CAL STATE LA

CALIFORNIA STATE UNIVERSITY, LOS ANGELES

Cal State LA Swarm Robotics

Swarmathon Technical Report

Spring 2016

California State University Los Angeles

Team Roster in Appendix A

Faculty Advisor: Dr. Nancy Warter-Perez

Dr. Warter-Perez certifies that she has read this report prior to submission.

Swarmathon Technical Report
California State University Los Angeles

Team Roster in Appendix A, Faculty Advisor: Dr. Nancy Warter-Perez

Abstract- **The Swarmathon project was created by UNM and NASA in order to advance research in the areas of swarm intelligence and swarm robotics while introducing students to emerging technologies. The Cal State LA Swarm Robotics Team began the project by testing hardware accuracy and seeking to improve accuracy. Software development began with the creation of a skeletal class structure. With the ground work laid out the team focused on developing an effective search algorithm with hardware constraints in mind.**

I. INTRODUCTION

The Swarmathon project was created by NASA and the University of New Mexico (UNM) in order to advance research in the areas of swarm intelligence and swarm robotics. The competition was designed to determine which search algorithm could collect the most AprilTags, small QR code-like objects, using three rovers for a preliminary round, and six rovers for a final. The project was tackled as an in-house competition between two separate teams in hopes that a combined final optimal strategy would be achieved. Both teams worked on a separate strategy for their search algorithm, however the testing of hardware, experimenting and troubleshooting of the various rover limitations were performed by an integrated effort of sub teams, comprised of participants from both teams.

One team tackled the algorithm with a segmented approach, giving each rover a unique area in which to perform a lawnmower style search discussed further in more detail later. State machines were also implemented to optimize searching and retrieving, depending on the time remaining within the round and the number of tags identified.

The other team took a grid themed approach, moving the rovers out in a mathematical star shaped pattern to reduce redundancy in search area. The segments of the grid were further divided into micro grids, in which a lawn mower pattern was used again, with separate modes for uniform and cluster distributions and a priority tag queue to access more valuable tags for maximum collection per unit time.

After extensive testing and experimenting, a final algorithm was created that attempted to circumvent the most significant issues facing the rovers, which were localization and obstacle avoidance. Evolving from the lawn mower approach and segmentation, a simple segmented "bicycle spoke" shaped pattern was chosen to eliminate the contact the rovers would have with each other, as well as being the most "rotationally light" method to reduce errors in the localization data.

II. RELATED WORK

A. Swarm Intelligence (SI)

Nature based search algorithms were the primary influence when approaching the constraints of this particular problem. Ant Colony based research has shown that while ants have limited capabilities individually, they can for example, work collaboratively to find the shortest path between a nest and a food source by laying down a trail of pheromones [1]. The synthetic algorithm work that came from studying this behavior began as a possible means of optimizing the age old Traveling Salesman Problem, where like the ants, the salesman would be trying to cover the most optimal route to get to points without redundancy. Some of the more popular variations of SI-style algorithms are the Ant Colony Optimization (ACO) [1], Particle Swarm Optimization (PSO) [2], Synthetic Predator Search (SPS) [3] and various other combinations of those such as the Group Search Optimizer (GSO) [4] strategies. There are four basic principles that encompass an SI based solution.

1. Positive Feedback – Something that improves the likelihood of a good solution reoccurring.
2. Negative Feedback – Something that impedes the chances of a poor solution being repeated.
3. Randomness – A randomness in path to increase the likelihood of a new and possibly better solution being found.
4. Multiple Interactions – Multiple sources following on a path and other paths to increase communication and discovery of best solutions.

An approach combining several principles from these strategies was used for the initial EagleSwarm concept. An optimal routing strategy must be obtained to maximize the searched area within the course constraints as gleaned from the ACO approach. A dynamic calculation of the location of the rovers must simultaneously be published to reduce collision, and optimize movement relative to each other as seen in the PSO formation. A weighted value must be assigned to the discovered resources to insure that an optimized retrieval strategy is called when the rovers begin to uncover resource locations on the map, similar to the SPS approach. Finally, the option of dedicating one of the rovers to be a Producing member if heavily weighted resources are already being collected by rovers in a closer proximity, similar to the GSO method.

