# Lab 6: Path Planning and Pure Pursuit for Autonomous Navigation of the Stata Basement

6.4200 Robotics: Science and Systems

April 26th, 2023

Joseph Camacho, Christina Chen, Timothy Kostolansky, Nicole Shigiltchoff, Jessica Wu

## I. Introduction

### A. Challenge Definition ()

The challenge is to develop a path-planning algorithm that can help our racecar robot navigate a maze and reach a specified goal location. The robot has a limited sensing range, and it must avoid obstacles in the maze while minimizing the distance traveled. To accomplish this goal, we will be implementing algorithms for path planning and path following, which are essential for the city planning segment of the final challenge of this class, where our algorithms must be able to determine a trajectory and command directions for our car to follow, given only a starting position, goal position, and the map of the cardboard city it will be navigating. We may use any path planning algorithm of our choice to find the trajectory for the maze. The trajectory must not cut corners or have the car run into any object. It should be as close to an optimal, shortest path as possible. And ideally, we also want the path-finding calculation to be efficient and fast so we can quickly change goal positions during the challenge as needed. Once we find a trajectory, we then want our robot car to follow it smoothly while not cutting corners and running into obstacles during the final challenge with a pure pursuit algorithm. We want evidence that our algorithm works, which we will obtain from both simulation and physical trials. We would start by using the ROS racecar_docker simulation environment to develop and test our algorithms. Ultimately, we evaluate our performance in the final challenge based on several metrics, including the distance traveled, the time taken to reach the goal, and the number of collisions with obstacles. We aim to use these metrics to compare the performance of different algorithms and to optimize our algorithms for speed and accuracy.

### B. Challenge Motivation ()

Both path planning and path following are common problems in many robotics applications that generally would be useful for us to understand and are crucial for the final challenge we face in this class. Our primary motivation for lab 6 is to learn and implement a working version of both algorithms and most importantly, ensure that they are integrated into our robot within the tight time schedule. Due to tight deadlines and the magnitude of the tasks at hand, we split into subteams to work on independent elements to increase efficiency while always guaranteeing that at least one team member is updated on all tasks in progress. From this project, we learned how to prioritize tasks effectively to ensure that the most critical tasks are completed first and how to be flexible and adapt to changing circumstances.

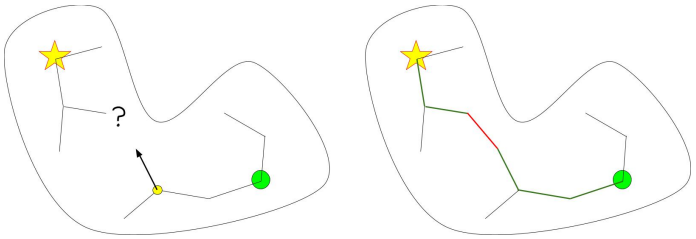


Fig. 1. (a) A tree is extended by sampling a random point not in an obstacle, finding the closest node on the tree, and adding a new node in the direction of the random point. (b) The two trees are bridged when two nodes become close enough and a path is found by going to the roots of the trees.

## II. Technical Approach

### A. Path Planning

For path planning, we used RRT Connect, a variant of rapidly exploring random trees (RRTs). RRT Connect has two such trees—one rooted at the start and the other rooted at the goal—and every iteration switches between extending the first tree and the second tree. The trees are extended by sampling a random point in open space, finding the closest node on the relative tree, and adding a new node in the direction of the sampled point $\delta$ away from the closest node. When the branches of the two trees become close enough (in particular, two nodes are at most $\delta$ away from each other), the trees are bridged and a path is found by iterating upward to the root from either side of the bridge. (See Fig. 1)

RRT-based methods have several properties that make them useful for pathing on a racecar. First, unlike grid-based search algorithms, they work natively over continuous space. Second, they rapidly explore the search space (as the name would indicate) by being biased to extend the tree towards the largest cells of the Voronoi diagram induced by the tree. Finally, they are very space efficient, especially in high-dimensional search spaces.

