Team Wolfpack: Multi-Agent Map Merging and Navigation

[redacted], Samuel Dalberg
CS4610 Robotics Science and Systems
Northeastern University

Abstract: In this project, multiple mobile robots were used to efficiently explore and map an unknown space. Two TurtleBot3s were used in simulation and physically to run SLAM, map merge, and then path planning algorithms to work together to explore an unknown space more efficiently than a single robot. Our project faced significant challenges and setbacks, including reinstalling different operating systems, faulty hardware, and inconsistent setup and maps. In the end, we were able to do the map merge perfectly in simulation, but not in a real-world environment. Further work can be done to explore the space with unknown starting positions.

1. Introduction

The field of robotics is heading towards more collaboration between robots to carry out tasks similar to how a group of animals or humans would approach certain tasks. Utilizing multiple robots has many of the same advantages as using multiple people does: tasks can be completed faster and more efficiently, tasks can be split up, and tasks can be more complex. Additionally, some tasks inherently require multiple agents such as scouting and gathering tasks, moving heavy object tasks, and other cooperative tasks. In this project, we explore a basic multi-agent scenario using a global controller (i.e. robots are not completely autonomous).

When formulating a task for the project, we centered around the idea of *multiple* robotic agents interacting with each other. Instead of focusing on multiple agents acting separately, we were interested in exploring how multiple agents could work together towards one goal. With this, we proposed the idea of having multi-agent map creation, merging, and navigation; after learning about localization and mapping with a single agent in class. Our task is formalized as follows:

> *'Using two mobile robots, explore and map an unknown space together by creating a global map and exploring intelligently'*

Our task is divided into several milestones to achieve the desired goal:

1. Setup simulation environment and physical robots
2. Run SLAM on each robot
3. Create a global map from each SLAM algorithm
4. Use the map to optimize frontier exploration with both robots

We made a couple of assumptions about a robot environment: it is flat and has some interesting features while being small enough to explore within a few minutes. We chose to use Jamie's apartment as our workspace due to it having large empty spaces for the robots to explore. Furthermore, it met our requirement of having interesting features by including chairs, tables, couches, a TV stand, etc. In addition, it had a hallway that served as a good testing ground for observation and movement. In Figure 1 we can see our real-world test environment.

Figure 1. Jamie's Apartment (Our Real-World Test Environment)

In this paper, we first lay out our approach to the problem and then define the software, hardware, architecture, and algorithms used in Section 2. Next, the process of implementing this task is explored more in-depth in Section 3. Our results for each milestone are also discussed in that section. Finally, we include further discussion on how to improve our project in the future in Section 4.

2. Approach

The focus of our project was to get the architecture in place and working before implementing the code ourselves. This way, we would know that if something went wrong, it was a code issue and not a communication/architecture/hardware issue. Thus, we found software packages that did similar or the same things as we wanted and used those in place of our own code. Then, we would test the system and if it worked, we would continue until the entire architecture was in place. Finally, we could implement our own algorithms using the same architecture one at a time, making sure everything worked together along the way.

2.1 Software

For this project, we initially chose to work with Ubuntu 20.04 LTS and ROS Noetic. However, it became apparent much later in the project that we needed to switch to Ubuntu 18.04 LTS and ROS Melodic. This switch is discussed in more detail in Section 3.3. For simulation, we used a combination of Gazebo and RViz. Finally, we used Python when programming, and pulled many available TurtleBot3 assets, including the TurtleBot3 Gazebo model, as well as packages such as `turtlebot3_gmapping`, `move_base`, and `multirobot_map_merge`.

2.2 Hardware

This project required multiple mobile robotics platforms to test on, and for this we chose the TurtleBot3. TurtleBot3 is a modular, open-source robotics platform that comes with two Dynamixel motors, a Raspberry Pi, an OpenCR board, a camera, and a 360-degree LIDAR. There are two configurations, the Waffle and the Burger. We were allotted two Burger TurtleBot3s named “Raphael” and “Bebop”. The TurtleBot3s also have a wide range of supporting packages that enable easy implementation of SLAM, navigation, and manipulation algorithms. In Figure 2. you can see our two TurtleBots.

