2017

NASA Swarmathon Technical Report

Team-Norco

Dona Sisk | Norco College | 3-22-17

Edward Abelian
Dan Haster
Michael Mendenhall
Nathan Montgomery
Derrick Uhrstadt
Jimena Garcia
Christopher Mckinley
Michael Montgomery
Thomas Schopper
Sherrie Zettlemoyer

This report was reviewed by the instructor, Dona Sisk, prior to submission

Nasa Swarmathon 2017
Technical Report

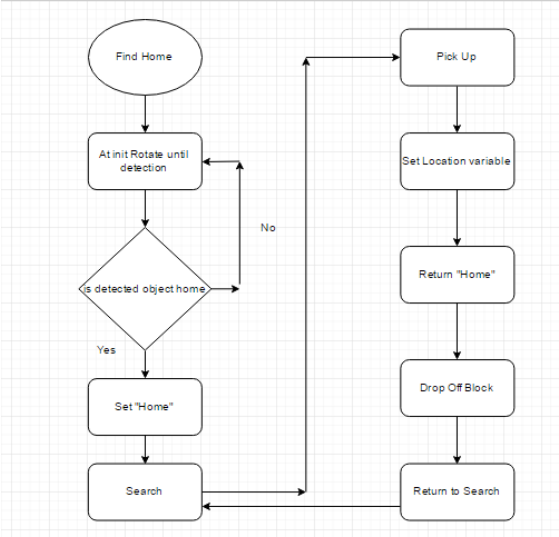
Team Norco

Abstract:
Through combined effort and determination our team has managed to decipher and improve upon the base code supplied to us for the NASA Swarmathon Competition. From beginning to end we faced technical challenges including the software and operating system required for the programming; through these obstacles we learned much of both our strengths and shortcomings as a team. The code we developed attempts to implement a linear outward spiral to cover as much of the arena as possible. We also generated code to have the swarmbots return to the location at which their search was interrupted when a tag was discovered.

I. INTRODUCTION

The goal of our program is to achieve a higher average block acquisition per round than the code supplied to us by NASA. Our goals were to make a more efficient code that would enable the rovers to find and retrieve resources along with remembering where the location was if multiple resources were found and go back to retrieve more until there were none left.

II. VARIABLES AND CODE



A. Variables Created by the Team

ResourceLoc

A Pose2D variable created to record locations where multiple targets were detected. The goal was to have the rover return to the set location if two or more resources were scanned at the time of search. This was far more efficient for cluster and power law distributions as the

rover would be able to repeatedly return to a location where resources were guaranteed to be found.

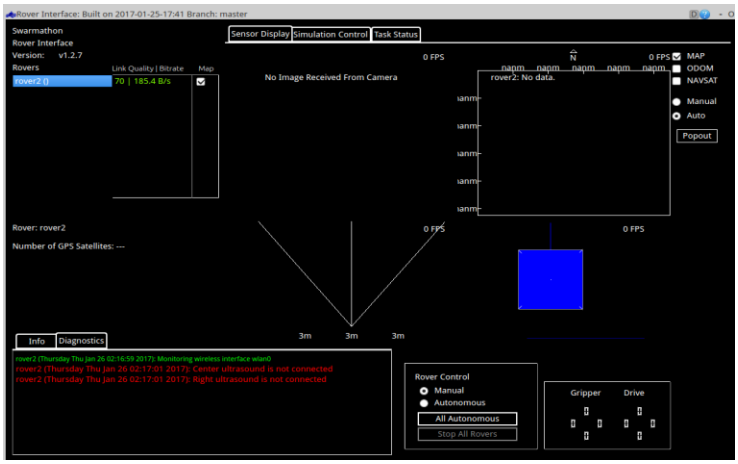
MultiTargetsDetected

A boolean variable that was made to determine whether or not a rover managed to scan two or more resources at a given location. Initialized as false, but set to true when the *targetHandler* managed to detected the required amount of resources simultaneously.

B. Code Generated by the Team

setResourceLocation()

A function in *PickUpController* that returned a Pose2D location variable called *ResourceLocation*. If the *nTargetsSeen* was within the set conditions, it would set *ResourceLocation* to equal *currentLocation* (which was sent over from *mobility)* and set passed from *mobility* by reference). Back in *mobility*, a variable called *ResourceLoc* was made to store that value for use.



III. ALGORITHMS

Search Pattern

The basic algorithm is based on the idea that each rover should do equal work when searching for resources. Each start off from their center location and extend a calculated distance to a position on the field where they then start their search. The distance covered by each rover should be about equal to each other without regards to the initial distance from their zero and the algorithm's starting point.

