Autonomous Resource Retrieval
Cabrillo Robotics Club,
Cabrillo College,
Aptos California

Student Team Members:
Kyle Park
Jaben Melville
Melissa Pardo
Andrew Boyd
Angel Martinez
Mitasha Malhan
Christopher Billingsley
Mauricio Rivera
Jim Turner
Garrett Mason
Kenny Murray
Christie Frushie
Gracie Harder
Zach Goulden
Billy Elisalde

Academic Supervisor:
Michael Matera

***Abstract*—This report documents the changes that the Cabrillo Robotics team, participating in the NASA Swarmathon Virtual Competition, made to the mobility package to improve the rover's' ability to navigate and collect targets in a virtual space. Special attention was paid to optimizing the ways the rovers navigated the environment, approached their goals, with scaling rotational and linear velocities, remembering food locations, and obstacle avoidance. Changes were tested in the simulation and run multiple times to test for bugs and anomalies. We found that decision making was an important feature to implement in the code which took a careful balance of desired objectives and variable settings. The things we thought at first would be most important to optimum tag collection e.g. maximizing the priority of food collection had to be complemented with things like obstacle avoidance and navigation of congested areas.**

I. INTRODUCTION

Our team's goal was to understand the Swarm rovers' behavior and create an optimized search algorithm to scout the area efficiently. Although we were beginners to collaborative programing, the experience felt like we were professional software engineers charged with taking an existing codebase and using it to develop a program with a real application. After watching the Webinars and example software demo videos we were all very excited to dive in and get programming, but it took a lot of time to get everyone up and running. Our Robotics Club was responsible for getting Cabrillo College STEM students to be a part of this competition and most of the participants were drafted from this club. Because of the excitement from being a part of a NASA robotics competition and the programming nature of the event, there were soon many computer science students willing to help from our lively STEM study center.

In order to run the simulation, code, and work to make changes from the school computers or from our personal computers, our academic advisor helped us preload USB flash keys with Ubuntu, Eclipse, and some bash aliases to start the GUI. Ubuntu was a new and exciting for most of us. Several tutorials through GitHub resources were necessary to get a better understanding of the theory, significance and use of version control to code as a team in a productive manner. The full learning came from the experience of actually using it to integrate our code. After a couple of weeks of getting used to all the different elements necessary to run, code, and test the rovers behavior, we began to develop small but steady changes to our code, improving tag collection efficiency.

Cabrillo Robotics Club organized three regular meeting times per week: Monday, Wednesday and Fridays. Here we would examine the code, observe the related performance, and discuss our objectives. To encourage people with little to no programming background we used "pair programming" methods pairing a novice with an experienced programmer. This way those with less experience had mentors to help them in developing code solutions to the problems we encountered. We evaluated rover performance through extensive testing and observation trials, before deciding how to go about making changes to the code. We split up the work of modifying certain parts of functionality to pair programming sub-teams early in the week in order to be more efficient. The novices were also given the job to test the code of different branches every day we met for effectiveness and give an update to the owner of each branch. After thinking about last week's observations and challenges over the weekend, the teams then proposed solutions and implemented code to attempt to fix problems from earlier in the week, testing them at the end of the week. When changes were thoroughly tested to show the rovers' behavior and tag collection numbers were improved, we would merge and push our code to our master GitHub branch.

Our advisor, Michael Matera, was a great source of help and motivation. His knowledge and enthusiasm fueled our potential to grow as a team and encouraged us to stay committed to the project. We were also very excited by the possibility that our program and code and could be implemented into future NASA rover and space exploration research and development. The fact that we being were given the honor and privilege to work under NASA and the University of New Mexico was a huge morale booster and propelled excitement through the competition for the whole team. Being that we had a time and the project was open ended, we felt like we had an enormous potential to succeed and learn a ton along the way.

