Adaptable Searching and Retrieving of Rover Swarms

Manuel Meraz, Mario Inzunza

Abstract— **Upon the necessary requisite understanding to start this project, we needed to analyze base swarm behavior. The base behavior of the template we received was random search, upon finding a 'resource' represented by a tag, the rover would return the resource to the center. The key element being interconnectivity through the sharing of information between rover. Thus the project would be separated into 2 parts, searching and retrieving, while utilizing the network capacity of the rovers to increase the cohesiveness of their work as a swarm. The searching aspect is a bit subtle to explore because in essence we're searching for something that we have zero information about its whereabouts, as a result a full scan of the area must be covered to find all possible resources. The second aspect of the project, the retrieval aspect is where we decided to begin, and utilize the swarm capacities of the rovers. We didn't collect data for this project and focused mainly on the ideas and analysis of swarm cohesiveness. The largest factor in determining efficiency as the discovery that there was an inverse correlation between obstacle collision calls and efficiency. The reason being that whenever a rover detected an obstacle, it had to then turn in order to avoid that obstacle. Thus, the more time spent turning as a result of obstacles collision calls the less efficient the 'path' a rover would take to collect resources. In essence, the fastest rovers at collecting are the ones with the least obstacle calls and being able to flow freely as a swarm.**

I. Introduction

The team's time was spent, for the most part, increasing the efficiency of rovers collecting capability. The approach we took was to first analyze what we were working with and what we wanted to do with it. First, we have to have a goal, increasing efficiency of collecting while utilizing the network capacity of the rovers. Second, we have to understand what it would take to accomplish this task. To understand, we look at the base patterns that the rovers currently have programmed and build from there. The base patterns being 'random' searching and retrieving in a straight line back to the collection disk, disregarding the other rovers. The next part is applying creativity to come up with a solution. Once an idea is concocted, we convert that idea into pseudo code and finally, that pseudo code into actual code. The rest is debugging and fine tuning until it runs the way we want. This pattern was repeated until the deadline.

Looking beyond the two parts of the project, searching and retrieving. Once enough efficiency was to be reached, the focus would then turn on the autonomy of the rovers and adapting to their patterns. In order to adapt, the rovers have to be able to not only receive information from their 'environment,' but be able to process that information in a way to increase the cohesiveness of the swarm behavior. To do this, data was eventually to be collected and the analysis of this data would turn into information the rovers can process to be able to adapt.

The cycle being to first put in our own ideas to get the rovers to be self sustainable. Next, once the rovers are sustainable, code in the capacity to learn about their environment and finally use that information to adapt. The data analysis section would feed the information to the rovers, along with rovers consistently sharing information with each other about themselves in almost real time, then that information is processed and the self sustaining algorithm would become more efficient by reducing by following the rover with the least obstacle collisions and imitating that rover, and repeating until all the rovers are following a path through intervals that would have the least obstacle collisions, and thus the least time turning and adjusting.

If the rovers require a path to follow, then there must first be a default path that is programmed into the rovers by us. For the rovers to follow a path, they must communicate. If they weren't originally communicating and sharing information, then we must program that in. To know if they are communicating, we must see them move, and to see them move means that we have to first program them to move in ways we want. In order to move them the way we want, we need to understand the necessary information required to accomplish this task.

The project can be broken down into this tree:

1. Searching

   A. Random Searching

   - Random searching is not random, it follows a pattern, thus instead of being an even distribution of likelihood of numbers, there would be a higher chance in certain portions of the map where a resource would spawn
   - The most likely location a resource could spawn would be between the outer radius and the center, in the form of an inner circle
   - We could use this information to scan the inner circle's radius edge to increase the likelihood of finding a resources
   - The next is noticing that all the rovers spawn facing the same direction. Following the same logic that randomness doesn't exist, we can see that if they're all facing the same direction then the rovers will follow a similar path with the central random numbers being the guide to where the rover would head directly to, and the amount of turning being done depending on the amount of obstacles it

  would detect, including other rovers and the wall

- In order to increase the maximum area scanned then, the best initial position being facing radially outward from the center and beginning from there

2. Retrieving

A. Clusters

- A cluster is defined by a a 360 degree scan by a rover that would then count the number of tags scanned
- If the tags scanned were greater than or equal to 5, then a cluster was formed at the center of the rover in terms of its coordinates with the radius of the circle defined by the upper edge of the camera view.
- The radius approximately being approximately 1.169m
- The next portion would be to analyze the cluster density in relation to the overall area of the map. If a new cluster was to be discovered, then in order to reduce the redundancy of cluster overlaps, cluster density would be required. Cluster density being defined as Area of all clusters taken divided by the overall area of the map
- Upon a cluster density of 0.8 > searching would stop, and retrieving would start being the only function
- 0.8 was decided as a first estimate, with other possible cluster densities to be considered being between 0.7 and 0.95
- A final walk through of all the discovered clusters for any left over tags would be done by a designated rover assigned to searching, which would stop searching and begin its final scan once the rover detected a high enough cluster density
- The rover would be designated by simply being the first rover to be created

B. Non-Clusters

- Everything that is not a cluster, or any resources discovered to not have 5 or more after a 360 degree scan by the rover.
- In the case of non clusters, the rover would simply drop the first collected tag on the collection plate and go back to collection plate to drop it. Then it would return back to random searching.