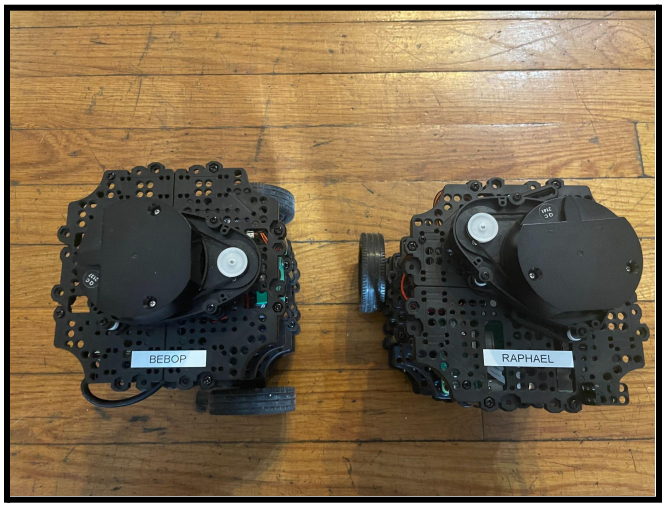

Figure 2. Bebop and Raphael

Immediately upon receiving our two TurtleBots, we found hardware issues. First, we were only given one charger to share between the two bots. This made setup particularly difficult as we could only charge one at a time. Furthermore, if we wanted to connect a TurtleBot directly to an outlet, we could not charge either battery at the same time as it used the same cord. Another pressing issue was Raphael's LiDAR sensor. The screws holding in the sensor on the top of the Bot had not been properly installed and had subsequently fallen out at some point before being given to us. This meant his sensor was completely detached from the body and only hanging on by the wires connecting it to the rest of the system. In Figure 3., you can see this issue. This problem was temporarily overcome by taping down the sensor as seen in Figure 4.

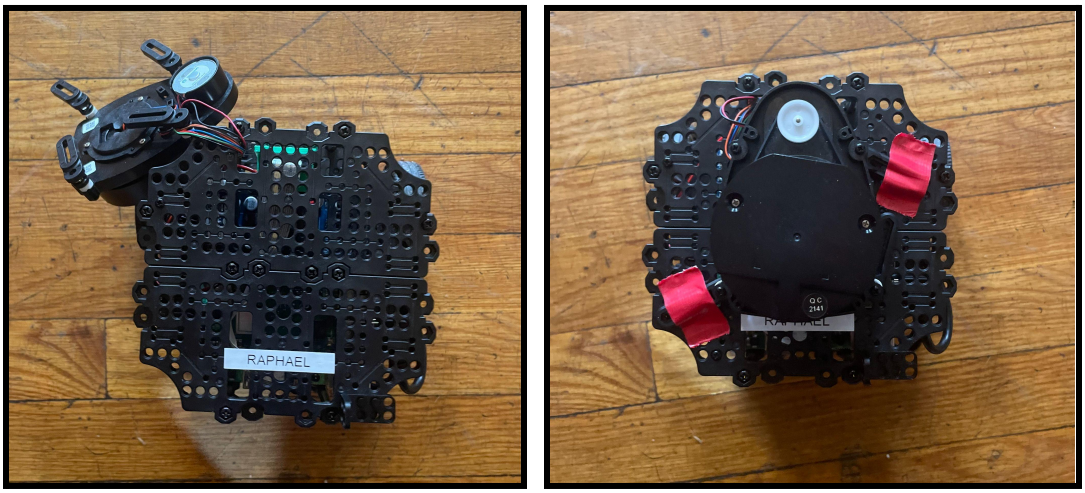

Figure 3-4. Broken and Fixed Lidar Sensor on Raphael

2.3 Architecture

The goal of our project is threefold: implement SLAM for multiple robots, use the SLAM to create a global map, and use the global map to optimize the path planning and exploration for the robots. SLAM takes in the LIDAR data and outputs a map. Each robot produces a map, both of which are input into the map merging algorithm and creates a global map. Finally, the global map is used with the robots' locations to produce goal positions to navigate to (which are input into a navigation algorithm). This cycle then repeats until the entire room is mapped.

