Morehouse College Swarm Robotics

Swarmathon Technical Report
Spring 2017

Morehouse College

Team Roster in Appendix A

Faculty Advisors: Drs. Dwayne Joseph and Amos Johnson

This report was approved and reviewed by Dr. Dwayne Joseph prior to submission.

NASA Swarmathon: Morehouse College

Abstract – **Implementation of robotics into everyday society is increasing exponentially. This increase includes the usage of swarm robotics, a field of robots with a coded mentality derived from the "swarm" instincts of insects like bees and ants. Swarm Intelligence is essentially the idea that a group approach to solving a problem or complete a task is much more efficient than an individual approach. This type of mentality is especially useful to NASA, and has already been implemented into ongoing missions such as the Mars Exploration Rover Mission. When the Rovers are conducting a task, there is a direct positive correlation with the success rate and the number of Swarmies involved, assuming the algorithm has been properly optimized. The task at hand for the students competing is to program the robots so that they can most efficiently search an area for blocks. In order for this to happen the students were required to conduct their own research and use prior knowledge from their S.T.E.M. courses. This report provides an in-depth analysis of the steps that the Morehouse students took in order to succeed in the NASA Swarmathon competition.**

1. Introduction

The NASA Swarmathon competition is a technical competition centered around developing search algorithms for robots that operate with "swarming" patterns and cooperation. Preparing for this competition was an opportunity for students to learn about the type of work done by computer scientists and engineers at NASA. The Morehouse Robotigers robotics team was very fortunate to receive 3 physical rovers to conduct real world tests of the code on. They were also responsible for attaching the grippers to the swarmies allowing the students to gain some added knowledge on the mechanical aspects of the robot. This year, the goal for Morehouse was to collect more blocks than last year. With the prominence of the brotherhood at Morehouse College, it was not difficult to recruit more students to help achieve this goal. With a team full of creative new minds the Morehouse Robotigers were able to create an algorithm with a high level of potential to succeed in the competition.

2. Related Work

Swarm robotics is truly a remarkable idea that builds upon the notion that primitive species can work together to accomplish large tasks. Most swarm intelligence research is inspired from how naturally occurring swarms (e.g. insects, fish, mammals, etc.) interact with each other within the swarm [1]. According to Dr. Ying Tan, a professor at Peking University of Beijing, China, swarm robotics is a fairly new approach to coordinating multi-robot systems to cooperate and achieve a common goal that consists of simple physical robots [2]. This implies that there is not a need for overly complicated robot systems; the actions of the individual robots adds to the complexity of the overall system. Before delving further into understanding swarm robotics, the nature of swarms and swarm mentality must be understood first..

A. Ant Swarms

Many organisms naturally operate in swarms from humans all the way to ants. To better understand swarm mentality it was necessary to do research on the practice of it in a natural environment. Deborah M. Gordon, a biologist at Stanford University, stated in an interview with the National Geographic magazine, "If you watch an ant try to accomplish something, you'll be impressed by how inept it is. Ants aren't smart, ant colonies are. A colony can solve problems unthinkable for individual ants, such as finding the shortest path to the best food source, allocating workers to different tasks, or defending a territory from neighbors. As individuals, ants might be tiny dummies, but as colonies, they respond quickly and effectively to their environment," [3]. The next question that was raised was, "how do these 'dummy' actions lead to such complex and intelligent solutions?"

The main sources of communication among ants are pheromone usage, sound, and even touch [4]. Ants sniff each other with their antennae to determine what type of ant is returning to the nest, where it has been, and whether it should join or choose a different job to perform. But the most important thing to know is that there aren't any particular ants telling any other ants where to go or what to do.

There are a set of instructions that the ant will naturally come to recognize and understand from these signs. That's exactly how swarm intelligence works; simple creatures following simple instructions stimulated by local information [3]. Doing so simplifies the actions performed by each individual ant, and it is obvious to see that, without the group, an individual ant would never be able to accomplish such a large task by itself.

