Lab 6 Report: Pathfinding on the Racecar

Team 5

Alvin Banh
Jialin Chen
Nevena Stojkovic
Haris Imamovic

6.4200: Robotics Science and Systems

April 24, 2025

1 Introduction

Author: Nevena Stojkovic, Editors: Alvin Banh, Jialin Chen, Haris Imamovic

In this lab, the objective was to develop an autonomous vehicle system capable of both planning and executing paths in a dynamic environment. The challenge was to create a system that could not only plan a collision-free path but also follow it accurately without hitting obstacles.

To achieve this, we implemented a Jump Point Search (JPS) algorithm for path planning. This search-based method efficiently calculates the shortest path in a grid environment by pruning unnecessary nodes, accelerating the pathfinding process. The algorithm was tested on an occupancy grid map, dilated to ensure that the planned path avoided obstacles. Once the path was generated, it was visualized in RViz to ensure it connected the initial position to its specified destination.

For trajectory tracking, we applied a Pure Pursuit controller to follow the planned path. The controller works by determining a lookahead point on the trajectory a fixed distance ahead of the vehicle and steering towards it. This method required tuning to handle different path curvatures and ensure smooth navigation, accounting for the vehicle's wheelbase length and motion constraints.

Finally, the path planning and trajectory following components were integrated to enable real-time operation. This involved subscribing to the vehicle's localization data from the particle filter to update the vehicle's position continuously and adjust the path as necessary. We tested the system in simulation and on the physical vehicle, iterating on the parameters to optimize performance for accurate and smooth path execution.

Overall, the lab provided hands-on experience in combining motion planning and control techniques, essential for autonomous navigation in real-world environments. It demonstrated how to plan, track, and integrate a system that allows a vehicle to navigate autonomously in a map, avoiding obstacles and reaching the goal with high precision.

2 Technical Approach

Authors: Alvin Banh, Jialin Chen, Nevena Stojkovic, Haris Imamovic

2.1 Technical Problem and System Overview

The technical problem is implementing an optimal and feasible path planning algorithm, following the shortest path, and integrating it with localization algorithms. Our planner has to generate the most efficient path (optimal) between a start and an end point. This trajectory should not cross any obstacles or walls (feasible). We implemented an optimized version of the A-star algorithm known as Jump Point Search (JPS) to generate this trajectory and used Pure Pursuit to follow the path. Our technical approach for our localization algorithm was fine-tuning noise deviations and particle amounts to see how accurately the racecar's position is in the map compared to the real world, extending from the previous lab.

2.2 Grid Processing

In order to navigate the Stata basement map, the trajectory planner is passed an occupancy grid containing pixel data that gives information on obstacles like walls and other objects that cannot be moved through. To use a search-based path-finding algorithm on the grid, some physical limitations of the robot need to be taken into consideration, mainly the fact that it most likely will drift from its intended path and that it has a non-1-dimensional width. To account for this, the occupancy grid is dilated where each occupied (meaning non-traversable) pixel is expanded into a disk, as shown on Figure 1, making its immediate neighbors occupied as well. This ensures that there is room for the robot to pass without colliding into the wall (though the safety controller will be run alongside to ensure no physical damage occurs).

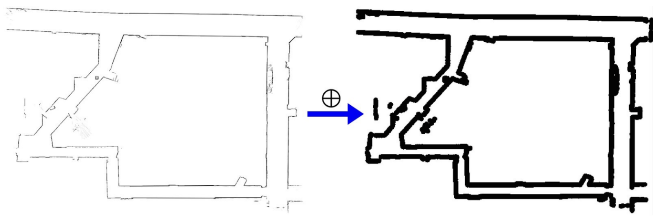


Figure 1: Each black pixel in the map on the left, represented by a maximum value in the occupancy grid, is expanded onto its neighbors using the dilate function, as represented on the right picture

2.3 Jump Point Search

One downside to the A-star algorithm is that it may make unnecessary checks to calculate the shortest path between two points. While A-star checks its immediate neighbors, JPS prunes the set of immediate neighbors around a node by trying to prove that an optimal path exists from the current node's parent to each neighbor and that path does not involve visiting the current node, hence eliminating many symmetrical (unnecessary) paths from being considered, as shown in Figure 2.

