# EZ-RASSOR 2.0 Gold Team - Sponsored by the Florida Space Institute

Autumn Esponda, Martin Power, Daniel Silva, Daniel Simoes, Chin Winn

Department of Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — In this project, we create a system for controlling a swarm of EZ-RASSOR rovers. The system handles dividing the task of collecting regolith among a group of EZ-RASSOR rovers, and will handle the management of each rover's battery level, their current objective, and the path which it will follow. The system utilizes a scheduling algorithm to coordinate the use of charging stations and local repair A* pathfinding to send rovers on energy-efficient paths.**

**The whole system is run in a Gazebo simulation and is made up of ROS nodes which communicate with the systems developed by the EZ-RASSOR 1.0 team.**

***Index Terms* — Multi-robot systems, Robot control, Robot programming, Simulation, Space Exploration**

## I. Introduction

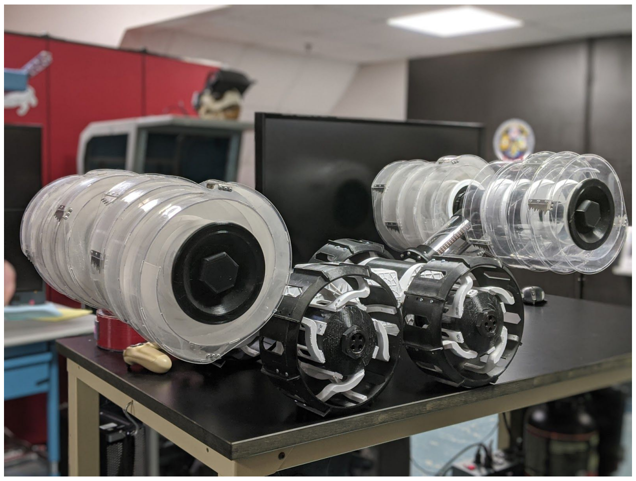


Figure 1. Mini-RASSOR rover at NASA SwampWorks.

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 a large group of EZ-RASSOR robots.

We have created an intelligent system capable of breaking up high-level tasks into individual assignments and waypoints for each EZ-RASSOR rover in the swarm to follow. Each EZ-RASSOR rover then uses this information to run it's on-board navigation routines. We have also created a ROS node which will track details about all of the rovers in the swarm, including current location and battery level. This data will allow the system to make smarter and more efficient decisions in regards to routing and task divvying. The system also attempts to avoid scenarios such as rover collision and dual tasking.

Our system also maintains a map of the surrounding environment. This map is used for path planning, where we run a path planning algorithm to find the shortest path which avoids obstacles such as craters and large rocks. The path planner also takes into account the slope of any incline/decline and attempts to avoid slopes which would be inefficient to ascend. If a rover encounters a previously unknown obstacle and veers too far off path, the swarm system will plan a new path and update the path the rover is following.

## II. Summary of Requirements

| |
|---|
| The EZ-RASSOR software shall remain open source |
| The EZ-RASSOR software shall continue to be well documented in order to facilitate further development and maintenance |
| The EZ-RASSOR software shall provide a solution that is presentable for demos around the Kennedy Space Center |
| The EZ-RASSOR software shall be compatible with and maintain consistent performance in a Gazebo simulation |
| The EZ-RASSOR software shall utilize ROS as a central middleware system to run the robot's external hardware |
| The EZ-RASSOR software should utilize swarm AI technology to enable a hive of rovers to handle tasks collaboratively and efficiently |

| |
|---|
| Tasks are limited to the mining and return of materials/regolith |
| Given the coordinates of a dig site or area, the Swarm 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 and hence, will only be as accurate as NASA’s data permits |
| Rover paths and scheduling should be constrained by a rover's potential battery life. The central system is responsible for ensuring rovers are brought back to the lander for charging in a timely manner |
| Individual EZ-RASSORs are responsible for the autonomous gathering and depositing of materials into the processing unit of the lander |
| The coordinate grid and rover exploration capabilities shall be within a limited range |
| 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 |

## III. Path Planning

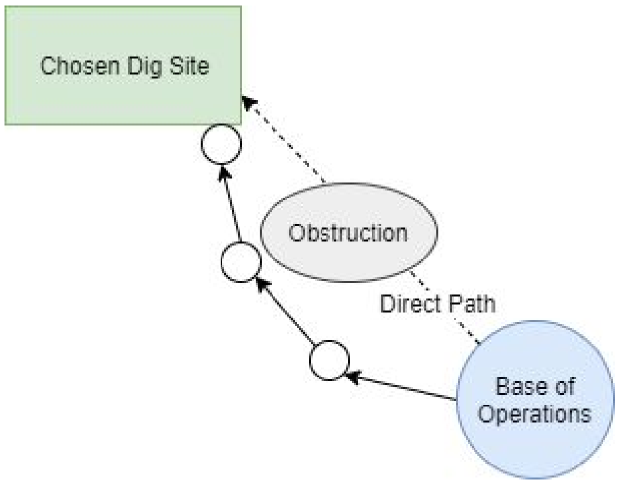


Figure 2. Path planning diagram.

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.