III. METHODS

A. *Overview of Major Classes*

Most of the work was done in the mobility package. As a group, we decided to follow the Robot Operating System (ROS) guidelines for the C++ programming language. Since the provided "mobility.cpp" file did not follow the ROS style guide, the mobility file was split into multiple classes to mitigate this issue. The development of a new mobility file was broken down into a Rover Class, Tag Class, and a Mobility class respectively.

1) Rover Class: The rover is a class that describes the state of the rover. The main features of the rover are current location, goal location, and a list of tags that have been found. The current location represents where the rover is located at that current time. The goal location is where the rover wants to be. The list of tags represents tags that have been detected and collected.

2) Tag Class: The tag class represents a tag object. One feature of the tag is position. The position represents where the tag was detected. The tag position is required in order for the rover to know where to return to retrieve the tag from in the future. Another feature is whether the tag has been collected. The rover needs to know if it has been collected so it does not try to pick it up again.

3) Mobility Class: The mobility class provides all of the call back functions and determines how the rover should behave. There were three major functions from the original code that required modifications. The search function and the target function were changed to fit our needs. The rovers' mobility search is based on the Eagle Swarm algorithm. The target handler deals with what the rover should do if a tag is detected by the camera. If the rover is collecting, pick up the tag and head home, but if the rover is searching, do not pick up the tag.

B. *Initialization*

The goal of the initialization process is to gather as much data as possible at the beginning of the competition so that the search algorithm can be called with all of the correct parameters and data sets. The initialization process has 4 functions:

1) Communications
2) Rover initialization
3) Map initialization
4) Tag storage initialization

The first area that will be initialized is communications between rovers. This is done through topic handlers, which comprise ROS topic publishers and subscribers.

Once communication among rovers is established, the rovers initialize themselves. During rover initialization, each rover will publish its own name and positioning information to a custom topic and at the same time subscribe to the same topic to receive information about the other rovers. The size of the set object can be acquired to represent the total number of rovers, which can in turn be used to determine the type of round the rovers will be in. Because all the objects within a set are sorted, the rovers are ranked, which will enable role assignment without prior knowledge of rover names.

The completion of rover initialization will enable map initialization. The initial position of all of the rovers can be used to approximate the center of the map and the number of rovers can be used to determine the type of round and map size.

Before the actual search algorithm starts, the tag storage will also be initialized, and be used to store and retrieve uncollected tags during the search. To prevent oversight and miscalculation, initialization data can also be updated and corrected in a later process.

C. *EagleSwarm Search Algorithm*

This approach combines the benefits of a random search and a more organized systematic search. There are two distinct methods of search, one optimized for a uniform distribution, one optimized for a clustered distribution. These two methods are further divided into versions for 3 rovers, and 6 rovers. The algorithm has the required logic to determine both the distribution of tags, and the number of rovers available for search.

1. *1) Uniform Tag Distribution Search:* The basis of the EagleSwarm search algorithm is within the uniform search procedure. This can be seen in
2.
3.
4. *Figure 1*. Each rover will first initialize and move into its starting position, labeled a1, b1, and c1. The layout for 6 rovers can also be seen in
5.
6.

Figure 1.

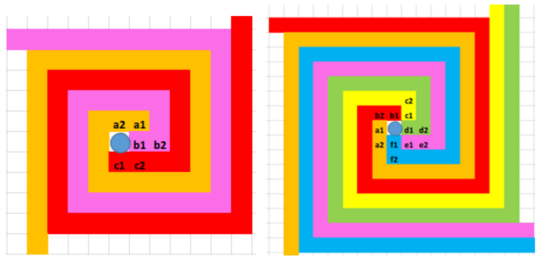

Figure 1 - Three Rover (right) and Six Rover (left) Uniform Search Pattern

After moving into starting positions, each rover will then orient themselves to face the path they will follow. This path is the colored lane on top of each rovers start position. Each square in the Figure represents a 1m x 1m sector. The rovers follow a simple formula in order to keep coding simple and organized.

