# Team Synchronized Swarmies
# Technical Report
# Pasadena City College

## Faculty Adviser: Jamal Ashraf

I, Jamal Ashraf, certify that this document has been reviewed and approved for submission.

## Team Members:

Anthony Guerra, Jarly Arcinsega, Stanley Chen, Karen Chu, Kyle Dean, Chan-Soo Kim, Xianmei (Sammi) Lei, Kevin Macias, Brandon Palomino, Kristiana Rendón, Joe Renzullo, Eli Selkin, Alex Su, Jason Wang, David Wu, Timothy Yao

***Abstract*—Team Synchronized Swarmies has overcome a number of challenges and has learned extensively along the way. As the first-ever robotics lab at PCC, the Swarmathon project presented new and interesting difficulties to team members - many of whom had never worked with robotics before. Much of the early effort in designing algorithms and approaches was predicated on the ability to know the robot's position with a high degree of accuracy. When it became clear that this would not be possible within the scope of this competition, given the tools and time available to the team, a re-appraisal of methodology was needed. The team successfully changed strategy to build rover behavior from the ground up, and has improved on the navigation routines that the rover uses. Additionally, the team developed a methodology for controlling the rotation of each wheel individually, to facilitate non-linear movement.**

## I. Introduction

Team Synchronized Swarmies has developed novel ways of operating the swarmie robots. This has been done despite the difficulties the team faced, and our results are shaped by the constraints we discovered. Some of the challenges faced were purely technical - none of the members involved with the team had significant experience with robotics. For almost all of us, it was our first time working with the hardware/software negotiation. Additionally, there was something of a leadership vacuum for the project at the outset - because nobody knew what needed to be done, nobody was in a position to guide the process, and much effort was expended in often contradictory directions. Group dynamics were at times problematic, and many people left the project. What we have been left with at the end is a core of dedicated individuals, who have self-organized, self-motivated, and have made commendable progress.

The project was greeted by participating classes with much enthusiasm. Over 90 people signed up to be part of the private team chat server - and the excitement was palpable. Immediately, the Python and C++ classes set about designing algorithms that could govern the movements of the rovers. There were a number of very interesting ideas that emerged from this process. Some teams were interested in cordoning off the map into concentric circles, and exploring arc-segments within these. Other teams were working on spiral search - in particular Fermat's spiral with curvature equal to the golden angle (~137.5 deg) leads to an even coverage of an arbitrary map, no matter the center of exploration. Other teams were interested in more esoteric ideas, such as sinusoidal movement. Development of these algorithms was on a per-group basis. Some were more successful than others, and for the first few weeks the teams were working on their own, more or less.

At this point, focus shifted to writing code that could be run on the rovers themselves instead of code that just worked in a vacuum. It was soon clear that much of the work and research that had already been done would not be possible to implement on the rovers. 100% of the algorithms that had been developed assumed that the robots would know their position with a high degree of accuracy. And even those which were a little more fault-tolerant required that the robots use nonlinear movement (e.g. spiral search).

The rovers are not capable of moving in nonlinear ways very naturally. They stop, turn, and then continue in what they think is a straight line. (Depending on the calibration of the rovers in question, this too may not quite work out as expected.) This meant that the spirals which were being planned were not being executed well, and the time-lag involved was immense. At this point, some members of the team began to work on the necessary changes to be able to run the mobility control in Python. Many of the more actively involved members of the team were doing this as a project in a Python class. This facilitated more rapid development of ideas and algorithms, and has broadly been a very positive move; however, there are still a few issues resulting from this translation.

Then work began on exploring how the rovers could be made to move in nonlinear ways. By modifying the code on the Arduino, our team was able to control the rotation of each wheel independently. Using this, development on movement algorithms proceeded, and one of the groups of students came up with the idea of using a force-matrix to govern navigation. This allowed for the straightforward incorporation of information from multiple sources, and the weighting and combination of these forces determined the behavior. It is an extensible method, which will allow for the creation of different priorities that the rover can then focus on - some forces can be attractive (positive magnitude) and others can be repulsive (negative magnitude). The robot can then calculate its own best vector, and travel along that. As information changes dynamically, this can be incorporated, and the rover's movement can curve around obstacles and toward goals.