B. Connection to Swarmathon Competition

The study of ants helped tremendously with the approach of how to program the robots. Instead of modeling the behavior of our rovers using a leadership structure, the Robotigers decided to model the approach to the competition after the basic ant model, where there is no clear leadership structure, just a set of basic instructions that each rover follows given certain situations. This approach, which other researchers have deemed credible, yields adequate results which are portrayed in the results section of this report.

3. Methods

During last year's Swarmathon competition, there were only six members from our college who competed on the team. For the 2017 competition, a total of fifteen students participated on the 2017 competition team. The increase of students from the previous year to this 2017 competition gave the Morehouse team an added advantage of dividing the tasks into several groups of students. The team mainly consisted of Engineering students along with a few computer science students. For the first meeting the students immediately decided which student would participate on what team. The three teams to choose from included the Assembly team (those who were responsible for putting the grippers on the rovers and fixing any hardware issues), the programming team (those responsible for editing and debugging the code that would be installed on the rovers), and the hybrid team (those responsible for connecting the linux machines to the rovers). After the assembly and hybrid teams completed their respective tasks, the members of those teams formed the outreach team (those who helped with a high school robotics team).

3.1 Assembly Team

The Assembly team was created to handle any physical modifications or repairs the rovers would need while testing various portions of code. The first assignment the assembly team was tasked with was to attach the grippers to each rover. The first issue the team ran into was connecting the gripper to the printed circuit board(PCB).The rover's PCB did not correlate with the manual visual instruction the team had been using with previous rovers. In effect, the rover had 4 sets of prongs protruding vertically from its PCB while the manual visual instructions only accounted for one set of prongs protruding horizontally. The team's second issue had to do with two rovers with various stripped bolts, and worn nuts with the front and rear fenders. These issues prohibited students from removing the battery, which also hindered the assembly team from interacting with the circuitry above the fuel cell. Additionally, the four support rods connected to the rover's circuitry cover had been broken and needed replacement.

The first step to attaching the gripper to the rovers was to remove all stripped bolts and nuts using a stripped bolt & nut kit, which allowed the team to unscrew the bottom carriage form the frame. Once unscrewed, the team had access to the rover's circuitry, which allowed them to replace the wires that were connected to the PCS unit prongs. Once the wires were replaced, the team was able to align and install the gripper. Replacing the broken support rods was a simple task. The team acquired the detailed drawings files for the support rods (from GitHub) and were able to 3-D print the replacement parts. .

Two of the team's 3 rover's had been deemed fully functional. While the final rover could have its gripper attached properly, the maneuverability of the gripper was hindered. After uploading the code onto the rover, it was found that something with the rover's front fender was stopping the gripper from having a full range of movement. The front fender was then installed properly and its gripper reattached.

3.2 Hybrid Team

To be able to run the team's code on each rover, the team had to learn how to "*-ssh*" into each rover from the linux devices. The first issue that occurred involved the rovers and linux devices being on a public wifi network, therefore connection between devices became cumbersome. Once we supplied our own router to connect the linux devices, we were able to locate the IP address for each device. We then faced a few version control issues between the code on the linux devices and the code on the rovers. For instance, one rover (No. 13) was fully operational; the second rover (No. 14) was able to establish a connection but none of the sensors were accessible; and the third rover (No. 11) had a faulty CPU that made it inoperable–further analysis indicated it could not receive power.

3.3 Programming Team

The programming team consisted of a few members who had coding experience. The programmers proceeded to download Ubuntu and ROS onto their devices before splitting into smaller subsets to better understand the code. Some members focused on how the search controller was implemented. Another subset of the team focused on how the *targetHandler* was called and how the search controller was affected by the *targetHandler*. A third subset focused on how *obstacleHandler* affected the search controller. Once the programmers became more affluent, they brainstormed different search algorithms before choosing the one they felt was best.

