Individual Lab Report 10
Progress Review 11

Shivam Gautam
sgautam@andrew.cmu.edu

Team Daedalus
Mohak Bhardwaj, Pranav Maheshwari, Shivam Gautam, Richa Varma,
Dorothy Kirlew

April 13, 2016

1 Individual Progress

My work till the final progress review revolved around finishing up tasks for the first mock Spring Validation Experiment (SVE). This specifically involved-

1. SVE Testing for Oculus Prime
2. Global Planner

1.1 SVE Testing for Oculus Prime

1.1.1 Oculus Prime Localization

As per the requirements of the SVE, we would be showing collaborative parking for two platforms instead of one. We previously demonstrated, the navigation of a single platform in the parking lot with collaboration with dummy vehicles. However, our major shortcoming was navigating from the constrained parking spots. Having walls at a separation of 50cm did not allow the ROS Navigation stack's DWA planner to generate vali trajectories while exiting the spot. As a result, we decided to remove the separation between the spots.

However, we had to leave the separations in the map unedited. This was done because the separations were needed in the cost-map of the robot so that it would not plan trajectories between spots. Removing them from the map would mean that the robot would be free to move through parking spots. We decided to test the navigation with the map depicted in figure 1.

Testing with this map revealed that even though the platform could find feasible trajectories to exit the spot, the robot localization was not as robust as before. This was attributed to the fact that the removal of walls from the real world affected the matching of the laser-scans during the localization. This resulted in the platform taking more turns (and more time) to eventually localize itself.

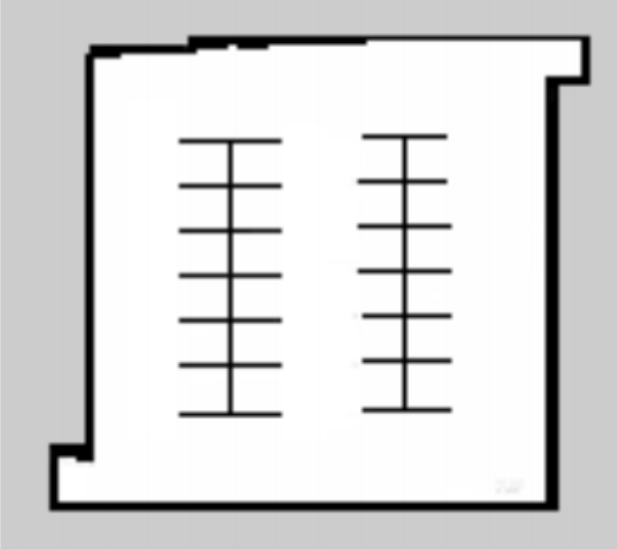

Figure 1: Map of the paking lot with adjoining walls between spots (Collaborator- Pranav)

I decided to find a middle-way between these two approaches and decided to test the navigation with the map depicted in figure 2. The rationale behind using this map was that leaving the a dot at the edge of each spot would restrict the robot's movement and wouldn't allow it to enter another spot while navigating to a spot. The removal of the walls meant that the platform was able to solve the problem it faced with the DWA planner. The obstacle-like 'dot' that is left on the map is sufficiently small to not hamper localization in the lot.

1.1.2 SVE Demos Dry-Runs

For this progress review, we demonstrated the tasks we would entail in the progress review. This involved practicing for the two demo's we have scheduled for that day.

1. DEMO 1

   This demo includes two platforms entering a parking lot and parking collaboratively. We had successfully tested the routine on a single platform but faced constraints regarding the availability of the second platform. Richa had been working on getting our other platform ready from the software and hardware end. After she had completed the setup for the robot to be tele-operated, I rewired a few electrical components and secured the electronics. Also, since this platform did not have a dedicated WiFi/Bluetooth card integrated because of the unavailability of the antennas, I decided to use one of the two antennas from our new platform. This weakened both the WiFi and bluetooth signal emanating from the platforms. However, WiFi communication was still possible because of the high-gain antennas on the router. Bluetooth capability, however, was severed and we could only communicate with the platform when the phone's bluetooth antenna was very close to the platform. To overcome