## II. Related Work

ROS learning relied on the ROS documentation[1] as tutorials for beginners. Nobody involved with the project had experience with ROS going in, so we were at square one, and made use of what we could find. Also useful were the ROSPy tutorial series[2], as well as some printed monographs, for instance “Programming Robots With ROS” [3] and “ROS by Example”[4].

The prototype of structure of rover’s movement was inspired by “Finite-State Machines: Theory and Implementation.”[5] This discusses how to apply finite-state machines to automatic processes in order to move in an environment under certain expected conditions. We established the following fundamental states based on the circumstances we knew our rovers would be encountering: searching, going home, returning to the previous tags location, and avoiding.

We also referenced “Understanding Steering Behaviors.”[6] This discusses the use of force vectors to determine the direction rover should move. It provides several examples of how a combination of forces can generate a resultant direction, seamlessly incorporating information on multiple (often contradictory) priorities, with low computational overhead.

Some open source materials and tutorial videos online helped to form our thinking - especially with respect to the topics of robotic motion with artificial intelligence, which are (at this time) completely absent from the curriculum at PCC. Some of team members attended an online course: “Artificial Intelligence for Robotics.”[7] This introduces concepts of certain algorithms and mechanisms which we experimented with in our process of development, for example, PID controllers, Simultaneous Location and Mapping (SLAM) algorithms, and Kalman filters. Covariance matrices for calibration of sensor data was suggested in the article “Covariance Matrices with A Practical Example.”[8] We also explored using some filters in the arduino code for getting more precise data in order to eliminate localization error. The articles “Kalman filter vs Complementary filter,”[9] as well as open code “Example Sketch for IMU including Kalman filter”[10] were considered extensively by team members, who wanted to translate the approaches described to our context in order to ameliorate some of the frustrating issues we were encountering.

Finally, a good deal of the learning that occurred during this project was from looking into the code base provided by UNM. Establishment of publishers and subscribers in mobility.cpp were very useful for building the ROS base of our code. State machines were well implemented in mobility.cpp, and we were able to build on that foundation to add our own functionality. We made use of the “rotate” state and the “translate” state in our code.

## III. Methods

We were exploring the idea of using force vectors to help simplify our movement calculations. Before attempting to incorporate the forces into our mobility functions, one of team members implemented state machines in javascript for his web development class. This was primarily done to ensure that the force vector idea wouldn’t clash with the finite state machine we had already planned. Figure.1 below is a snapshot of his simulator.

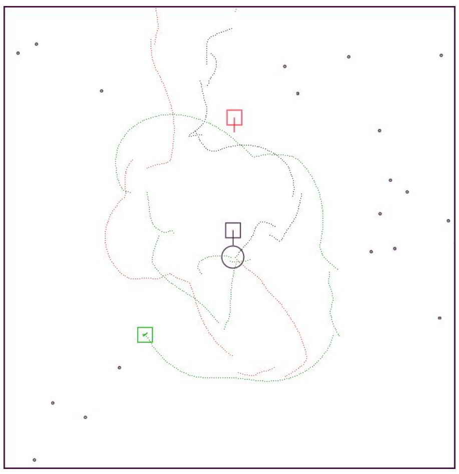


Figure.1 Simulator in javascript

Taking advantage of Python’s jump table, a method to create a menu directing to different functions without a number of if-else blocks, we are able to implement the state machine in a different way. We created a class called “Stack” with functions to operate functions in an order in the stack as a tuple. This let us avoid needing to check the conditions for each iteration. Instead there is just a callback function on a timer calling the function at the top in the stack if stack isn’t empty. Also implementing a new state is as simple as writing your function and pushing the function name into the stack if the conditions for its execution are met.

