Lab # 6 Report: Path Planning

Team # 2

Isaac Lau
Lilly Papalia
Mario Peraza
Joshua Rapoport

6.141/16.405: RSS

May 4, 2021

1 Introduction

Author: Joshua Rapoport
We will describe the development and testing of a Robot Operating System (ROS) package that, given perfect map knowledge and position data, plans and follows a path to a specified goal position.

This lab represents the integration of components from several past labs - path following and localization - with path planning. In light of upcoming efforts to integrate many ROS packages into a single robot, this week's goals extend to preparing for this final integration:

Path Planning: Implement an algorithm that will find a complete, collision-free path from an initial position to a goal position. The algorithm(s) in question must be reasonably time-efficient, and produce a path that is close to optimal.

Path Following: Rework the selection of the transient goal use by a pure pursuit controller, and tune the controller accordingly.

Localization: Tune parameters of a Monte Carlo Simulator to maximize performance when running alongside other ROS packages.
Developed in Python 2 using ROSPy Melodic, and it can be configured to run in a 2D environment (visible in "RViz") or a 3D environment (visible in "TESSE" simulator).

2 Technical Approach

2.1 Path Planning

Author: Mario Peraza

The goal of a path planner is to input a map, a starting location, and an ending location and output the shortest or most efficient path between the starting and ending location. There are multiple different algorithms to consider when creating a path planner. For this lab, search-based planning and sampling-based planning algorithms were compared to each other to determine the most efficient and accurate method of path planning. There are benefits and drawbacks to each method which were taken into account when designing the path planning algorithms.

2.1.1 Occupancy Grid

Author: Lilly Papalia

The first step to path planning is converting the given occupancy grid into a useful data structure for searching. To use the occupancy grid, converting from the map frame to the pixel frame is necessary. First a rotation to align the points in the same orientation as the pixel frame was applied, as shown in Equation 1. In Equation 2, a translation was added to shift the origin of the map frame to intersect with the origin of the pixel frame. Third, the points were scaled using the map to pixel resolution factor, as displayed in Equation 3.

$$\begin{bmatrix} X_r \\ Y_r \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{1}$$

$$(X_1, Y_1) = (X_r + X_t, Y_r + Y_t) \tag{2}$$

$$(u, v) = (\frac{X_1}{resolution}, \frac{Y_1}{resolution}) \tag{3}$$

Another important aspect of the map is dilation. Figure 1 shows the grid without dilation, also known as the work space. The work space defines any point that is technically unoccupied, but in practice points close to the wall will result in collision if the robot travels there. Figure 2 displays the grid with dilation or the configuration space. The configuration space defines the points that are feasible for the robot to occupy. The dilation adds an extra buffer between the robot, and the walls. Next the data from the grid was used to create a graph with a tuple of indices as keys, and a list of k nearest neighbors as the values for randomly sampled points.

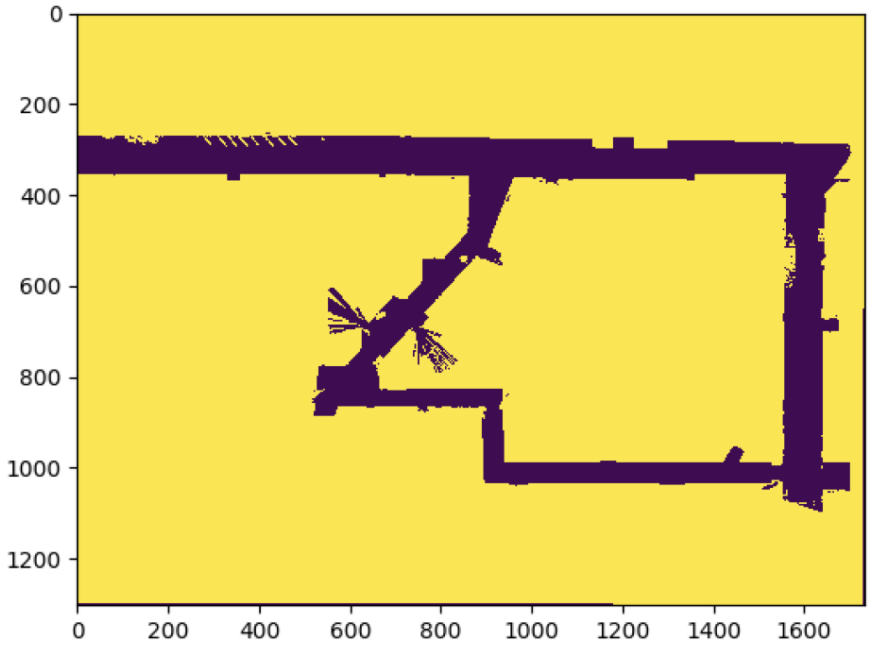

Figure 1: Visualization of the occupancy grid without dilation, also referred to as the work space. Purple shows the areas that are available, while yellow shows the areas that are not available. The work space defines all points that are technically unoccupied, but does not factor in the size of the robot.