Under this configuration, the rovers will slowly search outward from the center and cover the maximum amount of ground with minimal overlap. Any gaps within this search can be partially filled with returns to the home base.

Phase Two will have all rovers conducting a "lawnmower" tile search described later, in an individual sector to determine if there are any tags. Once a positive result is achieved, it is added to the priority queue and a quick local search is conducted for any additional tags. If none are found, the tag will be collected, returned home, and the rover will resume its search at the previous location. If more tags are found, these are added to the priority queue and the rover collects the nearest one. This is the core processing loop for a uniform distribution

2) Clustered Tag Distribution Search: Once the priority queue is updated to have a tagClusterSize > 3, the algorithm will switch into cluster search mode. The difference between a clustered and a uniform search is the addition of an outer perimeter "bot" which will abandon its lane and proceed to search the outer areas of the course. It will randomly pick a position and heading greater than 5m away from home base and then conduct a short, 20 second local search for tags. The objective of this rover is to find an elite tag source. The priority queue will automatically keep track of the number of tags corresponding to a position, and will guide the rovers to the location where they should look for tags. The 3 and 6 rover cluster searches are the same, apart from the addition of a second outer perimeter bot.

3) Quad Class: This class is paired with the TileSeach class to be defined next. While the TileSeach searches a 1 x 1 sq. meter tile, this class tells TileSearch which 1 x 1 sq. meter on the gird to search. The first few tiles are determined by the directional heading. This is due to the fact that the start of the square spiral pattern is slightly irregular as a result of the number of rovers. After the first few searches of the irregular tiles (three in the worst case scenario) the pattern then becomes repeatable for each rover.

4) Tile Search Class (Lawn Mower): As discussed earlier, the EagleSwarm algorithm works in such a way that we must search in a micro grid. These micro grids are each tiles that are 1m x 1m in dimensions. While observing the rover's cameras and making calculations it was determined that the rover was able to view a width of approximately 0.3m. Taking this into consideration it is possible to divide the tile into 9 smaller parts, such that their length and width is approximately 0.3m. Using this method it is safely possible to cover the entire area of the tile if the rover moves in the pattern as shown in Figure 2 below.

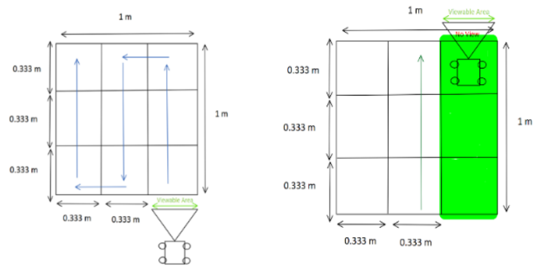

Figure 2 - Visualization of Grid Tile Search Movement (left) and Range of Coverage Area (right)

After following this lawnmower style pattern, the 1m x 1m is able to be scanned and searched and then end at the top left corner. The rover then reorients the microgrid to the bottom right corner of the next neighboring grid to search and resumes the next tile search.

5) Priority Queue: The priority queue was created to keep track of the best tag sources. A tag source is assigned a strength value, pushing it higher on the priority the higher the strength. The equation for strength of tags can be seen in Equation 1,

$$Strength\ of\ Tag = \frac{\#\ Of\ Tags}{Distance\ to\ Home + Distance\ to\ Rover} \quad (1)$$

which accounts for the two primary parameters related to tag cluster viability, the distance and number of tags. As the number of tags increase, the strength increases. As the distance to a tag increases, the strength decreases. This ensures that close tags are always collected first, unless the number of tags far outweigh the distance.

The goal of identifying strong tag clusters is to help the rovers work together. When a rover A has no tags to collect, it can be informed by other rovers that there is a tag at position (x,y), and rover A can proceed to collect that tag without ever having encountered the tag itself.

