project's significance lies in both the direct cross-department learning opportunities as well as the general impact this field of technology will have on our planet's future. Working in unison with the University of Arkansas' Razorbotz team highlights the significance of multifaceted engineering projects and the breadth of knowledge required to reach an acceptable product. The robot and its programming we plan to produce are the foundation for future Lunar and Martian mining expeditions to help alleviate Earth's resource exhaustion and environmental damage from current resource extraction.

1.0 Problem

At the rate at which Earth's population is consuming non-renewable resources, mankind will not be able to sustain its growing population [18]. One of the options mankind has been looking into is searching for resources off-planet. A prime candidate due to its proximity and signs of resources such as water and Helium-3 is the moon which led NASA to create the Artemis program [19]. NASA's Artemis program plans to establish the first human presence on the moon since the 1972 Apollo 17 mission. These plans include the development of a lunar base camp which would possibly be built in Shackleton Crater due to its suspected access to ice and minerals. There are many concerns with the Artemis missions, especially surrounding the sustainable deployment of humans to the moon. It is both dangerous and expensive to send people to the moon with each pound of commercial cargo costing anywhere from $9,100 to $43,180 [20]. This is one of the main reasons behind the development of and demand for precursor lunar robots. To facilitate human travel to the moon, Mars, and other extra-terrestrial sites, we need to engineer efficient robots that can accomplish the pre-requisite task of mining and research. This is where the problem that our project, the Lunabotics competition, and the NASA Artemis program all hope to address arises.

Lunar vehicles have been in the works since the 1950s and experienced a huge increase in development in the 1960s thanks to the Space Race between the US and the USSR during the Cold War. As NASA went through many iterations of lunar robots, the expectations rose from travel and navigation to mining, on-site research, and continuous exploration. A key problem with deploying robots in space is their dependence on manual control. For a radio signal to reach the moon, it can take several seconds which does not seem like much but to reach a robot on Mars from Earth, it can take around 7 minutes [21]. Because of this time delay, human control would be difficult and possibly dangerous for the robot since any decisions/movements made from Earth would be anywhere from 7-20 minutes delayed. This is where the development of robust autonomy for lunar and martian robots becomes crucial. As such, NASA's Artemis Student Challenge was created to be a college-level competition where teams must engineer the most efficient robot that can reliably navigate and mine the simulated lunar environments.

The importance of developing an efficient semi-autonomous robot for space mining and exploration grows with every second. This challenge is something that NASA has been trying to solve for several decades and mankind is rapidly approaching a deadline. The impact of not solving this problem of efficient precursor lunar robots is the exhaustion of resources on Earth and the growth in environmental risks of our current on-planet mining. Our current mining processes for rare-earth metals are causing contamination of air, water, and soil due to leaching chemicals and toxic waste [22]. If mankind can't rapidly reduce its consumption of the resources in question and the environmental consequences of their extraction, we will only be digging our graves. To combat our growing population and resource demands, off-planet mining becomes almost necessary and so the challenge of researching and developing the most efficient and inexpensive mining robots is imperative.

2.0 Objective

The primary objective of this project is to design, engineer, and program a semi-autonomous robotic lunar excavator for the NASA Lunabotics competition that can effectively maneuver through complex, rough terrain and extract regolith simulants with high accuracy and efficiency. Specifically, we will develop a multifaceted control schema that combines autonomous and manual functionalities. Our team will achieve this objective through the integration of advanced sensors, robust software, and hardware components that enable the excavator to detect, navigate, and excavate the regolith safely and reliably.

3.0 Background

3.1 Key Concepts

3.1.1 Object Detection

The latest advancement in artificial intelligence has allowed us to solve previously difficult problems. One such task is object detection, which can be described as the task of detecting individual instances of various objects found within an image or a video and classifying them correctly [12]. It is one of the most pertinent challenges in computer vision, as many problems in the field require object detection to accomplish various other tasks such as pose estimation, image segmentation, etc. There are a lot of factors that have allowed us to finally start solving this task. Firstly, hardware has improved to the point where we can implement multilayer perceptrons, which can be thought of as neurons represented in a computer. Multilayer perceptrons were first theorized during the 1950s [13], however, it was impossible to implement them efficiently with the technology at the time. With the rapid improvement of technology during the 21st century, we've finally achieved implementation. The reason why perceptrons are so important is that we can solve problems without having to explicitly code for them. Rather, we use data that we know the results of and allow the computer to come up with a function or a model so that it can solve this problem on its own. This is because a perceptron will fire if the inputs scaled by a weight manage to exceed a threshold amount [14]. The computer can solve problems based on the perceptrons that fire, thereby creating solutions to a new set of problems that were previously hard or impossible to solve. A model might take many perceptrons to solve harder tasks, and they are organized into layers, where each perceptron takes input from the previous layer [14]. A crucial part of using such models is to have a dataset that it can run over so that the model can optimize the amount by which the inputs are scaled such that the model outputs the correct result. Each incorrect answer will further optimize the model by shifting the scaling factor by a small amount, eventually converging at an optimal value so that the model is accurate for the training data [14]. It is also important to test the model on randomized data to ensure that the model can accurately predict the output to various kinds of input rather than overfitting to the training dataset [15].

While multilayer perceptrons are very good at solving rudimentary tasks, we will need something more complex to solve tasks such as object detection efficiently and accurately. This is where the idea of a Convolutional Neural Network comes in. Essentially, we will have certain layers in the network which will convolve a kernel matrix with portions of the input data. These layers will extract the various features of the input, therefore, allowing the model to classify these features. In these layers, the kernel matrix is optimized so that the model extracts the features correctly [16]. This method is commonly used in object detection and implementations often try to maximize speed or precision. Oftentimes, these are inversely related as faster models tend to be less precise. A primary drawback of these models is that they require lots of data to predict the answer accurately. Additionally, training some of the more complex models takes a long time and requires lots of calculations. One of the best datasets for the object detection task is Microsoft’s COCO dataset, which is a joint project by many companies and universities to classify various objects in everyday scenes that anyone can use to train a model [17]. A popular library to implement these Deep Learning concepts is PyTorch [9].

3.1.2 Client-Server Communication

Client-server communication is commonly used when one system cannot access the resources of another system. It allows a system to share necessary information with another so that each system does not have to store duplicate information. In this relationship, there is a client and a server as the name implies. The client is primarily responsible for requesting and receiving information, while the server is responsible for handling this request and responding with the appropriate information. One way this is accomplished is through the Transmission Control Protocol (TCP) [23]. This allows the client and server to first establish a connection before transmitting data to ensure the information requested is sent and received to the correct destination. First, the client sends an initial request to begin this connection. Second, the server acknowledges the request so that the client knows a connection is being established. Next, the client sends an acknowledgment of the connection so that the server knows to begin transmitting the data. TCP connections may persist for future communication, however, they are commonly terminated due to the number of requests many servers receive. In the case where the connection persists, information can continue to flow from the server to the client with no interruptions. In the case where the connection is terminated, the connection must be reestablished later before more data may be transmitted [24]. The application of this Client-Server model is useful in many multi-system technical methods such as transmitting control signals for motors or publishing images from a camera.

3.1.3 Controller Area Network (CAN bus)

The Controller Area Network (CAN bus) is a communication protocol that allows multiple electronic devices to exchange data in real-time, even in challenging environments. It utilizes a two-wire system with a resistor on each end (Figure 1) that facilitates simultaneous communication among devices on the network using dominant and recessive signals.

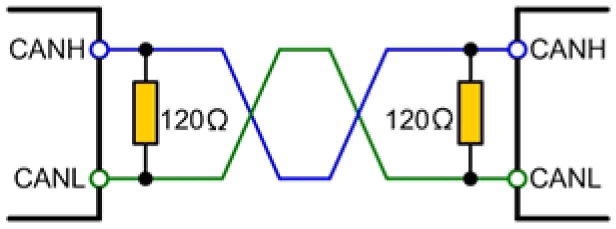


Figure 1 - CAN bus

Data transmission is carried out using a message-based system, with each message comprising a unique identifier and up to eight bytes of information. The identifier determines message priority, ensuring that those with higher priority are transmitted first and given more bandwidth, making efficient and effective control of various subsystems critical in managing complex electronic systems. The CAN bus is distinguished by its sophisticated error detection and correction mechanisms. The cyclic redundancy check (CRC) mechanism verifies data integrity by identifying and rejecting errors and notifying the sender to resend the message. This feature significantly reduces the likelihood of data corruption, improving electronic systems' overall reliability and safety. The CAN bus is widely used in modern vehicles and industrial automation systems, enabling real-time subsystem communication. Its resilience, error detection and correction capabilities, and priority-based message transmission make it a dependable solution for managing complex electronic systems. The CAN bus enhances safety and performance for various applications by transmitting data quickly and precisely [29].