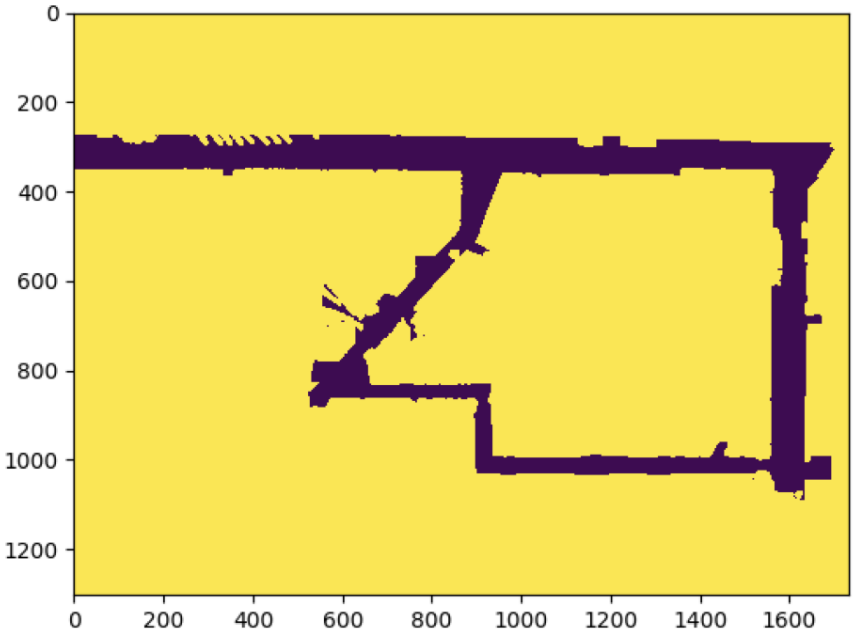

Figure 2: Visualization of the occupancy grid with dilation or the configuration space. Purple shows the areas that are available, while yellow shows the areas that are not available. The configuration space defines all points that are unoccupied, and distanced away from obstacles so that the robot can occupy the space without collision.

2.1.2 Probabilistic Roadmap

Author: Mario Peraza

Sampling-based algorithms use a set of points sampled from an occupancy grid, potential trajectories, and a collision checking module to create a graph of feasible trajectories otherwise known as a roadmap. One of the leading sampling-based algorithms is the probabilistic roadmap (PRM) (1). The PRM algorithm samples the occupancy grid for collision free points and creates nodes at these locations. These nodes are used to create a roadmap where each point is connected by a path which does not collide with objects. These paths are then used to determine the most efficient way to get from a starting location to an ending location. Variants of the PRM algorithm have been developed and compared by Sertac Karaman and Emilio Frazzoli (2). These comparisons were used to determine the design of our teams path planner (Figure 3).

	Algorithm	Probabilistic Completeness	Asymptotic Optimality	Monotone Convergence	Time Complexity		Space Complexity
					Processing	Query	
Existing Algorithms	PRM	Yes	No	Yes	$O(n \log n)$	$O(n \log n)$	$O(n)$
	sPRM	Yes	Yes	Yes	$O(n^2)$	$O(n^2)$	$O(n^2)$
	k-sPRM	Conditional	No	No	$O(n \log n)$	$O(n \log n)$	$O(n)$
	RRT	Yes	No	Yes	$O(n \log n)$	$O(n)$	$O(n)$

Figure 3: Summary of results for the variants of the PRM algorithm (2).

Of the sampling-based methods, a simplified PRM that evaluates k nearest neighbors for paths (k-sPRM) adequately met our needs. Given a fraction of the thousands of nodes in the free configuration space and sufficiently large $k \geq 5$, k-sPRM produces a reasonably smooth path compared to the other variants. In terms of completion, across dozens of runs, there were zero failures to complete a path. Of the algorithms considered, k-sPRM had a low implementation cost, was easily visualized for debugging, and matched the time and space efficiency of traditional PRM.

To implement the k-sPRM algorithm, randomly sampled points in unoccupied spaces must be chosen. The occupancy grid is used to locate unoccupied space which allows a helper function to randomly select 2% of these points to become nodes on the roadmap. The roadmap is then passed into the *simple_prm* function

```
def simple_prm(k, nodes, graph):
    '''
    Builds a k-nearest simplified probabilistic roadmap
    (k-sPRM). Saves valid paths in the edge map 'graph'.
    '''
    for v in nodes:
        dest_nodes = k_nearest(v, k)
        for u in dest_nodes:
            if u in graph[v]:
                continue
            elif collision_free(v, u):
                graph[v].append(u)
                graph[u].append(v)
    return
```

where k is the number of nearest neighbors to a specified node, v. Nearest neighbors are determined by finding k nodes that are the shortest distance away from the specified node. After the k nearest neighbors are found, a graph is created by connecting paths between each specified node and its neighbors as long as the path is collision free.

The k-sPRM algorithm is able to generate a roadmap given an occupancy grid by randomly sampling unoccupied points and linking collision free paths between

them. However, there can be multiple pathways from a given starting node to an end node. To choose the most efficient path, a search-based algorithm must be applied to the generated roadmap.

2.1.3 A^* Search Algorithm

Author: Isaac Lau

Once the probabilistic road map has been created, the A^* search algorithm is run to find the shortest path from the start to end nodes with the PRM.

Out of the many different search-based algorithms, the team decided to implement A^* to maximize search efficiency. As a hybrid search-based method that uses Dijkstra's shortest path algorithm in the background in addition to an informed heuristic, A^* does not naively branch out from the start node, look at its neighbors and the distances to its adjacent nodes, and then repeatedly picks the next node with minimal distance. Instead, A^* utilizes a heuristic to guide the search and alter the ordering of which nodes are explored first.

In the A^* search algorithm, the heuristic function is comprised of a $g(n)$ and $h(n)$ value. In the context of this algorithm, $g(n)$ represents the exact cost of the path from the starting point to any vertex n. Conversely, $h(n)$ represents the heuristic estimated cost from vertex n to the goal node. As A^* moves through the the map's free configuration space, it balances the two of these values, examining the vertex n that has the lowest summation of $g(n)$ and $h(n)$ scores until it arrives at the goal node.