D. Segmented Spoke Search Algorithm

Due to difficulties with localization a second search algorithm that minimized the number of turns and collisions was developed. This algorithm is referred to as the Segmented Spoke algorithm because the rover's movement mimics that of a bicycle spoke. The course is segmented by the number of rovers initialized. For instance, two rovers would segment the course by 180 degrees. One rover would start at zero degrees and the other would start at 180 degrees. From there, the rovers move out towards the wall. Once the rover approaches the wall, it will turn counter clockwise, then move 50cm while hugging wall. The rover will then turn counterclockwise again to face the "home" heading of (0,0). Once the rover reaches its destination near the center, it will turn around and head back out towards the wall. All of the rovers will be also be progressing to the left to avoid collisions with each other. If a tag is detected, it will be immediately collected and then the rover will continue to search on the next "spoke".

IV. EXPERIMENTS

When the physical rovers were received, baseline testing and research was conducted on all the sensors and mechanisms that would be publishing data within ROS. Testing was conducted to discover any limitations or issues with the hardware, and to be able to troubleshoot and calibrate to maximize the rovers' final potential.

A. Sonar

Each rover has three Ultrasonic Distance Sensors, used for the distance measurement between any obstacle and the rover itself. These three sensors are placed in front of the rover; one is facing straight ahead while the other two are to either side of the middle sensor, both angled at 45° to their respective side.

1) Limits: The Ultrasonic Distance Sensor can detect an object within the range of .02 m and 3 m. Any value outside of this range is very inaccurate. The ultrasonic sensor has a burst frequency of 40 kHz and cannot detect objects that are too small to deflect enough of the ultrasonic burst, nor can it detect objects that cause the signal to bounce back at an angle.

2) Testing: The sensors were tested by placing objects around the rover at specific distances and having the rover rotate in place to examine each sonar sensor. The distance recorded by the rover was then compared to the theoretical distance. The distance measured was accurate with about a 7% margin of error. The error decreased when the rover was not moving and was facing an object for more than 3 seconds.

3) Obstacle Detection Problem: The obstacle detection problem, as a result of the sensitivity of the sonars, went unnoticed until near the end of the development phase. The rover would detect obstacles in the middle of nowhere, absent any physical obstacle and for no apparent reason. After careful inspection it was attributed to the sonar sensitivity and heightened obstacle detection.

4) Troubleshooting: As a quick solution, a feature was implemented that would trigger obstacle avoidance only when there were continuous calls. The resulting solution is not ideal, however time constraints required its implementation until another more elegant solution could be developed.

B. Visual Tag Detection

The rovers come equipped with a camera, used to symbolically pick up tags and return them to home base. The camera uses the provided AprilTag detection library to detect tags. It also publishes a picture through ROS to signify a tag collection.

1) Limits: The Logitech C170 camera onboard the rovers has a resolution of 1024 x 768 pixels rated at 15 frames per second (FPS). The low FPS hinders the detection ability when moving at full speed due to inherent blur. The horizontal viewing angle for the camera is 58°, but its useful range becomes limited as the camera is pointed downward towards the ground reducing the vertical viewing angle. The horizontal viewing range was also measured at 0.3m.

2) Change in Detection Rates: In the preliminary testing phases, image detection at full speed was marked at approximately 0% for single tag detection, and at about 27% at 25% of full speed. After examining the original image detection code it was discovered that the Sobel edge detection function was commented out. After editing the code, full speed image detection increased to 83% for single tag detection, however Sobel edge detection was removed in later code releases.

C. *Robot Localization and Coordinate Frames*

The robot_localization package is a library provided by Charles River Analytics, and provides nonlinear state estimation through sensor fusion of an arbitrary number of sensors [5]. The configuration of our sensors can be seen in Figure 3.

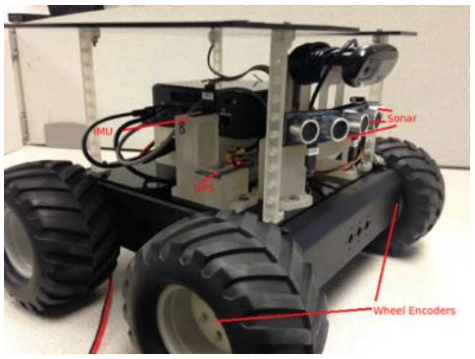

Figure 3 - Image of Physical Rovers Sensors

Navigation is one of the most important factors the project depends on. If the rovers do not know where they are, they will have no idea where to go either. By utilizing this package, data is obtained on the positioning of the rovers, and therefore positioning of the tags that need to be collected.