3.1.4 ROS 2

Robot Operating System 2 (ROS 2) is an immensely powerful framework for building distributed robotic systems. One of its core features is the ROS graph, which organizes the various data processing elements that make up the robot's operating system. The ROS nodes are at the heart of the ROS graph, representing individual executables responsible for specific modular purposes. Nodes communicate with each other through a variety of mechanisms, including topics, services, actions, and parameters. Topics are a convenient way to transfer data between nodes in a ROS 2 system. They operate on a publisher-subscriber policy, allowing nodes to publish and subscribe to data on a shared topic. For instance, a temperature sensor node can publish temperature data to a topic named "temperature," while other nodes subscribe to the same topic to receive the temperature data. Topics can have multiple publishers and subscribers, making sharing information across the ROS graph simple. Another mechanism for data transfer in ROS 2 is services, which operate on a call-and-response model. Unlike topics, services have only one node acting as the server, while multiple nodes can be clients. For example, a service node can provide data only when a client node requests. This model is similar to the client-server communication described earlier. ROS 2 also supports actions and parameters, which provide additional flexibility for building distributed robotic systems. Actions allow nodes to perform long-running goals in a non-blocking way, while parameters provide a mechanism for storing and sharing data between nodes. By leveraging these various mechanisms, ROS 2 enables developers to build complex, sophisticated robotic systems [11].

3.2 Related Work

3.2.1 Servi Robots

Service robots are becoming increasingly more common with the advancement of artificial intelligence. The robot which we are programming, at a high level, should be able to drive itself to a location while avoiding obstacles to do a certain task, and then return to where it started. This is closely related to a recent development from Bear Robotics partnered with Softbank Robotics. They created a service robot, called Servi, which autonomously delivers food to customers in a restaurant as well as busses tables [1]. This Robot as a Service (RaaS) allows businesses to replace existing employees with a much cheaper and seemingly more reliable option, or add the robot to their staff. This is related to the robot which we are programming because Servi, at a high level, drives itself to a table, does some task whether it be to serve customers or bus the table, and drives itself back to where it started all while avoiding obstacles.

Although this is a good solution to the inconvenience that having a human employee can cause, it is not a perfect solution. The Servi robot is simply a moving tray. Some human intervention is still needed for it to operate how it was intended to. People still need to load and unload it, and while bussing is an ability that Bear and Softbank are advertising it can do, it can only hold the dishes, not pick them up. The robot we are programming should be able to do all of its tasks without any human intervention. When it gets to its location, it will dig up rocks and store them without needing the help of humans.

Autonomous robots will likely replace many humans in a lot of different fields of work. They have great benefits for many industries due to their productivity and precision. With a precise robot, there are next to no human errors that can occur which would decrease efficiency or set back a company. Transporting goods is the most relevant task where we can see the potential of autonomous robots. Not only can the robot be a solution for human error, but it can also result in more safety for humans. With regards to having an autonomous robot mine on worlds other than Earth, the benefits of autonomous robots become even more apparent. Instead of having a human go into space and mine by hand, NASA can send a robot or even a group of robots to do the same thing faster without the risk of the human being harmed or becoming fatigued and less efficient. There are some obvious concerns when replacing humans with robots. There is always the possibility that the robot does not work the way it was intended to. If this should happen, then it can cost a company a potentially very large amount of money, it can slow down progress, or it can even put humans at risk if they are working around the robot if it were to fail. The solution to this is to test the robot extensively before and during production, implement safety features that can immediately stop a robot so a potential problem can be investigated, constantly search for ways of improving the robot, and study the robot's performance closely.

3.2.2 Roomba

Autonomous functionality in robots is nothing new. For example, Roomba by iRobot has been in many homes across America since 2002 [2]. While it is much smaller, it utilizes many technologies that will need to be considered for the lunabot, primarily calculating location and path using infrared emitters and cameras. While the Roomba is running, it is mapping out the area and uses this information to plan efficient cleaning courses and remember where the robot has been. This allows it to run for hours since it can automatically return to its home dock and recharge when the battery is low, and then return to the location and path it was on before the battery was low [4]. While the lunabot will not solely use a camera to understand its location, it is a large source of its direction and enables object detection. We will need to use the camera in a similar way to avoid rocks and potholes while still making progress toward our goal. However, we need to identify and excavate regolith whereas the Roomba is focused solely on the navigation of its field. This difference in task complexity signifies the additional challenge in the autonomous development of our robot. Also, we do not have the luxury of moving around the map and bumping into things to map out the location like the Roomba, we must be able to navigate it our first time through and avoid hitting things to prevent potential damage to the robot. Luckily, we can make use of a more powerful camera and more advanced frameworks for the robotics competition, but that does not cancel out the additional requirements that we are held to.

4.0 Approach/Design

4.1 Requirements

The competition has a strict set of rules for the robot that must be followed. These rules are the set of requirements that goes as follows ([3], page 29-30):

- The robot, before moving, must be contained within the dimensions: 1.1m length x 0.6m width x 0.6m height. After it begins, it may expand beyond these dimensions, but can not exceed 1.5m in height.
- The robot must not be any heavier than 80kg.
- There must be at least 4 points which humans can lift the robot by. These points must be clearly marked (ISO 7000-1368) so the robot can be safely moved.
- The robot can be launched in any orientation. The axes X, Y. and Z correspond to length, width, and height and must be declared to the inspection judge.
- The robot must have a Reference Point Arrow marked on it which points in the direction that the robot should begin moving in before it starts. This arrow does not indicate what direction the robot should always move in.
- Subsystems which are attached to the robot that transmit commands, data, or video will be counted toward the final weight. Equipment used for these purposes that are not attached to the robot will not be counted in the final weight.
- Multiple robots are allowed, but must comply with the dimension and weight specifications as 1 unit.
- The robot may be controlled either with a remote control or autonomously.
- Touch sensors are not allowed.
- The robot must be equipped with a "Kill Switch". This Kill Switch must be a red, easily accessible emergency stop button which satisfies the following conditions: It must be implemented in a way that it uses trusted engineering practices and principles. It must have at least a 40mm diameter. It must be placed on a surface that is easy to access and requires no additional steps to get to. Only 1 Kill Switch may be placed on each robot. Disabling the Kill Switch without authorization from the staff will get the team disqualified. The Kill Switch must immediately disable the robot after 1 press of the button. It must be extremely reliable. Due to these reasons, a Commercial Off-The-Shelf red button is required. A closed control signal to a mechanical relay is allowed if it stays open to disable the robot. The button should disconnect the batteries from all controllers and it should isolate the batteries from the rest of the active sub-systems. If a system is powered by its own, independent, internal computer battery, then it may stay powered on in the event of the Kill Switch being pressed.
- The robot must provide its own power. No power from the facility will be able to be used for the robot during the attempt. The robot can use as much power as it wants, there is no limit. The energy used by the robot must be recorded with an Commercial Off-The-Shelf logging device. The energy used must immediately be shown to the judges after the attempt.
- The robot can not use any processes, gasses, fluids, or any consumables that would not work at a place other than Earth. Something like a dust wiper controlled from Earth that

operates on the robot is acceptable. Closed pneumatic systems can be used by the robot if the gas is supplied by the robot.

- Due to limited budgets, the robot does not have to use equipment that is able to be used off-world, but only if the components can be swapped for off-world environment rated equipment. Examples of equipment that the robot cannot use are: GPS, rubber pneumatic tires, air/foam filled tires, open/closed cell foam, ultrasonic proximity sensors, or hydraulics due to the fact that NASA cannot anticipate these to be used in an off-world mission.

