Lab 6 Report: Path Planning

Team 4

Rohan Wagh
Oliver Rayner
Gabriel Jimenez
Kairo Morton

6.4200/16.405: Robotics Science and Systems

April 27, 2023

1 Introduction

In lab 6, we worked on solving path planning in a known environment. Path planning is an important part of autonomous navigation and serves as the backbone of most navigation systems. Solving path planning can allow an autonomous robot to travel from its current position to some goal position, and aid in positioning the robot for other tasks or objectives.

2 Technical Approach

As previously introduced, the problem of path planning is multi faceted and extremely useful for mobile robot navigation. Below is a system diagram of the team's approach to path planning:

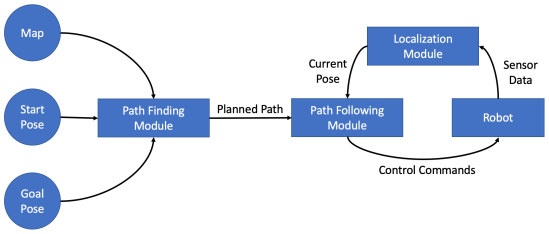


Figure 1: General system diagram

Our technical approach to solving this problem can be broken down into two parts: path finding and path following. The first being defined as follows: given the current pose of a mobile robot in a known map/environment and a goal pose in the same map find a valid and feasible high-level motion plan through the environment. The second aspect of the technical approach, path following, focuses on translating the high-level path to low-level robot controls in order to have the robot's motion replicate the planned path through the environment. To solve the first challenge of path finding, we rely heavily on well-known search algorithms over space of possible robot poses guided by the dynamics model of the mobile robot and the features of map in order to ensure the feasibility and validity of the planned path. Finally, to tackle the challenge of path following we implement a version of the pure pursuit algorithm. Both of these methods are discussed in greater algorithmic detail throughout the upcoming subsections.

2.1 Path Finding

As stated, the goal of path finding is to discover a feasible and valid path through the environment that originates at an staring pose $(x_{\text{init}}, y_{\text{init}}, \theta_{\text{init}})$ and ends at a goal pose (x^*, y^*, θ^*). In our solution we represent this path as a list of poses $[(x_1, y_1, \theta_1), ..., (x_n, y_n, \theta_n)]$ where the path between any consecutive pair of poses (x_i, y_i, θ_i) and $(x_{i+1}, y_{i+1}, \theta_{i+1})$ is governed by the bicycle motion model with parameters p_l, and, δ where p_l specifies the path length and δ specifies the steering angle. Given this definition of a path, we need to find the sequence of actions and resultant poses such that $(x_1, y_1, \theta_1) = (x_{\text{init}}, y_{\text{init}}, \theta_{\text{init}})$, $(x_n, y_n, \theta_n) = (x^*, y^*, \theta^*)$ and all poses along the path are not in collision with any obstacles specified by the known environment map. In our specific scenario we assume the map is given as a 2D binary image where a pixel of value 0 represents free space in the environment and 1 represents a prohibited obstacle. Finally, we assume that there is a known conversion between 2D map pixel coordinates and real world environment coordinates.

A simple algorithmic approach to path finding given this map representation would be to plan a path using search algorithm such as A*, BFS or DFS run directly on the free space in the map starting from the nearest 2D coordinated to $(x_{\text{init}}, y_{\text{init}}, \theta_{\text{init}})$ and finishing at the nearest 2D coordinated to (x^*, y^*, θ^*). However, this approach encounters two key issues. First, it incorporates no notion of robot heading or dynamics and as such while the path produced may be valid in that it does not collide with obstacles, it is unlikely to be feasible for all but straight paths. To solve this problem of feasibility while maintaining the desirable property of validity we re-frame the problem as a search over a discretized version of the continuous robot configuration space. Specifically, we implement a modified A star search starting from the initial pose and compute neighbors to a given pose (x, y, θ) using a discrete set of actions $A = \{(p_l, \delta) | \forall p_l \in P, \forall \delta \in D\}$ where D is a user defined set of usable steering angles and P is a user defined set of usable path lengths. Due to the use of actions in this way the edge weights needed for A* are simply set to the path length between neighboring poses, p_l,

and a simple euclidean distance heuristic to the goal position is used to guide the search overall. These actions are applied to the current pose, $\mathbf{x}_{\text{old}}$, to produce feasible neighboring poses $\mathbf{x}_{\text{new}}$ in the procedurally generated configuration space graph as follows:

$$\Delta x = \frac{L \sin\left(\frac{p_l \tan(\delta)}{L}\right)}{\tan(\delta)}$$

$$\Delta y = \frac{-L \cos\left(\frac{p_l \tan(\delta)}{L}\right) + L}{\tan(\delta)}$$