II. RELATED WORK

We were inspired by watching videos of the latest in robotics from Boston Dynamics[1]. Robots have the interesting effect of making people notice the complexity of tasks humans accomplish effortlessly when they see highly advanced robots clumsily mimic what has been perfected through biological evolution. It put some of our challenges with getting our rovers to behave properly in perspective. At the same time it was inspiring to see what can be done by developing robots that are able to do the task of humans or, in our case, a small colony of robotic ants. Kyle and Jaben read part of the book recommended, "Programming robots with ROS" [2], which discussed how much of the prepackaged code was highly complex open source software. Having the valuable API functions of image detection, odometry, GPS and communication available to all those interested in making developments in the field of robotics without having to write all this from scratch will enable more rapid advancements in Robotics. We also learned, although most of us were learning the fundamentals and just starting programming classes that professional programmers often inherit code to develop and improve as part of their job.

III. METHODS

A. Distance to Goal/Angle to Goal/ Desired Heading

We began the project by organizing and attempting to understand the code in mobility.cpp. We noticed that several constants were used in multiple places and we started by

determining their significance and naming them (See TABLE I). We also noticed that some calculations were done and redone in different places in *mobilityStateMachine()*. We factored those calculations into three variables (See TABLE II). Our refactoring made the code more readable and greatly improved our understanding of the states in *mobilityStateMachine*(). Also, with constants properly named we were able to experiment with values to improve overall driving performance.

B. wtfAmIDoing()

The first challenge we had as a team was sorting through the hundreds of lines of code from the source code and figuring out how these rovers operated. To help us modify how the rovers acted while holding targets, we separated the high-level decision making from the rest of the navigation processes in Transform State by adding them into a separate function we called *wtfAmIDoing*().

C. State Machine Additions

We added some additional states in the state machine that enabled us to expand our functionality. To aid in collision avoidance at the goal location we added the *BACKUP* state, which caused the rovers that were returning home with a target to backup if they encountered another rover within 3 meters

TABLE I.

Constant	***Unit***	***Description***
c_TRANSLATE_THRESHOLD_ANGLE	Radians	If the angle between us and the goal is bigger than this the rover goes from *TRANSLATE* to *ROTATE* to fix the angle and drive on
c_ROTATE_THRESHOLD_ANGLE	Radians	The allowable difference between the target angle and the real heading. When the angle is smaller than this the rover goes from *ROTATE* to *TRANSLATE*.
c_ROTATE_THRESHOLD_ANGLE	Radians	When wandering pick a new heading that's on average this many radians from the current heading.
c_ROTATE_THRESHOLD_ANGLE	Meters	When wandering pick a new goal that's this many meters along the random heading.

TABLE II.

Variable	***Unit***	***Description***
distance_to_goal	Meters	The distance between our current location and the location specified in *goalLocation*
angle_to_goal	Radians	The shortest angle between our current heading and the goal. This may be positive (when the goal is on left) or negative. This angle is used in the TRANSLATE state.
desired_heading	Radians	The shortest angle between our current heading and desired heading as specified in *goalLocation*. This angle is used in the ROTATE state.

from the center of the arena. To aid in our rovers covering a larger area, we implemented a *CIRCLE* state, which the rovers enter while they are wandering. By providing the robots the ability to break out of most of task oriented states after a set amount of time we could prevent them from getting stuck in futile continuation of an objective.

D. Remembering Food Location

The next major hurdle we tackled was finding a way for the rovers to "remember" where they had last seen a target so they could return to that location. We set the variable *foodLocation* when the target was acquired and verified by the GUI that it had not been recorded yet and, in the *wtfAmIDoing* function, set the goal to that location. This change helped us dramatically in clustered mode, but a significant problem arose when the rover ran out of targets in a pile and remained in the last location indefinitely. To solve this we implemented a *giveUp* counter to reset the *foodLocation* and continue wandering.

E. April Tag Detection

The initial version of the code forced the rovers to only detect one target in an image at a time. To improve this feature, we added a loop that checked all the tags in the image to ensure we returned to the location with a large number of tags. Implemented within this is our fix to allow the scoring system to verify that the tags we are sending and the tags the scoring system is detecting match up.

We started to work on an algorithm for placing all targets seen during scouting and collection into an array to create a map to find areas. We started to implement this by creating another node in the code that would contain an array of seen target coordinate information. This would necessitate knowing from what distance the targets are first recognized, what direction the rovers are facing, and their location using odometry possibly in combination with GPS. This biomimicry idea would be similar to what bee or ant insect colonies do to keep track of food locations. Because the fact that any node we added would not be compiled or used in the competition we did not make further use of mapping but may want to do this next year if the rules would allow us to make a new package with this function or we could write it into some other existing node(s).