By running A^* in conjunction with PRM, the path planning algorithm is able to quickly find a path from the start to end node.

2.2 Pure Pursuit

Author: Joshua Rapoport

Once a path has been generated by the PRM-A^* pipeline, it is converted into a trajectory of transient points and handed off to the path follower. A pure pursuit controller was evaluated using a transient goal position chosen along the trajectory.

The first step is to identify the closest point on the trajectory to the robot. This was done by calculating the closest point on every segment of the trajectory.

Given the start and end vectors on a given segment v, W, and a robot position p, the trajectory segment can be parametrized:

$$p_{close} = v_i + t_i^*(w_i - v_i)$$

$$t_i^* = \frac{(p - v_i) * (w_i - v_i)}{|w_i - v_i|^2}$$

where t_i^* is the parameter $t \in [0, 1]$ of the closest point p_{close} on any line segment i to the robot. Calculating p_{close} for all segments and finding the minimum distance $|p_{close} - p|$ will return the index i of the absolute closest segment.

Second, we define a look ahead point, or transient goal point, which the pure pursuit will use to select a steering angle. In the past (see Lab 4), transient goal was defined naively as the nearest trajectory point to the robot. While successful on dense, clean trajectories, this approach was not robust on sparse or noisy trajectories.

Parametrizing all trajectory segments, there will be at least one point $p_{tran} = v + t^*(w - v)$ that intersects with the look ahead radius. (Note: solving for t^* can return two real or complex values. The solutions relevant to the transient goal are real values of the form $t = \frac{-b+\sqrt{b^2-4ac}}{2a}$).

Only solutions after segment i_{close} (i.e. in front of the robot) are considered valid. If no solutions at the look ahead are found, the robot is either far away from the trajectory, or approaching the final goal position.

Upon the robot's odometry being updated (either by the "ground truth" simulator or by the localization package), a callback is made to the path planner:

```
def odom_cb(msg):
    p_robot, heading = pose_from_msg(msg.pose)

    # 1. Closest Trajectory Point
    p_close, i_close = self.get_closest_point(p_robot)

    # 2. Transient Goal Position
    p_goal = self.get_transient_goal(i_close)

    # 3. Pure Pursuit
    turning_angle = self.evaluate(p_goal)

    # Publish drive instructions (speed, turning_angle)
```

which finds the appropriate turning angle for the robot of wheelbase length L, look ahead distance d, and velocity V_{max} to coincide with the transient goal.

2.3 Integration

Author: Lilly Papalia

2.3.1 RViz Integration

Integration of the path planner, pure pursuit, and localization was tested in the 2D rviz environment. To do this, all three processes were run in unison. Execution of the three processes resulted in the path being planned first, then localization was used to locate the car and help control the pure pursuit to successfully follow the path.

2.3.2 TESSE Integration

Author: Lilly Papalia

Integration in the 3D photorealistic TESSE simulator has a few differences from the 2D rviz version. While running pure pursuit in 2D, localization was utilized. On the other hand, in 3D the TESSE ground-truth pose was used to locate the car. Another difference was adding a transform listener between the map and world frames. The transform listener converts the odometry pose from the world frame to the map frame.

Another major difference between rviz and TESSE was the change in maps. A visualization of the TESSE map with dilation is shown in Figure 4. Unlike the rviz simulation, TESSE has obstacles that must be taken into account while path planning. To account for the obstacles, a map with collision-free paths was utilized. The collision-free map is shown in Figure 5.

Figure 4: Visualization of the TESSE map with dilation for the walls. The yellow represents the walls, or unavailable area, while the purple represents the unoccupied areas.

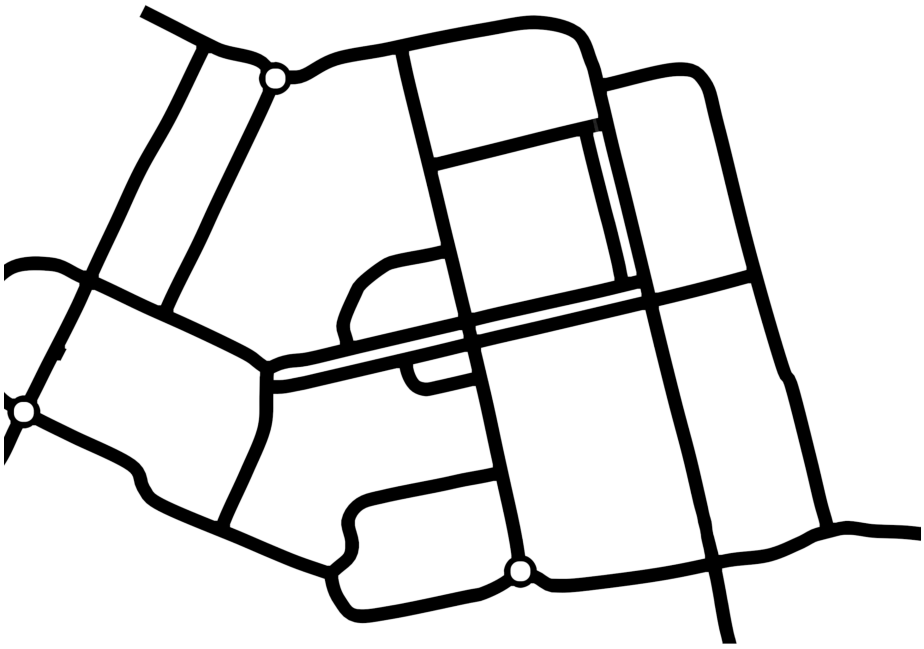

Figure 5: TESSE collision-free map which accounts for the obstacles in the TESSE simulation. The collision-free map can be used to improve the path planning in TESSE.

