Save it, Build it, Run it Tech Report
NASA Swarmathon Virtual Competition
Montgomery College

Team Members:

Jalalah Abdullah	Daniel Levine
Denys Fedorchuk	Shayan Taslim
Denis Tra Bi	Suriya Iqbal
Rebekah Newby	Huyen Vu
Michael Bailey	Abhi Sharma
Michael Tang	Logan Wallace

Academic Supervisor:

David Kuijt

Abstract

The NASA Swarmathon Virtual Competition requires teams from schools around the country to develop a search algorithm for cooperative robotics. Our team focused on creating a server (hive) that allowed the rovers to communicate at the beginning of each round. The hive was created within the ROS environment and acted as a socket that allowed the rovers to send requests to it. The hive then responded uniquely to each rover. The hive was then used to determine the unique search paths for each rover and to allow them to communicate information regarding discovered clusters. The search algorithm implemented was one of simple zigzagging paths within each rover's boundary. The collision detection code was also alerted in order to optimize the behavior of the rovers after they collided.

Introduction

The original goal of the team was to establish what the code provided to us was capable of doing. A majority of the knowledge gained from this phase was uncovered through trial and error. Various aspects of the code were deleted or modified to see how the rovers behaved with the new code. In order to make this process simpler, Qt creator was the IED of choice.

The first thing that was understood was that the rovers were rather egocentric. They each thought that their starting location was the center of the map so any form of uniform movement was almost impossible to achieve. It was decided that before any significant modifications to the code could be made, the information that the rovers possessed had to be uncovered. Fiddling with how to subscribe to topics allowed us to display crucial information in the console.

After a baseline understanding of what the rovers were capable of was established, research was conducted regarding what algorithm to implement. Most of the research was performed using Montgomery College's online database. From the information gathered, our team decided that the rovers must search in an outward expanding path and that clusters had to be prioritized.

The work was split up into two teams. One team was in charge of optimizing the rover's behaviors so that they were more predictable, and another team was in charge of figuring out what the exact search path should be. The teams then met every weekend to share progress and obstacles with one another.

Discussion

I. Hive

Our first goal was create a communication network between the rovers. This was done to minimize the amount of collisions, overlaps within the rover's paths, and to priorities clusters. This was done using ROS's publishers, subscribers, and servers, (which work like a typical socket where you send and receive requests).

Immediately an obstacle was encountered. The competition rules did not allow us to modify the package where NASA's code published its messages (where we wanted to build the server). The solution to this was to initiate the server within the launch file that was provided to us. Another obstacle was then encountered as each rover launched this file upon initialization, creating multiple servers. The solution to this was to give each server the same name. This overriding resulted in one server at the end of initialization.

The server consisted of two parts, hive services and hive. Hive services are the jobs that the server can do and hive is collection of those services. Because the hive uses hive services, hive services had to be built before the hive.

The hive was then used to assign each rover a unique identification number and to count the number of rovers. The number of rovers was used to determine the size of the arena and the search paths of each individual rover.

II. Collisions

Our next challenge was to normalize the rover behavior of the rovers upon detecting a collision. The original code resulted in a chaotic scramble to avoid the collision and negatively impacted the colliding rover's search path. The collision detection class was altered so that the rover's misalignment upon returning to the search path and total distance traveled to avoid the collision was minimized.

This was done placing each collision in a stack. Each time a rover detected a collision, if would receive a collision goal location that was a short distance 90 degrees away from the direction of the collision. Once the collision goal was reached, the rover received a return goal to go to which was a location in between where the collision occurred and where its original goal location was. To minimize the amount of space that was left unsearched by this collision, the return goal was placed closer to the original pre-collision location than it was to the original pre-location goal.

III. Paths

Once the rovers had the ability to differentiate themselves from one another and acted properly upon nearing one another, the search algorithm was implemented. The main goal of the algorithm was to have the distance between the rovers and the distance constantly increase so that each cube located could be immediately returned to the center and have the shortest distance to do so. A simple zigzag between boundary functions y=x, y=-x, and x=0 then constructed.

Collisions only occurred while returning targets or nearing a boundary, but the chances were of that synchronization happening were so minimal that it barely affected the algorithms performance. An illustration of the three-rover and six-rover search paths can be seen below.

Unfortunately, our team ran out of time and never fully implemented the 3-rover path. The behavior of the rovers upon reaching the wall was never fully "mathed out". Instead the prelim round did simple square spirals with different starting locations and spiral width ratios.