We developed the targetCallback function in order to read tags and decide what to do with them. It takes input from the camera which is represented by the tag’s ID number and a 2D array of unsigned char for the raw image data. When the rover is trying to go home: If the tag is new, it is queued in a list; If the tag is old it is ignored. When the rover is not going home; if the tag is new it is queued and picked up; if the tag is old, but not picked up yet, it will be picked up. Otherwise it is ignored. This solved how to deal with seeing multiple tags per trip - previously the rovers could do this, but they didn’t save it in a queue, so what this added is the ability to go pick up something later that it saw already.

A rover is driven by vectors introduced by forces and added on its current coordinate. For example, in Figure 1, rover 1 is in goHome state since it just finished the search state. State switching is as pictured in Figure 2. State stack has “search” and “goTo” by default: Figure 3.1. When “goHome” is added to the stack, “swirl” , “dizzy” and “goTo” will follow if it does not hit its target right away, shown as Figure 3.2. If the rover arrives at (0,0) but fails to detect the home tag, it will spin in “dizzy” mode: Figure 3.5 and if it still doesn’t see the home tag, it will explore the four corners of a square spiral with side length 1 meter (initially).

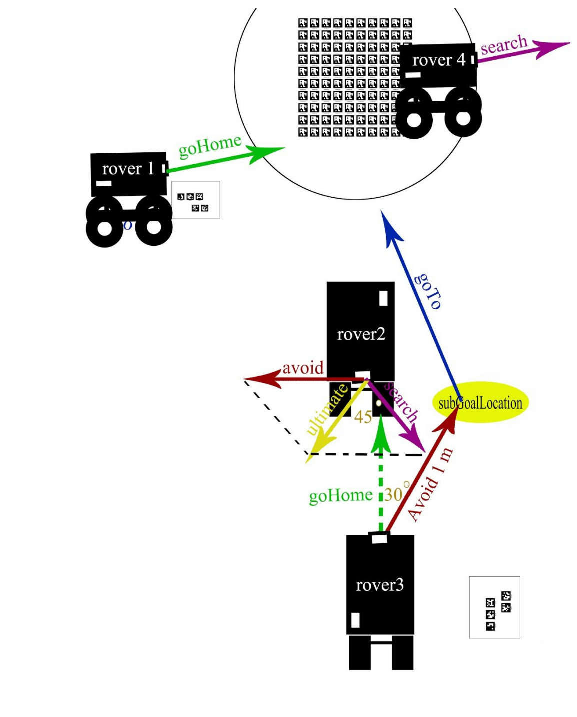


***Figure 2: Four situations of states switching***
Rover 1: search -> goHome
Rover 2: search -> avoid
Rover 3: goHome -> avoid
Rover 4: goHome -> search

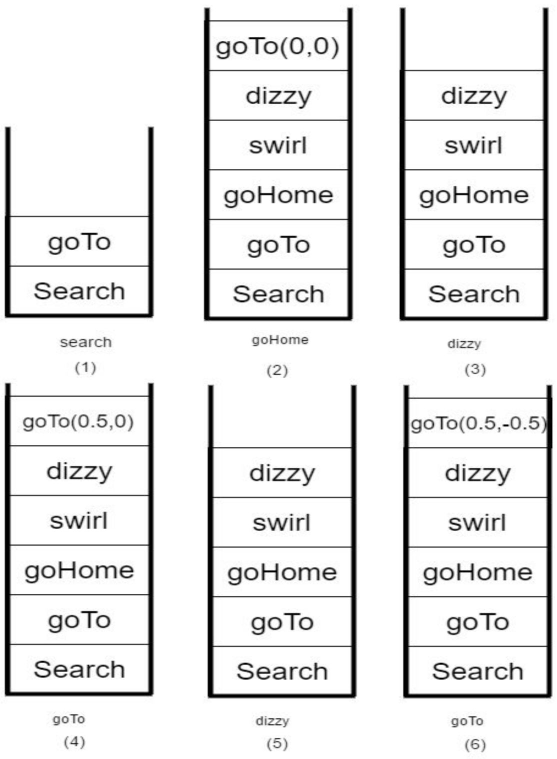


***Figure 3: state stack: search to goHome***

When a rover switches from goHome to Search(Rover # 4 in Figure 2), when it reach home(0,0), all states associated with goHome will be popped out, then the rover will be assigned a random vector to add on to its current coordinate: Figure 4.