F. Proportional speed and turning

The speed the bots would translate to a desired location and the turning rate was able to be adjusted by changing the values of these variables. We wanted to accelerate to the desired locations that were farther away and then slow down once they got closer. This faster rate of travel would enable the bots to cover more ground when traveling long distances and maintain accuracy to see targets to collect. Also going slower in areas of high traffic would reduce the likelihood of collisions occurring by allowing the obstacle avoidance time to work. To implement this solution the rate of speed (c_SPEED_SLOPE) was multiplied by the *distance_to_goal*. This change didn't work right away because the rovers needed to maintain some speed and the fact that computation of their speed decreased to zero as they became close to their targets would cause problems. A maximum safe speed was needed to avoid any

collisions or shoot past targets. Now the proportional speed is calculated when it is more or less than these variables. This improvement allowed the rovers to pick up many more tags when tested in different competition configurations. (Fig.1)

Using the provided code the rate of turning towards the desired heading could be increased or decreased by changing the variable's constant. We observed the rovers turned too slowly or too fast and slipping past the desired heading with the momentum from the rapid turning velocity. Our solution was to make a rate of turning that was proportional to the desired heading (Fig.2). The greater the angle the bot was facing relative to the desired heading (up to 180 degrees), the rate of turning would increase towards the desired location. This allowed the bots to turn rapidly through the larger part of the angle difference and then slow down approaching the desired heading, allowing momentum and reduce the tendency to overshoot the intended direction. This change increased the accuracy of the direction of travel, therefore increasing the likelihood of returning to the same pile of tags which sped up collection rates. This function using trigonometry and with similarity to the proportional translational speed equations, the rate of turning speed required steps in the slope of the equation to make a minimum and maximum turning rate.

Fig.1

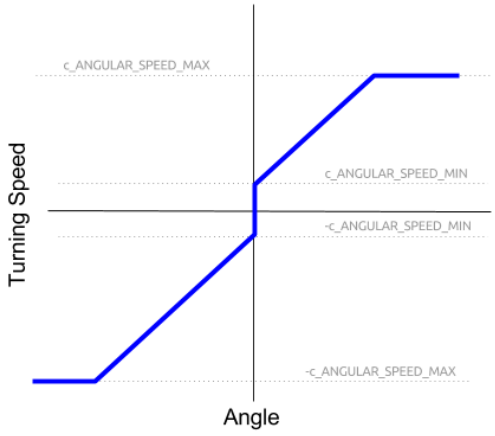

Fig. 2

G. *Turning while translating*

To allow a more dynamic adjustment to the rover's travel direction we decided that we should implement code to allow the rovers to recalculate their direction while translating. This was done by connecting the now finely tuned *desiredHeading* to be considered by *wtfAmIDoing* continuously in translational mode. This helped the rovers smoothly arc back to the intended direction if they still had error in their initial trajectory path. We had to adjust sensitivity of this by scaling up and down the variable. Implementing this function allowed the rovers to hone in on their *desiredHeading* efficiently, and without intention arc around the paths of other rovers teaming up on a pile of resources.

H. *Collision Avoidance*

One major recurring conflict was getting stuck along the walls. We solved this by adding a state in the state machine that prevented the rovers from getting stuck crawling along the walls. The function first checked if there was something large in front of it and then checking if it was not carrying a tag. It was unlikely to encounter a wall if it was heading back home with a tag, they could deduce that the large obstacle was a wall. If both of these conditions are true, then the rover will turn 60 degrees and continue searching for tags.

The other problem we would often encounter was rovers slamming into each other, sometimes getting stuck, unable to pass. This was a problem especially with a group of tags close to home and a higher density of rovers in the area trying to collect and return the tags, leading to collisions. We solved this by implementing a break out method into the state machine.

For many problems that the rovers encountered with obstacles we had them give up on their objectives after a certain amount of time and change states. By breaking out of the routine they were able to reorient themselves and reevaluate their *desiredHeading* which seemed to fix most jams.

IV. Experiments

Our method for experimentation was incrementally executed following code development. Statistical analysis diagnostics was made possible on the fly during testing by changes to our code that output "chat" statements to our bash. This way we could pause the simulation, check the output statements on the bash that would direct us to the part of the code that was last trying to operate and insert a timed break out if the rovers were trying to translate and had not moved for 15 seconds.