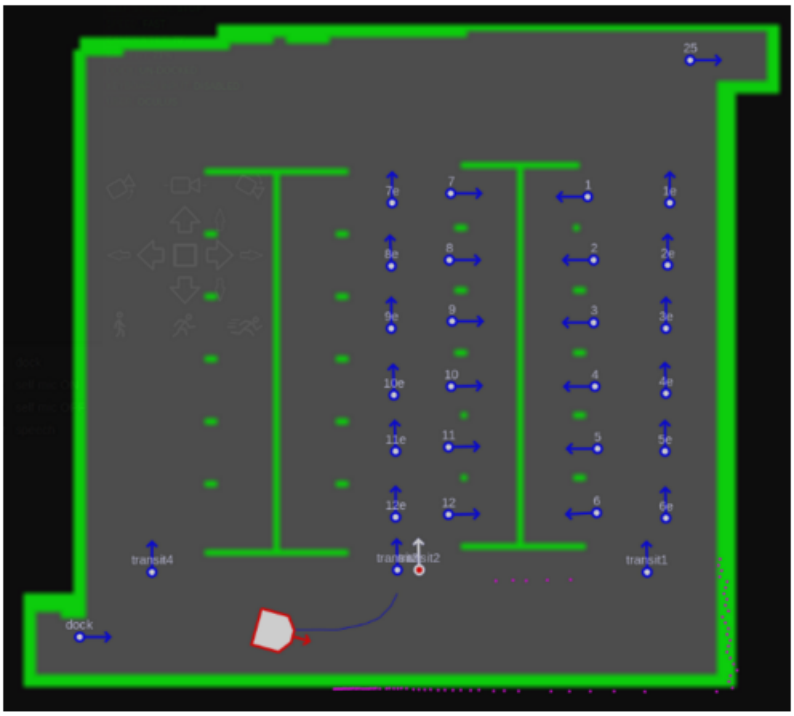

Figure 2: Map of the parking lot with adjacent walls removed and spoofed obstacle (Collaborator- Pranav)

this, we wrote a separate script to spoof the phone to test our other subsystems
.

After testing the SVE Demo 1, we observed the platforms parking collaboratively and satisfied the requirements of the project. The platforms could also avoid obstacles and navigate around them. Once parked, sending a return command ensured the platforms exited the parking lot.

2. DEMO 2

This demo includes demonstrating the capability of our planning algorithm on a large scale. For this, ran our planner to park a hundred cars in a simulated parking lot. The demo showed how the cars took different strategies to select a parking spot depending upon different heuristics.

This is showed in figures 3 and 4. The environment was rendered in RViz. The decision to select a spot was taken by a global planner and the local planner was used to navigate the vehicle to the selected spot.

On testing the simulation, we observed that the spot assigned to a car determined on the tuning of weights associated with the following heuristics-

