Lab 6 Report: Path Planning

Team 23

Alazar Lemma
Benjamin Soria
Caden Moore
Henry Heiberger
Vittal Thirumalai

Report edited by Henry Heiberger and Alazar Lemma

Robotics: Science and Systems

April 27, 2023

1 Introduction

Authored by Caden Moore, Edited by Henry Heiberger

In this lab, we were tasked with the goals of developing a path-planning algorithm for our autonomous racecar that could generate an efficient and navigable path to a target goal along with a trajectory-following algorithm that could allow the robot to efficiently follow this planned trajectory. These two algorithms are essential for our autonomous racecar because they allow it to reach the required objectives during the final challenge.

To achieve these goals, our robot first needed a way to plan its motion. To do this, we made use of a search-based path planning algorithm that employed an A* shortest path graph search implementation along with an added goal distance heuristic. This allowed us to provide the algorithm with a target goal to generate an optimal path from the robot's initial position to that goal. To compare against our search-based implementation, we also implemented a sample-based path planner that made use of random map sampling in order to generate an efficient goal path.

Once we were able to generate planned trajectories using our path planning algorithm, we needed a method to follow this trajectory as quickly and closely as possible. We did this by implementing a pure pursuit trajectory following an algorithm similar to what we employed in the visual servoing lab. This pure pursuit algorithm worked by successively finding points on a generated trajectory a designated lookahead distance away from the robot and adjusting the wheel's steering angle accordingly to pursue them.

This lab was very exciting to complete because it meant we really had a fully autonomous racecar that could task plan, drive, and navigate the Stata basement all on its own. Doing this required a lot of the progress we made in previous labs, such as localization, pure pursuit, and even things we learned from the first lab and combined them into one large system that got everything working together. All of our effort put into previous labs led up to this lab and achieving a fully autonomous racecar ready for the final challenge.

2 Technical Approach

2.1 Implementing a Path Planning Algorithm

Authored by Henry Heiberger

Path planning or the process of generating a trajectory that maps an efficient and drivable route from the robot's location to a target goal was the first main goal of this lab. This section highlights and rationalizes our design decisions when solving this planning problem as well as discusses our implementations in depth.

2.1.1 Choosing a Path Planning Algorithm

Authored by Henry Heiberger

Before beginning to implement our path planning algorithm, we found it useful to first consider which of the two major categories of such algorithms we wanted to focus on, search-based or sample-based. At the highest level, search-based path planning algorithms discretize their environment's map into a high-resolution grid such that a shortest path graph searching algorithm can then be directly applied to create an optimal goal path. In contrast, sample-based algorithms rely on a random distribution of points across the environment's map that can then be connected in order to generate an efficient goal path. We compared these two types of algorithms using the following metrics:

Regarding resulting path optimality, because search-based path planning algorithms rely on deterministic shortest-path graph searching, they come with guarantees that they will always return the optimal path for the map they are searching on. In contrast, while sample-based planning algorithms are probabilistically complete, meaning as the number of random samples increases, the probability of the algorithm returning a solution approaches one, in most cases, such algorithms can only be expected to return a relatively efficient path to the target goal, not a guaranteed optimal one.

Regarding algorithmic complexity, in order to achieve their optimality guarantees, search-based path planning algorithms tend to be computationally intensive, running in exponential time with the increasing degrees of freedom of the robot. These graph search algorithms also tend to require large data structures, making such methods rather memory intensive. Conversely, sample-based methods, while much harder to characterize asymptotically, tend to terminate more quickly in polynomial and even linear time, thus requiring less time and space than their search-based counterpart. This makes them ideal in cases with limited resources.

Regarding driving feasibility, both algorithms face unique drawbacks. Because they attempt to find the most optimal path to the target location and straight line paths are shorter than curves, search-based path planning algorithms can create paths with sharp edges that don't meet the turning constraints of real-world vehicles. Furthermore, these paths also tend to hug walls and cut corners, further making them potentially difficult to navigate. In contrast, due to the nature of random sampling, sample-based algorithms tend to produce paths that are more irregular, making them potentially difficult to navigate and filling them with extraneous turning. It is worth noting though that through careful map preprocessing, path post-processing, and augmentations that consider driving feasibility, the effects of these drawbacks for both path planning methods can be mitigated.

Beyond these factors, it is also worth noting that standard sample-based path planning algorithms tend to be even more efficient than search-based ones in multi-query scenarios because the sample-based algorithms compute representations that do not need to be regenerated for additional queries.