The main challenge of a state estimation package is the increasing error as time increases. Several attempts were made to mitigate this error, some were more helpful than others.

1) IMU: The rovers have an Inertial Measurement Unit (IMU). This device serves to report x, y, and z acceleration, as well as the orientation, with magnetic north as a reference.

a) Testing: According to ROS standards (REP-103), the magnetometer should read 0 when facing East. In addition, values should increase when going clockwise, and decrease when counter-clockwise. These are the only requirements and constraints for the IMU.

b) Troubleshooting: Initial testing had shown that our rover orientation was reading a “0.5” while facing east. Even though the IMU was very precise, this violates the requirement of the robot localization package. This had the effect of confusing IMU and GPS headings, as they no longer matched each other, resulting in a very confused position estimate. A second problem was revealed when the Swarmathon Technical Team announced we should calibrate the IMU’s. This was also a major contributor to the confusion of our position estimate.

c) Results: The orientation problem was fixed by adjusting the yaw_offset value found in the Swarmathon Arduino code. Originally thought to be set in the rover_onboard_launch, it was found that the Swarmathon Arduino code was actually responsible for this measurement. The yaw_offset can be calculated as shown in Equation 2 by

$$0 = yaw\_offset + sensor\_raw\_value \quad (2)$$

Once this parameter was adjusted, readings were in line with expected values. This had the result of reconciling the IMU and GPS headings to match the rest of the system, giving an overall greater output to the state estimation. The second fix came with the IMU calibration. All 3 rovers’ previous calibrations were far off from the new calibration, with an average difference in minimum values of -759 and an average difference in maximum values of 196. These two changes fixed the IMU inaccuracy problems while maintaining more precision in measurements.

2) GPS: The rovers utilize a GPS Click, manufactured by U-Blox. The GPS is a core component of our state estimation system.

a) Testing: Testing consisted of taking the rovers outside and driving along a predetermined track. The pathway was then compared to the output of the Rover GUI. The predetermined paths were of multiple types, including squares, octagons, zigzags, and random movement.

b) Troubleshooting: Baseline testing was conducted by driving the rover along a line that traced a square. An example output is shown in Figure 4. The yellow square represents the actual path, while the red line represents the GPS output.

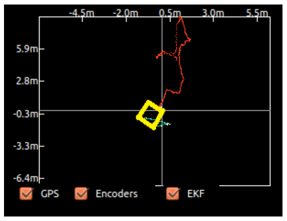

Figure 4 - Actual Path vs GPS Data

The initial discrepancy was coming from an unknown error. While the GPS understood the velocity, the orientation was pointing in the wrong direction in every test. This posed the problem of how exactly we can utilize a GPS signal if it does not match the real world position of the rovers.

c) Results: The following conclusions were made: The first error of the GPS headings not matching the real headings was explained by a magnetic declination setting. This can be explained as a δ difference between true north and magnetic north as seen in Figure 5.

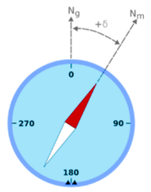

Figure 5 - Example of Magnetic Declination

The original stock settings of the rover had a configuration much different than one required for Los Angeles. After adjusting this setting, GPS readings became more accurate. Figure 6 shows the output when this problem was fixed.

Note that the output in Figure 6 varies greatly when the rover is stationary. This is because it is not raw GPS data, but rather converted coordinates into an x-y coordinate frame using a Kalman Filter.

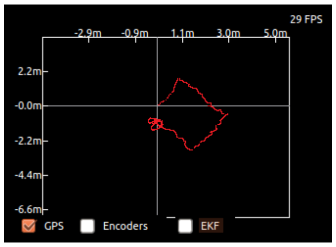

Figure 6 - GPS Output After Correction of Magnetic Declination

3) Kalman Filters: The Kalman Filter is responsible for the mathematical calculations predicting our rover's position. With two official options for our Kalman Filter, extended (EKF) and unscented (UKF), further testing was conducted to determine which of these filters to use. The testing setup was identical to the testing for GPS.

a) Testing: Baseline testing for both filters proved difficult. There seemed to be huge error in both the encoders and GPS. An example output of the square test can be seen in Figure 7. The red line traces the actual route, and the green traces the filter output.