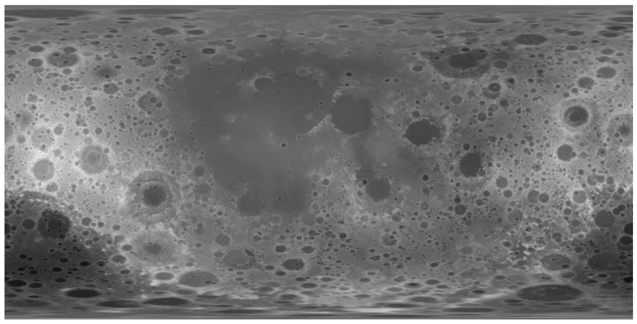


Figure 3. Image of the surface of the moon created using NASAs lunar elevation data.

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 has been generated for that rover by the swarm controller.

## IV. Rover Status Tracking

The swarm control system has the ability to track data about each rover through the use of ROS nodes. The swarm control node subscribes to the path planning node and reads the status of each rover from it. This data is used throughout the whole swarm system. The path planner uses location data to determine if a rover needs to have a new path generated. The scheduling subsystem uses the battery and location data to track whether a rover should be allowed to complete another journey or if it should be sent to the charging stations. The system also keeps track of what each rover's current activity is, such as whether it is driving back to home base or charging.

## V. Scheduling

The scheduler is in charge of ensuring that multiple rovers are able to coordinate their movements without creating an excessively long queue for charging. Because of this, the scheduler mainly interfaces with the rover statuses 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.

## VI. Gazebo Simulation

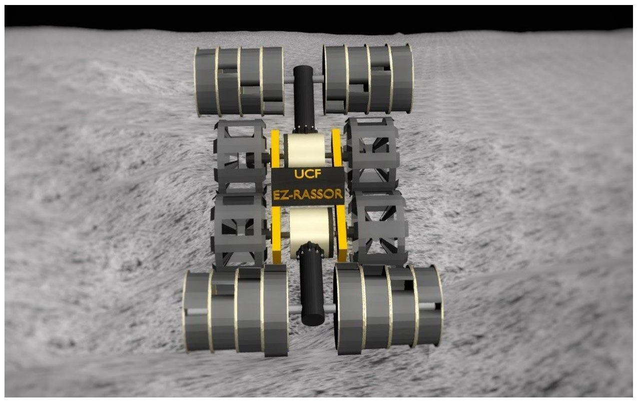


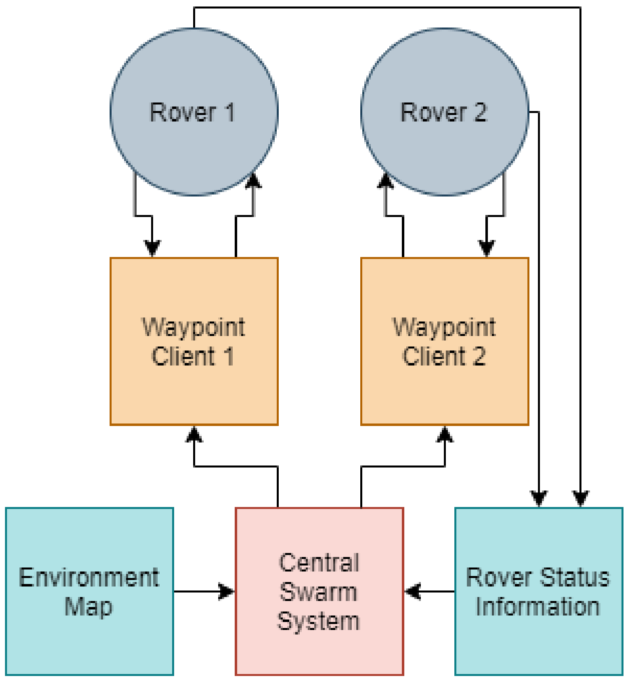
Figure 4. EZ-RASSOR rover model in Gazebo.

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 aren't able to access the physical rover particularly often to figure out how to interface with and process data coming in on the real cameras.

The simulator also allows for multiples of the same model to be loaded into one world at a time, which is useful when it comes to the swarm intelligence as discussed in Section III, IV, and V. 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 uses a similar approach, where we are using the simulation for 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.

## VII. ROS Architecture Overview



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 one 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 processes. 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.

Regarding our implementation of ROS, ezrassor_swarm_control package, consisting of two nodes, handles high-level scheduling and path planning for a swarm of EZ-RASSOR rovers. The system is able to manage an arbitrary number of EZ-RASSOR rovers and dig sites, and will efficiently divide the task of collecting regolith between the rovers. The system tracks details such as current battery level about each rover and uses that information to decide if the rover should be mining or queueing for a charging station. The paths generated by the path planning subsystem of the swarm control system are based on the elevation map provided to the system. The paths will search for paths with minimal elevation change, which will help conserve the battery of each rover.

The first node is the swarm_control node. This node functions as the central control hub for the swarm control system. This node is in charge of tracking and publishing the paths for each rover. The swarm control system will make decisions about when each rover needs a new path supplied to it and where that path should go. For instance, if the rover's battery is too low to safely continue working, the swarm control system will send it to go to the charging station. This node publishes sets of waypoints for each rover.

The second is the waypoint_client node. This node acts as a communication client between the path planning system and the swarm of EZ-RASSOR rovers. It subscribes to the waypoint_client publisher assigned to each EZ-RASSOR rover and alerts the rover when there is a change to it's path. The use of SimpleActionClients allows previous actions to be cancelled and updated to new paths provided by the swarm control system. The waypoint_client will receive messages from each rover indicating that the rover is ready for the next waypoint, then the waypoint_client will provide the next one to it.

## Acknowledgement

The authors wish to acknowledge the assistance and support of the Florida Space Institute and NASA.