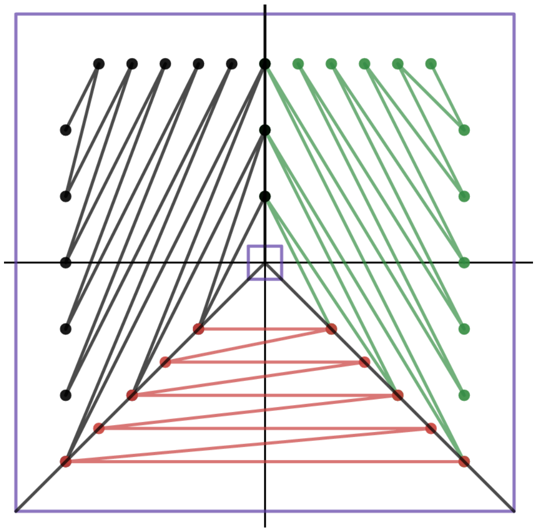
Figure I: Intended Three-rover path

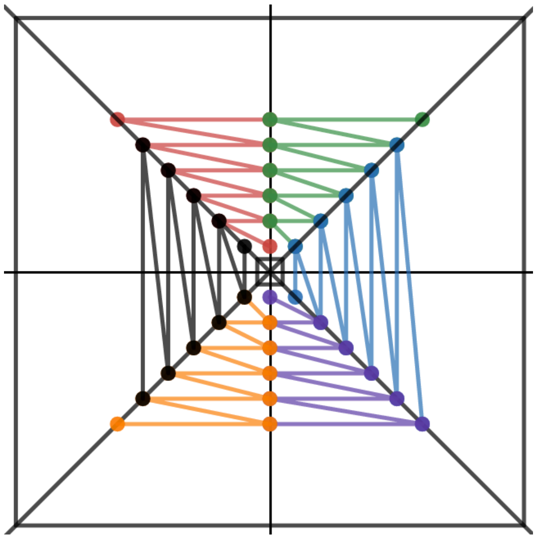
Figure 2: Six-rover path

IV. Picking Up Targets

Another problem encountered was that the rovers had difficulty picking up cubes. Because completion is time limited, a decision was made that after 5 seconds of trying to pick up a cube, the rover would give up. Thankfully, because of the way the rovers approach the cube, that has not occurred in our testing.

One huge obstacle that had been encountered, and not dealt with, is that cubes sometimes get stuck underneath the rover's claw. This blockage incapacitates the rover for the duration of the round. The only possible solution we could think was to have the rovers ram into each other until the cube got displaced. As much as we enjoyed that solution, our team never got around to implementing it, maybe next year.

V. Clusters

Because there was a possibility of a round with only clusters, the team needed to develop some method for the rovers to know when to use the hive to call the other rovers for help with a cluster. The decision was made that if a rover searched its path for five minutes without finding any tags, it would switch check the hive to see if a cluster had been located.

A cluster was defined to be any time a rover saw at least six tags at once, picked up one of the tags and brought it to the center, and then returned to where it picked up the tag and still saw six tags. Once a cluster was found, it was given higher priority dependent on how many times the previously stated cycle occurred. The higher the priority, the more rovers who found nothing would come help with it.

Unfortunately again, our team ran out of time and never worked out how to get the rovers to reach the called to clusters without crashing through the middle, so the cluster code was commented away.

Results

The rovers have done a very good job picking up cubes. For one, the rovers push targets near the center into the center when they return with ones they found on their search. The rovers also do a very good job returning to clusters and fully collecting the cluster. There is still some error with the precision of destinations, but most of it is unavoidable and related to aspects of the code we were not given control over. On average, a rover picks up and returns a cube every 30 seconds, and we are quite happy with that.

Conclusion

The team's main goal was to set up a communication network between the rovers. This was accomplished using the ROS server and allowed us go give each rover a unique search path. The communication network also allowed the rovers to communicate important information about clusters to each another. The combination of fixed search paths with dynamic reallocation of rovers to collect clusters worked very well, averaging a pick up every 30 seconds. To this point we have not tested a full round but look forward to doing it in the coming days. Some problems that still remain are that cubes get stuck within the claw and that the rovers are inaccurate at reaching their destination.

Our team learned a lot during this competition and every team member learned something different. We learned how to use environments ranging from Unix to Gazebo to ROS. We learned how to split work into teams and how to properly implement that code using GitHub. And we learned how challenging a simple task like picking up cubes can be, and how many scenarios we need to consider before we can solve a problem.