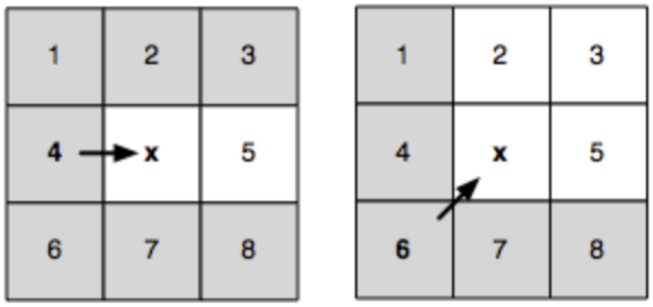


Figure 2: Node x is the current node being processed, with the arrow indicating the direction of travel from its parent. Both in diagonal and straight movement, all grey neighbors are not considered as these can be reached optimally from the parent of x without ever going through x.

Because some paths are ignored due to symmetry, we need to re-include some neighbors that might not be considered due to an obstacle blocking the path from the current node's parent, demonstrated in Figure 3

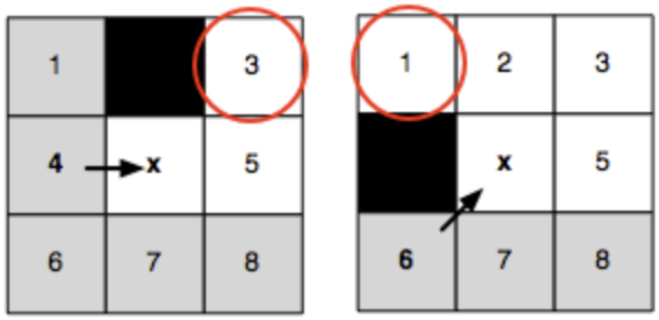


Figure 3: The arrow indicates the direction of travel to node x from its parent. When x is adjacent to an obstacle, the highlighted neighbors cannot be pruned and need to be included, since any alternative optimal path, from the parent of x to each of these nodes, is blocked.

To further reduce symmetries, the algorithm recursively "jumps" over all nodes that can be reached by a path without visiting the current node, stopping the recursion once an obstacle is hit, or a node that has neighbours not reachable via other symmetrical paths, illustrated in Figure 4

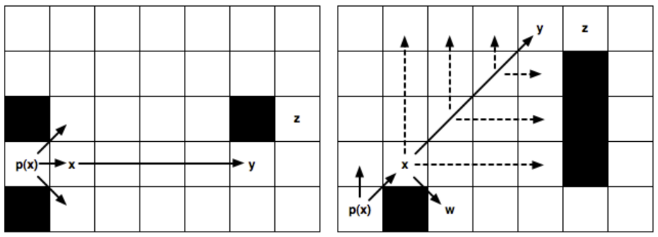


Figure 4: Straight pruning is performed, and y is identified as a jump point successor of x, since it has a neighbor z that cannot be reached optimally except by a path that visits x then y. The intermediate nodes are never explicitly generated or even evaluated. When the diagonal pruning is performed, y is again the jump point successor of x. Before each diagonal step, it first recurses straight (dashed lines), and only if both straight recursions fail to identify a jump point, it steps diagonally again. Node w, a forced neighbor of x, is generated normally.

2.4 Particle Filter/Monte Carlo Localization

The pipeline of our localization implementation is as follows in Figure 5:

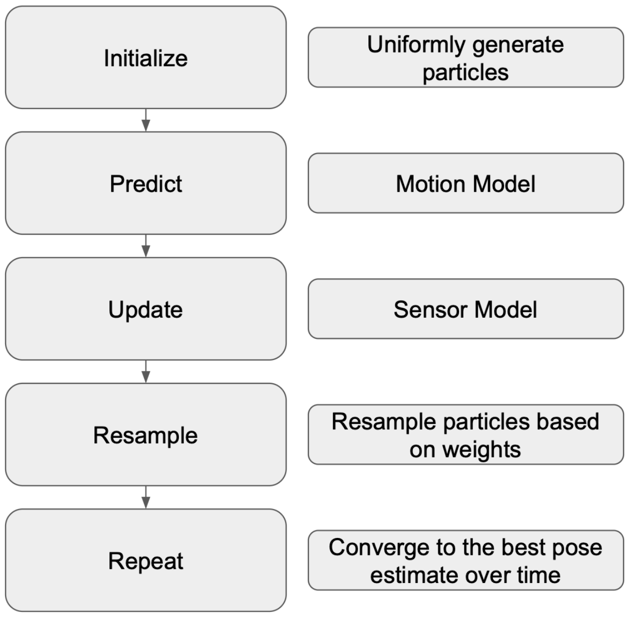

Initialize
Uniformly generate particles
Predict
Motion Model
Update
Sensor Model
Resample
Resample particles based on weights
Repeat
Converge to the best pose estimate over time

Figure 5: The left column of this figure shows the high level series of actions in implementing MCL. The right side is the method of implementation.