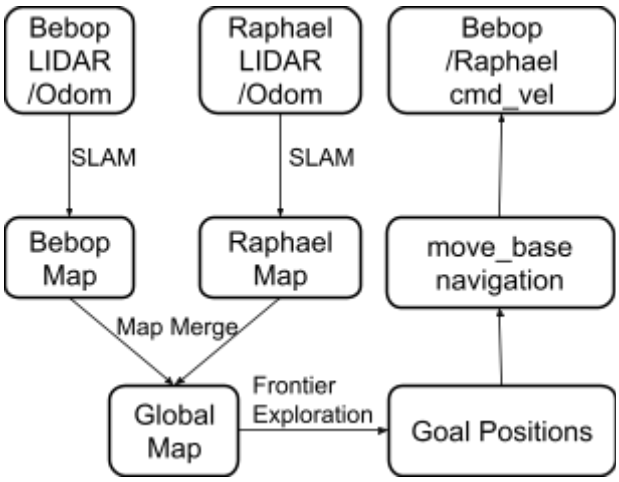

Figure 5. SLAM, Map Merge, Exploration, Navigation Cycle.

2.4 Algorithms

While we did not implement several of these ourselves, it is important to understand how these work and what they are doing when we subscribe to the ROS topics they publish to. Our

approach combines the use of SLAM, Map-Merging, and Navigation to achieve fast multi-agent mapping.

2.4.1 SLAM

Simultaneous Localization and Mapping (SLAM) is an algorithm that tracks both a robot's position and the environment relative to it. It consists of a prediction step and an update step. In the prediction step, the new expected state is computed based on the odometry readings from the robot. This may include some noise as the hardware is not always (never) perfect. The update step then uses sensor data, in this case, the point cloud coming from the LIDAR, and updates the map and the location of the robot relative to the map.

2.4.2 Map Merge

When working with multiple agents, the individual robots' maps need to be merged together to create a singular global map. There are many different ways to merge maps together. This ranges from a rudimentary approach of simply using transformations from a given initial position to a more complex approach involving feature-matching algorithms such as SURF or ICP. For the `multirobot_map_merge` package that exists in ROS, the approach is a combination of taking in the initial start positions of each robot and then using feature-matching to align the edges and obstacles.

2.4.3 Navigation and Path Planning

With a global map, we can optimize the exploration of a space using multiple robots. Using the `move_base` package for navigation, we can figure out goal positions for each robot to go to in order to maximize space exploration. This is known as a frontier exploration problem. This algorithm takes in the occupancy grid from the merged map and outputs a goal position for each robot. The algorithm iterates over the occupancy grid to find a group of points that represent unknowns in the map that are between known occupied spaces. A center point for the group is calculated and added to the frontier list. Then, from the frontiers calculated, the appropriate frontier is selected for each robot to maximize the efficiency of exploration. This could be having a heuristic encouraging robots to move away from each other, or a minimum distance calculation that sends the robot to the closest frontier (but not to the same as the other robot).