3. Data Collection and Analysis

A. Real Time Data Collection and Analysis

- Real time data collection is the collection of data as rovers are active
- The relevant real time data to be collected being the environment, which includes the rover itself collecting the data, the map and other rovers
- Ideally the rovers would always know everything about each other at all times. First the rover gathers information on itself, and shares it with all the others. If all rovers do this, then the rovers know where all other rovers are.
- Next is mapping the area. Converting data into a hypothetical space the rover can travel through. If all rovers have a self map and they're all sharing information, then a larger map can be created by utilizing the overall map information to create a larger map

B. Collection of Historical Data

- Historical data is all past data based on previous simulations
- if our goal is to decrease the amount of time to accomplish the task, then we need to analyze which patterns the fastest rovers follow, adapt those and repeated
- The time between searching and discovering resource and the time between the collection of a tag to the next tag. For example, if a tag is found, the rover picks it up, then drops it at the center, and finally goes back to the cluster to collect another tag. The time it took for the rover to complete that loop, would be the collection efficiency.
- The fastest rovers are the rovers that turn the least, and thus have the least obstacle collision detections
- If rovers have memory of historical data, then previous paths can be used to accomplish similar tasks and adapted to the current path by following the path the rover with the least obstacle calls, and so on.

C. Analysis

- As rovers collect historical information and real time information, they use this information to calculate and predict the most efficient path.
- If a prediction is made, and the time decreases to accomplish the task, the rover adapt this patterns

- If the rover accomplishes the task in less time, then the new pattern is ignored
- If the rover finishes in equal time, then the simplest pattern with the least information required is utilized

II. Related work

We did not research any papers or projects that have been similar to this. All of the knowledge we acquired came from:

- Our previous understanding math analysis, specifically trigonometry and physics
- https://cse.sc.edu/~jokane/agitr/agitr-letter.pdf
- www.ros.org
- Any specific libraries and tools we used were discovered by using the Google as a search engine to find the tools we needed to accomplish the task
- The readme.md file provided by the base Swarmathon template code, along with the code base itself to understand how the rovers operate

III. Methods

A. *Calculation of base path to build upon*

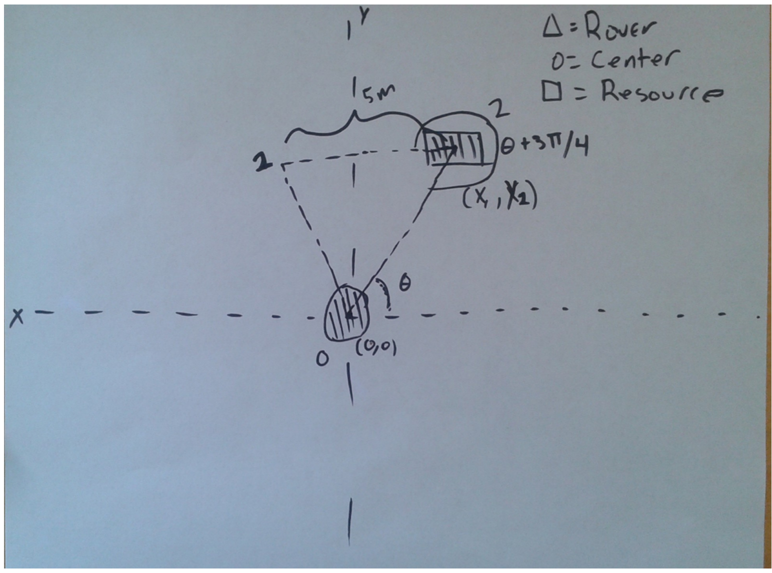
The first attempt at getting rovers to collect resources as a group from a designated cluster, was to simply follow a straight line back to the center, then back to the resource and so on. This proved to be an issue because the rovers kept running into each other.

The next attempt was to create a map where each key was a checkpoint in a path, and the key pointed to GPS information for that checkpoint. This path information would be stored in Resource object containing all relevant information about a cluster, and default path created by a the rover that discovered the cluster of resources.

The original default path was a square, but too much time was spent turning by the rover as it hit each check point, or corner of the square. The square would be created by using the center and resource cluster as side of the square. Then building the square around this side. This was an issue because the size of the square would conflict with the outer edges at times.

The next choice was a triangle path. The triangle path means 1 less checkpoint or turn, meaning faster collection times. The triangle was to be designed so that each side of the triangle was parallel to the walls, thus never intersecting with them.

The center would be a corner, the resource itself would be a corner, then with the angle between the x-axis and the line the resource creates plus 3 *pi* / 4 would create a parallel line the. length of 5 meters from the resource parallel to the wall would result in a triangle that always fits.