3 Experimental Evaluation

3.1 Path Planning

Author: Lilly Papalia

The first test for path planning was a visual test. Before using the probabilistic roadmap to speed up the execution of the path planner, a simple A^* implementation was tested. While A^* is the most efficient of the search-based planners, the runtime for executing on the stata basement map was significant. An image of the path using A^* is shown in Figure 6.

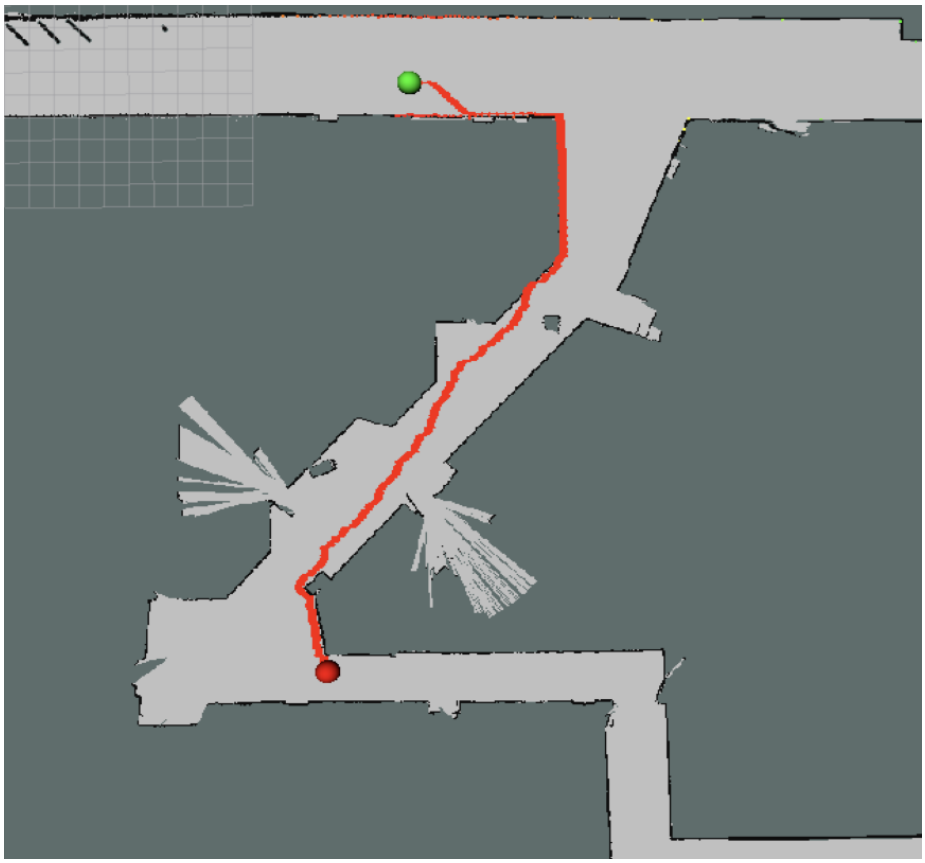

Figure 6: Visualization of the path using just A^*. The path is very clean and short, but took a significant amount of time to run.

Compared to A^* alone, which scales poorly with increasing number of nodes, implementing a PRM-to-A^* pipeline resulted in faster runtimes. The path was not as clean as the full A^* path, but the reduction in smoothness was made up for by the speed of the algorithm. An image of the path using PRM and A^* is shown in Figure 7.

Figure 7: Visualization of the path using PRM and A^* together. The path is not as smooth as the full A^* path, but executed in a fraction of the time.

Another check to confirm the validity of the path planner was through the use of the gradescope autograder. Figure 8 shows the PRM and A^* path overlayed with the staff solution. As shown in the image, the path planner followed the staff solution very closely, visually confirming the success of the path planner.

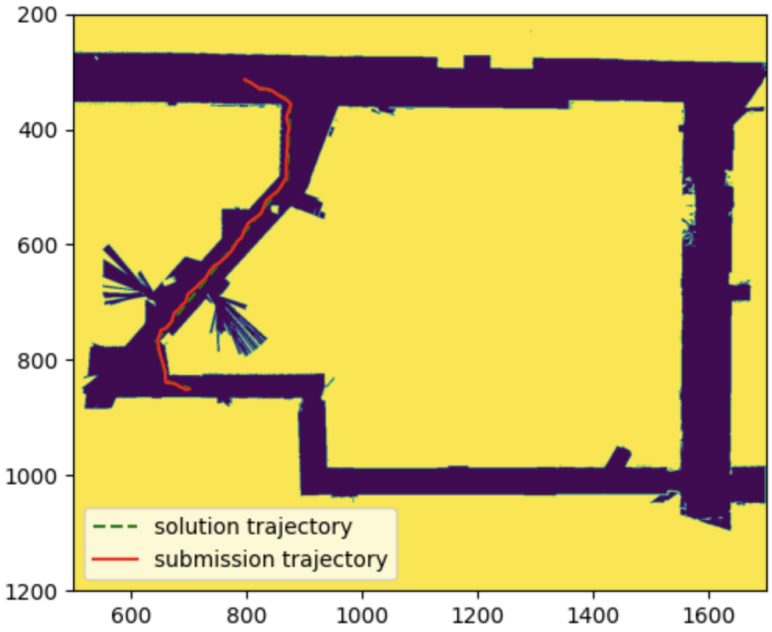

Figure 8: Path planner compared to staff solution. The two paths are visually very similar confirming success of the path planner.