The particle filter is implemented to perform MCL based upon updates from the motion and sensor model as the robot moves. It creates pose estimates from a set of particles from the odometry data from the VESC and sensor data from the LIDAR. The purpose of a particle filter is to compute likelihoods of a set of particle and resamples to get better positional estimates. We have added noise to our motion model to account for odometry drift on the real car.

We then compute the average of the particle pose and publish the odometry message. This process calls upon the previous sensor and motion model and processed in the order outlined in Figure 5.

2.5 Pure Pursuit

The goal of the Pure Pursuit controller is to enable an autonomous vehicle to follow a predefined trajectory by continuously calculating a steering angle that directs the car toward a lookahead point. The lookahead point is defined as a point a fixed distance ahead of the vehicle, guiding the steering behavior along the path. The main challenge lies in selecting an appropriate lookahead point to ensure smooth navigation while accounting for the vehicle's dynamics.

2.5.1 Path Segmentation and Lookahead Point Calculation

To implement the Pure Pursuit method, the trajectory is divided into segments. A segment refers to a straight-line portion of the path between two consecutive points on the trajectory. These segments are the basic units the vehicle follows. Each segment is represented by two points: the start and end points. Dividing the path into segments simplifies the calculation of the nearest point and the lookahead point, making the algorithm more computationally efficient.

At each timestep, the car calculates the distance to every segment in the trajectory and identifies the nearest segment. The segment with the smallest distance to the vehicle's current position is selected for the next calculation. From this segment, the lookahead point is computed, which is located a fixed distance ahead of the car along the path.

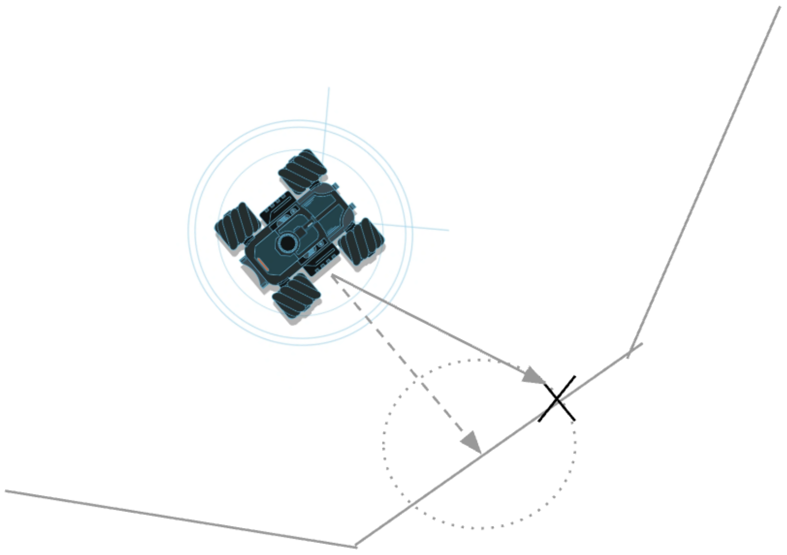


Figure 6: The robot calculating the lookahead point, which is a fixed distance ahead along the trajectory to guide its movement.

If the lookahead distance is larger than the length of the current segment, the remaining lookahead distance is carried over to the next segment. This ensures the vehicle always targets a point ahead on the trajectory, even when the lookahead distance exceeds the length of the current segment.

2.5.2 Steering Angle Calculation

Once the lookahead point is determined, the next step is to compute the steering angle. This angle is required to steer the vehicle toward the lookahead point. The steering angle is calculated using the following formula:

$$\delta = \arctan\left(\frac{2L \cdot \sin(\alpha)}{d^2}\right) \tag{1}$$

Where:

- δ is the steering angle.
- L is the wheelbase length of the car.
- α is the angle between the car's current heading and the line connecting the car's position to the lookahead point.
- d is the lookahead distance.

This formula is derived from the geometry of the vehicle's motion, where the curvature of the path is inversely proportional to the steering angle. A larger lookahead distance leads to a smoother curve, while a smaller distance makes the vehicle more responsive and capable of tighter turns.