$$\Delta \theta = \frac{p_l \tan(\delta)}{L}$$

$$\mathbf{x}_{\text{new}} = \begin{bmatrix} \cos(-\theta_{\text{old}}) & -\sin(-\theta_{\text{old}}) & 0 \\ \sin(-\theta_{\text{old}}) & \cos(-\theta_{\text{old}}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} + \mathbf{x}_{\text{old}}$$

where L is a constant parameter specifying the distance between the front and rear axle of the robot. These neighboring poses are then filtered keeping only the valid poses which are not within an object when projected on to the 2D map. While these neighboring poses are generated using continuous functions, for the purposes of keeping track of the nodes and their respective costs during A* search they are discretized to the nearest coordinate on a grid with a user specified real world resolution. The angles are also discretized by remapping an angle $\theta \in [0, 2\pi]$ to $\theta' \in \{0, \frac{2\pi}{N}, \frac{4\pi}{N}, ..., \frac{2N\pi}{N}\}$ where N is the number of angle bins. This discretization in turn makes the search tractable and allows for another two parameters to control the behaviour of the algorithm as speed can be traded for accuracy by increasing or decreasing the search resolution in the angular or spatial domain.

Overall, this approach enables the planning of feasible and valid plans efficiently while still allowing for user tuning and control of the quality and accuracy of produced paths.

2.2 Path following with Pure Pursuit

The pure pursuit component leverages an input trajectory in order to calculate the necessary control inputs to move the racecar from its current pose to a target position. The pure pursuit algorithm followed by the component is as follows:

1. Determine racecar's current location
2. Evaluate closest path point
3. Find a new goal point or look-ahead point

4. Calculate steering required

Below is a sample of the geometry of Pure Pursuit:

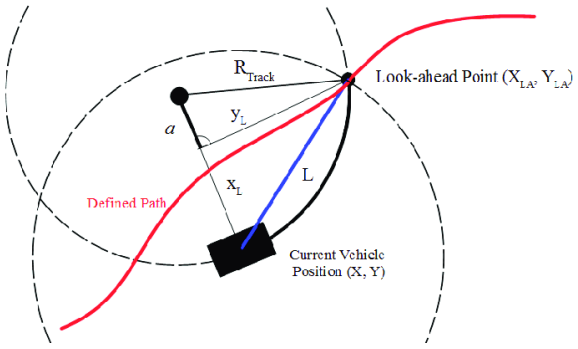


Figure 2: Geometry of pure pursuit

Determining racecar's current location. While executing in the physical platform, there is uncertainty when it comes to the ground position and orientation of the robot. As a result, we leverage the particle filter from previous implementations in order to achieve a best estimate of the car's pose.

Evaluating closest path point. The position estimate is then used alongside the input trajectory to evaluate which point in the path is closest to the robot. We iterate over all the segments that make up the trajectory, compute the distance from current position the a projected point onto the current segment, and then identify which segment index yielded the minimum projected distance.

Finding look-ahead point. The closest segment index serves as the starting index for finding the next target look-ahead point. We followed the recommended approach of checking segment by segment for intersections between the current segment and the look-ahead radius. Our look-ahead radius was defined by $turning\_radius + 0.1 + (0.2 \cdot speed)$, where turning radius is 1 meter and a manually tuned weight on the speed. We experimented with other factors, such as a rolling average of steering angle to represent path 'curviness', but did not observe good enough impact on performance to warrant persisted the experimental changes. Another element we tried to address through the look-ahead radius is potentially clipping through obstacles if a path doubles back on itself around the obstacle, since the search logic prioritizes intersecting points that are further along the path. We did not find that changing the look-ahead radius solved all relevant scenarios, instead choosing to limit the number of segments we iterate on past the starting segment index.

Calculate steering required. Once a target position is set, then the steering angle follows from the curvature of the arc containing the current estimated position and the target point, and accounting for current orientation. To facilitate this operation, we transform the target point from the world coordinate frame

into the car coordinate frame, with the relevant formulas as:

$$\alpha = \arctan(\frac{y^C_{target}}{x^C_{target}}) \ , \ steering\_angle = \arctan(\frac{2 \cdot wheelbase\_length \cdot \sin(alpha)}{lookahead\_radius})$$

The pure pursuit component maintains a loop to continuously compute the steering angle as long as there is a loaded trajectory at least one segment long. Additionally, in our control updates our target speed decreases linearly to 0 on the last meter to the final point in the trajectory.