RRT Connect adds one more optimization to a basic RRT algorithm: It uses two trees. It is much easier for the branches

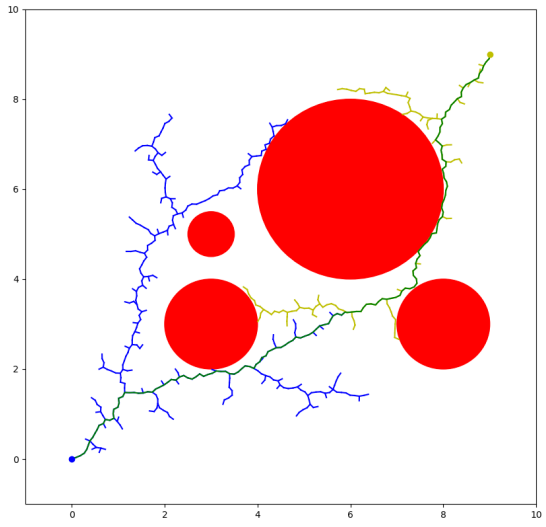

Fig. 2. An example of using RRT Connect to path around red circles from a start (blue) to a goal (yellow).

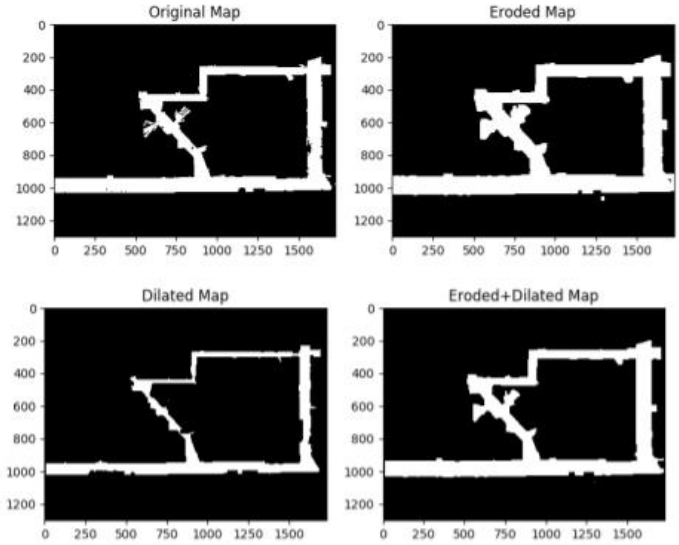


Fig. 3. Images of eroding and dilating the map. (a) the original map (b) erosion alone (c) dilation alone (d) the final map with erosion and dilation.

of two growing trees to connect than for a single tree to extend to the goal (as RRT alone does), which makes RRT Connect significantly faster than RRT.

Unfortunately, RRT Connect (and other RRT algorithms) do have two drawbacks. First, the paths they output are not smooth. Second, it tends to cut corners when $\delta$ is large. The first drawback did not pose an issue for us because we used a path-planning algorithm (pure pursuit) that naturally smooths out paths. The second drawback we fixed by dilating and eroding the obstacles to give a larger area the path cannot cut through, which we explain further below.

### *B. Obstacle Dilation and Erosion*

A simple way to help RRT Connect avoid cutting through corners is to dilate the obstacles so the path stays farther away from the walls. We used scipy.ndimage to first erode the obstacles (to get rid of small noise) and then dilate them by a larger radius. For our specific map, we found that an erosion of 18 pixels and a dilation of 22 pixels (corresponding to 0.91 meters and 1.11 meters, respectively, at our resolution) worked fairly well.

As we'll explain later, pure pursuit also tends to cut off corners, and this same erosion and dilation technique helps avoid that as well.

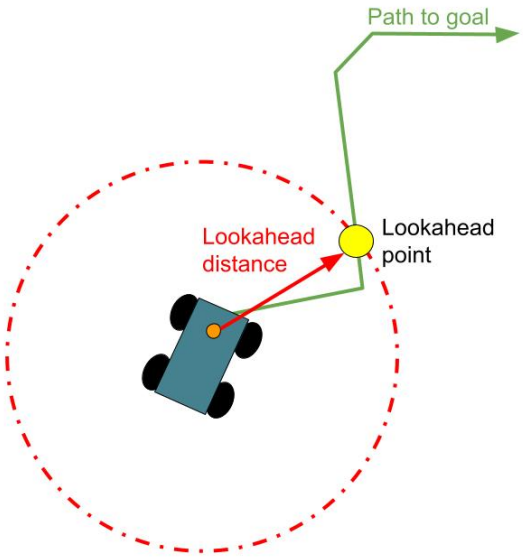


Fig. 4. A visual representation of the Pure Pursuit algorithm. A circle with a radius of the lookahead distance is created around the car, and its intersection with the planned path that is closest to the goal is determined to be the lookahead point. The robot will now drive towards the lookahead point.