The decision to divide the trajectory into linear segments simplifies the Pure Pursuit method. By focusing on straight-line segments, the algorithm can efficiently compute the nearest point and determine the steering angle required to follow the trajectory. This segmentation is also critical when navigating complex environments with varying path curvatures, as it allows the lookahead point to be calculated dynamically as the vehicle moves. The lookahead distance plays a critical role in controlling the vehicle's response. Larger lookahead distances result in smoother turns but may reduce the vehicle's ability to respond quickly to sharp changes in the path. On the other hand, smaller distances enable quicker reactions, especially when navigating tight turns, but may lead to jerky steering behavior. By adjusting the lookahead distance based on the terrain or path curvature, the system can balance smoothness and responsiveness.
In practice, the Pure Pursuit controller is integrated with a path planning system that generates the trajectory. The car subscribes to the real-time localization data from the particle filter, which provides its current position. The vehicle then follows the trajectory using the Pure Pursuit algorithm, continuously

updating the steering angle to maintain its course. The integration of path planning, localization, and control allows the vehicle to autonomously navigate its environment, avoiding obstacles and reaching its destination.

3 Experimental Evaluation

Authors: Alvin Banh, Haris Imamovic, Jialin Chen

3.1 Testing Methodology

Our goal was to optimize both the trajectory planner and the follower. For the planner, we wanted to compare the performance of JPS versus A-star to determine whether JPS was faster at planning paths and how much time is saved. As A-star is the standard approach taken by most teams in this lab, we wanted to confirm whether our new approach, using JPS, would outperform the standard approach. Thus our first testing procedure involved using both algorithms to generate paths between three pairs of start and end points of increasing distance, and timing how long it took to generate that path. See Figure 8 for the generated paths from both algorithms on increasing trajectories.

Our second testing procedure was motivated by finding the best lookahead distance for different speeds. We decided to test speeds of 1 to 2.5 m/s, and lookahead distances of 0.5 to 1.5 with step sizes of 0.5 each for both. Figure 9 shows the path ran in all trials. This path has both straight and turning segments, which would best evaluate the robustness of our chosen parameters: speed and its corresponding lookahead.

3.2 Algorithm Optimization

Performance Metrics

We evaluated performance using:

1. **Computation Time:** The time for the A* and Jump Point Search algorithm to compute the shortest optimal path.
2. **Root Mean Square Error of Distance:** The root mean square distance error between the car and the closest point on the trajectory.

Computational Time Comparison of A* and Jump Point Search

We implemented two different path-finding and compared the time to compute a path using Python's time module. Figure 7 plots the average time over three trials each that both algorithms took to generate paths of three different lengths. The JPS optimization significantly improves the computation time, effectively halving it for long paths, making it a significantly more feasible option for real-life applications.