3. Implementation and Results

3.1 Milestone 1: Software and Hardware Setup

As previously mentioned, we had numerous hardware problems from the start. These aside, we also had trouble getting the software to work properly in the beginning. We started by installing Ubuntu 20.04 LTS and downloading a TurtleBot3 Ubuntu image onto a single TurtleBot3. This proved troublesome as we did not have a microSD card reader at first. Once acquired, the image was installed and the next step was to get it connected to the wifi. This was done by connecting the Raspberry Pi to a monitor and keyboard and manually adding the wifi configuration in the netplan YAML file and applying the changes. We could now ssh into the TurtleBot3! A software update and firmware update were done, and after configuring the IP addresses for ROS, we got teleop working on a single TurtleBot3. We quickly found out that we

were unable to run teleop via a virtual machine, and had to use a dual-booted Windows computer. All of this took several hours as we tried different things in the YAML configuration to get the wifi connected, couldn't get the firmware to update, and the ssh wasn't working properly.

When we had one TurtleBot3 working, we booted up the second one using the steps we figured out along the way. This setup took much less time. Finally, we wanted to get them both running at the same time. We would start one up and it would work fine, but when we started the second TurtleBot3, both would shut down. After some digging online, we found a resource for using multiple TurtleBots, and just needed to add `ROS_NAMESPACE=name` in front of the bringup scripts. ROS namespaces allow us to have unique node names, which is a requirement for ROS to work. This allowed us to run both TurtleBot3s at the same time.

3.2 Milestone 2: Running SLAM

Getting SLAM to work on a single robot is rather trivial once everything else is set up. It simply involves running a ROS SLAM launch file that is already included in the TurtleBot3 image. SLAM mode uses the TurtleBot's LiDAR sensor to make observations about the environment around it. In Figure 6 we can see SLAM running on a single turtle bot within a simulation.

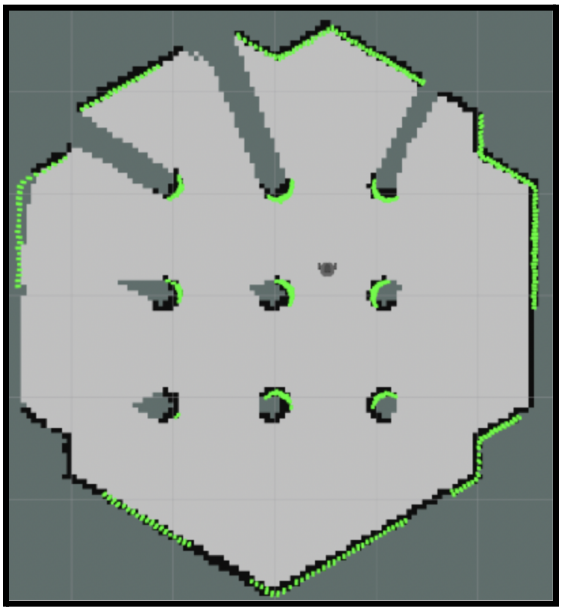

Figure 6. TurtleBot3 Running SLAM in Simulation

Getting Slam to work for two robots proved significantly more difficult. Along with the aforementioned issues with running two TurtleBot at once, we had no way to 'combine' the maps outputted by each bot. This leads us to our next section.

3.3 Milestone 3: Map Merging

SLAM worked for both TurtleBots at the same time, now we needed to take those maps and merge them together somehow. We brainstormed numerous ideas for achieving this.

- Using Iterative Closest Point algorithm (ICP) to align point clouds.
- Random Sample Consensus (Ransac)
- Using a pre-made package for Map-Merging

Although we did explore ICP and Ransac, it proved too difficult to extract, use, and redistribute the map and point cloud data with our limited knowledge of ROS. We were able to find a package called 'multirobot_map_merge', and some demonstrations of this working on multiple robots. This package takes in multiple ROS map topics, along with accompanying robot position data. During our first attempt, we failed to produce any usable results and were running into constant obscure errors. Initially, our thought was that the positions were not being properly transmitted to the map merge package. After sifting the internet for some time and spending days trying to work with our results, we came across blog posts explaining that map_merge ***would not work*** with ROS Noetic. This is due to a feature removal that came with Noetic, where the TF topic will not take on a prefix. This meant that all data from each robot's individual maps were being sent to the same TF topic, and there was no way to distinguish them. This problem can be seen in Figure 7. This prompted us to switch to ROS Melodic. This meant we had to redo our dual-boots to switch from Ubuntu 20.04 to Ubuntu 18.04 and reinstall ROS and all accompanying packages. Additionally, we had to again flash the proper version of Raspberry Pi OS to each TurtleBot. This set our progress back significantly.