2.2.1 Observations on Pure Pursuit

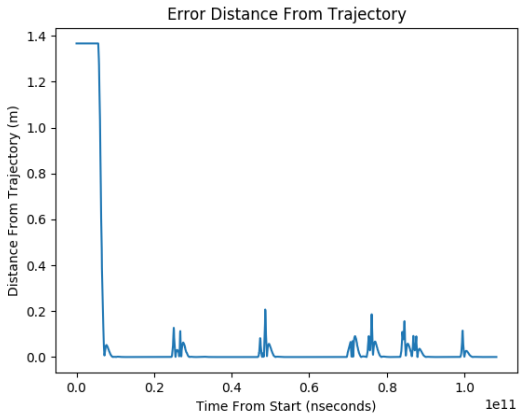


Figure 3: Evaluation of Pure Pursuit Following

Our pure pursuit implementation performed well on the simulation testing we did. On Figure 2, we can note that the delta from the closest point on the path quickly drops to near 0, while only reaching a peak of 20 centimeters on turns, which effectively meant most of the car was still on the path. Our mean error from the path after filtering out the path locking process is about 2 centimeters.

3 Observations and Evaluation

In order to evaluate the path planning algorithm, we ran the system in both simulation and real worlds environments. The simulation testing allowed us to evaluate the efficacy of the path planning algorithms as well as evaluate the pure pursuit path follow in a noiseless environment. In addition the simulation environment could function without the use of localization, instead using ground truth robot pose. This helped debug and fix problems with the path creation and follow algorithms. During testing however, the localization code was utilized

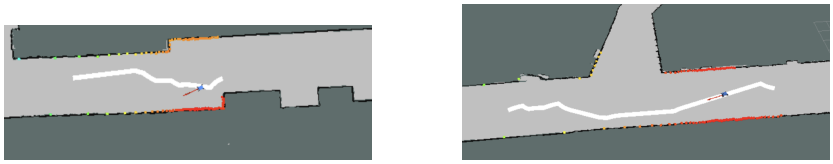
and both path planning and following were evaluated using the predicted pose from localization.

3.1 Evaluation in Simulation

Running the system in simulation allowed us to directly evaluate the planned paths. For the path planning to be deemed successful, it should be able to design a feasible path in a reasonable amount of time. We evaluate feasibility by both observing the path visually and testing if the path leads to collisions.

3.1.1 Observations on Plans

We pathed plans at three different lengths. The shortest paths had a goal location a couple of meters ahead of the robot, as can be imagined, these paths were fairly easy for the algorithm to compute. Below is an example of the smaller paths:



(a) Short path (b) Medium path

Figure 4: Comparing oscillations in short and medium path lengths

One observation we noted was the path's tendency to "drift" toward the goal location, even if that moves the robot towards one side of the hallway. As will be described later, this is likely a result of the Euclidean distance heuristic.

While the small paths were easy to compute, we noticed that they often ended up having a lot of small oscillations. This could potentially be a result of the A* algorithm taking a series of smaller length segment steps to reach the goal, as it has to position itself into the goal pose within a smaller distance. As can be seen in Figure 4b, the medium-length path starts out smooth with long segment sizes and starts to hold more direction changes as it got close to the goal, acting more like the path in Figure 5a.

3.1.2 Effects of Heuristics

One interesting observation made was the effect of the Euclidean distance heuristic on the path. As described in prior sections, this heuristic prioritized taking steps that lower the distance to the goal pose.

In Figure 5, the effect of the heuristic can be best seen in the middle section of the path, slightly ahead of the robot. On long paths, when the goal is orthogonal to the robot's direction, the robot will try and drift towards the wall and get

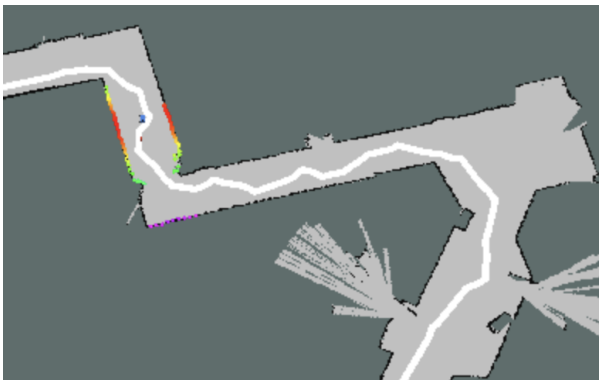


Figure 5: Oscillatory path caused by Heuristic

closer to the goal. This is a direct result of the path planning algorithm tending towards the goal, then turning away from the wall repeatedly. This behavior is also not seen in sections where the hallway is in the direction of the goal, as can be seen in the lower part of the path in Figure 5, and thus can be attributed to the heuristic's effect on the trajectories created by the path planning algorithm.