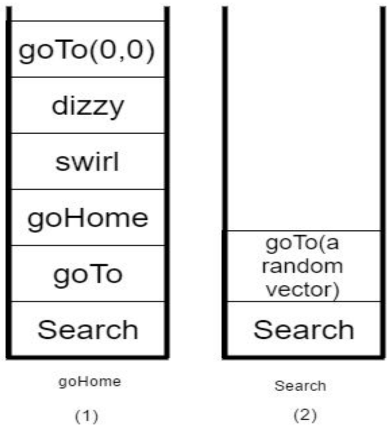


***Figure 4: state stack: goHome state to Search***

If a rover is in a search state (Rover # 2 in Figure 2) and tries to pass through ground occupied by another rover in front, the avoid state will be added to stack. The rover will have a new avoid vector (red) as 90 degree combined with search vector (pink) and ends up traveling along a new vector (yellow). (Figure 5)

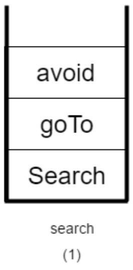


***Figure 5: state stack: search to avoid***

When a rover (Rover #3 in Figure 2) meets another rover on the way home, the avoid state will be triggered and the goTo state will be pushed with a sub location at 30 degree off the original location, in order to avoid the other rover. After reaching sub location, the rover will continue goHome state, reassigning goTo(0,0): Figure 6. This is one of the most important improvements we made over the original code, in terms of the performance of our rovers.

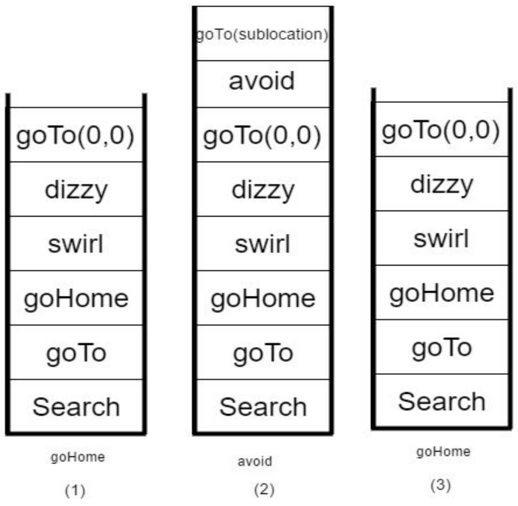


***Figure 6: state stack: goHome to avoid***

## IV. Experiments

Our test results from rover motion experiments showed that our initial hypothesis was wrong. We did not fully expect the robot to go straight when we send it a go signal. But from our test results we encountered a much larger than expected drift. Our experiment is to test viability of motor controller on arduino. Our experiments on rover 23 showed on average a 20 centimeter drift for every meter traveled.

Our experimental procedure is as follows.

1. Upload the default arduino code.
2. Verify arduino is operational.
3. Set the rover on marked starting line.
4. Mark the ground with a piece of tape perpendicular to starting line, measuring 10 meters
5. Align the robot to perpendicular of starting line
6. Flip the motor switch to off position.
7. Send in rostopic pub message.
    a. rostopic pub -r 10 /swarmie0/velocity geometry_msgs/Twist '{linear: {x: 0.3, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
8. Flip the motor switch to on position
9. Watch as the rover travels 10 meter distance
10. Stop the rover by flipping the motor switch
11. Measure the distance from the parallel tape to center of robot and note down the distance as drift.
12. Repeat step 4 to 11 to collect more data

13. Repeat step 4 to 11 with different linear velocity, by changing x value in step 6

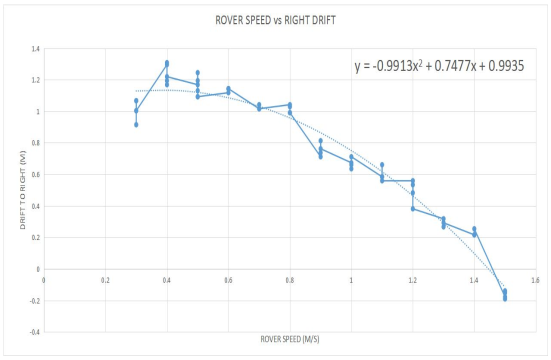
Our experimental results yielded great results of correcting drift by editing the arduino code. But further tests showed that other rovers has different drift profile and our results could not be used without modification. This was a huge roadblock for us, for more than a month. It meant that all of the work we had done on mapping algorithms and higher-level behavior was not currently implementable, and that we had to go back to square one with most of our work.