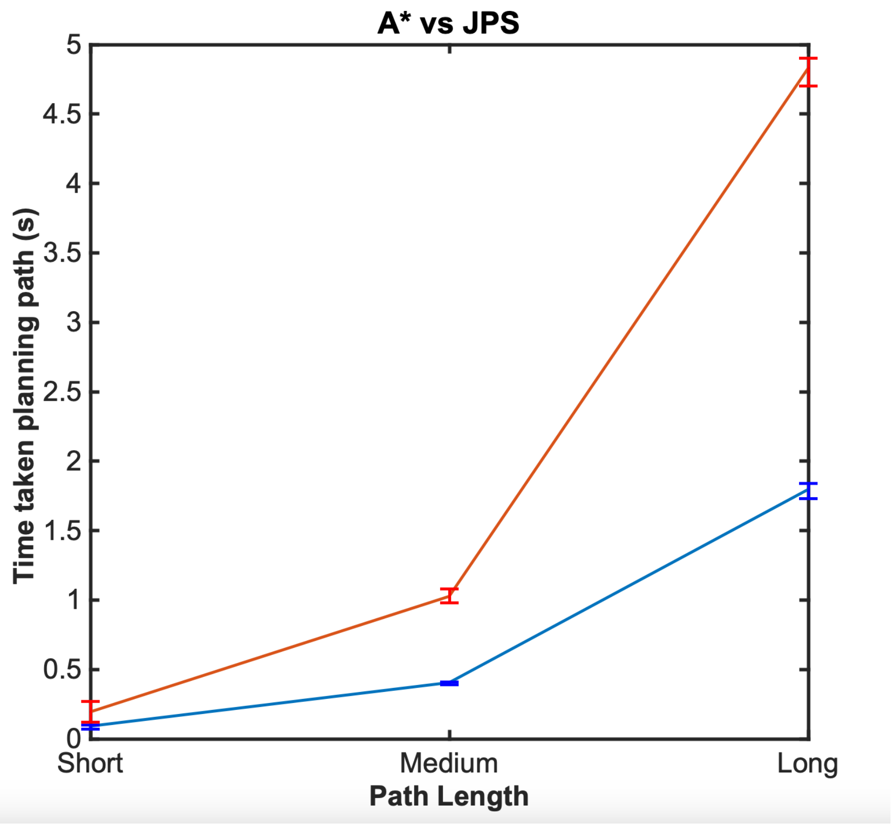
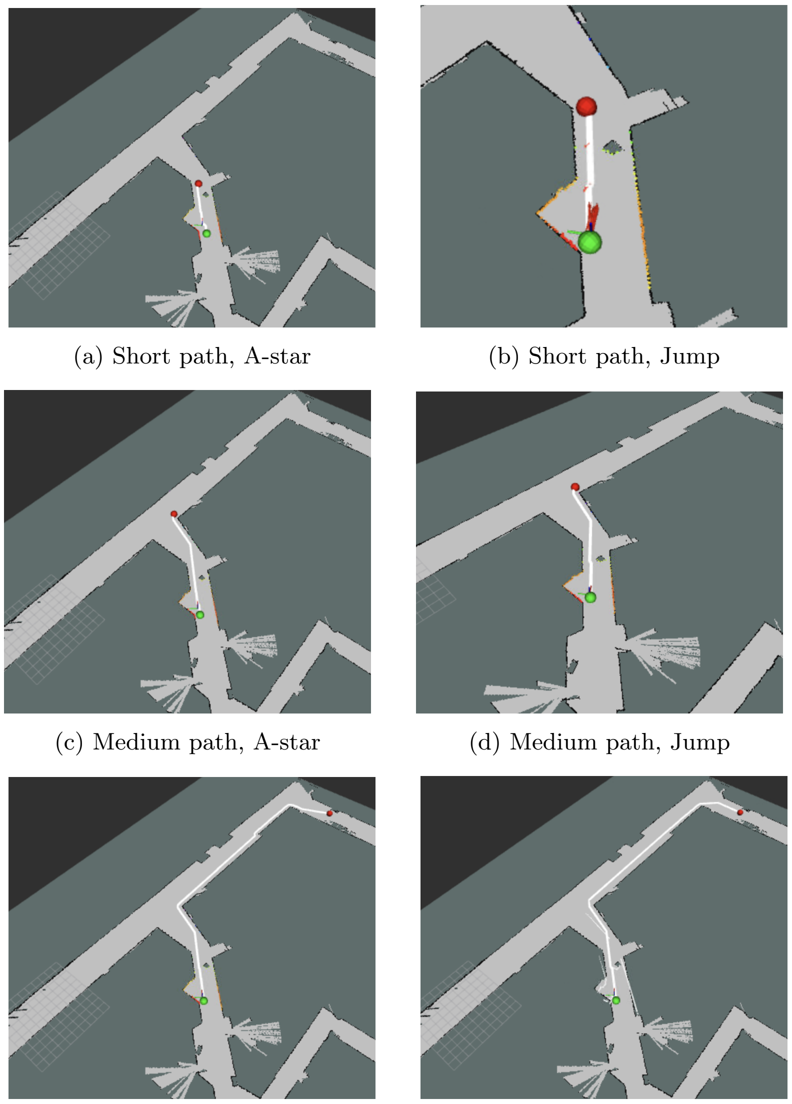


Figure 7: The red line shows the timing of A*. The blue line shows the timing of JPS. The efficiency of JPS is significant for larger paths, demonstrating a 62% decrease in time taken to find the shortest path.

Figure 8 shows the paths generated by both JPS and A-star using the same endpoints. JPS is able to find the path faster in all three cases.



(a) Short path, A-star (b) Short path, Jump

(c) Medium path, A-star (d) Medium path, Jump

(e) Long path, A-star (f) Long path, Jump

Figure 8: Paths in which the computation time is calculated as an average of 3 time trials

Distance Error

Figure 9 below is the path ran for all trials testing speed and lookahead distance. From our data, we saw that a lookahead of 1.0 m and 1.5 m worked best across all speed levels. Therefore we decided to compare more closely those two lookahead distances with the four speed levels in Figure 10.