Return To Resource Locations

The rover was initially programmed to, as it repeatedly called back *mobilityStateMachine* on a timer, to set the *goalLocation* to follow the search algorithm (which was initially created to be random) after a certain amount of time elapsed without having found any resources or avoiding obstacles. That meant that after having found and dropped off a resource, the rover would eventually revert back to performing its programmed search algorithm. We felt that it was more efficient to let the rover return to a location where it was guaranteed to find more resources, if any were available. Hence, we made *ResourceLoc* to save the location of where it detected said targets. We then called the function *setResourceLocation* to set the *currentLocation* where the rover found the resources equal to *ResourceLoc*. Then we

set *goalLocation* equal to *ResourceLoc*. However, problems arose when our added code failed to function because *goalLocation* was repeatedly reset to follow the search algorithm instead. To counter this, we made *multiTargetsDetected* to determine where or not the rover detected multiple targets during its search patterns. Then we created the condition that the *multiTargetsDetected* must be set to true to return to the resource location. Otherwise, if set to false, it would simply revert to the search pattern.

IV. DATA

1. *Explanation of the Data Titles on the following tables*

Base Code- Code provided by the NASA Swarmathon Competition

Rev#- The relative code update number provided by the team

RovNo. – Average number of rovers in simulation

Time- The average length of the simulation

2. *Explanation of Testing Methods*

All experiments were conducted on the same computer under similar system loads to ensure consistent results. All tests were conducted multiple times with exception to one code revision which broke the movement ability of the rovers, and on the same day under all three tag distribution options in order to verify that under real test conditions the program will perform consistently and results were recorded immediately after in a Microsoft Access database so that all data may be referenced and compiled to see any abnormal variations.

3. *Response to Abnormal Variations*

Code Rev	RovNo.	Time	Blocks Collected
Base Code	3	30	26
Rev1	1	30	18
Rev2	1	30	18
Rev3	1	30	0
Rev4	1	30	24
Rev5	1	30	26

1) In the event that a test resulted in an unexpected result, having a variation of ten or more tags from the rest of the data set for that code revision an additional ten tests were performed to determine the cause and likelihood of this event happening again.

V. TROUBLESHOOTING

1. *Operating System*

Linux Ubuntu 14.04 (Trusty Tahr)

- Network Issues
  - The majority of computers used by the team were unable to use the wireless adapter in their computers

due to the operating systems kernel

    - Solution: Hook into the hardwired Ethernet connection.
    - Alternate Solution: use WiFi USB Dongle

- Installation
    - The BIOS of the team's laptops made it exceedingly difficult to actually install the operating system because the manufacturers have locked it down to prevent people from accessing it.
    - Solutions: Crash Windows during boot and force a restart from disk. A program called UNetBootLogIn is capable of booting the computer into the operating system from a ISO file.
- During install certain computers were unable to install the video drivers.
    - Solution: Purge and reinstall Nvidia Drivers

2. *ROS Indigo*

- Sim speed
    - Simulations running at .2 real time corrected by installing proprietary drivers
- Catkin errors
    - Catkin would not build directory, the team discovered that manually building the directory would allow "catkin build" to succeed in all tasks

3. *QTCreator*

- Compiler errors various
    - Reinstall QTCreator, reinstall ROS build directory, make sure compiler was properly set up

VI. SUMMARY

Throughout the many different ideas to come up with a more efficient way to search and find resources we have come up with a more efficient version of the code supplied to us by NASA. After comparing our results with the results gotten by running the simulation with the NASA

source code and averaging the two, we found that our code has made an improvement on the amount of resources gathered in the 30 minutes given time.

Given the chance, we would like to implement some more safety functions into the program, as well as fine tune the movements of the rover. For example, we found that the rover, when arriving at the *centerLocation*, would spend a lot of time looking for the center. We would have liked to program a different algorithm for finding the center tags, or at least try and speed up the process. Even when the rover finally finds the center tags, it could end up scanning the corner tags and drive right over the corner, rather than into the center. This would cause it to drop the tag outside of the center. It would also either drive parallel to the side of the center instead of into it and proceed to, once again, drop the tag outside the designated space for collection.

Finally, we believe that the most effective improvement would be the inclusion of Publisher and Subscribers. We feel that the rovers should work in tandem with one another, like the ants they mimic. By having the rovers communicate important information like resource locations, we could have the rovers work more efficiently in recovering said resources. For example, in a cluster or power law distribution, the rovers would not be forced to individually look for resources the entire time. If one were to find the location of resources, it could transmit the information to the other rovers, directing them to the same spot, thereby eliminating the need to search the entire map.

With these additions to the program, we expect a substantial increase in search and retrieval performance of the rovers. Unfortunately, due to the fact that the requisite class for the competition merely started in February, we did not have very much time to overview the code, figure out its mechanics, and apply our own changes. The issue was exacerbated due to troubleshooting errors and a lack of proper equipment. In the end, we only had a couple of weeks to program the rovers' algorithms.
We hope to continue working on the code in the weeks following the competition so that we may further improve on our understanding of the mechanics and networking abilities of robotic systems.

I.**References**

O'Kane, Jason M. A Gentle Introduction to ROS. N.p., Oct. 2013. Web. 15 Mar. 2017.

"The C++ Resources Network." Cplusplus.com. N.p., n.d. Web. 26 Mar. 2017.

"Qt Documentation." Qt Documentation. N.p., n.d. Web. 26 Mar. 2017.

"Home." NASA Swarmathon. N.p., n.d. Web. 26 Mar. 2017.