Author: Isaac Lau

In terms of efficiency and runtime of the combination of A^* search algorithm, it is noted that as a search-based method, it is more efficient than the simpler BFS, DFS, or Dijkstra implementations. However, it is still subject to slow runtimes since it will be searching a large section of the occupancy grid. However, to reduce runtime, A^* was run on the probabilistic roadmap instead. [See the comparison of runtime between A^* run with PRM and without PRM on the occupancy grid in Figures 9 and 10]. By reducing the number of nodes that A^* needs to traverse through PRM, it executes several orders of magnitude (approx. 2^3 times) faster than before.

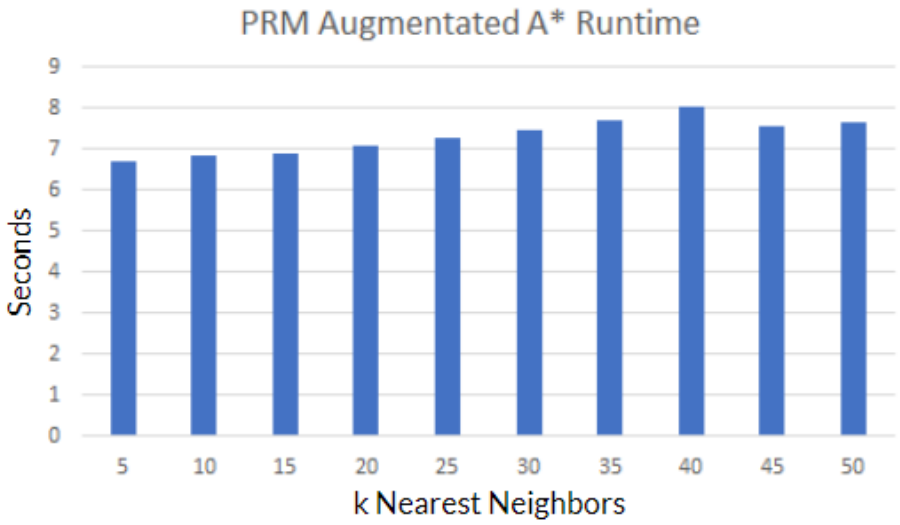

Figure 9: Visualization of the runtime of the combined A^* search algorithm and probabilistic roadmap (PRM) implementation with k-sPRMs generated from a random sample of N/100 nodes

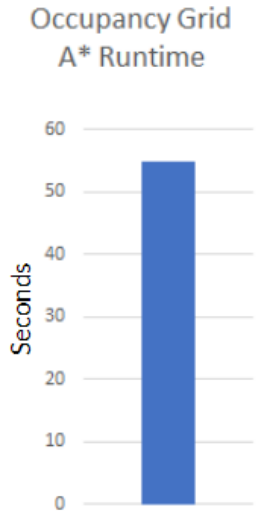

Figure 10: Visualization of the runtime of the A^* search algorithm on the occupancy grid.

Author: Lilly Papalia

Path planning is split into two sections, PRM and A*. Six trials of the path planner were run using the same start and end points, shown in Figure 11. As shown in Figure 12, the total run time for the same start and end nodes varies slightly. This is due to the random sampling of points, leading to each trial being slightly different. The maximum time difference was a roughly 3 seconds, approximately 7 percent of the total run time. Further analysis revealed, the large majority of the run time is spent making the probabilistic road map, while under a second is spent running A*, as shown in Figure 13. Since a small map is being passed into A*, the search algorithm can run quickly and efficiently. Dedicating run time to building the PRM pays off with the speed of A* on a smaller map. The A* runtime is also more consistent than the PRM runtime over the six trials.

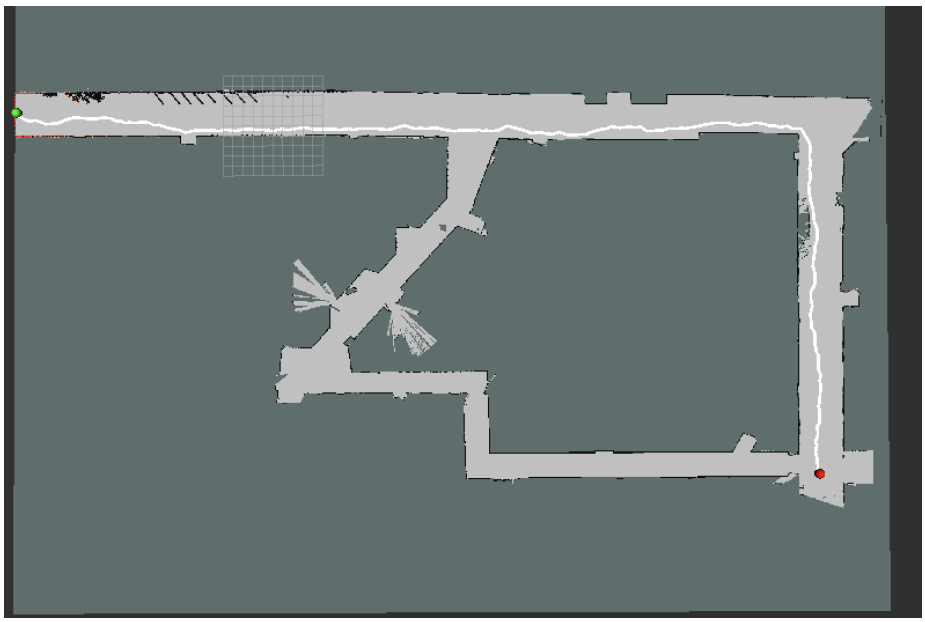

Figure 11: The path used in the six trials of path planner. The start and end points were the same for every trial, but the actual path varies slightly because of the PRM generated.