(a) Distance to exit- Determined by the global planner

(b) Cost to navigate to a spot- Obtained from the local planner

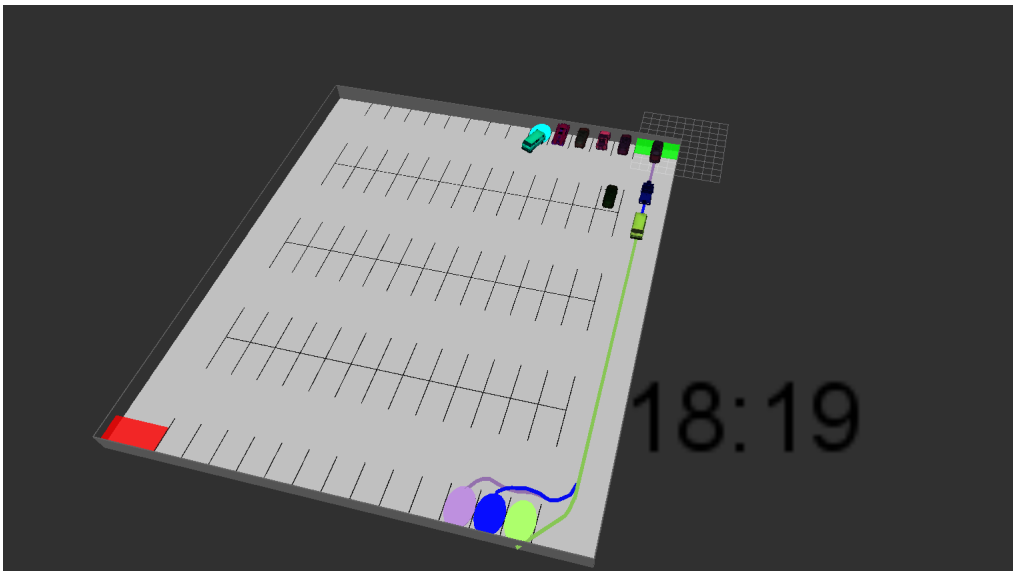

Figure 3: Initial state of parking lot

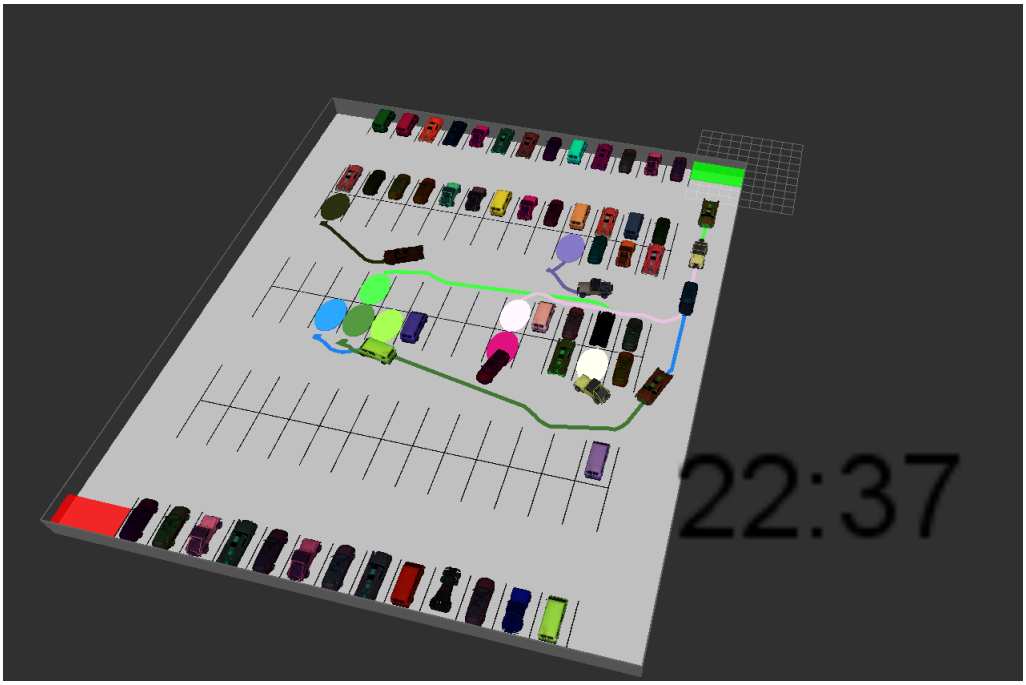

Figure 4: State of the parking lot after more cars enter

(c) Time of the day- Whether peak or non-peak hours.

(d) Cars in Queue- number of cars waiting to be parked

(e) State of the parking lot- Spots that are occupied vs claimed vs empty. This shall be discussed in detail in the section covering the global planner.

1.2 Global Planner

The Global planner is complete and has been integrated with the local planner and rendering engine. The global planner has been divided into four separate classes and header files which ensure modularity in code. They are organized as-

1.2.1 ROShandle

This is the class that deals with the ROS interface of the planner. Implementing it as a class essentially means that multiple global planners can be run at the same time using separate ROS initializations for each.

The ROShandle includes services to communicate with the local planner as well as the rendering engine. The services have been discussed at length previously and no modifications were made to this architecture. Additionally, the ROShandle deals with subscribing to the worldTime node (Node that reflects the current time in the world). The class has an object of the gplanner class which is discussed in the next section.