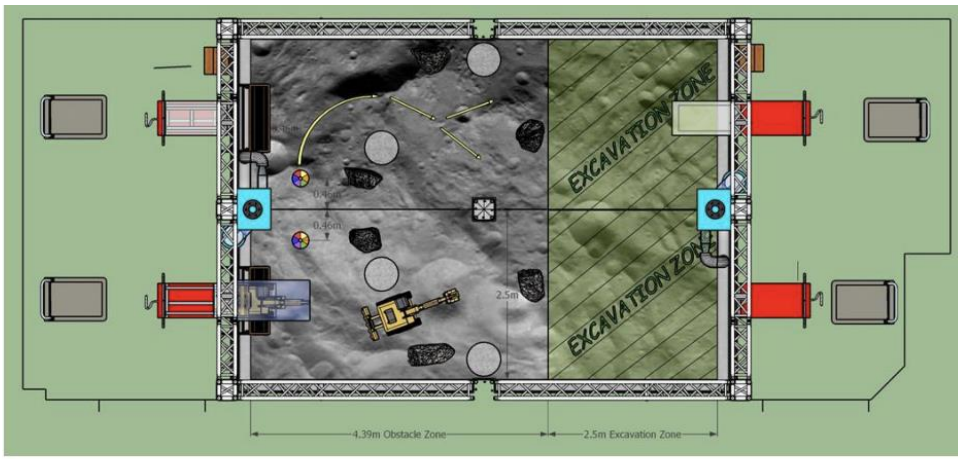


Figure 2 - Mining Arena ([3], page 32)

4.2 Use Cases

Use Case #1: Gather resources from dangerous/inaccessible terrain
Author: Jackson Newman
Primary Actor: NASA researcher/Robot supervisor
Goal in Context: To gather information based on samples that the robot retrieves or to use the resources that are found outside of Earth or in extreme environments on Earth.
Preconditions: Regardless of where the robot is mining, the robot supervisor must be well-educated and prepared to ensure the robot's mission is successful. If the robot is going into space, there must be proper extra-terrestrial transportation and the researchers must be well-equipped to conduct useful research with the data.
Trigger: There is a demand for resources from dangerous environments or off-planet. If from space, NASA may request for research to be done on certain resources or environments to prepare for human space travel.

Scenario:

1. If the mission is for space, NASA sends the robot to another planet.
2. The robot is supervised whilst autonomously moving through the terrain and gathering resources.
3. Once the robot has gathered a satisfactory amount of resources, it and its harvest return from the possibly dangerous or off-planet dig site.
4. The resources are then studied or used.

Exceptions:

1. The robot becomes obsolete because other research or resource extraction surpasses the robot's importance, value, and/or simplicity.
2. The regions are too difficult to reach or the resources simply don't exist in feasible quantities of nearby terrain/planets.
3. The resource may not be necessary enough to justify spending the money to send the robot on a trip to collect it.

4.3 Detailed Architecture

We transitioned from the planning and preparation phase to the production phase this semester. Furthermore, we deployed the knowledge gained through communication with our CSCE mentor William Burroughs and the Mechanical Engineering sub-team to adapt our goals. Specifically, the plans of starting this semester with robot control code development and unit tests on previous years' movement logic had to be postponed due to unexpected delays from the Mechanical Engineering sub-team. Rather than freezing our task schedule, we opted for restructuring our AGILE sprint plans to work on elements of the project that didn't depend on robot completion. This meant starting on prerequisites for the autonomous code such as position tracking, video streaming, and motor controller driver software. Besides enabling autonomous software development further down the line, moving these tasks up allowed us to hasten some aspects of manual control as the robot's completion caught up to the software. Near the start of March, the Mechanical Engineers had completed the basic chassis of the robot (Figure 3) which enables more movement logic-related tasks.

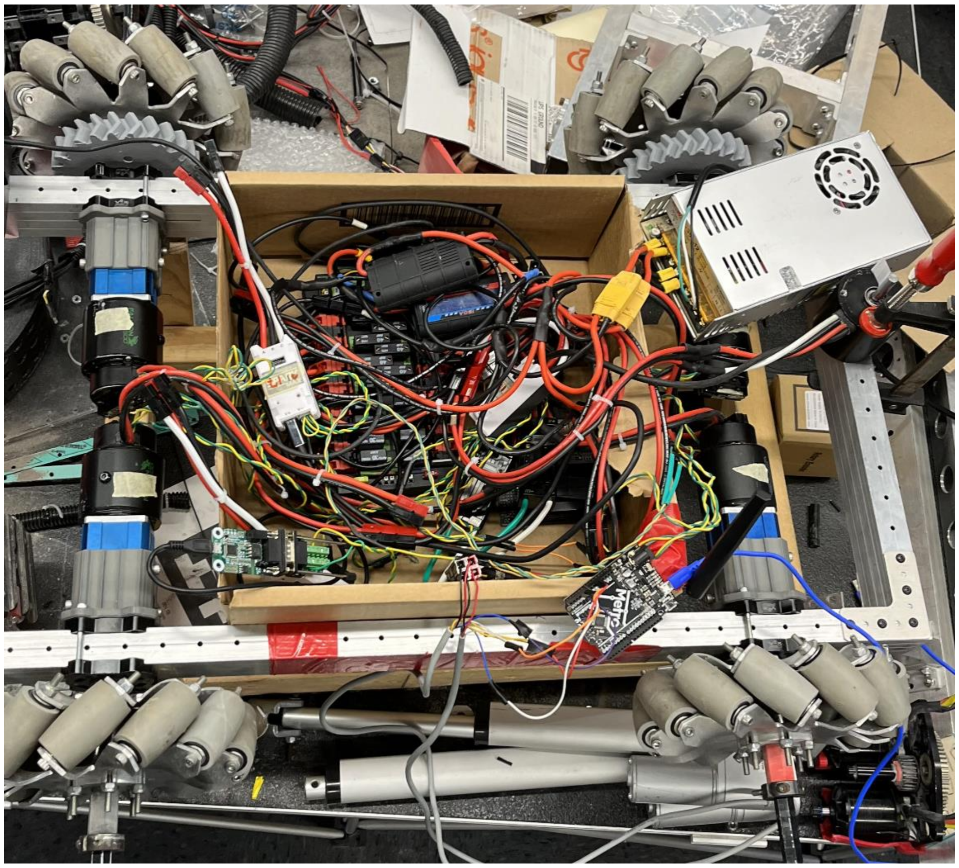


Figure 3 - Robot Chassis (as of 3/01/2023)

A key aspect of our progress this semester has been splitting the manual control and autonomous aspects of the robot apart and finding the overlapping elements. Before writing any new code, we first analyzed previous years' code and updated user documentation based on the context we got from William and the Mechanical Engineers. An example is the change in the robot's excavation system from a flywheel bucket to a conveyor bucket which has many implications for the ROS 2 nodes that need to be created. After reviewing and updating previous years' code, we were able to effectively summarize the node breakdown and architecture of the software design behind the Lunabotics excavator.

The Communication node is the most crucial and central element of the Lunabotics excavator's software architecture. As a ROS 2 node, it takes in data from other nodes and publishes data through *rclcpp* APIs to an established socket, allowing the robot to move according to user input. The Communication node's inputs are typically integers or floats, which it converts to hexadecimal to standardize the published output. Its main job is to set up the connection to the socket address and continuously run its processes until the *rclcpp::ok* condition becomes false due to an error or the robot being turned off. However, the Communication node cannot function alone. It relies on other nodes to provide it with data, including the Client node and publishers such as Logic, Excavation, Falcon 1 and 2, Talon 1 and 2, Zed, and the Power Distribution Panel nodes.

4.3.1 Manual Control

Since a significant portion of the competition's score is based on the robot's ability to excavate using manual control, we have made it our priority to ensure the robot is fully functional as soon as possible. Therefore, we have prioritized aspects of manual control that are prerequisites to autonomy. Despite not having kinematic simulations or a physical robot two months into the semester, we have focused on software development for individual components such as motors, motor controllers, cameras, and data publishing/receiving. These tasks impact multiple nodes within our software architecture and their connection with the Communication node. To ensure effective manual control, we have defined the responsibilities and details of each node and how they are set up. By doing so, we can enable efficient manual control and prepare for autonomy development later in the project.