By outputting tag collection and simulation run time, we could compare the results statistically when we tested. By changing the simulation speed in setup world, we could run more simulations to get more data about how our rovers were performing. Although it was necessary to do fine tuning and look for errors such as stuck rovers and wheel slip at speeds closer to real time to observe what was going on and for how long.

We did basic mapping of these results dividing up the parking lot area into a grid and plotting the location of the clustered or power law tag piles to observe relative behaviors

between rover performance when large amounts of tags were close and farther apart. Some of our changes made big gains for some layouts and were terrible in others, which showed promise which only required minor tweaking.

Also useful was the ability to pick up the rovers and put them in situations we were trying to develop methods for helping them get better at. If we wanted to see how they were behaving around tags to collect, if they were seeing the tags and from how far, we simply could grab them and place them at varying distances from the objects we were curious about their responses to.

V. Results

We immediately noticed that the rovers numerous problems navigating congested areas. So we could optimize the rovers' functionality for the most difficult situations we ran most of our experiments using clustered target distribution. Early on we made a big improvement in performance when we separated the navigation from the decision making by adding the function *wtfAmIDoing*. From there we tried improvements and to develop functions to mapping target pile locations that could be communicated to the other rovers. During the last weeks of April, we struggled to make significant target collection improvements with these changes. We decided to work on optimizing the code we were given from there on because we found out we were not going to be able to have additional packages compiled for the competition. Making rate of speed and turning proportional to the distance and angle proved to help overall efficiency when tested, 3/7/2016. Obstacle avoidance improvements tested on 3/11/2016 helped the robots not get stuck along the walls. Observing behavior showed us how and what code caused the rovers to get stuck so including the timer give up breakout routines to these states allowed them to free themselves and proceed. Turning while translating gave the rovers a smooth arc in traveling to their destinations and this added the benefit of them curving around each other's paths to and from tag piles when they were going for them at the same time. All these incremental improvements helped move our code to be optimized to the point where the rovers were picking up most or all of the tags in the times given for preliminary or final round competition. See Fig.3 for our results.

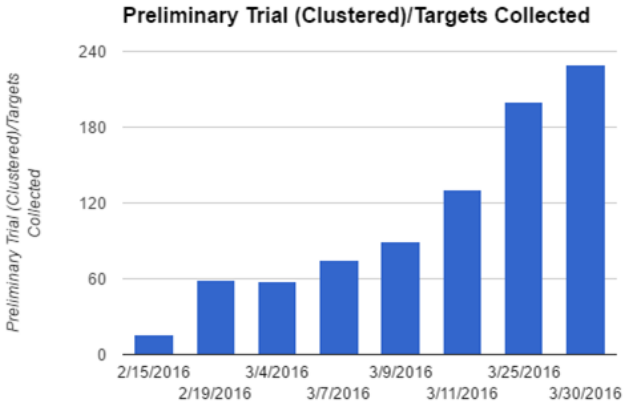

Fig. 3

VI. Conclusion

Our changes to the code were generally focused on mobility, which made much more of a difference than we expected. As important as well designed swarming algorithms are to making the rovers as efficient as possible, getting the rovers to move about in the most time efficient manner possible made huge differences. It was incredibly inspiring to see that we could make a huge change in the performance of these robots with nothing but first year programming knowledge. On top of that, being able to work with NASA and the University of New Mexico team? That just makes it even better.

It was also very nice to get experience working with a pre-existing codebase and in a team. Lower level courses such as the ones community college students take don't really give much experience working with pre-existing libraries. They also very rarely give us an opportunity to develop software in a team setting. Both of these skills are very commonly used in professional software engineering jobs. The vast majority of programming is done in a team setting, and it is very rare to start from the ground up completely when developing software. In addition to giving us the wonderful chance to work with NASA, it has also been great to learn development skills that can be directly applied to a professional setting.

VII. References

[1] http://www.bostondynamics.com/index.html, Web, March 16, 2016

[2] Morgan Quigley, Brian Gerkey and William D. Smart, Programming Robots with ROS, O'Reilly Media, 2016