To aid readers, a figure summarizing the most critical differences between search-based and sample-based path planning algorithms can be seen in **Figure 1**.

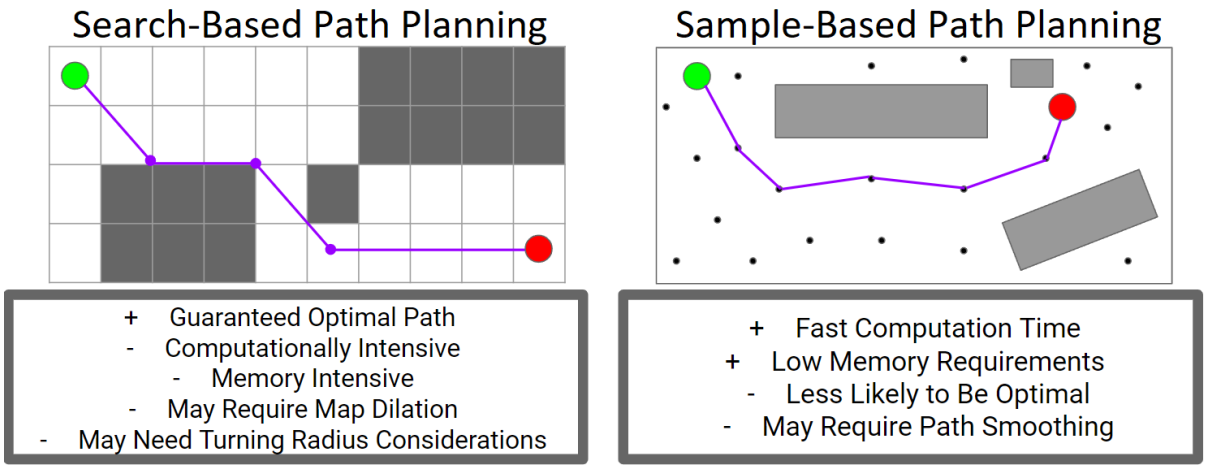

Figure 1: The figure displays a small visual highlighting some of the key differences between search-based and sample-based path planning algorithms, specifically with regard to the strong optimality guarantees but high computation cost of search-based algorithms versus the weaker optimality guarantees but low computation cost of sample-based ones.

In the end, though we knew that both algorithmic approaches would allow us to successfully achieve our goals for the lab, we knew that, in most cases, our path planning algorithm would be given the time to run to completion prior to running on the course. Because of this, the majority of our testing and evaluation during this lab was done using a search-based path planning algorithm due to its enhanced optimality guarantees and likely reduced need for path post-processing. However, to fully evaluate the performance of both methods, we did still implement one of each path planning algorithm and quantitatively compared their performance. This numerical comparison can be found in **Section 3.3**.

2.1.2 Implementing Search-Based Path Planning

Authored by Henry Heiberger

After completing our analysis, we first developed a search-based implementation of a path planning algorithm. As stated above, search-based path planners discretize their environment's map into a high-resolution grid. With this, they are able to apply a shortest path graph searching algorithm in order to search the map and find the optimal path to the target. We implemented this type of path planner in the following way.

As mentioned above, one major concern with implementing a search-based path planning algorithm is that, in order to find the most optimal path, the algorithm can generate paths that closely hug walls and cut corners, making the path more difficult for the robot to successfully drive. To deal with this risk, we performed an erosion operation on our map image which enlarged all dark areas. This grew the walls and obstacles within our map beyond their actual size, providing additional leeway in the case that generated paths followed closely to the wall. The result of performing this preprocessing on our map of the Stata basement can be seen in **Figure 2**.