To calculate the distance between to points, the Pythagorean formula was used with the GPS coordinates.

To calculate the resource location, we designated the resource location to be the rovers location.

The plan was to use this base path, and as rovers travel through this path they record their a new path based on the GPS coordinates they specifically took along with the number of obstacle collisions. The path with the least obstacle collisions at the present moment based on the data the rovers collect would then become the new designated path that all rovers take, and so on.

This is called adaptable path, because as rovers loop and collect a cluster of resources, the path they take changes based off of the number of collisions.

We never got to the point of implementing an adaptable path, but that was the end goal.

B. *Rover queue along path for the purpose of synchronized collection of resources*

One of the major issues was congestion along path check points. If all the rovers attempt to head to the resource to collect at the same time, they will inevitably run into each other and start a dance where nothing productive is being done.

In order to account for this a waiting system was implemented. All the rovers know at all times where all the other rovers are at and where resources are location. This information is published to the topics */resource* and */condition* through custom messages that were created to store relevant information.

The rover's synchronization takes advantage of a message that keeps updated information of every rover's position, checkpoint and collection states. The message type being named *status*. The task of stopping and moving a rover is delegated between a “master rover” (the first rover to discover a cluster and form a path to the center) and the other rovers on the same path. At each checkpoint, every rover is tasked to stop if it's too close to another rover. In contrast, the “master rover” coordinates rovers that are moving towards the same checkpoint. This separation of tasks was done to avoid burdening a single rover with calculating every possible collision and achieve greater more efficient traffic flow.

The synchronization code is divided into two sections, depending if a rover has been told to stop. The code that orders rovers to wait was divided into two subsections in the condition handler, the first part can only be iterated through by the “master rover” and ensures that the distance between any rovers that are moving towards the same section are at least 1.5 meters apart from each other. The second part is accessible to every rover and ensures that the distance between any rovers that have different checkpoint (as is possible when rovers are at the same checkpoint, with one moving away from it and another moving towards it) is at least 1.5 meters. Should any distance be smaller than 1.5 meters, the rover that is farthest from the checkpoint, or has the lowest checkpoint (meaning that its behind another rover) is told to wait. Rovers that have been told to wait are dedicated to ensuring that they are far enough away from every moving rover that it can end its stopped state. A rover in a waiting state is excluding from code that is used for navigation to ensure that each waiting rover can return to its path as fast as possible.

This was as far as we got in terms of the actual implementation of the code.

IV. Experiments

Most of the experimentation done was done by simply testing the code after we experimented. If the code didn't compile, then we fixed syntax errors until the code compiled. If we had run time errors, then we would analyze in real time using ROS's tools such as *rostopic echo*, to see the data being published along with using ROS_INFO_STREAM() to send out relevant information to */rosout*. This information was viewed using *rostopic echo /rosout | grep “relevant string”*.

If everything seemed to be fine, then we had to analyze our logic and make sure we account for edge cases to account for unexpected behavior.

V. Results

One of the major obstacles to overcome was getting started due to the initial learning curve and scarcity of resources to begin. The initial plan was to use a dedicated computer as the main computer to work on, but after many issues the computer we were provided could not run the simulations due to not being strong enough to process the simulation. The first chunk of time was spent learning the essentials to begin, and sharing that information to the other team members by spending individual time with them on how to start. Our team ended up having to use personal laptops, and by the end there were only 2 of us working on it.

We did not collect data and did not use statistical analysis to present data. We focused on coming up with ideas and implementing those ideas by coding them in the *mobility* package, and tested them by watching the simulations and observing the data in real time to make sure things ran as planned. So the only way to verify our results, is through the simulation itself.

We updated our code for the first few weeks, but we had quite a few obstacles in our school work and it slipped out minds to keep our code updated with the original code base. When we got to the end of the competition and reread the rules, we realized we hadn't applied the latest patches.

As a result, our code might not run as expected due to late bug fixes (such as publishing by picking up and dropping the target).

VI. Conclusion

In this work we analyzed the base ideas for rovers to share information and work efficiently, and partially put them into our code base.

The ideas and the coding weren't the most difficult part. The most difficult part of the competition was putting forth the effort to learn the material, then dispensing this information to everyone else in a digestible manner and having the rest of the team use that new information to move onwards towards coming up with new ideas to implement with a scarcity of resources available to us. Many of our team mates never got past learning the very basics of the Linux terminal, and much time was spent trying to help them with futile effort.

In the future,striving towards team cohesiveness by observing commitment, independent learning, motivational drive, and interest in the subject will be something to look for in people before accepting people in to a team. The key being the capacity to be an independent learner and the application of creativity to solve problems.

Acknowledgments

The UNM team for hosting such an interesting event

NASA for their sponsorship

Professor Bruce Smith for the support and advice

The MESA Director at our school, Raul Soto, for providing us a place to work and help analyzing our math and ideas