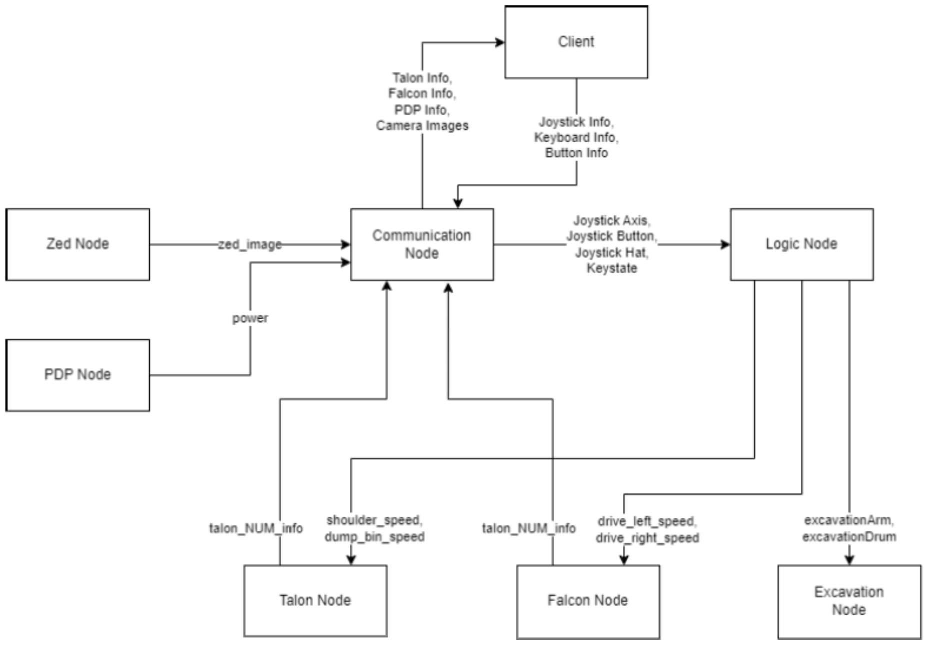


Figure 4 - Flowchart of Nodes

Figure 4 above presents a flowchart that outlines the interactions between nodes relevant to manual control. The Client is responsible for transmitting joystick information such as key states, button presses, and throttle to the Communication node as well as receiving and processing motor, power, and image data. A competition detail that impacts our design is the score deduction for publishing data to the Client during the actual competition. As such, we have a *silent run* mode which will be enabled outside of testing. Further down the chain is the Zed node, a crucial element for both automation and manual control, which is responsible for publishing images from our Zed2i camera to calculate the robot's position and detect obstacles. The Power Distribution Panel (PDP) node is responsible for reading current and voltage measurements, as well as controlling the state of individual power outlets. By publishing our power-related data to the Communication node, we are able to monitor each electric component of the robot through its Controller Area Network (CAN bus) id.

The Logic node is a crucial component of the robot's control system responsible for various critical functionalities, such as updating the wheel and excavation motor speeds. It receives real-time information from the Communication node, including the joystick axis, joystick button, joystick hat, key state, and zed camera position. Based on this information, the Logic node computes necessary adjustments to the coordinate planes and throttle for both the right and left wheels to update the wheel speeds. In addition, the Logic node can reset the robot's speed to zero when needed. The excavation system is also controlled by the Logic node, which adjusts the shoulder speed, arm speed, and drum speed based on the published data from the Communication node. This enables the excavation system to operate smoothly and accurately. The Logic node's ability to execute different states of the robot based on joystick buttons is another vital aspect. For instance, button two toggles between the drive and excavation state, while button three inverts the direction of the drum. Such functionalities provide greater control over the robot, allowing it to perform various tasks efficiently. To ensure that the Logic node operates as expected, we plan to create unit tests that simulate different control scenarios. These unit tests will help to identify any potential issues and improve the overall reliability and risk mitigation of the robot. With proper testing and validation, the Logic node will operate smoothly and accurately, making the robot an efficient and reliable tool.

The Falcon node is responsible for controlling the Talon FX motor controllers and setting the speeds of the Falcon motors attached to them. These motor controllers are integrated into the Falcon 500 motors and feature built-in drivers that enable us to easily control them using APIs in our code. Additionally, these drivers automatically convert control signals into the CAN bus format for transmission. The Falcon node receives input for *drive_left* and *drive_right* speeds, which it then utilizes to execute precise turns or maintain a straight path by adjusting the speed of the drivetrain motors accordingly. The Falcon node receives information published by the Logic node, processes it, and transforms it into movement commands that are executed by the Talon controllers. This node also publishes *talon_NUM_info* to the Communication node, which provides real-time updates on the motor controller status.

The Talon node is responsible for controlling the Talon SRX motor controllers that regulate the movement of the linear actuators responsible for vertical motion and height control of the excavation apparatus and bucket system respectively. Although not built-in, the motor controllers come equipped with APIs, and the Talon node leverages these APIs to automatically convert control signals to the CAN bus format. The node receives *shoulder_speed* and *dump_bin_speed* values from the Logic node to control the motors responsible for moving the shoulder and dump bin. It publishes *talon_NUM_info* to the Communication node to keep the entire system updated on motor performance. Upon receiving the data from the Logic node, the Talon node translates the information into movement by the motor controlled by the Talon instance.

The Excavation node is a crucial component of the robot's manual control and autonomous functionalities. It enables us to control the excavation system via the controller and call an excavation macro within the autonomous section. Last year, the Excavation node received *excavationArm* and *excavationDrum* data from the Logic Node. The *excavationArm* value controlled the linear actuators that determined the height of the excavation apparatus and will likely maintain this functionality. The *excavationDrum* value controlled the speed at which the flywheel spun and will now control the motor that spins the conveyor that the buckets are attached to. This year, we will be using a NEO brushless motor hooked up to a SPARK MAX Motor Controller. This new type of motor and controller means we cannot leverage our

Talon/Falcon nodes and will need to develop a new Neo node responsible for converting speed values into motor movement. We went through the development process to develop the new Excavation and Neo nodes. This started with the initial analysis and reverse engineering of the SPARK MAX Motor Controllers, which are necessary for sending signals over the CAN bus to set specific motor values. We connected to the onboard Jetson Nano via SSH to sniff addresses traveling through the CAN bus as we executed commands on the REV Hardware Client. We then wrote and executed C++ files to test specific addresses and payloads. The culmination of this analysis was the SPARK MAX Driver Software, which includes functions such as inputting speed values between -1 and 1, which are converted into the IEEE Standard for Floating-Point Arithmetic (IEEE 754). This allows us to set the speed for the motors and is an essential step in developing the Excavation node.

4.3.2 Autonomy

The NASA Lunabotics competition assigns a portion of the score on the robot's autonomy. Therefore, it is in our best interest to ensure the robot is autonomous to maximize the team's points in the competition. To do so, we must ensure the robot understands its spatial location as it traverses the competition ground. In other words, the robot must be able to navigate sections of the course by itself. To enable this, we have many sensors that collect various movement metrics of the robot, such as the SparkFun LIS2DH, which will measure the robot's acceleration without consuming much power. It contains capacitive plates, which move in relation to each other as the system accelerates. This changes the capacitance, which is proportional to the system's acceleration, thereby allowing the robot to measure its acceleration [5]. It is a 3-axis accelerometer; thus, it can measure the acceleration in the x, y, and z planes of motion [6]. Next, we use a ZED2i stereo camera to ensure that the robot can avoid various obstacles present throughout the competition's course. The first advantage of the ZED camera is that it can estimate the depth of various objects using binocular front-facing cameras, much like how human eyes function (illustrated in Figure 5). Another advantage of the camera is that it can map the entire 3D plane that the camera sees, thus giving our robot a way to create a point-cloud map of the whole competition ground so that it can optimize its traversal by finding the path of least resistance from the extraction zone of the regolith to the target hopper. The camera augments the estimated location of an object from the front-facing cameras using a neural network [8]. The camera will also feed all the information to the team's controlling computer allowing us to see what the robot sees in addition to the robot's depth map. These are the primary hardware equipment used for the automation of the robot.