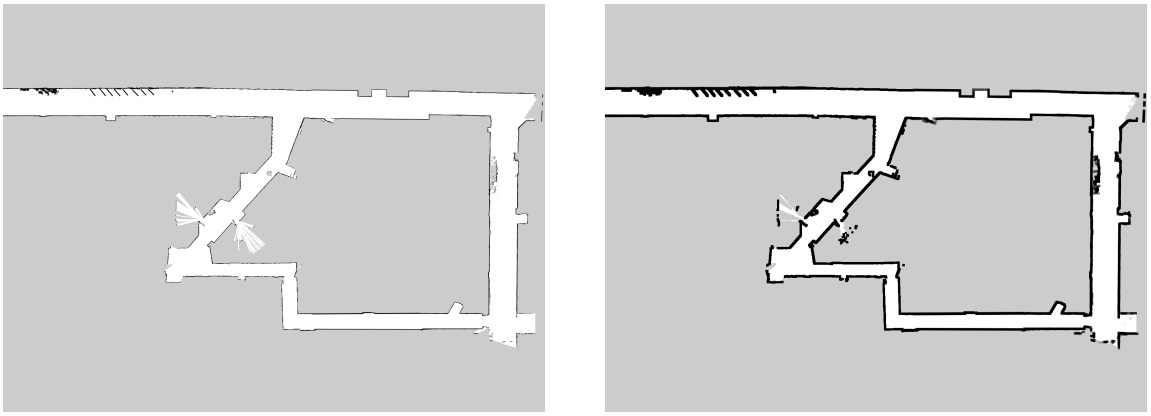

Figure 2: The left image shows the original Stata basement map used for path planning. The right image shows the same map after erosion has been applied, enlarging dark regions on the map and making potential obstacles take up more space.

After preparing our map, the only other major design decision we had to make when implementing our search-based path planner was which graph-searching algorithm we wanted to use. After considering several options, we ended up deciding to use an implementation of **A* search**, a widely known graph searching algorithm which, as it explores, prioritizes nodes that have the lowest total distance to reach from the robot's starting location along with an added heuristic of the Euclidean distance of that node to the target goal. This algorithm was chosen because of its widespread use for similar use cases in robotics literature, ease of implementation, and added heuristic that reduces the time needed for a path to be found without sacrificing path optimality. By reconstructing the path found by the algorithm through the use of parent pointers indicating where each node in the path came from, we were able to publish efficient trajectories from the robot to the target goal in both simulation and the physical Stata basement environment. Examples of these trajectories can be seen in **Figure 3**.

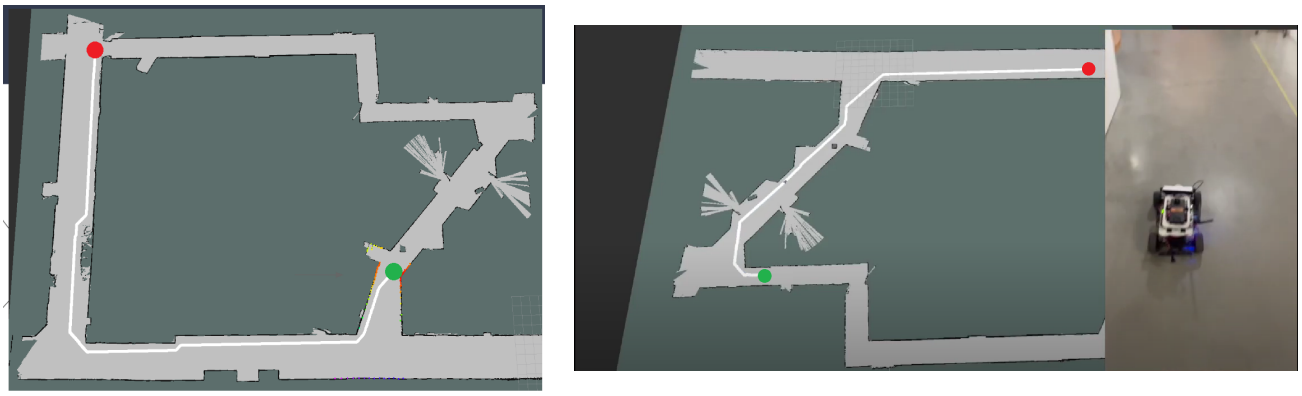

Figure 3: The left image shows the result of applying our A* search path planning algorithm in simulation to find the best route from the green start marker (the robot's location) to the red target marker. The right image shows the same thing but applied on a live robot running Monte Carlo localization (MCL).

It is worth noting that our A* search algorithm does not consider driving feasibility when trying to generate an optimal path, leading to the possibility of paths being generated that require turns that exceed the turning radius of our robot. However, during extensive testing in both simulation and real life, though we did see some examples of tighter edges such as the ones displayed in **Figure 4**, we found that, due to the layout of the eroded Stata basement map in which we tested our robot, even the sharpest turns produced by our A* search path planning algorithm were still fairly navigable. Because of this, we concluded that the impact of adding a feature such as Dubins curves, the shortest curve which connects two points with a constrained curvature, to our planned paths in order to consider a limited turning radius would likely not be very significant. However, we may consider this addition in the future if the problem becomes more apparent on different racetracks.