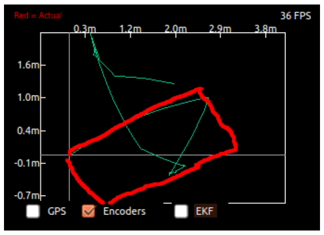

Figure 7 – Kalman Filter versus Actual Route

The filter and encoders seemed ill-equipped to handle any turns where a pure rotation was present. No matter what kind of path was taken, the error was always greater than 1m on the x axis and 1m on the y axis. Both EKF and UKF outputs were tested as well, however there was little difference between outputs, contrary to the literature and math which state that the UKF makes the filter more stable [6].

b) Results: After correcting the GPS declination setting and calibrating the IMU, a new baseline test was conducted. After initial tests confirmed a square pattern was fairly accurate, the test moved onto more complex patterns as seen in Figure 8. The initial position can easily be compared to the final by way of where the green line stops, compared to the origin.

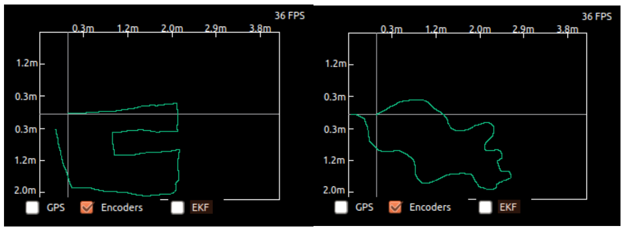

Figure 8 - More Complex (left) and Varying (right) Routes.

The accuracy of the EKF output once the GPS and IMU issues were fixed, improved significantly. There was an average standard deviation of approximately 0.4m. However, this did nothing to fix the problems of increasing error as time passed, and position error eventually increased past 0.4m and far beyond.

Testing moved on to adjusting pose and twist covariance. The wheel encoders were being trusted too much, with a covariance of 0. In order to add more data from our GPS and IMU, we increased these values. There was very marginal improvement seen in overall accuracy, but the drift was definitely slowed and allowed almost an average of a full extra return home before diverging too far to recognize. The limit to adjusting these covariance values, was seen when EKF output began to become increasingly noisy as to confuse the rovers with big, rapid position changes. This limits us from increasing the covariance too high. Results can be seen in Table 1.

Table 1 - Covariance Adjustment Data

Covariance	x(m) %error	y(m) % error	Returns home (mean)
0	0.45	0.45	1
0.05	0.4	0.4	1
0.1	0.4	0.4	1
0.15	0.35	0.35	1.5
0.2	0.35	0.35	1.5
0.25	0.35	0.35	1.5
0.3	0.4	0.4	1.5
0.4	0.4	0.4	1.5

V. SIMULATION RESULTS

With the difficulties in getting the hardware running successfully, simulation runs were both effective and useful for data collection. The two major search algorithms, EagleSwarm, and Segmented Spoke, were both tested through simulation to judge their effectiveness. The first algorithm, EagleSwarm, was defined by its approach of maximizing covered ground and minimizing overlap of rovers. This proved to not be very effective due to innate inaccuracies in rovers, both in simulation and in real hardware. As seen in Table 2, the average amount of obstacle avoidance calls was very high in the EagleSwarm search, the leading cause of its ineffectiveness.

Table 2 - Results of Simulation Runs

Algorithm	Distribution	% of Tags	Average Avoidance Calls
EagleSwarm	Cluster	7.77	1073
EagleSwarm	Uniform	14.1	2878.7
Segmented Spoke	Cluster	12.96	533.4
Segmented Spoke	Uniform	27.3	780.1

With this new data in mind, the team sought to construct an algorithm that had similar ideas, although this time with an emphasis on avoiding the obstacle avoidance call. The resulting algorithm proposed was the aforementioned Segmented Spoke algorithm. This algorithm was much more effective in keeping the rovers functional and on task, rather than avoiding other rovers. These results can also be seen in Table 2. Inevitably, because an efficient obstacle avoidance protocol was unable to be implemented and the precision in the robot localization issue was not solved to a satisfactory extent, the final choice in algorithm was the Spoke method. By maintaining a consistent collection rate by circumventing some of the precision issues, it was the simplest and safest choice.