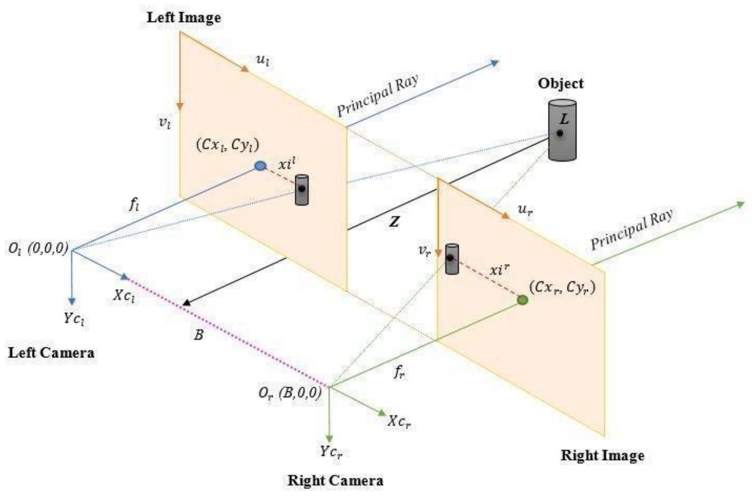


Figure 5 - An illustration of binocular vision [7]

Our initial breakdown for the Autonomy is the following steps:

1. Tracking the position of the robot.
2. Tracking the orientation of the robot.
3. Calculating the angle between an ArUco marker and the robot.
4. Turning the robot a certain amount of degrees.
5. Driving the robot a specific number of feet.
6. Running the Excavation Macro.
7. Turn 180 degrees and return to the starting position.
8. Running the Dumping Macro.

Position tracking can be challenging because motion is relative to the observer's position. For example, a robot standing still on a bus is not in motion concerning the bus but in motion regarding the road. The ZED API accounts for this by providing two reference frames: the camera and world frames. The camera frame represents the difference in camera position between the last and current frames. In contrast, the world frame sets a reference point and describes the difference in the camera's current position with that reference point. The API calls always return the position behind the left camera rather than the center of the ZED2i camera. This means we need to account for the distance between the two points if we want to use the position of the camera's center. Finally, the ZED2i camera defines the position as (x, y ,z) coordinates in a 3D space [26] as shown in Figure 6.

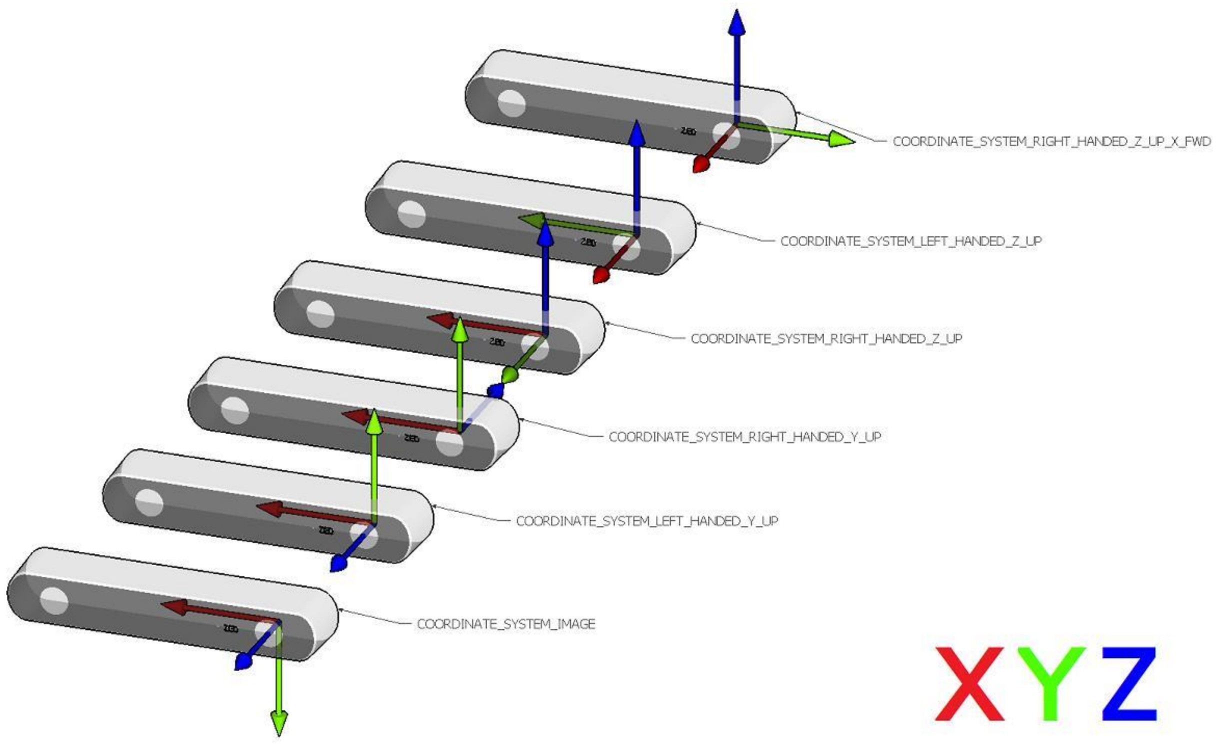
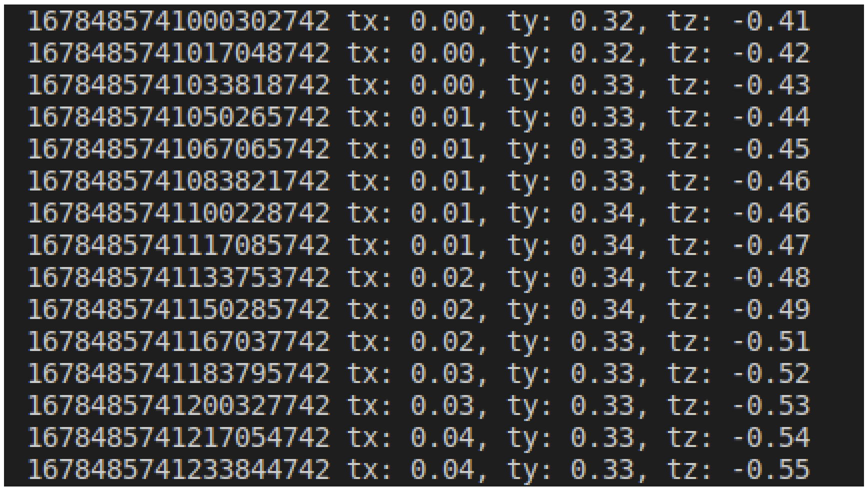

COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP_X_FWD
COORDINATE_SYSTEM_LEFT_HANDED_Z_UP
COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP
COORDINATE_SYSTEM_RIGHT_HANDED_Y_UP
COORDINATE_SYSTEM_LEFT_HANDED_Y_UP
COORDINATE_SYSTEM_IMAGE
XYZ

Figure 6 - Illustration of the various coordinate systems in the ZED API [25]

Our implementation uses a right-handed coordinate system with the z-axis pointing upwards and the x-axis pointing forwards, as shown at the top of Figure 6. To facilitate this, we create a Zed node that performs several key functions:

1. It gathers data from the various ZED sensors using the API.
2. It publishes this data so that other nodes can access it.
3. It streams the ZED video to a client's computer.