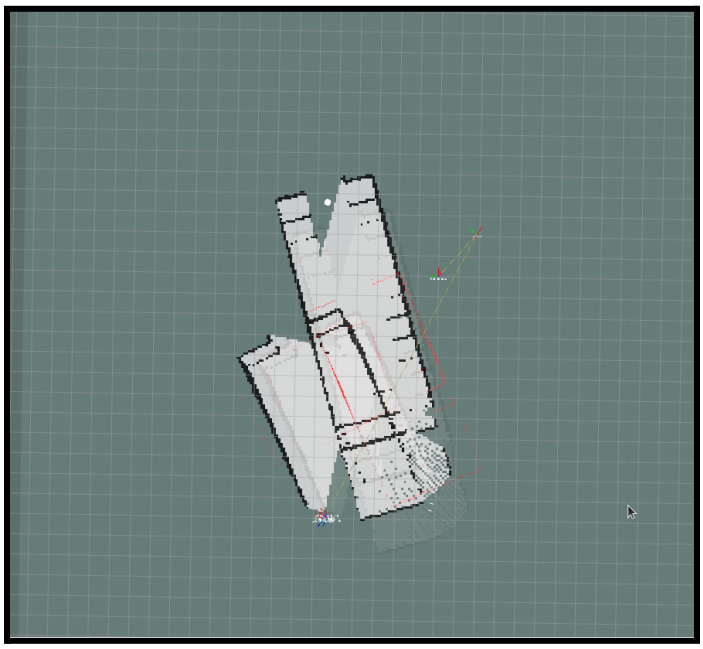

Figure 7. Initial map merge on ROS Noetic.

Once we had the proper versions of ROS installed, testing out the map merge package yielded much better results. At first, our maps had an unusual offset that, after hours of debugging and combing launch files, was resolved by adding a map transform of 0.4 in the x and y direction for each robot map being input. Our map was then successfully merged in simulation, showing the proper location of each bot, and connecting together parts of each individual map that were overlays.