## Drift calibration Testing:

1.Arduino test

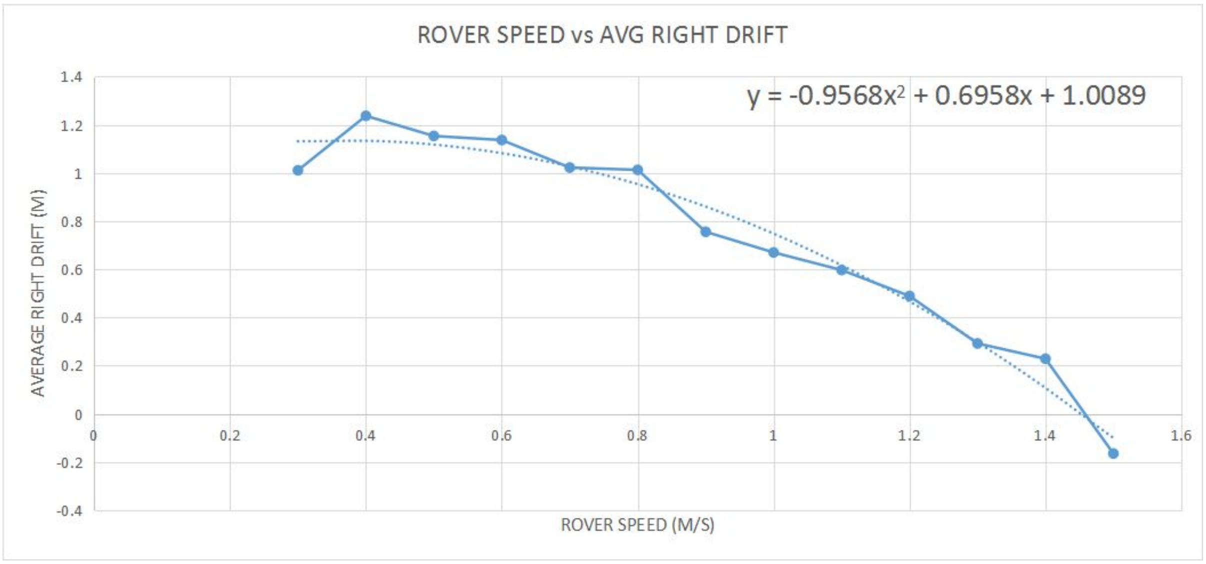
raw data:

https://docs.google.com/spreadsheets/d/16j8ks_QugHC6Nb89f11tXxkG4k6N4s_nt9vQTdQgbAA/edit#gid=0



1.These measurements are recorded from Rover 23 that had modified arduino code;

2.left wheel speed was all set to * 0.95.



The experiment described above is by far the most mathematically analyzable of the ones we conducted. Many of our other experiments were qualitative in nature.

For instance, when we set out to characterize whether we were capable of returning home (once the calibration code was pushed from UNM), we didn't seek out to make absolute measures of the inaccuracy as we improved our algorithms. Instead, we ran tests to see if we were off consistently in one direction, or whether our errors were random. This was a significant result - all of the errors we were seeing with simple "out and back" journeys were biased in one direction for each rover. This suggested that the issue lay with the calibration and not with our algorithm. This was particularly obvious when we discovered that the movement bias was different for each rover.

Further hand-tweaking of the IMU values using a procedure akin to the one described above (in 14 steps) led to our ability to reliably return home and removed much of the sensor drift we were experiencing with the odometry.