1.2.2 Gplanner

This class, abbreviated from Global Planner, is the main crux of the global planner. It calculates the cost of all the heuristics for all the spots in the lot and scores each of the spots. The implications of the various heuristics on the parking spot are modelled as follows-

1. Distance to exit: The distance to exit is a static distance for each of the spots and is calculated using the SBPL. The calculation is based on the motion primitives of a car in the x-y-theta space. The more the distance from the exit, the more is the cost assigned to that spot.

2. Cost to navigate to a spot: This cost is obtained from the local planner. The global planner queries the local planner for the cost of any particular spot according to the heuristics implemented in the local planner. The local planner returns a cost associated with navigating to the queried spot based on its set of heuristics.

3. Time of the day: This assigns a cost to each spot based on the time of the day. The proximity of the current time to the user-defined peak hours determines a cost assigned to each spot. If the time is close to the peak timings, the planner assigns a higher cost to spots closer to the entrance. This has the effect of sending cars farther away to prevent crowding.

4. Cars in Queue: This heuristic assigns a cost to all the spots based on the number of cars in the queue. If more vehicles are present in the queue, the planner assigns costs such that the planner disperses the vehicle to a far away spot.

5. State of the parking lot: The state of the parking lot also determines the cost associated with a spot. A spot that is near to an occupied or claimed or occupied spot is assigned a higher value than a spot that is surrounded by several free spots.

To ensure fairness, all the heuristics are first normalized before combining to determine a final score for each spot.

1.2.3 PathPlanner

This class is used to calculate the distance of a car to the exit as part of the distance to exit heuristic. Instead of computing this distance in a 2D grid, which is relatively trivial, this planner uses the motion primitives for a car - discretized at 16 angles and non-holonomic motion in seven directions. The auto-park simulation environment was converted to an environment compatible with the SBPL format. I used the Anytime A* planner to compute the path after comparing its performance with the other planners.

To run the planner for multiple spots, the planner and the environment variables from a previous search were cleared and reinitialized for each new spot. Caching the costs for all the spots after running the path planner increased the speed of the global planner. The cost is calculated at the initialization of the global planner and the environment and is cached to save computation for future calculations on the same map.

2 Challenges

One of the challenges I faced was during the navigation and testing of the Oculus Prime platform in the parking lot. I faced issues related to the WiFi and bluetooth communication which were solved by using one WiFi antenna on the platforms each, as has been documented earlier. Also, it was really frustrating working with the current electronics of both the new and the older platform as they would stop responding mid-way during testing. I attributed the problem to faulty connectors of the Solid-State Drive which would come loose during testing. These connectors were secured using a light coating of glue to avoid coming loose.

During testing, the depth sensor for the older platform returned incorrect data- the platform would see an obstacle in front of it always. The sensor was replaced with the backup.

The challenge I faced while coding the Global Planner was using the SBPL library for the global planner. There are two options to using the SBPL- as a C++ library or as a ROS library. During my development work, I was using the SBPL as a C++ library and it worked fine. But, when I migrated my path planning code to ROS, the package could not include the C++ library. Support and API documentation for the ROS version of the library was limited. To mitigate this, I built the SBPL as a ROS package and included it in my global planner as a dependency. I faced a few issues while coding the global planner which were resolved appropriately.

3 Teamwork

I collaborated with Mohak and Pranav for the simulation environment. This involved integrating the three planners and testing. Since we had already laid the groundwork for all the ROS services, this was achieved smoothly. Mohak, Pranav and I also worked together to test the navigation of the Oculus Prime. Mohak completed his local planner

and implemented the heuristics that were decided in the architecture(distance to goal, proximity to obstacle, motion segments and proximity to occupied spots). Dorothy and Richa worked on a desirable requirement of the project- having multiple vehicles park simultaneously. For this, they worked on implementing a modified A* in simulation.

4 Plans

Our plans for the SVE include testing our system on the new platform that arrived on the day of our progress review. Mohak, Pranav and I would be working on analyzing different parking scenarios in the simulation environment with different weights. Dorothy and Richa would be working on improving the multi-agent planner. If feasible to integrate in the given time frame, we would integrate and test it in the parking lot.