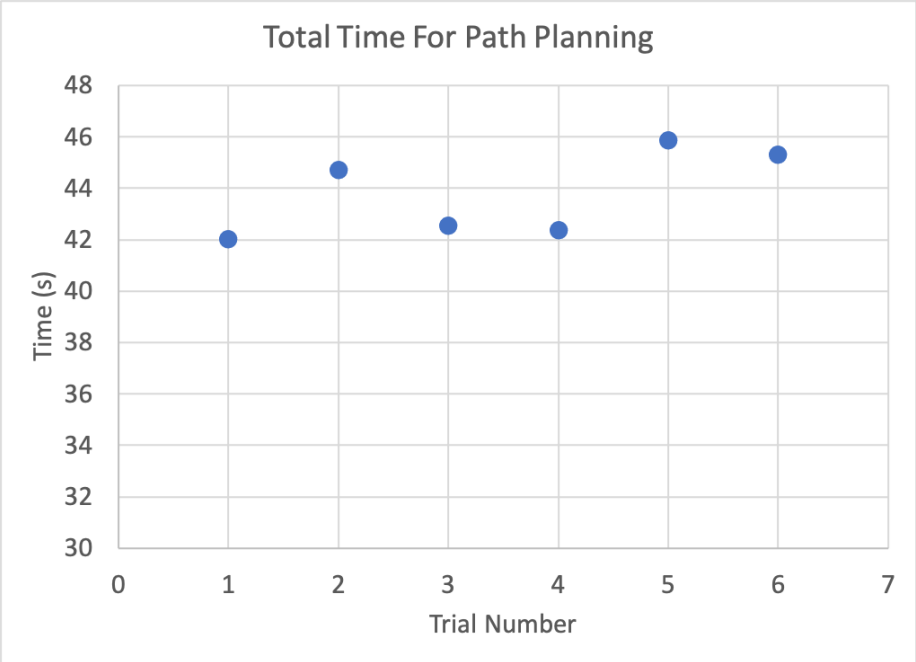

Figure 12: This chart displays the total run time of the path planning algorithm over 6 trials using the same start and end points. There is slight variation between the six trials, but overall the algorithm is consistent.

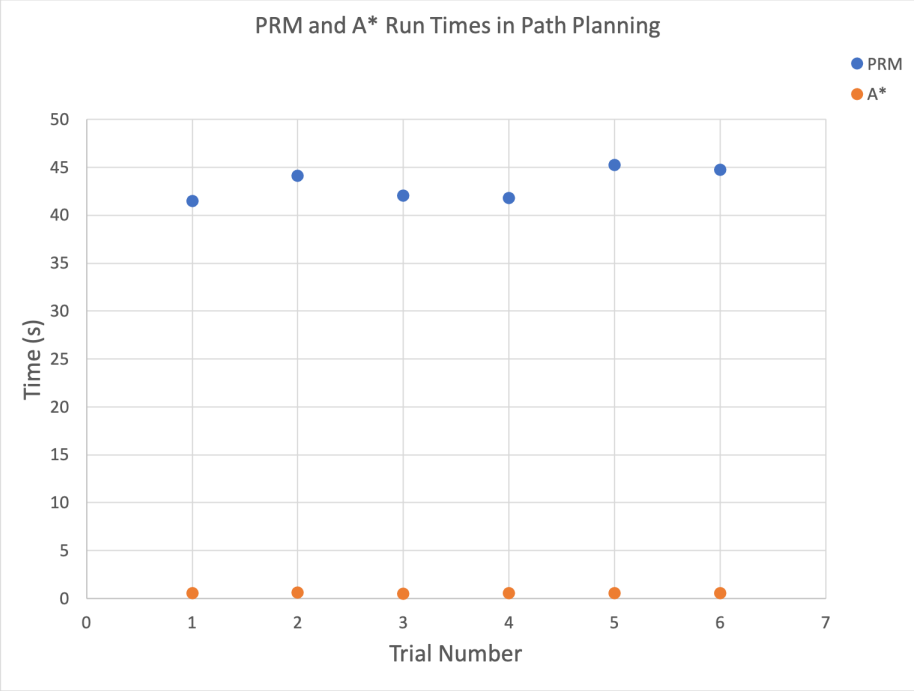

Figure 13: This chart displays the run time of PRM compared to the runtime of A* in the path planner. The run time of PRM is significantly larger, but the speed of A* running on a smaller map makes PRM worth the invested time.

3.2 Pure Pursuit and Localization

Author: Joshua Rapoport

Although the path follower was given initial parameters (for look-ahead and velocity) before integrating with the path finder, those parameters required refinement when converting from ground-truth odometry (GT) to localized odometry.

Using localization, robot performance was much more sensitive to changes in heading. Using constant look-ahead $L = 2.0m$ and velocity $v = 2.0\frac{m}{s}$, the robot handled parts of the path in Figure 14 poorly. In particular, the robot failed right-angle turns by either clipping the inner corner of the wall, or missing the turn in a wide arc.

Figure 14: A visualization of the path in rviz generated by the PRM-A^* pipeline.

After re-tuning the parameters to $L = 1.8m, v = 2.2\frac{m}{s}$, the robot no longer clips the corner, but the turns misses still occasionally occurred. Generally, the solution would be to slow down at turns, which allows the localized odometry to adjust accordingly (further tuning to the localization package could also yield better results).