We also discovered that it was safer and easier, if we needed to introduce error in our goTo function, to do so in a way that undershot rather than overshot the mark. This is especially significant when we consider that the rover will be trying to avoid collision with the walls and with other rovers. It is easier to fix and undershoot error than an overshoot one - and if we are likely to get stuck against walls, then we especially would like to avoid hitting them!

We also ran experiments to ensure that the behavior of our finite state machine was as expected. We found that the behavior was suboptimal in the case of a collision avoidance call - the function would overwrite the overall destination and not keep track of what its goal was! So instead, we worked on implementing an alternative state (avoid) and augmented that with a separate avoidGoalLocation to avoid overwriting the total goal. So, when the rover reaches its avoidGoalLocation, it then pops back to its goHome location, and ends up going back where it should have been heading the whole time, if the obstacle had not intervened.

We also ran experiments to see what sort of random pattern we could implement when we missed a target, that would bring us closer to it and likely let us detect it. We experimented with spiral patterns, but these rely on curving motions, which the robots do not do very well. (They stop every time they need to turn, and they need to turn constantly in spirals). So we tried other shapes, and settled on a box around the rover’s current location. If we have danced around that box and still not found the target, then we increase the dimensions of the box and keep looking. This proved effective in most cases when we were near home but not quite detecting it.

Finally, we ran experiments to figure out what the optimal value would be for our undershooting, described above. Our initial guess for how large this value should be was 20 cm. This almost universally led to the robot stopping short of its goal and entering the dizzy state (and then spiral searching after) which was very inefficient. It would usually find a large target this way (e.g. the nest) but it was rare for it to actually re-detect a small tag it had cached. So, we experimented to discover the optimal value for this undershooting distance. Eventually, we settled on 5 cm, which improved the results dramatically, while still having the additional padding room for collision avoidance that we wanted.

This last experiment was typical of our work. We encountered problems, and had to come up with tweaks we could make to our code (or to its order of execution) to try to overcome them. Toward the end of the project, most of our experiments were qualitative: e.g. “Is a .95 speed multiplier right, or is .93 better?” Here “right” and “better” are fuzzy terms that measurement would not have much helped us to satisfy. In hindsight, however, some of our quickest progress came when we sought to measure things quantitatively, and that will be among our suggestions to teams going forward in the competition next year. Measure early, measure often.

## V. Results

The results section of a paper such as this is typically the preserve of statistical analysis of concrete improvements made by changes to algorithms and configurations. The work we undertook this semester has had some definite results that we can quantify.

For instance, the code we started with was able to return home none of the time on purpose, and occasionally by accident - now we are able to return home the majority of the time for simple trips (over 90% of 0-10m trips ended successfully at the nest in our testing).

For instance, the rovers were initially drifting by up to 20cm per 1m traveled - our own attempts at manually tuning to correct for this were relatively successful, and reduced the drift to < 5 cm per 1m traveled. (A later code update by UNM also helped with this, through the IMU calibration - our approach was to alter the speed at which certain wheels turned).

For instance, the rovers initially could not turn and move at the same time. We developed a method that would allow us to do this, by extending the Arduino code, and were able to implement curved movement for the rovers. This ended up introducing other complications. Due in part to those complications, but also to the difficulty in altering the Arduino code while still getting the advantage of the tuning done on each rover by UNM/NASA prior to the competition (which we found to be vital in our own testing) we decided to scrap this.

But our work can be built upon if those two challenges are overcome, and there are definite advantages to doing so.

For instance, the rovers initially could not avoid an obstacle and remember where they were supposed to be going. The collision avoidance was implemented in a way that overwrote the location goal, so that if a robot was meant to be going home, it would forget what it was doing until triggered by another event (most often, avoiding something else again, like a wall it was about to run into). This meant that the rovers very rarely made their way home if they were in cramped settings (and our classroom testing area was certainly that!) We developed tools that would allow us to maintain this information while still implementing effective collision avoidance behavior.