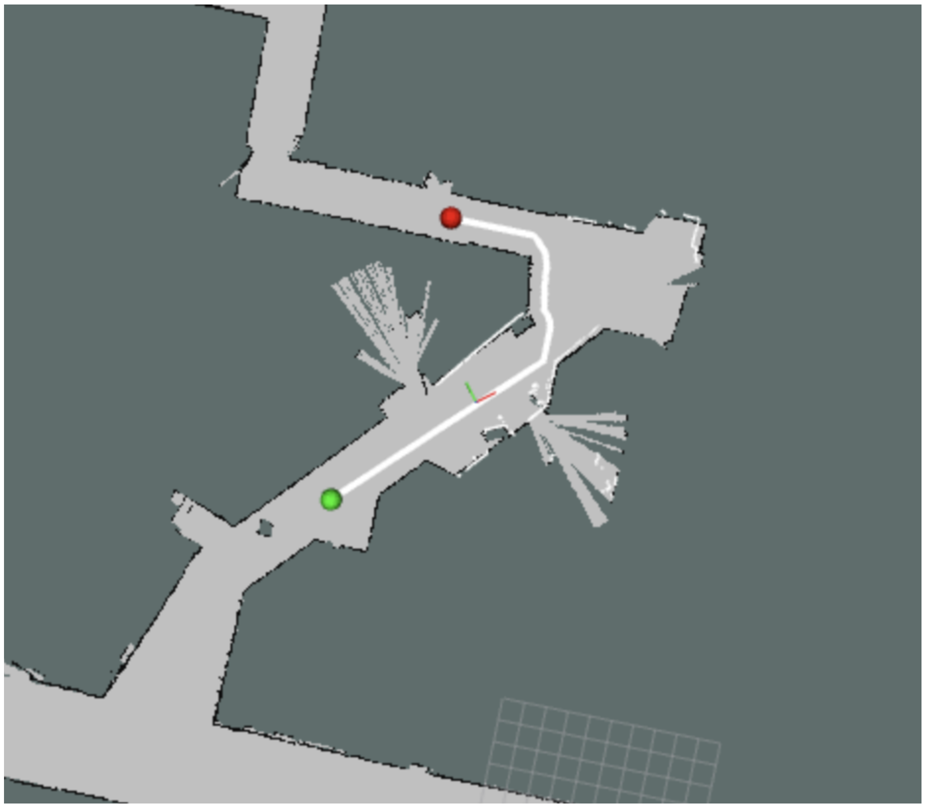
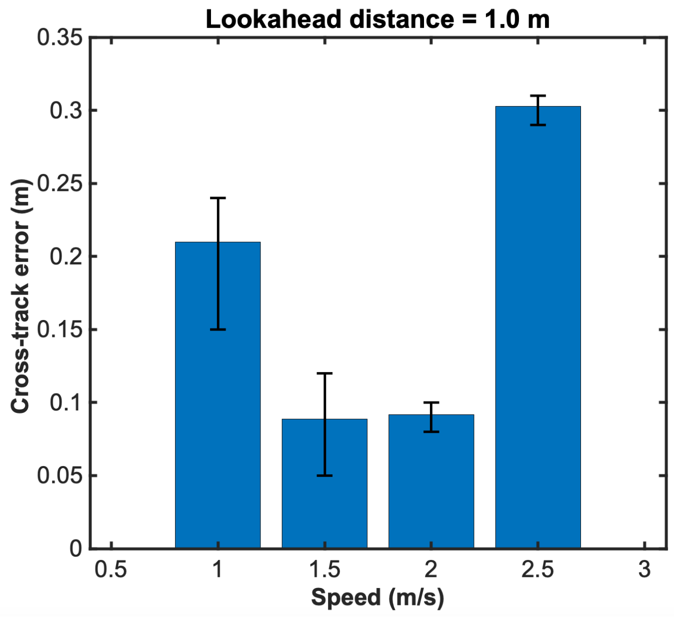


Figure 9: The path used for testing varied lookahead and speeds on the distance error