Focusing on pure pursuit: if the parameters are re-contextualized as the maximum look-ahead L_{max} and maximum velocity v_{max}, then they can be scaled using normalized curves $\in [0, 1]$ dependent on significant factors - in this case, turning angle δ.

After several combinations were tested, two curves showed promising results:

$$v(\delta) = \frac{v_{max}}{\cosh\left(\frac{\pi}{2}|\delta|^{1.8}\right)} \tag{4}$$

$$L(\delta) = \begin{cases} 0.2L_{max} & \text{if } \frac{v}{v_{max}} < 0.2 \\ L_{max}\frac{v(\delta)}{v_{max}} & \text{if } 0.2 < \frac{v}{v_{max}} < 1.0 \\ L_{max} & \text{else} \end{cases} \tag{5}$$

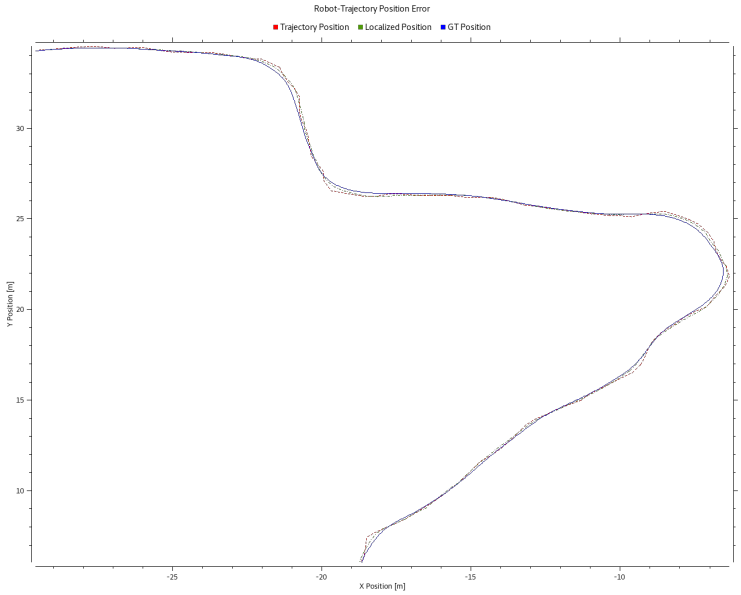

Figure 15: The parametric position error of the robot with respect to the trajectory in Figure 14. This plot compares the closest trajectory position (red), localized robot position (green), and ground truth robot position (blue).

As seen in Figure 15, by re-tuning the pure pursuit controller to use localized odometry, and by damping the velocity on sharp turns, the robot's deviation from the trajectory is greatly reduced. Observe that, although there is a slight overshoot of the localized and ground truth positions of the robot, it corrects quickly.

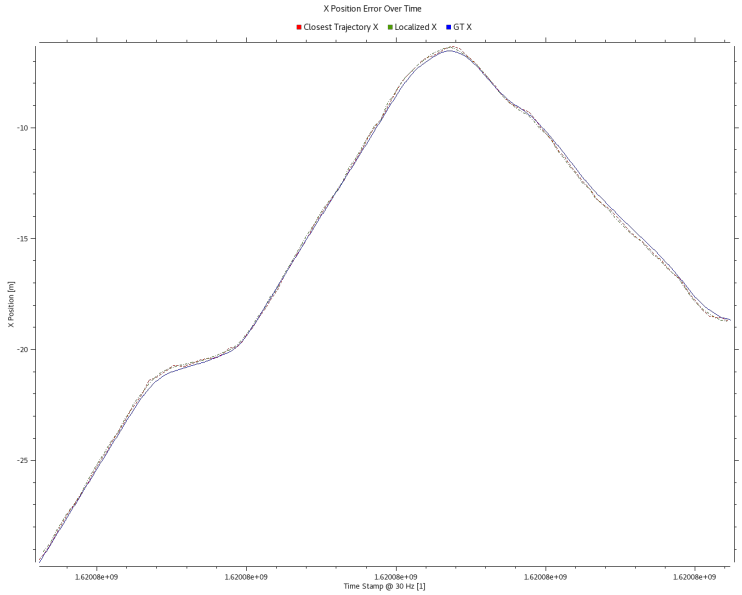

Figure 16: The position error of the robot about the x-axis of the map frame, with respect to time.

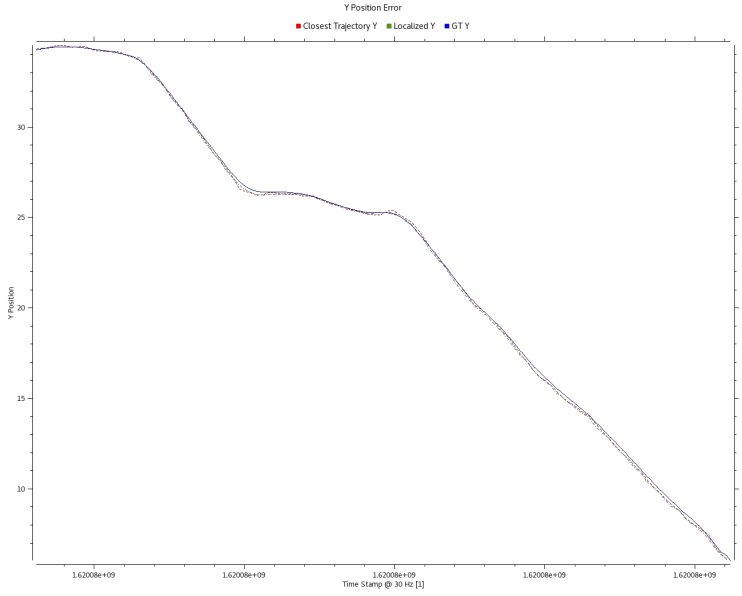

Figure 17: The position error of the robot about the y-axis of the map frame, with respect to time.