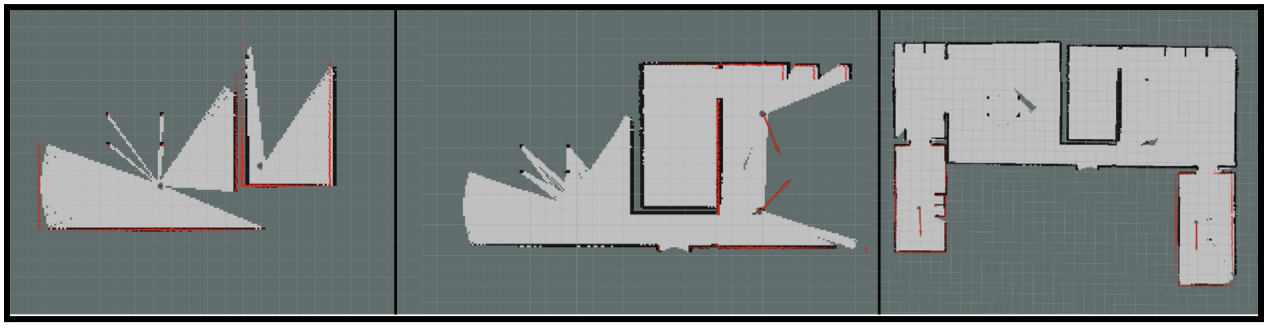

Figure 8: Multi-Agent SLAM and Map-Merge in Simulation

In the figure above, the three sections represent different stages of SLAM as more area was filled out. The robots first started in two separate rooms, with the `map_merge` placing each individual LiDAR map in the correct location. In the second image, the two robots both are navigated to the same room, where the individual LiDAR maps are overlaid onto each other. At this point, the `map_merge` package uses feature-matching to convert the overlaid parts of each map into one map. Finally, in the last image, each robot is navigated to separate parts of the world, with the first TurtleBot3 mapping out the left side of the world, and the second TurtleBot3 mapping out the right. The `map_merge` package produces one combined map showcasing the results of the TurtleBot3 mapping. When comparing this map to the original world layout, the edges and borders in the map are very accurate and to scale. The obvious benefit of simulation is the lack of noise and other real-world obstacles. When testing the map merge in a real-world environment, the merged map is less accurate due to unwanted noise.

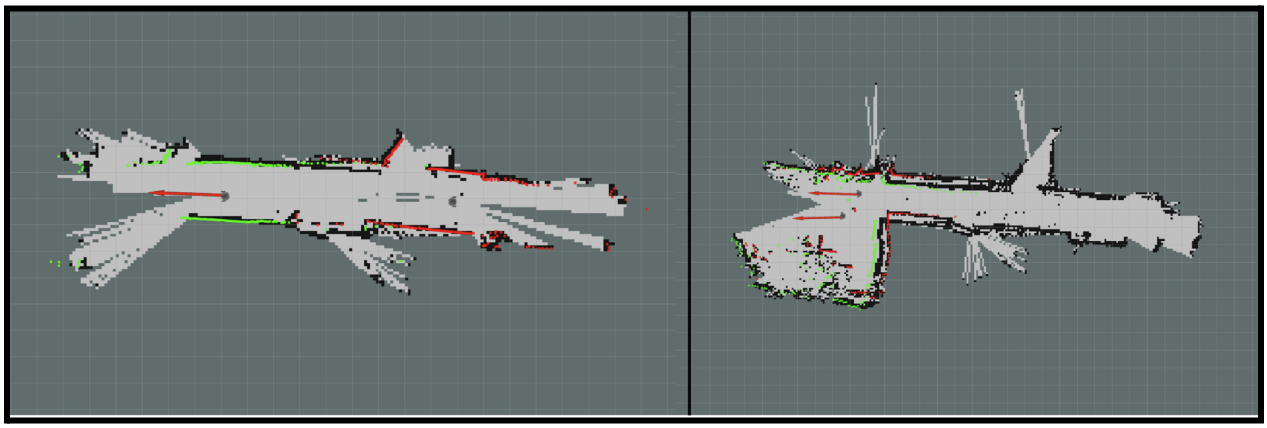

Figure 9. Multi-Agent SLAM and Map-Merge in Hallway

The figure above shows our resulting merged map from a real-world TurtleBot3 test. The first image was captured before any bot movement took place. The `map_merge` package was given initial robot starting positions and was then able to accurately combine each individual map and align the hallway walls. In the second picture, each robot was navigated to the same end of the hallway, jointly mapping out the larger room. As the robots moved down the hallway, the edges of the map became less defined. This is due to the noise introduced from odometry readings and LiDAR sensors not being fully accurate. Additionally, the larger room in the second picture was filled with many obstacles, such as tables and chairs, making the feature mapping especially difficult. Videos will be included in the Section 5 of a real world test.

In summary, we are satisfied with the results of this milestone. The simulation final map is almost perfectly accurate. Additionally, our real-world test is promising. The map is as accurate as it can be, with almost all inaccuracies coming from noise we can't control. We believe the simulation will provide us with the mapping necessary to path plan and navigate.