3.3.1 Hill Climbing Algorithm

The Hill Climbing (*HC*) Algorithm starts with a suboptimal solution to an issue (i.e. finding the best location for tags) then begins to iterate, evaluate and update the solution until the best solution is executed at every iteration. To implement this algorithm the rovers would first be assigned a random heading. After receiving a heading, the rovers would travel on a straight line for a random length. If a rover was to find a tag prior to reaching its set distance, it would collect the tag and then restart the *HC* process. Otherwise after reaching its destination the rover would begin to scan its surroundings. While scanning, the rover would evaluate a new position with respect to another and determine which route would be most effective. If a new position was deemed more effective than the previous, the rover would begin its *HC* algorithm again with the new position. Ideally, with each iteration, the rover would be able to determine where tags would most likely be located. Ultimately it was decided that the *HC* algorithm would not be best suited for us due to the limitations on time we had to implement it since the majority of the time would be spent on the implementation of a cost function. The biggest problem that we faced with implementing this algorithm was finding a way to quantify whether or not one position was better than the last. Instead, we focused our efforts towards putting a numeric value on the attractiveness of a position and decided to implement a structured random search algorithm.

3.3.2 Random Square Algorithm

The Random Square (*RS*) algorithm allows a rover to pick a random length between two meters and six meters (incremented in size by half meters). The rover then randomly picks one of four initial thetas. Those options include 0, $\pi / 2$, π, and $(3 * \pi) / 2$. After those two values were picked the rover would move *x* meters in the initial random heading. Once the rovers searched a distance of x, it would precede to turn $\pi / 2$ radians to the left and continue to search for another distance of x. The rovers again would turn $\pi / 2$ radians to the left (See figure 1). At this point the length would increase to $2 * x$ to ensure a greater portion of the field is being searched. The same process is repeated twice more with the new value of $2 * x$ (see figure 2). Once the square is almost finished, the length reverts back to *x*. The rover then completes the square by searching for another distance *x*, turns left by $\pi / 2$ radians and would return back to the original location by traveling a final distance of *x* (see figure 3). This code is based off of a counter that increases each time the rover turns, thus once the rover returns to the center the counter resets to 0. When the rover completes its first square, it will rotate π radians and then select a new distance *x'*. This algorithm is the default algorithm of the rovers. Therefore if the rover does not see a resource or obstacle (such as the wall or another rover) the code should run flawlessly.

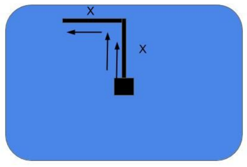

Figure 1

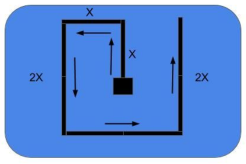

Figure 2

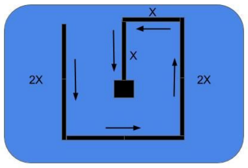

Figure 3

3.3.3 Obstacle Detection

In order to effectively optimize the *obstacleHandler* Function, we asked: "What is considered an obstacle when the rover is in the arena?" We concluded that there were only two types of obstacles: walls and rovers. From there we thought of different cases for each obstacle - for example, one case would be a rover running face to face with another rover. We focused on two possible cases, these cases that had the highest probability. The first case being when a rover approaches a wall. The second case being when another rover crosses a rover's path, however not face- to-face (we decided not to include the case of face-to-face rovers because the likelihood was low considering our search algorithm). In order for the rover to differentiate between a wall and a rover we decided to make the rover stop for three seconds then check if the obstacle is still there. If the answer was "yes", the rover would make a left turn and continue the search algorithm. To accomplish this we focused on the obstacle handler callback function and the machine state case in the main function. The obstacle handler gets called repeatedly when there is an object in the way of the rover. The first time the handler gets called the rover's velocity is set to *0* and a counter variable is initialized along with a flag. When the counter reaches *3* seconds another flag is sent that checks if the obstacle handler gets called again. If the obstacle handler is called again, that indicates the obstacle is a wall and the rover will turn. However, if obstacle handler is not called after *3* seconds, then the obstacle is no longer in front of the rover (i.e. a rover passed in front of it), and the rover continues its path.