For the entire run, the x and y errors (Figures 16 and 17) were less than the wheelbase length of the robot (0.35m for the 2D simulation), suggesting that the robot stayed on the path at all times.

3.3 Integration

3.3.1 RViz Implementation

Author: Lilly Papalia

Integration of localization, path planning, and pure pursuit was successful in the 2D rviz environment. While turning around corners, the robot gets very close to the edge. Increasing the dilation or changing the path planning to account for corners could improve this issue. A video of the whole system running in rviz can be seen here. The green marker represents the localization, the start position is the green sphere, and the end point is the red sphere. As shown in the video the path is created, then the robot uses localization and pure pursuit to successfully follow the path.

3.3.2 TESSE Implementation

Author: Lilly Papalia

For the TESSE implementation, the number of nearest neighbors was reduced from the 2D version, and the number of random nodes selected was reduced due to the increase in map size. Unfortunately, due to transform issues from converting from rviz to TESSE, integration in TESSE was unsuccessful. On the other hand, the VDI worked super well, and will definitely be useful for the final challenge. As a team we would like to continue working on the TESSE implementation to better prepare us for the final challenge.

4 Conclusion

Author: Mario Peraza
The path planner, pure pursuit model, and localization were successfully integrated into the 2D environment. The path planner was able to determine the most efficient path from a given start point to a given end point. This path was accurately followed through our pure pursuit model with the help of localization. Multiple path planning algorithms were tested and compared, contributing to the final design of our algorithm which utilized both sample-based and search-based algorithms. The pure pursuit model was tuned to ensure the simulated car was able to accurately follow a given path with few disturbances. In the future, the algorithms and models which were developed in this lab can be scaled and tuned to be useful in other environments.

5 Lessons Learned

5.1 Joshua

The lab this week was incredibly taxing and rewarding. Among other feats, our team made the jump to developing and formally comparing multiple approaches. This is an extension of our balance of parallelization of work and pair collaboration.

I think that, in addition to stumping several TAs with a bug in our system, we had issues related to a breakdown in communication. Among many errors found while debugging, we found two errors that stood out. One was in our utils file, in a method that had not been peer-reviewed or unit-tested, and it was only discovered by the person who originally wrote the code. The other was to do with expectation of numerical convention that had not been kept in the code. This change, however, was not uploaded, and was kept on a local branch until the discrepancy was discovered.

Some of the more meaningful changes we can make going into the final ROS integration is to (1) have two eyes on everything written, and (2) consolidate unit testing for all methods. Working in pairs has worked well (when it is actually done), but formalizing frequent code review could mitigate the impact of solo work. Unit tests could save hours more of debugging, and guarantee expected function for our code prior to integration.

5.2 Lilly

The lab this week presented a challenge with many moving parts, but our team tackled the challenge leading to a successful implementation in the end. Working together on sections proved to be helpful, as in the past few labs. Working in smaller pairs is definitely an approach we will be using going into the final

challenge.

One lesson learned from this lab is to take into account all files while troubleshooting errors. A small error in our utils file sabotaged our whole system, and this bug was very difficult to find. In the future, we now know to consider looking into the helper files for small bugs.

Overall, integration of our separate sections went smoothly, something that can not be said for previous labs. Communication about variable names, and what data structures we are using was definitely important in this lab and in the final challenge.

5.3 Mario

As a team, this was one of our most successful labs. We were able to communicate a lot better. We learned to leave very descriptive comments when writing code to allow everyone to easily read through new code. Once again, we paired off and handled different sections which was very effective. It works well when someone is constantly checking the code for bugs and it helps to have someone who is working on the same item as you as more ideas can be easily generated.

There are always technical challenges which hold us back such as the TESSE environment not being able to run or small bugs which take hours to resolve, however everyone on our team is dedicated to finding solutions to these problems.

5.4 Isaac

As one of the final labs of RSS 2021, both the code base and technical material was certainly challenging for me. However, once again, peer programming was extremely helpful and allowed us to maximize efficiency and learn from each other. Working with larger sections of code, this lab also gave me more experience working with algorithmic implementations in a more professional setting.

In terms of personal and communications growth, we were able to continue building off of the progress from prior weeks. Compartmentalizing code sections did lead to some initial integration challenges, but by coming together earlier and talking through our code over Zoom, we were able to successfully resolve any problems.

References

[1] Geraerts, Roland, and Mark H. Overmars. "A Comparative Study of Probabilistic Roadmap Planners." In Algorithmic Foundations of Robotics V, edited by Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth Hutchinson, 7:43–57. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. https://doi.org/10.1007/978-3-540-45058-0_4.

[2] Karaman, Sertac, and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning." ArXiv:1105.1186 [Cs], May 5, 2011. http://arxiv.org/abs/1105.1186.

[3] Shortest distance between a point and a line segment. (2018, April 01). Retrieved from https://stackoverflow.com/questions/849211/shortest-distance-between-a-point-and-a-line-segment/15017251501725

[4] Line segment to circle collision algorithm. (2015, Oct 17). Retrieved from https://codereview.stackexchange.com/questions/86421/line-segment-to-circle-collision-algorithm/8642886428