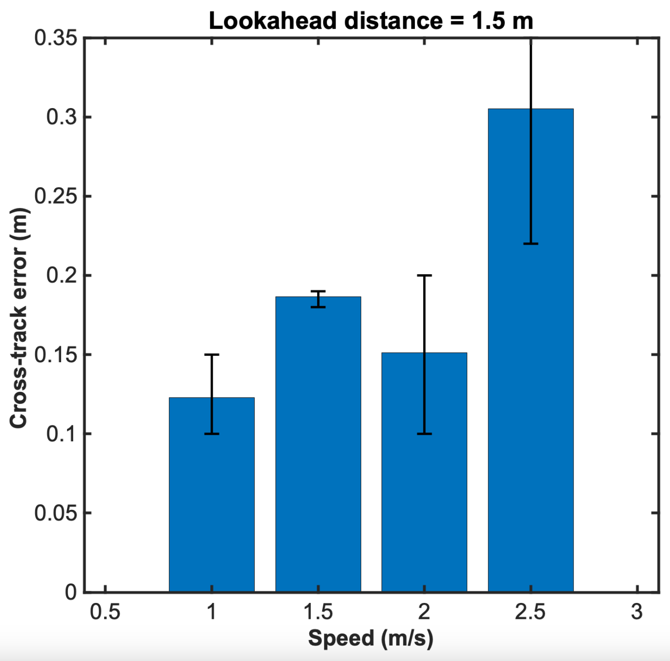


Figure 10: The measured cross-track error demonstrates comparable performance for both 1.0 m and 1.5 m, with the RMSE error represented by the bar plot staying within the 10-35 cm range, generally performing better for smaller speeds

Particle Optimization

Another optimization evaluated was the amount of particles used in our particle filter. The tradeoffs to using more or less particles is the amount of resolution and the costs of computational load. High computational load would be a problem due to error accumulation from latency issues and speed of determining an accurate pose estimate, which is important for more complex navigation. Table 1 summarizes these tradeoffs.

Table I: Number of Particles on Performance

Particles	Observation:
50	Low latency, lower accuracy
75	Medium latency, medium accuracy
100	High latency, high accuracy

We settled on 50 particles as our algorithm was robust enough to handle lower-resolution localization while maintaining accuracy.

3.3 Real-life Performance Assessment

After determining the robustness of the localization in simulation, we tested how well the robot performs in the Stata basement. We conducted both qualitative and quantitative tests to determine if the robot can accurately draw its trajectory around the Stata basement as shown in Figure 11.

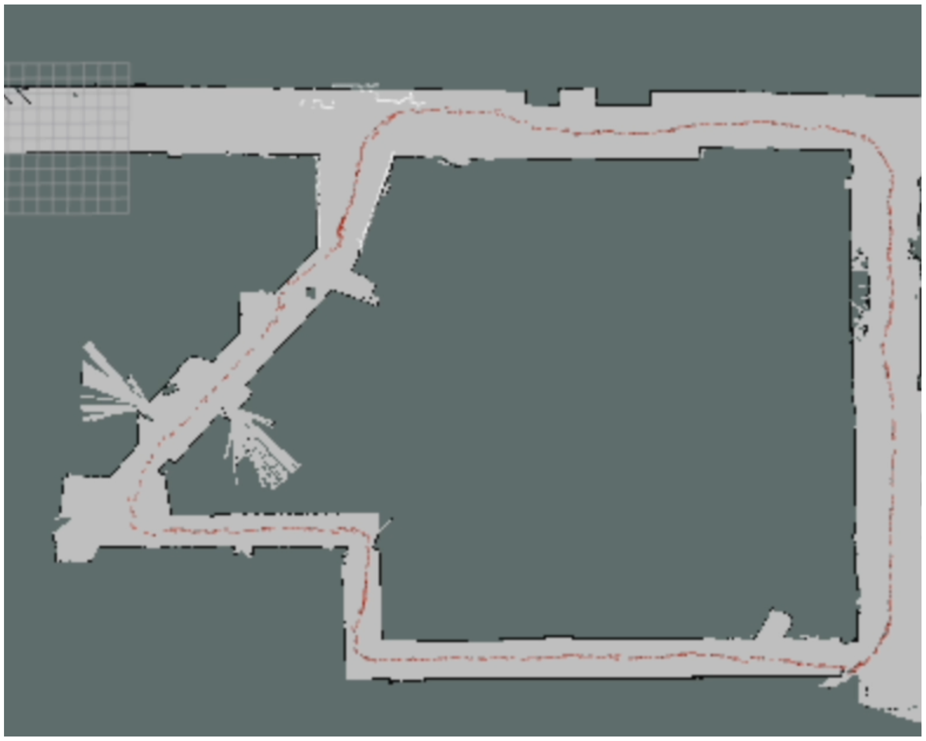


Figure 11: The robot can successfully trace its predicted path in Rviz that aligns with its actual movement through the Stata center basement.

The error in the predicted position was calculated across 3 runs for both the straight driving and turning as shown in Table II, showing a consistent and satisfactory performance with different magnitudes of the change in the odometry due to turning.

Table II: Mean errors across different path shapes

Path shape	Mean error (cm):
Straight	8.0
Turn	9.3

The video of the full basement loop can be found here or at the URL: `https://youtu.be/7KwP99XGfAk?si=eRJbWDbUIBhba4Pe`.