4. Experiments

A. Swarmathon-ROS Simulations

In Swarmathon-ROS, there are several options that were at the disposal of the team. The first option was choosing the surface upon which the rovers would be moving: Kennedy Space Center concrete, parking lot concrete, or gravel. The group decided to leave the surface as Kennedy Space Center as a control variable for all the simulations. It was also possible to choose how the tags were distributed around the simulation zone using the uniform, cluster, and power law tag distributions. There were two different types of tests that could be conducted. The first was a preliminary test which ran for 30 minutes with a 15 x 15 simulation zone and 3 rovers. The second was a final test with a 60 minute run time, 6 rovers, and a 30 x 30 simulation zone. After the simulations were complete, it was time to move on to the physical rover testing.

B. Swarmathon-ROS Physical Test

When testing the first rover it would search the area, but it wouldn't pick up any blocks. After troubleshooting the issue it was determined that the wires in the gripper were crossed. Once they were in the proper location the issue was resolved. The rovers were tested indoors on linoleum tile on the floor and fluorescent lighting above. The supports on the rovers snapped multiple times which slightly delayed testing. After creating new supports, the team was able to manually control the rover around the room. Unfortunately the wheels were not spinning at the same speed which created difficulty testing out the code. The issues with the wheels were caused by bad Inertial Measurement Unit (IMU) calibration. The difference between the linoleum tile and the wheels of the rovers, as well as difference in lighting, caused by indoor lighting fixtures. Overall, the rover was successful in being able to detect objects and implement the code that was uploaded to it.

5. Results

As shown in the table, when using the uniform and clustered tag distributions, the rovers collected fewer tags and received the error message fewer times. When using the power law distribution there was a heavy increase in the amount of tags collected, but the error message was also received multiple times.

Table 1:

(Simulated) Preliminary Tests		
Tag Distribution	**Simulation Time**	**Targets Collected**
Uniform	30m 27s	23
Cluster	31m 34s	27
Power Law	32m 25s	41

Table 2:

(Simulated) Final Tests		
Tag Distribution	**Simulation Time**	**Targets Collected**
Uniform	60m 12s	44
Uniform	60m 43s	46
Cluster	60m 21s	48
Cluster	60m 34s	49
Power Law	60m 3s	51
Power Law	60m 29s	55

6. Conclusion

Overall our time spent with the rovers was productive. Splitting the team into two subsets- one for coding and another for assembly - allowed for time to be maximized on every aspect of the rover. In regard to coding, the decision to use a “no-leader” approach, where no rovers are in contact with another came about due to time constraints, inexperience with ROS and linux based operating systems, and lack of advanced programming knowledge. With the data, and experience the team has collected this year, it is clear that membership needs to be shifted from individuals with a physics background to individuals with a strong background in programming. In the future, it is the team's goal to execute the previously mentioned *HC* algorithm, and move away from a “leaderless” swarm.

7. References

[1] E. Bonabeau, M. Dorigo, G. Theraulaz. “From Natural to Artificial Swarm Intelligence.” Oxford: Oxford University. 1999. Press.
[2] Y. Tan, Z.Y. Zheng, “Research Advance in Swarm Robotics.” Science Direct, vol. 9. Issue 1. Beijing. pp. 18– 39.
[3] P. Miller, “The Genius of Swarms.” National Geographic Magazine. 2007. pp. 1-7.
[4] D. Jackson, F. Ratnieks. “Communication in ants.” Curr Biol 2006; 16(15): 570-4.

Appendix A - Team Roster

First Name	**Last Name**
Xavier	Bonner
Na’Qari	Bryant
Jeffery	Butler
Malik	Buton
Caleb	Davis
Nyles	Fleming
Trent	GIlliam
Ernest	Holmes
Justin	Johnson
Carl	Johnson
Tai	Lewis
Kgensu-Ra	Love El
Philip	Rucker
Jordan	Scott
James	Scott
Christopher	Williams