However, other parts of the results of this project are not so easily put into numerical form, but it is our view that these results are at least as important as the numbers sketched out above. When we started this project, none of our members had ever worked with robots. At all. Now, we all have had a very good introduction to the subject. And we know our way around (and, unfortunately, into) some of the more common pitfalls.

When we started this project, none of our members had used git or github to organize code as a team. Its advantages were soon made clear, when trying to manage in-development versions of Arduino code and ROS code on 3 different machines, simultaneously being worked on by two classes working in different programming languages. Eli and Joe helped out where they could in that respect, and as a result of the work put in by the team members with the information they provided, now most of the members of the team are comfortable using git to manage code.

When we started this project, everyone expected that the primary challenges would be in the algorithms we were developing (for ideal robots behaving themselves). Now we know better.

When we started this project, we designed our algorithms in a “waterfall” method - which resulted in a very long and uninformative feedback loop. This killed the motivation of a large number of the early participants in the project, a number of whom decided not to continue with the project when their hard work and research turned out to be impossible to implement. Toward the end of our work together, we had migrated to a more “agile” method of development, with a tighter loop between problem, solution, test, and analysis. The gap between the hardware and the software that we had experienced at the beginning of the project was finally beginning to close.

The learning curve for a project like this is large no matter what your circumstances as a team are. We think that we have grown tremendously through the process. We have done our best to document the pitfalls we encountered, and will be passing that knowledge on to the group who will take up the challenge at PCC after most of us are gone. We have been encouraged by the response to our first outreach event to hold several more in the coming days and weeks. Seeing the eyes of the kids whose lives we touch light up is probably the best result any of us could have hoped for.

## VI. Conclusion

Pasadena City College - Team Synchronized Swarmies - are, today, of one mind. This has been a fantastic experience for everyone involved, despite the challenges, and despite the long hours producing work that often led nowhere or required too much reworking to be worth further pursuit. When we were discussing the results of our project, we came across a number of things.

We learned how to manage teams and personalities - and that friendships are hugely important to the success of projects like these. We learned how to use git and github to manage our code and easily change versions on physical hardware being shared by multiple people. We learned that data is not perfect (very unlike the examples we had been trained to solve by our texts and courses) and that an open mind is required to find ways of surmounting this. We learned that we need to adapt our goals to match what we as a team are able to do. We learned how to use tools like vim and ssh and Linux more broadly. We learned how to negotiate with robots - and that the best stance is one of compromise and enticement, not of demands and force. We learned about the structure and function of ROS. We learned how to communicate information from one rover to another using ROS topics. We learned how to keep running averages of unreliable information like GPS readings and apply filters to them to improve their utility.

More important even than all of that, though, we learned the value in reaching out to those who will come up after we are gone. We learned the importance of outreach, of inclusion, and of diversity. We did not end up implementing all of the functionality that we planned to in the first few days and weeks of this project. Robots are hard. But we have developed the kind of knowledge that will serve us well in our futures, and we are doing our best to pass that on.

## VII. References

[1] Isaac Saito, “Document,” Wiki, 11 January 2016.

[2] Dirk Thomas, “rospy,” Wiki, 31 December 2013.

[3] Morgan Quigley, Brian Gerkey, William D. Smart, “Programming Robots with ROS,” O'Reilly Media, November 2015.

[4] Patrick Goebel , “ROS By Example,” 10 January, 2014

[5] Fernando Bevilacqua, “Finite-State Machines: Theory and Implementation,” Envotatotuts+, Envato Pty Ltd, 24 Oct 2013.

[6] Fernando Bevilacqua, “Understanding Steering Behaviors,” Envotatotuts+, Envato Pty Ltd, 2012.

[7] Sebastian Thrun, “Artificial Intelligence for Robotics,” Georgia Tech, 2012.

[8] “Covariance Matrices with A Practical Example,” Mania Labs, Mania Labs, 6 August, 2012.

[9] robottini, “Kalman filter vs Complementary filter,” robottini, digitalnature, 25 September 2011.

[10] Lauszus , “Example-Sketch-for-IMU-including-Kalman-filter,” Github, GitHub, Inc., 2 Sep, 2014.