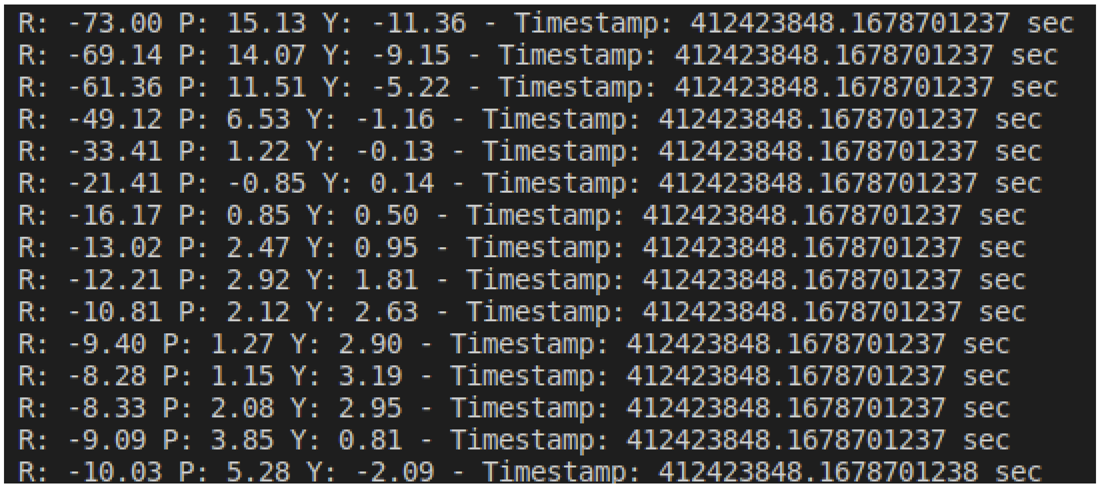
For position tracking, we create a topic that publishes a vector containing the (x, y, z) coordinates of the camera with respect to the world frame, along with a timestamp. Figure 7 illustrates this by printing out the camera's coordinates to the console as it is moved, along with a UNIX timestamp. The large number printed first represents the UNIX timestamp, which measures the number of seconds that have elapsed since January 1, 1970. The tx, ty, and tz variables represent the camera's translations along the x, y, and z axes, respectively. By default, the units of translation are in millimeters, but for our purposes, we convert them to meters.



```
1678485741000302742 tx: 0.00, ty: 0.32, tz: -0.41
1678485741017048742 tx: 0.00, ty: 0.32, tz: -0.42
1678485741033818742 tx: 0.00, ty: 0.33, tz: -0.43
1678485741050265742 tx: 0.01, ty: 0.33, tz: -0.44
1678485741067065742 tx: 0.01, ty: 0.33, tz: -0.45
1678485741083821742 tx: 0.01, ty: 0.33, tz: -0.46
1678485741100228742 tx: 0.01, ty: 0.34, tz: -0.46
1678485741117085742 tx: 0.01, ty: 0.34, tz: -0.47
1678485741133753742 tx: 0.02, ty: 0.34, tz: -0.48
1678485741150285742 tx: 0.02, ty: 0.34, tz: -0.49
1678485741167037742 tx: 0.02, ty: 0.33, tz: -0.51
1678485741183795742 tx: 0.03, ty: 0.33, tz: -0.52
1678485741200327742 tx: 0.03, ty: 0.33, tz: -0.53
1678485741217054742 tx: 0.04, ty: 0.33, tz: -0.54
1678485741233844742 tx: 0.04, ty: 0.33, tz: -0.55
```

Figure 7- Implementation of Position tracking

The orientation of a camera, which is attached to the robot, can be described by its rotations around three different axes: pitch, yaw, and roll. These measurements are typically represented using a mathematical concept called a quaternion [27], which the ZED API uses to store the camera sensor's rotations [26]. To convert this quaternion into more familiar roll-pitch-yaw angles, we use the Matrix3x3 class provided by ROS 2's *geometry2* library [28]. Once we obtain these angles, we publish them alongside a timestamp from the Zed node so that other nodes can access this information. Figure 8 shows this process, where the R, P, and Y variables denote the roll, pitch, and yaw angles in degrees, respectively.



```
R: -73.00 P: 15.13 Y: -11.36 - Timestamp: 412423848.1678701237 sec
R: -69.14 P: 14.07 Y: -9.15 - Timestamp: 412423848.1678701237 sec
R: -61.36 P: 11.51 Y: -5.22 - Timestamp: 412423848.1678701237 sec
R: -49.12 P: 6.53 Y: -1.16 - Timestamp: 412423848.1678701237 sec
R: -33.41 P: 1.22 Y: -0.13 - Timestamp: 412423848.1678701237 sec
R: -21.41 P: -0.85 Y: 0.14 - Timestamp: 412423848.1678701237 sec
R: -16.17 P: 0.85 Y: 0.50 - Timestamp: 412423848.1678701237 sec
R: -13.02 P: 2.47 Y: 0.95 - Timestamp: 412423848.1678701237 sec
R: -12.21 P: 2.92 Y: 1.81 - Timestamp: 412423848.1678701237 sec
R: -10.81 P: 2.12 Y: 2.63 - Timestamp: 412423848.1678701237 sec
R: -9.40 P: 1.27 Y: 2.90 - Timestamp: 412423848.1678701237 sec
R: -8.28 P: 1.15 Y: 3.19 - Timestamp: 412423848.1678701237 sec
R: -8.33 P: 2.08 Y: 2.95 - Timestamp: 412423848.1678701237 sec
R: -9.09 P: 3.85 Y: 0.81 - Timestamp: 412423848.1678701237 sec
R: -10.03 P: 5.28 Y: -2.09 - Timestamp: 412423848.1678701238 sec
```

Figure 8 - Implementation of Orientation tracking

The next major hurdle in making the robot autonomous is calculating the angle between our robot and an ArUco marker, which designates the dumping zone for the rocks the robot must collect and other landmarks in the arena. The big idea behind the implementation derives from the geometric concept of calculating the angle between two vectors. As explained previously, we know the robot coordinates using the ZED camera. Let us denote this as v1. The next big task is to find an ArUco marker using the ZED camera’s images. Once we do, we should be able to retrieve the marker's estimated x, y, and z coordinates from the point-cloud map, which we can denote as v2. To calculate this angle, we can use the following formula:

$$v_1 \circ v_2 = |v_1| \circ |v_2| \circ cos(\theta)$$

We can now rearrange for θ:

$$\theta = arccos(\frac{v_1 \circ v_2}{|v_1| \circ |v_2|})$$

To implement all of this, we use an OpenCV library to detect a marker. Then, we will call the ZED API to retrieve the point-cloud data on the position of the marker. Next, we create a function to implement the equation above to calculate the angle between the robot and the ArUco marker. Finally, we publish this information so that other nodes use it to move the robot correctly. After properly detecting and processing the ArUco markers, we can create the Autonomy node, which parses information from the Zed and Logic nodes to turn the robot a certain number of degrees and locate ArUco markers in the arena. To accomplish this, we use the camera reference frame from the ZED API and keep track of the total degrees turned until we reach the desired angle. We also need to account for the alignment difference between the center of the robot and the camera. After detecting and locking onto the desired ArUco marker, we call the function that drives the robot a set distance forward. This function will set the motors to spin at a constant speed and repeat the message until the robot has moved the desired distance. Ideally, this process does not require any ZED API calls since we are only working with the motors in a linear motion, but for the sake of making the action robust, we use the ZED2i’s distance-measuring ability for confirmation.

The challenge of our autonomous flow is determining the timing of the macros. For our base plan, we orient the robot based on the ArUco marker, then drive toward the excavation site. This is followed by an excavation macro to extract the material. This involves sending messages to start the excavation mechanism, dumping the material into the robot’s stockpile, and determining when to stop the process. After extracting the regolith, we will rotate the robot 180 degrees to return to the dumping site. Then, the robot will drive forward until it is at the dump site. Finally, we dump the material. If time allows it, we will continue to iterate and build upon the complexity of our strategy, with the ultimate goal being to implement a fully autonomous robot that would implement Dijkstra’s pathfinding algorithm, use computer vision models such as YOLO [10] to avoid obstacles, detect regolith, and square up with the dump site. However, this is a stretch goal and will most likely only reach foundational development such that future Razorbot teams can use and improve upon it to make the robot fully autonomous.