3.1.3 Path Feasibility

One key component of the path planning algorithm is the feasibility constraints. The A* algorithm only takes steps that are reasonable for the robot to also be able to complete. One scenario where this can be seen is when the robot has to turn around.

Figure 6: Turn in path

In Figure 6, an example of such a scenario can be seen. When the goal pose is placed behind the robot, it would not make sense for the robot to have to either drive the entire loop or for the path to ignore what is feasible for the robot to follow (for example simply ignoring starting direction and having a line from

robot to goal). As a result of the constraint in A*, the path planning algorithm has to find a way to turn the robot around by driving with the max steering angle and moving in a circle to rotate the robot.

3.1.4 Time to Plan

The last evaluation in the simulation was the time to plan longer paths. We noticed that as the path got more complex or longer, the time to form the path also scaled. We tested the time to path 3 different paths at multiple combinations of distances and the number of turns to identify how both parameters affect the time to create a path.

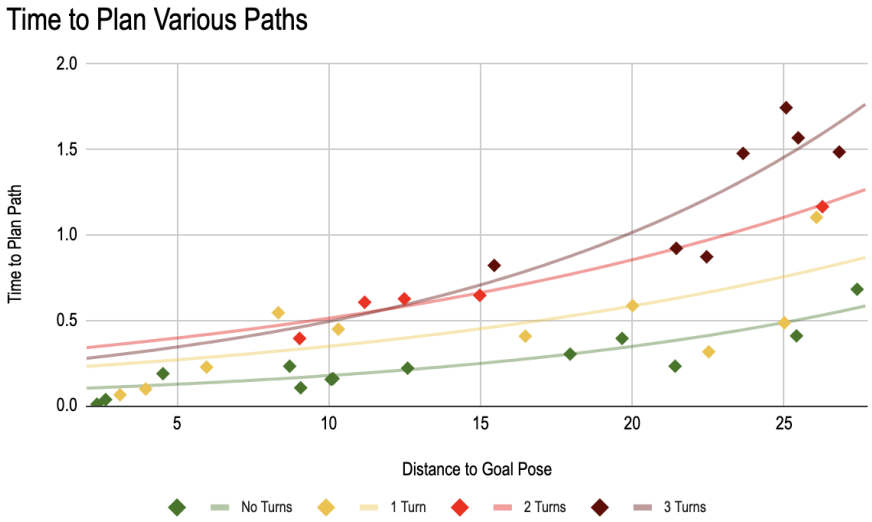


Figure 7: Time to plan path for various configurations

As can be seen in Figure 7, the time to plan the path increases with both the distance to the goal as well as the number of turns that the path needs to navigate through. Both of these factors increase the complexity of the path, which could be the reason why it may take longer to plan. Regardless of the complexity, almost all paths are planned within a reasonable amount of time, in under 2 seconds. This is within the bounds of success the team set for the lab and the as a result we deemed the path planning algorithm to be successful.

3.2 Difficulties in Hardware

XXX To be written

3.3 Evaluation on Robot in Real World Environment

XXX To be written

4 Conclusion

5 Reflections

5.1 Gabriel Jimenez

My main technical contribution was the pure pursuit implementation, and a challenge there was troubleshooting all the possible workflow steps that could be the root-cause for bugs. The hardest one to debug was the transformation step from world coordinate frames to car coordinate frames, since we could not just lift the transformation from the transformation tree and had to compute it ourselves. The values from the transformation tree served as a useful sanity check, but we still had multiple failed iterations on that math logic. Initially we did not even realize the transform step could be an issue, since we had ported the buggy code from a previous code section we thought was bug-free. Once again, collaborating through the troubleshooting process proved to most efficient when the root-cause bugs prove hard to find.

5.2 Rohan Wagh

I was helping with the integration and testing part of the lab as well as worked through some ideas for Pure Pursuit. This lab was a stronger test of debugging skills, especially in getting the localization code to feed into the system as a whole and identifying when a problem was a result of localization, path planning, or pure pursuit. We also had some interesting hardware problems that required working through alot of our system setup. In the end, learning how to better break down the problem and make steps towards the solution was vital to success in the lab.

5.3 Kairo Morton

During this lab I spent the majority of my time working on the path planning and search aspects of the system. I found this work in particular to be technically challenging and engaging due to the open-ended nature of the problem and the multitude of possible solutions. In the process of developing the path planning algorithm, the most valuable lesson I learned was the importance of creating sandbox simulations and unit tests for my algorithms outside of the ROS framework, as it allowed for quicker and more efficient in developing our solution. However, I did realize that I could have been more helpful and available when it came to integrating the code onto the robot after it was working in simulation. Nevertheless, overall it was a great learning experience and I

am excited to possibly improve on the code in the future for use in the final challenge.