4 Conclusion

Author: Jialin Chen, Editor: Haris Imamovic

Our goal was to design an optimal and feasible path-finding algorithm to determine the shortest path between a start and end point in the real world, and follow it as accurately as possible. We use Jump Point Search and Pure Pursuit algorithms to plan and drive the robot in the real world. For the robot to know where it is in its environment, we used a particle filter based Monte Carlo Localization from our previous lab in addition to a given map of the Stata Basement. Pose estimation is achieved when the particle filter predicts and updates particles to converge to a location representing the highest likelihood of robot pose. We found through our tests that a lookahead distance between 1 and 1.5 meters provides ideal performance and generally performs better with 1.5 and 2 m/s speed.

Throughout the development process, many technical challenges occurred in path planning and localization. Initially, we faced inconsistency issues with not always finding the shortest path. We addressed this by refining our function for checking which cells are traversable. After fixing our path planner, we then used our previous implementation of Pure Pursuit and adapted it for the planned trajectory. For our Monte Carlo Localization integration, we faced challenges mainly with debugging a small error with how we flipped the sign of odometry data. Key insights learned from these issues is to validate each stage of integration to avoid critical errors.

Future work that could be done could include interpolating between our current lookahead distances and gathering more data to further improve the performance of the trajectory follower. We may also need to modify the safety controller so that it can stop the car in time at higher speeds, which is necessary for the Final Challenge.

5 Lessons Learned

Alvin Banh

This lab was difficult from trying to understand ways to improve the code and being stuck on problems for a long time. Many issues came up with debugging, especially for Pure Pursuit and integration with localization. As a team, we were stuck on trying to figure out why our code would not run in simulation and we spent hours thinking and looking through the code to check for any bugs. It turned out that the reason why the code was not working in simulation was because of a flipped sign in our odometry data. We were able to get to the root of problem by bouncing ideas off of one another and listening to suggestions, reinforcing the importance of good collaboration and teamwork. In terms of technical growth, I learned a lot more about how different path finding algorithms work and how they can be optimized with Jump Point Search. I was able to see ways old modules from other labs are directly useful when running tests for our path finding using our safety controller and localization.

Jialin Chen

This lab was probably the coolest to see working in real life since we can set a start and end goal and watch the car navigate through the best trajectory in real life. In order to get different modules of the lab to work together from path planning to path following and localization, we debugged as a team, which taught me that teamwork is extremely helpful and essential when trying to troubleshoot, since we have more ideas on where could go wrong. One of the bugs that took us a while to catch was forgetting to flip the odometry signs when switching between simulation and real life, which was so trivial we had overlooked it but it would make a huge impact on the localization. We ran into some hardware issues and it took some trial and error to fix the issues, including replacing the battery, but I'm glad that we were able to overcome it for our data collection. Group debugging was a major takeway from this lab, as well as communicating as many ideas as we have for data collection and what type of measurements we should take. It was useful to sketch out ideas on whiteboards when sharing with the team.

Nevena Stojkovic

This lab taught me a lot about both teamwork and robotics. On the teamwork side, I learned how important clear communication and dividing tasks are. At first, it was tough to get everyone on the same page, but regular check-ins and planning helped us work more smoothly. I also saw how helpful it is to have different perspectives when solving problems and how patience and flexibility are key when things don't go as planned. For robotics, I got a better sense of how path planning and control systems work together. Implementing Pure Pursuit helped me realize the challenge of balancing smooth steering and quick responses. I also learned a lot about how Jump Point Search (JPS) works in obstacle avoidance and why it's important to keep things efficient in real-time systems. Integrating everything showed me how crucial it is to make sure all the parts—planning, control, and localization—work together seamlessly.

Haris Imamovic

Coming from an algorithm-intensive background, exploring shortest path finding algorithms in a real-life setting, on a working, physical robot, was so satisfying to watch. I felt very comfortable working on the algorithm itself, and learned a lot about optimizations used to transform something performed on an ideal grid to something that works on a map representing actual space. It made me realize that I would like to continue doing work like this moving forward in my academic career, and potentially research similar algorithms. With each briefing, I feel like my presentation skills get better, and I am more comfortable speaking in technical jargon in front of a highly knowledgeable group. I also refreshed my knowledge of MATLAB and data analysis, which is a convenient skill to keep finessing.

References

[1] MIT RSS Teaching Staff, "Path Planning," GitHub, 2025. [Online]. Available: `https://github.com/mit-rss/path_planning`. [Accessed: Apr. 23, 2025].

[2] MIT RSS Teaching Staff, "Motion Planning," Massachusetts Institute of Technology, 2025. [Unpublished lecture notes].