NASA SWARMATHON

TECHNICAL REPORT

College of the Sequoias

(COS Team presenters in picture from left to right: **Alexis Contreras, Ahsen Baig, Cory Lewis, Paul Gonzalez-Becerra, & Gerald Jumper**) Other contributors not pictured: **Zachary Stafford, Sabin Adams, Davis Hang, Graham Frazier, Aaron Johnson, Isabel Lambert and Rachel Owens**)

Faculty Advisor: John Redden Staff Advisor: Ahsen Baig

Report reviewed by: John Redden 3/27/17

Abstract

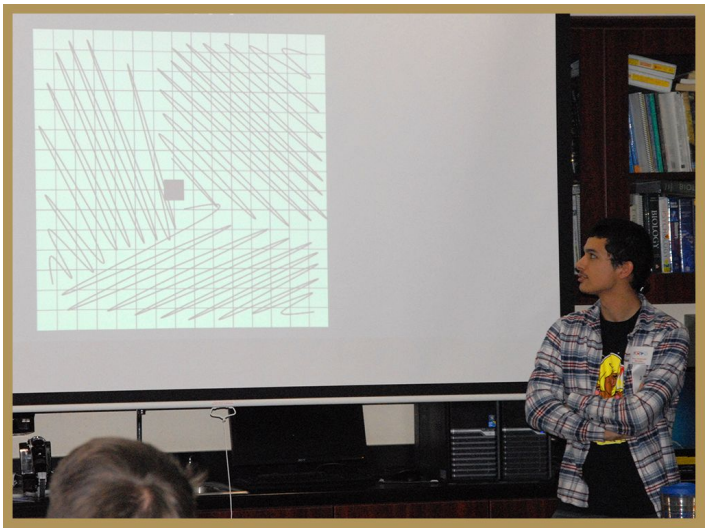
The NASA Swarmathon project provided our Friday Night Lab program with a challenge that tested our ability to persist within the scope of an existing complex system. We learned that we could push through technical problems and work outside of our zone of comfort. This competition has, most importantly, instilled in our group the confidence that we could, with time and effort, tackle the requirements of any seemingly impossible challenge. We used our current knowledge, which is predominately game development and mathematical based, into this project to expand what we know.

Introduction

The NASA Swarmathon project at COS began with 8 members and was an ongoing project in the Friday Night Lab program. Over the course of several months, we had many contributors and a number of discussions as to how to approach creating a search algorithm for the rovers; however, we did have some rough patches implementing these ideas. With minimal Linux experience, our group ran into challenges from the beginning. The learning curve pushed us through installation issues, new operating systems, and C++ frameworks. This ultimately lead us to have a functioning master copy on a computer where we worked together in the lab; individually, there were numerous technical issues. In the end, this was a positive learning experience where we gained experience in navigating components of the preconfigured code, search algorithm research, testing and outreach.

Related Work

We began by reading the *Gentle Introduction to ROS* and the existing code to see how things worked. The segment of code that caught our initial attention was the case system in mobility.cpp. At the very beginning we tried to rework this system and then realized that we should focus on the searchController. Unfortunately, this realization took more time than expected; although, the tinkering was a valuable way to learn more about ROS and how everything works.



Once we scrapped our initial work and started over, we began to make progress working on the search algorithms in the searchController.cpp. After some time learning how the system worked, we decided to implement a waypoint system. It is something similar to the one we were working with in Unity, which we used to both further understand the 3D aspects of mathematics and to build a game in virtual reality.

Methods

Our beginning assumption was that we were to only modify the SearchController. With this restriction our methods were as follows:

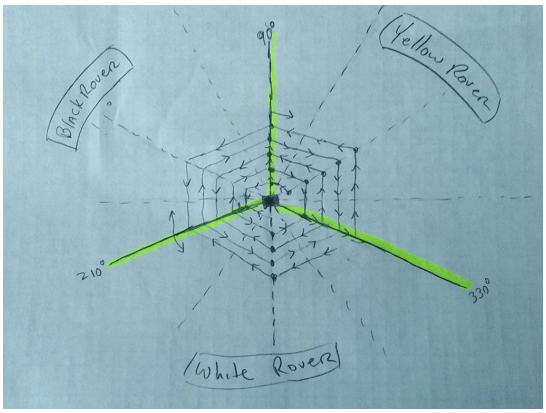
1. We devised a waypoint system that is simply an array of 2D position objects found in the SearchController header file. `Pos2D goals[GOAL_SIZE]`

2. We passed the `publishedName` from mobility into the searchController so that we could distinguish between the rovers and assign them their tasks accordingly.