VI. CONCLUSION

There were numerous areas of development required for this project, each requiring much research and work to implement. Understanding ROS and state estimation were two of the biggest factors that impacted the Cal State LA Swarmathon Project's level of success.

A. Understanding ROS

Topics are one of the most important concepts to grasp in ROS. The sensor data is communicated through topics. The map data is shared through topics. Timers, objectives, states, and goals can all be set, controlled, or monitored through topics. Topics also offer a convenient way to debug software; one can see how accurate or inaccurate data processing truly is in real-time.

B. Understanding State Estimation

Navigation is an essential area that must be optimized. The positioning of rovers reflects not only their ability to navigate themselves around, but also the quality of information they share. If rovers have inaccurate position estimates, the data they share with other rovers is almost useless. With accurate positioning, everything else in the Swarmathon project is improved including the search algorithm, the ability to share information, and the ability to collect tags consistently. This remains one of the key final components that needs to be optimized.

There are several ideas that are in the process of implementation:

1) Sensor agreement: Getting the sensors to report data in the same coordinate frame.

2) Obstacle Avoidance Protocol: Implementing an efficient movement strategy algorithm for rovers that come into contact with each other by establishing a global position. Several options were discussed, but time did not permit a working implementation.

3) Enhanced Version Control: One of the hardest problems with the Swarmathon project was successfully putting together multiple pieces of code. Inconsistent documentation and uploads prevented the team from organizing code as optimally as possible. This problem came to a head when compilation time arrived and many problems had to be ironed out during the later stages of the project.

REFERENCES

[1] G. Sharvani, N. Cauvery and T. Rangaswamy, "Different Types of Swarm Intelligence Algorithms for Routing," *International Conference on Advances in Research Technologies in Communications and Computing,* pp. 604-609, 2009.

[2] Y. del Valle and G. K. Venayagamoorthy, "Particle Swarm Optimisation: Basic Concepts, Variants and Applications in Power Systems," *IEEE Transactions On Evolutionary Computation,* vol. 12, no. 2, pp. 171-196, 2008.

[3] A. Linhares, "Synthesizing a Preditory Search Strategy for VLSI Layouts," *IEEE Transactions On Evolutionary Computation,* vol. 3, no. 2, pp. 147-153, 1999.

[4] S. He, Q. Wu and J. Saunders, "Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior," *IEEE Transactions And On Evolutionary Computation,* vol. 13, no. 5, pp. 973-990, 2009.

[5] ROS Wiki, "ROS/Introduction," Wiki.ros.org, 2016. [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 21 March 2016].

[6] ROS Wiki, "catkin/conceptual_overview," Wiki.ros.org, 2016. [Online]. Available: http://wiki.ros.org/catkin/conceptual_overview. [Accessed 21 March 2016].

[7] ROS Wiki, "robot_localization," Wiki.ros.org, 2016. [Online]. Available: http://wiki.ros.org/robot_localization. [Accessed 21 March 2016].

[8] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," *Charles River Analytics,* 2014.

APPENDIX A – TEAM ROSTER

First Name	**Last Name**	**Major**
Christoph	Anderson	CS
Aram	Atamian	EE
Sergio	Castillo	EE
Saul	Castro	CS
Jason	Green	EE
Holly	Griffiths	ME
Layla	Habahbeh	CS
Ray	Han	CS
Abner	Hernandez	EE
Donna	Hernandez	EE

Ceasar	Jimenez	CS
Kyle	Kinsey	EE
Jose	Lemuz	EE
Brian	Martinez	EE
Mariah	Martinez	CS
David	Rojas	EE
Jonathan	Sahagun	CS
Josh	Saunders	EE
David	Trejos	EE
Jonathan	Valladares	EE