Therefore, we are implementing three major nodes that will handle the automation of the robot. The first node is the Zed node. As mentioned above, it will use the ZED API to publish information gathered by the camera to other nodes. The most important information gathered is the position of the robot, the orientation of the robot, the point-cloud map of the robot’s surroundings, and the images that the robot sees. This node is also responsible for streaming the data from the robot to a client computer via a TCP connection over the network. We leveraged the ZED API’s feature to stream the camera images to a remote client for our robot. While the Zed node in and of itself does not enable our robot to do anything autonomously, the information gathered by it is crucial for the robot to be semi-autonomous. The next node which plays a crucial role in automating the robot is the Logic node, which calculates the robot’s position from the Zed node and publishes this information to the Automation node. The final node we are implementing is the Automation node, which contains the logic for moving the robot, rotating the robot, and our semi-autonomous strategy, which has been described previously. This node will take in the data from the Zed and Logic nodes to determine which action the robot must perform next. Then, it will publish this information to nodes such as the Excavation node, Talon node, and Falcon node. Finally, to elaborate on why the robot is semi-autonomous as opposed to fully-autonomous, we are implementing the nodes in such a manner that the robot will disregard autonomous commands while we are manually controlling the robot. This is because the competition scoring emphasizes extraction over autonomy, and thus this provides a way for us to maximize the team's points in case of autonomous failure. To implement this, we will have a button on the joystick that will flip a boolean indicating whether autonomy functions are on or not.

4.4 Risks

Risk	**Explanation & Risk Reduction**
Lack of Communication	Since this project has a very large separated team, any lack in communication can lead to slow progress. To increase our inter-team communication and avoid miscommunication, we started the project off with a Teams channel dedicated to open-communication and weekly check-ins. We also communicate task scheduling through the Razorbotz Trello board. The combination of the Teams channel, the Trello board, the Razorbotz discord server, and consistent emails/meetings with our supervisor (Andrew Burroughs), we can reduce communication risks.
Mechanical Failure	There are many points of possible mechanical failure related to the physical construction of the robot. Some major points of failure include the motors and motor controllers with possible short circuits. To reduce the chance of these risks, the MEEG team is using different motors & motor controllers which have a lower failure rate, and we will avoid any programmed angles/moves that might overload any components.
Differences in physical and simulation	In previous years, there have been issues with the coded tests not translating to the physical robot well. To reduce this risk, we are planning to start earlier and work in better conjunction with the MEEG team to understand their CAD designs as well as the physical robot. This allows us to notice any differences between CAD designs and the physical robot when making the autonomous program.
Library/Framework Version Mismatch	With the differences in member operating systems and updates in necessary libraries & frameworks, we may easily run into version control issues. To reduce the risk of this, we are comparing our ROS 2 install procedures with previous years and getting all the installations

	done as early as possible. We also use the virtual machine VirtualBox when possible to unify our installation and coding processes.
Autonomous Failure	When we start programming for autonomous control, there are many risks that can lead to autonomous failure including failed object detection. We can reduce the effects of any autonomous failures through a "Kill Switch" button on the robot to shut off power and possible manual override to prevent a serious crash remotely.

4.5 Tasks

1.0 Researching competition

- 1.1 Understand objectives of the competition
- 1.2 Understand our responsibilities as the CSCE team
- 1.3 Meeting with supervisor to clarify questions and concerns regarding the CSCE team's responsibilities

2.0 Learn ROS 2

- 2.1 Install on computers
    - 2.1.1 Install VirtualBox (if not using Linux or M1 mac)
    - 2.1.2 Run Linux Distro
    - 2.1.3 Install ROS 2 in the Linux OS
- 2.2 ROS 2 Python & C++ tutorials

3.0 Review previous code

- 3.1 Understand what different components are doing
- 3.2 Understand when C++ vs Python is required

4.0 Streaming from the camera

- 4.1 Investigate potential solutions to enable camera streaming

5.0 General control

- 5.1 Implement nodes to control desired motor
    - 5.1.1 Will subscribe to manual/autonomous nodes that publish motor speeds
- 5.2 Excavation
    - 5.2.1 Enable manual control of motor

- 5.2.2 Expected operations include:
  - 5.2.2.1 Mine into the crater
  - 5.2.2.2 Collect the resources mined
  - 5.2.2.3 Store collected resources into temporary bucket
- 5.2.3 Implement algorithm that performs the expected operations for the excavation tool
  - 5.2.3.1 Create node to control each motor based on the algorithm

6.0 Manual control

- 6.1 Translating joystick commands into motor speeds
  - 6.1.1 Program motors to drive
  - 6.1.2 Program motors to begin mining process
  - 6.1.3 Program motors to dump mined materials
- 6.2 Communication with Client
  - 6.2.1 Implement communication so controls can be sent to robot
- 6.3 Manual control nodes testing
  - 6.3.1 Create unit tests for each node to ensure proper functionality
  - 6.3.2 Create integration tests to ensure overall proper functionality

7.0 Autonomous control

- 7.1 Driving autonomy
  - 7.1.1 Create object recognition nodes with ZED 2i stereo camera
    - 7.1.1.1 Implement node for camera to publish images
      - 7.1.1.1.1 Send robot’s image stream to GUI
    - 7.1.1.2 * Implement node to receive images and handle object detection
  - 7.1.2 * Create nodes for the robot’s path planning
    - 7.1.2.1 * Implement node to create path based on object detection
    - 7.1.2.2 Implement node to translate path into motor speeds
  - 7.1.3 Communicate motor speeds to the desired motors
- 7.2 * Excavation Autonomy
  - 7.2.1 * Utilize excavation node when robot has been positioned
- 7.3 * Dumping Autonomy
  - 7.3.1 * Utilize object recognition and path planning nodes to detect and navigate to dump site

7.3.2 * Implement node to control the position of the bucket for dumping once at dump site

7.4 * Autonomous control nodes testing

7.4.1 * Create unit tests for each node to ensure proper functionality

7.4.2 * Create integration tests to ensure overall proper functionality

**To do be done if time permits based on Mechanical Engineering team's progress*

4.6 Schedule

Tasks	Assignee(s)	Dates
Researching competition	CSCE Team	10/1-11/27
Install ROS 2 on personal machines	CSCE Team	10/10-10/17
Complete ROS 2 Python tutorials	CSCE Team	10/17-10/24
Complete ROS 2 C++ tutorials	CSCE Team	10/17-10/24
Investigate potential solutions for the video stream for the camera	CSCE Team	10/24-11/7
Review and understand code from previous years to ensure smooth transition into next semester	CSCE Team	11/1-11/15
Documentation of previous years code	Jackson Burger Ahmed Moustafa Jackson Newman Justin Kilgo	1/30-2/6
Manual controls testing	Jackson Burger Jackson Newman	1/30-2/6
Investigate Zed2i camera and software to begin object detection	Ahmed Moustafa Rohit Kala	2/6-2/13
Reverse engineer Spark Max motor controllers	Ahmed Moustafa Jackson Burger Jackson Newman	2/11-2/25
Implement node for camera to publish images	Justin Kilgo Rohit Kala Jackson Newman	2/13-2/27
Implement node to receive images and track position	Rohit Kala Ahmed Moustafa	2/20-2/27

Tasks	**Assignee(s)**	**Dates**
Communication with client	Ahmed Moustafa Rohit Kala	2/27-3/11
Test/Fix previous driving automation code	Jackson Burger Jackson Newman	3/25-4/1
Implement autonomous functions to test expected operations of autonomy (turning, driving certain distance, etc.)	Jackson Burger Jackson Newman	4/1-4/15
Handle ArUco marker orientation	Ahmed Moustafa Rohit Kala Justin Kilgo	4/8-4/15
*Implement algorithm that performs expected operations for the excavation tool		
*Create nodes for the robot's path planning		
*Utilize object recognition and path planning nodes to navigate to the dump site		
*Communicate motor speeds to desired motors		
*Create unit and integration tests for each node for manual control		
*Create unit and integration tests for each node of autonomous control		

**Could not be finished due to time and hardware constraints; can be accomplished by future teams.*

4.7 Deliverables

- Project Website
  - We will create a page on the Capstone website for our project which will contain our project information, tasklist, proposals, reports, and code.
- Project Proposal/Report
  - We will be creating a proposal for the original idea of how we are going to carry out the development of this project. Upon completion, we will create a report detailing our solution for the project. There will be clear indications of decisions that strayed away from our project proposal. We will also include the difficulties that we have faced and the process we took to overcome them.
- Manual Control Code

    - We will need to create a robot that can also be manually controlled using a joystick. This will enable the robot to navigate the map and excavate material normally. We will need to program the client/server connection on the Jetson Nano to allow us to send joystick inputs to the robot from a controller via our ROS 2 code and have the robot respond accordingly.
- Autonomous Code
    - We will be programming the robot such that it is semi-autonomous and will be able to accomplish the goals set by the NASA robot mining competition. To achieve this we will need to implement path planning, depth perception, and position tracking to navigate through the arena. This deliverable will include the multiple ROS 2 nodes for publishing images from the camera and planning the path to correctly send the motor speed data.