3. For each rover, we populated it's instance of the array `goals` once in the searchController method. This gives us full control on how to set the waypoints by applying mathematical models that returns points on the map.

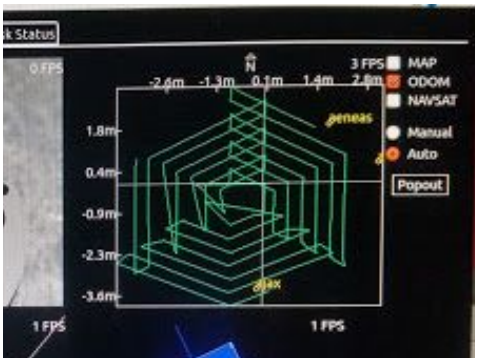
4. We then set the rover's goalLocation by using its current waypoint array index and then increasing the index by one each time the search method is called. If the index is out of bounds, then the index resets to zero and starts the cycle again.

5. We adjusted the goalLocation to world coordinates for each rover to simplify our path creation because the rovers were offset.



By the time we were able to get a handle on the system, we were only able to test a few search algorithms using this waypoint system. We tried sending the rovers in a spiral going outward, then we sent the rovers out far and spiraled back in. Testing along the way using the 10 minute 3-rover uniformly distributed target configuration.

We noticed that the initial scrum of obstacle avoidance calls adversely affected results. Finally, we realized that we should partition the area to minimize collisions. We tried many partitions, ultimately settling on a 3-sector inside-out sweep, as pictured. We created the sectors for 3 individual rovers to sweep without colliding. This greatly improved results. For the 6-rover configuration, the initial plan is to use this partition while two rovers search each sector. By the time of this writing, we have not implemented a 6-rover search.



Experiments and Results

We began by establishing a baseline with the given random walk algorithm using the uniformly distributed target world with 3 rovers. In several ten minute tests, we had on average 10 targets collected with 279 obstacle avoidance calls. From here we tested at each stage of our learning curve. We began with a spiral algorithm, which did improve the collection rate and obstacle avoidance calls over the original random walk algorithm;

	A	B	C	D
1	Run	Targets Detected	Targets collected	Avoidance calls
2	1	0	13	326
3	2	0	11	502
4	3	0	9	514
5	4	0	9	143
6	5	0	7	59
7	6	0	15	174
8	7	0	6	236
9				
10		avg	10	279
11	New Algorithm hard coded			
12	1	0	10	671
13	2	0	12	132
14	3	0	11	54
15	With wrist angle change			
16	1	0	14	77
17	2	0	16	127
18				
19	QI, QII and QIV w/wrist change			
20	1	0	8	461
21	2	0	9	181
22	3	0	14	236
23	w/drop off changes			
24	1	0	11	226
25	2	0	12	228
26	3	0	15	159
27	4	0	12	527
28	5	0	11	830
29	drop off restored mostly			
30	1	0	10	610
31	2	0	16	32
32	3	0	7	98
33	4	0	5	216
34	World Coords in Mobility			
35	1	0	22	53
36	2	0	16	356
37	3	0	19	463
38	4	0	15	384
39	Some spacing in Quads			
40	1	0	15	159
41	2	0	15	111
42	Values in Dropoff back to default			

however, this brought up position offsetting in our mathematical calculations. After more experimenting, we found out that we could obtain better results by separating the rovers to avoid collisions. We came up with a method to separate the entire area into a specific segments for a specific rover to search through. From here, we thought we might start at the outer edge then work back; in hindsight, it would not have worked out as the rovers would have to travel long distances and collect fewer objects while on a time constraint. Finally, it became clear that we should obtain as many targets nearby then zig-zag out as far as time allows.

As of the writing of this report, we have just began to modify the dropOffController and mobility.cpp to streamline our results. We have only

tested in the 3-rover configuration, using the uniform targets example world and 10 minute tests.

Conclusion

We found that working in a team to tackle problems while encountering difficulties individually to be quite the challenge. Most of the hard work came from trying to understand the new environment and to adapt accordingly. We did however come up with different patterns and algorithms, which was much easier than the environmental change. We chose the best performing algorithm from a handful of other algorithms we created. The experience that came from participating in Swarmathon gave us a profound insight to a broader category of programming that we may not have had the pleasure to work with. Most of the team has worked on game development, which balances performance with efficiency and interactivity, and we were able to use this kind of mindset along with this opportunity to help in scientific endeavours. We appreciate the opportunity to participate and thank you for your time implementing this program.

College of the Sequoias
NASA Swarmathon Team - 2017