### *C. Pure Pursuit*

To follow the path defined by the RRT Connect algorithm, we implemented the Pure Pursuit algorithm (Fig. 4). This algorithm searches for a target point at the intersection of a circle centered at the location of the car and the path and uses Ackermann Steering to guide the car toward that point. A new circle and intersection point are calculated each time the localization algorithm determines a new location for the robot, at a frequency of 50 Hz.

To determine the location of the car, we used our implementation of Monte Carlo Localization from lab 5. This algorithm used a combination of exteroceptive LiDAR and proprioceptive odometry data to determine the most likely location of the car and worked effectively to find the car's pose in the Stata basement.

The circle around the car was parameterized with two values: the origin, which was set to equal the position of the car, and the radius, which is the lookahead distance. The lookahead distance is how far away from the car the lookahead point should be, and is a value that can be optimized manually depending on the path that the robot is expected to follow. For planned paths without sharp turns where the robot can

safely deviate from the path, the lookahead distance can be set to be large, such as 6x the length of the robot. This will allow for the smoothest robot trajectory. However, in situations where it would be important for the robot to adhere to the planned path as closely as possible, such as in a narrow space with sharp corners, the lookahead value should be set low, such as 1x the length of the robot. Unfortunately, this also increases the likelihood of the robot oscillating around the planned path due to constant attempts to overcorrect after a sharp turn. In the pure pursuit simulation for this lab, we found that a lookahead distance of 3 m, about 6x the length of the car, produced the smoothest trajectory while minimizing obstacle collisions. Despite the robot cutting the corners of the planned path, obstacle dilation and erosion prevented any part of the path from being close enough to a static obstacle that corner-cutting did not cause collisions in the majority of cases.

In many situations, there are multiple points of intersection of the planned path and the circle around the robot. When multiple possible candidates for the lookahead point are found, the point that is closest to the goal at the end of the planned path is chosen as the lookahead point. This is done to prevent the robot from ever driving away from the goal at the end of the planned path.

After determining the location of the lookahead point, Ackermann Steering was used to drive the car towards it (Fig. 5). The steering angle of the robot was determined to be such that the robot would move towards the lookahead point along the arc of a circle of radius $\frac{L_1}{2\sin(\eta)}$ of angle $2\eta$, with $\eta$ defined as the angle between the vector of the robot's initial velocity and the lookahead distance vector from the robot to the lookahead point and $L_1$ defined as the lookahead distance. This angle $\delta$ was calculated as follows:

$$\tan^{-1}\left(\frac{2L\sin(\eta)}{L_1}\right)$$

$L$ is defined as the wheelbase length of the robot, or the distance (in m) between the front and back wheels of the robot.

## III. Experimental Evaluation

We were not able to take quantitative data within the time constraints of this lab, but we are actively working to measure and test our path planning components, as this is paramount in usage for the Final Challenge. Here we will discuss some challenges that our team faced and the successes that we had. Within the path planning module, RRT Connect is able to efficiently find a valid path through the map within X seconds. This was sufficient for the purposes of the lab, as the driving itself took Y seconds, which is much larger than the negligible X seconds that were spent on planning the path. One challenge we overcame within path planning was the cutting of corners, which we resolved using map dilation and erosion techniques. Our paths were qualitatively safe for the car to drive, i.e., not cutting corners, after we applied the dilation and erosion to the map before planning the path. Within the pure pursuit module, our algorithm successfully uses a lookahead distance to plan car driving directions.