5.0 Key Personnel

Ahmed Moustafa - Moustafa is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Foundations I, Programming Foundations II, Programming Paradigms, Software Engineering, Artificial Intelligence, and Algorithms. He is currently a Software Development Engineer at SupplyPike working on full stack web development. Last summer, he attended the NACME-Google Applied Machine Learning Bootcamp working on a Reinforcement Learning model for a scale self-driving car. As a member of the CSCE portion of this project, he will be working on robot programming.

Jackson Burger - Burger is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Challenges I, Programming Challenges II, Programming Paradigms, Software Engineering, and Database Management Systems. He is currently an Annual Engineering & Technology Intern at J.B Hunt where he does full stack web development. As a member of the CSCE portion of this project, he will be working on robot programming.

Jackson Newman - Newman is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Foundations I, Programming Foundations II, Programming Paradigms, Database Management Systems, Software Engineering, and Algorithms. He is currently an annual Engineering and Technology intern for J.B. Hunt and works with the Continuous Integration and Continuous Development team. As a member of the CSCE portion of this project, he will be working on robot programming.

Justin Kilgo - Kilgo is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Foundations II, Programming Paradigms, Software Engineering, and Algorithms. He has experience programming robots from his high school robotics team and was a Software Development Engineer Intern for Amazon over the summer working on Full Stack Web Development. As a member of the CSCE portion of this project, he will be working on robot programming.

Rohit Kala - Kala is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Foundations I, Programming Foundations II, Programming Paradigms, Software Engineering, Artificial Intelligence, and Algorithms. He is currently an undergraduate research assistant for the CVIU lab at UARK. As a member of the CSCE portion of this project, he will be working on robot programming.

Uche Wejinya - Dr. Wejinya is an associate professor in the Department of Mechanical Engineering at the University of Arkansas. He received his Ph.D. in Electrical Engineering from Michigan State University in August 2007. Dr. Wejinya's research relates to mechatronics with an emphasis on nanotechnology. His research delves into nanomaterials for nanosensors, nanoelectronics, and robotics which is relevant to this project.

William Burroughs - Burroughs is a Master’s student in the Computer Science and Computer Engineering Department at the University of Arkansas. He has been on the Razorbotz team since 2018 and was the Controls sub-team lead in 2020. He is the current Computer Science sub-team lead and mentor for the RMC project.

6.0 Facilities and Equipment

- Facilities
  - Mechanical Engineering Robotics Laboratory - Lab in the Mechanical Engineering building where the robot is built and tested
  - Test Pit - Room in the Engineering Research Center used for larger tests that cannot be performed in the robotics lab.
- Equipment
  - ZED 2i stereo camera
  - SparkFun LIS2DH accelerometer
  - SPARK MAX Motor Controller
  - Jetson Nano processor
  - Pololu Glideforce GF23-120512-3-65 High-Speed LD Linear Actuator with Feedback: 12kgf, 12" Stroke (11.8" Usable), 3.3"/s, 12V
  - Falcon 500 Motors
  - USB to CAN converter

7.0 References

[1] Staff, The Robot Report. “Servi the Product of Bear Robotics and Softbank Robotics Partnership.” The Robot Report, 29 Sept. 2020, https://www.therobotreport.com/servi-bussing-robot-product-bear-robotics-softbank-partnership/

[2] Spectrum, IEEE. “Roomba.” *ROBOTS*, 18 May 2018, https://robots.ieee.org/robots/roomba/

[3] NASA, https://www.nasa.gov/sites/default/files/atoms/files/00_guidebook_2023_ver._0.3_tm.pdf.

[4] “Find Answers.” *IRobot*, https://homesupport.irobot.com/s/article/19541.

[5] “Accelerometer Basics.” *Accelerometer Basics - SparkFun Learn*, https://learn.sparkfun.com/tutorials/accelerometer-basics/all.

[6] “Triple Axis Accelerometer Breakout - lis2dh12 (Qwiic).” *SPX-15760 - SparkFun Electronics*, https://www.sparkfun.com/products/15760.

[7] *The Operation Principle of the Zed Camera*. https://www.researchgate.net/figure/The-operation-principle-of-the-ZED-camera_fig2_325854193.

[8] *How Does the Zed Work? – Help Center | Stereolabs*. https://support.stereolabs.com/hc/en-us/articles/206953039-How-does-the-ZED-work-.

[9] “Pytorch.” *PyTorch*, https://pytorch.org/features/.

[10] Redmon, Joseph. *Yolo: Real-Time Object Detection*, https://pjreddie.com/darknet/yolo/.

[11] “Ros 2 Documentation ℑ.” *ROS 2 Documentation - ROS 2 Documentation: Foxy Documentation*, https://docs.ros.org/en/foxy/index.html.

[12] “Object Detection.” *Papers With Code*, https://paperswithcode.com/task/object-detection.

[13] Credit: Division of Rare and Manuscript Collections, et al. “Professor's Perceptron Paved the Way for AI – 60 Years Too Soon.” *Cornell Chronicle*, 25 Sept. 2019, https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon.

[14] *MIT Deep Learning 6.S191*. http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf.

[15] “Training and Test Sets: Splitting Data | Machine Learning | Google Developers.” *Google*, Google, https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data.

[16] “What Is a Convolutional Neural Network?” *What Is a Convolutional Neural Network? - MATLAB & Simulink*, https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html.

[17] “Common Objects in Context.” *COCO*, https://cocodataset.org/.

[18] “Earth 'Will Expire by 2050'.” *The Guardian*, Guardian News and Media, 7 July 2002, https://www.theguardian.com/uk/2002/jul/07/research.waste.

[19] Gilbert, Alex. “Mining in Space Is Coming.” *Milken Institute Review*, https://www.milkenreview.org/articles/mining-in-space-is-coming.

[20] Kramer, Sarah. "Here's How Much Money It Actually Costs to Launch Stuff into Space." *Business Insider*, Business Insider, https://www.businessinsider.com/spacex-rocket-cargo-price-by-weight-2016-6.

[21] "Space Technologies at California." *Home*, https://stac.berkeley.edu/project/rover#:~:text=An%20autonomous%20rover%20system%20deployed,these%20resources%20is%20incredibly%20difficult.

[22] Nayar, Jaya. "Not so 'Green' Technology: The Complicated Legacy of Rare Earth Mining." *Harvard International Review*, Harvard International Review, 12 Aug. 2021, https://hir.harvard.edu/not-so-green-technology-the-complicated-legacy-of-rare-earth-mining/#:~:text=This%20stems%20from%20the%20fact,in%20especially%20detrimental%20health%20effects.

[23] GeeksforGeeks. "Client-Server Model." *GeeksforGeeks*, 15 Nov. 2019, www.geeksforgeeks.org/client-server-model .

[24] "What Is TCP/IP? | Cloudflare." Cloudflare, www.cloudflare.com/learning/ddos/glossary/tcp-ip/.

[25] "Coordinate Frames." *Stereolabs*, https://www.stereolabs.com/docs/positional-tracking/coordinate-frames/.

[26] "Stereolabs." *Stereolabs Docs: API Reference, Tutorials, and Integration*, https://www.stereolabs.com/docs/.

[27] "Quaternion Fundamentals." *Quaternion Fundamentals - ROS 2 Documentation: Foxy Documentation*, https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Quaternion-Fundamentals.html.

[28] "Matrix3x3 Documentation." *TF2: TF2::MATRIX3X3 Class Reference*, https://docs.ros2.org/foxy/api/tf2/classtf2_1_1Matrix3x3.html.

[29] Corrigan, Steve. "Introduction to the Controller Area Network (CAN) (Rev. B)." *Texas Instruments*, https://www.ti.com/lit/an/sloa101b/sloa101b.pdf.