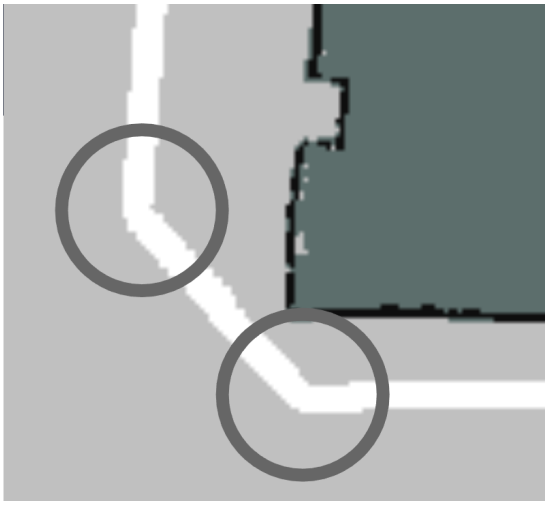

Figure 4: The figure highlights some of the sharpest turns that our A* planning algorithm produced that would be rounded off if Dubins curves were applied. However, it is worth noting that due to the layout of the Stata basement map, even these sharpest turns were fairly navigable by our robot, making the impact of adding Dubins curves to our planned paths less substantial.

Following the above procedure, we achieved a performance and effective search-based path planner that allowed our robot to easily generate navigable paths to any target that was fed to it.

2.1.3 Implementing a Sample-Based Path Planning

Authored by Vittal Thirumalai, Edited by Henry Heiberger

To compare with our search-based implementation, we also developed a sample-based implementation of a path-planning algorithm. In contrast to search-based ones, sample-based algorithms rely on connecting a random distribution of points across the environment's map in order to generate a path to a target goal. A couple popular sample-based algorithms include Rapidly-exploring Random Trees (RRT/RRT*) and Probabilistic Roadmap (PRM). We decided to implement RRT/RRT* instead of PRM because RRT/RRT* is a sample-based algorithm that relies more purely on sampling random points and finding the paths between them, while PRM uses graph search algorithms as a subroutine, making it closer to search-based methods in its approach of finding the optimal path. We believed that seeing the performance of RRT/RRT* against our search-based path planner would give us a better sense of the trade-offs between search-based and sample-based algorithms. Between RRT and RRT*, we chose RRT* because it is a more advanced version of RRT that rewires the tree to make a more optimal path.

Similar to the search-based path planning method, we used the same preprocessing of the map using erosion so that paths are not too close to a wall. Then in RRT*, the algorithm incrementally adds nodes through random sampling and local optimizations. In each iteration, a new point is randomly sampled and it is connected to the closest point in the existing tree. With the RRT* optimizations, the tree is gradually rewired by seeking and keeping only the edges contributing to minimal cost paths. This rewiring leads to faster convergence to an optimal path, with a guarantee of convergence to the optimal path as the number of sample nodes goes to infinity. Finally, analogous to in search-based planning, the path is reconstructed using parent pointers, and the resulting trajectories are published.

The number of sample nodes in RRT* is a key parameter that affects the runtime and accuracy of the generated path. **Figure 5** shows a visualization of a generated path for 1000 and 2000 nodes. For 1000 nodes, the time to generate the path was 0.67 seconds and the runtime of pure pursuit was 60.56 seconds. And for 2000 nodes, the path generation took 0.83 seconds while the runtime for pure pursuit was 57.77 seconds. So with more nodes, the time for path generation increases while the path becomes closer to optimal, as expected. It is interesting to note that in this specific run, with more nodes it seems to have a rougher path, but the total time was less in any case.

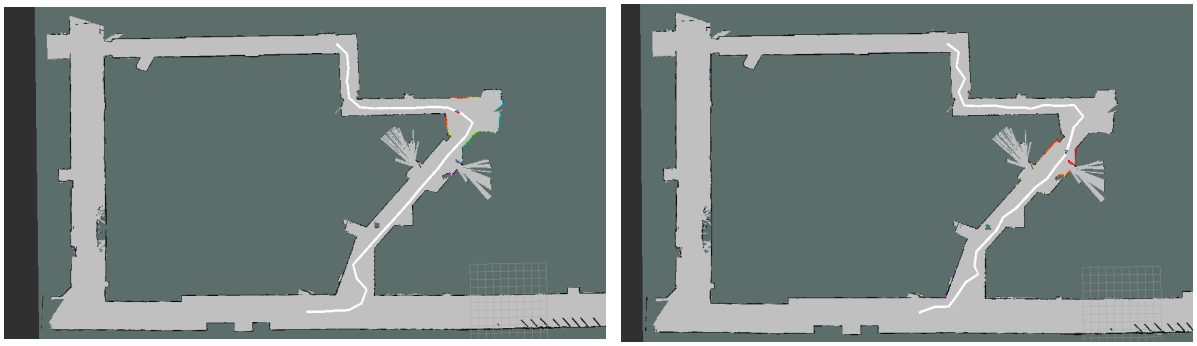

Figure 5: The images show the result of applying the RRT* sample based algorithm in simulation to find a good route from the start (top middle) to end (bottom middle). The left image has the number of nodes in RRT* set to 1000, while the right image has this set to 2000.