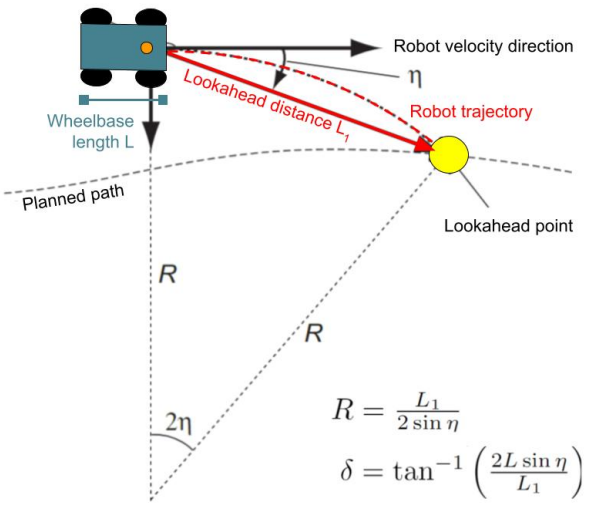


Fig. 5. A visual explanation of the geometry of Ackermann Steering. The steering angle $\delta$ is calculated using the wheelbase length $L$, the arc angle $2\eta$, and the lookahead distance $L_1$.

Qualitatively, the paths that our car takes in simulation and in reality are both safe and direct. The car is able to take path directions and drive from a starting position to a goal position within the lab's time constraints.

## IV. Conclusion ()

In lab 5, we successfully implemented path planning on the racecar. This included developing a path planning algorithm using RRT Connect, programming odometry controls for the car to follow paths using pure pursuit, and combining the path planning and pure pursuit to allow the robot to find and follow a path in simulated and real environments. We did this within the Stata basement, which includes many obstacles, turns, and irregularities, not to mention passersby and objects that are not part of the virtual map programmed onto the car. Through the course of the lab, we faced many challenges, such as corner cutting in path planning, pure pursuit not transferring as expected when moving from simulation to reality, and multiple members of the group getting sick over the course of the work. Despite these and other challenges, we learned a lot and successfully created a path-planning car, of which we are proud. We are excited to transfer and apply this knowledge in future work, especially in the upcoming 6.4200 Final Challenge.

## V. Lessons Learned

*1) Joseph:* I learned that it can actually be very easy to get some algorithms (in particular RRT Connect) to work the first time, and that it is very useful to first get inefficient code working correctly before trying to optimize for speed.

*2) Christina:* I gained more technical insights in this lab by tackling the code head on. I learned to take ownership of parts of the code, even if I'll need more time and support than others. I'm trusting myself more to seek answers, understand concepts, and execute sufficiently. For example,

I had to scout and understand the entire code base to insert map dilation logic and integrate the simulation code for the real environment. It's an unfamiliar feeling to trust myself to be the only one who understands something, so be quick and adaptable if there's no one to help, but I'm learning to accept it and trust that I'll still be able to find the answers and convey the crucial details to the team. I'm also learning about clear expectations to keep the team organized and motivated; for example, I hope the constant "Accomplishments and Todos" accomplishes that.

*3) Timothy:* I learned about a variety of path planning algorithms and their respective advantages and drawbacks. I also reviewed how to implement pure pursuit and helped Nicole debug her code. It was a valuable experience to be able to help others work on their respective parts this lab, as I was sick at the start of the lab and was not able to take a lead on one specific part. As a result, I learned about everyone's work and was able to help in a variety of places. I also re-learned the value of time management and working as a team, as we worked very well in crunch time this lab.

*4) Nicole:* This was the first group lab where I wrote the majority of a coding component from scratch (the pure pursuit algorithm), and while others contributed to debugging, I'm glad I got experience putting together the structure of an algorithm independently. I also learned a lot about pure pursuit and its different pros and cons through this process. Additionally, I got practice communicating urgency to my team in a way that effectively sped up our work process despite the obstacles we encountered while finishing the lab.

*5) Jessica:* I learned about path planning algorithms such as RRT and RRT Connect and how to tune the parameters of the RRT algorithm and alter a map for our specific purpose. I am now familiar with the geometric principles behind the pure pursuit algorithm. We also all went through illness and a time crunch for this lab which taught me how to better manage a workload under pressure and ask for extensions when I need them.