3.4 Milestone 4: Path Planning and Navigation

Unfortunately, we ran out of time due to Milestone 3's setback and were not able to complete our final goals. However, we were able to get navigation working through a ROS package called `move_base`. This package can command a robot and navigate it through a given map to a desired end goal. Using RViz, we can easily place down a location goal on the map and

`move_base` will navigate the TurtleBot3 to it. This navigation can be seen in the videos included in Section 5. Using this package, we intended to implement a frontier exploration algorithm that would assign goal positions to each TurtleBot3 to optimally explore the world. Although we were not able to get an implementation working, we do think it is plausible. Building off our successes with map merging, we hypothesize that one could effectively build a multi-agent navigation system to efficiently explore a space.

4. Further Discussion

4.1 Next Steps For Further System Improvement

One clear, scalable area for improvement is in the number of agents used. For our task, we were only allotted two TurtleBot3s to work with. In an ideal scenario, our methods and tasks could be expanded to use any number of Turtlebots. Since there are more agents, a smaller space could perhaps be more accurately mapped since there are multiple viewpoints observing it. More agents bring another possible result. Two TurtleBots limit the possible exploration space to only a small room. With numerous more TurtleBots, we hypothesize that we could explore much larger spaces. Perhaps an apartment, a house, or an entire building. Even outdoor spaces could be good tests for our system.

Further use cases for our system are using the generated map to complete more complex Multi-Agent tasks. Many of these tasks we pondered while initially formulating our task. Some examples of these tasks include:

- Multiple agents searching for something within a space.
- Multiple agents looking for *each other* within a space.
- A leader and a follower robot(s), one (or multiple) robots following another robot (Mother Duck and Ducklings)
- Multi-robot sweep search.
- Multi-robot pushing task.

4.2 Advice For Future Students

Our main advice for future students is to start earlier. The project involves a large mixing of software and hardware, which comes with many unexpected problems. Especially for students who have limited Robotics knowledge (like us), it is critical that they take the time to get acquainted with the systems they are using. Seemingly simple tasks often took far longer than expected with lots of debugging necessary. All things considered, the project can be rather intimidating.

Perhaps the most critical aspect of the project was with hardware and personal equipment. Unbeknownst to us, a correct Linux install would be a make or break for the project. It became clear to us once we started working on the project, that only one of our team members had a computer that was even capable of running the correct Ubuntu Linux and ROS version. If we were prepared early, it's possible that we could have figured out how to acquire another working computer.

Having a working Linux environment is *pivotal*, and should be figured out at the *beginning* of the semester, even *before* project brainstorming, team-forming, etc. Week 1! This is also something we suggest course staff *press* on students.

In terms of how much time certain components took to get working, getting a single TurtleBot to move took an entire day. Getting two TurtleBots to move at the same time took another whole day. We ran into countless problems where the solutions were extremely unclear. Often the hardware documentation can be sparse and intimidating. Familiarizing oneself with ROS and the TurtleBots (or whatever system you are using) early is key.

We also advise that students start small. Formulating small, scalable goals is the best way to success. Whatever crazy project ideas you may have, one would likely be unable to achieve such ideas given the time constraints of the semester. Having a *scalable* goal means one can create achievable milestones along the way. Having subgoals is critical, if one's entire project banks on everything working at once, failure is far more likely.

5. Supplementary Material

5.1 Links

Map Merge*:* http://wiki.ros.org/multirobot_map_merge
Move Base: http://wiki.ros.org/move_base
Robotis E-manual: https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/
Loading multiple TurtleBots (hidden link):
https://emanual.robotis.com/docs/en/platform/turtlebot3/applications/#load-multiple-turtlebot3s
TF: http://wiki.ros.org/tf

5.2 Videos

Bebop and Raphael in the Hallway:

Hallway (Feel free to speed up):
https://drive.google.com/file/d/1vNja5uHJjI_m3pm16n4uAY3ypVjHDsc4/view?usp=sharing

Computer SLAM:
https://drive.google.com/file/d/1micMV3-cWo807Rt3kBwl-kfdcgj4pJ1W/view?usp=sharing