2.2 Implementing a Pure-Pursuit Algorithm

Authored by Benjamin Soria

2.2.1 Algorithm Overview

The role of pure pursuit in our implementation is to direct the car through a series of checkpoints provided by our path planner. The PurePursuit class accomplishes this by subscribing to the */trajectory/current* topic, which our path planning implementation publishes a *PoseArray* object to. Once the trajectory is received, the Posearray is unpacked and line segments are created between consecutive points. The nearest line segment is found by calculating the closest point on each line segment to the robot. The nearest point among the set of points identifies the nearest line segment. A goal point is identified by searching for a point on the nearest segment which intersects a circle of radius *lookahead* around the car. This point may not exist when the car is at the end of the line segment or if the lookahead distance is too large. In this case, the line segments up the trajectory towards the stop goal are searched until an intersecting point can be found. Once a goal point has been identified, we implement a pure pursuit strategy to publish a steering command.

2.2.2 Identifying the Nearest Line Segment

The nearest point to the robot on all line segments is found by projecting the robot's x and y position onto each line segment. For one line, we consider three points P_0, P_1, and P_2, each with coordinates (x, y) in the */map* frame. The nearest point lying on the segment connecting P_1 and P_2, segment B in **Figure 6**, intersects a perpendicular line, segment C, stemming from P_0 to the line segment. We can use the property that the dot product of B and C must be zero to find the intersecting point by calculating the scalar u which defines the distance along the segment at which the intersecting point is located.

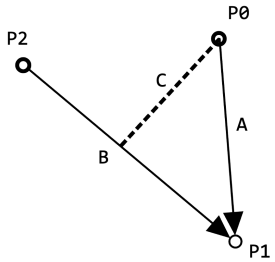

Figure 6: Geometric representation of the projection of a point P_0 onto a line segment connecting points P_1 and P_2.

$$u = \frac{\vec{A} \cdot \vec{B}}{\|\vec{B}\|^2} \tag{1}$$

Our implementation uses numpy arrays to compute the nearest segment at each stage that a new odometry estimate is received; Figure 7 provides a visual example of the process for one stage.

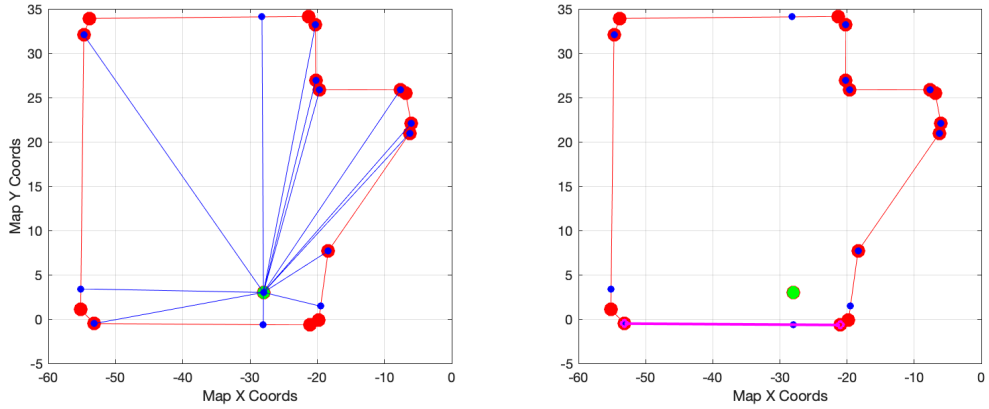

Figure 7: The plot on the right displays the nearest point on each line segment to the robot. The nearest point among this set of points corresponds to the nearest line segment to the robot. This nearest line segment is highlighted in the plot on the right.

2.2.3 Identifying a Lookahead Point

Once the nearest line segment is identified, we search for a goal point to perform pure pursuit on. A goal point must satisfy three conditions:

- The goal point must lie on a segment connecting two points provided by the trajectory's PoseArray
- The goal point must lie at the intersection of a line segment and a circle of radius *Lookahead* centered at the estimated car position
- The goal point must orient the car towards the end of the trajectory.

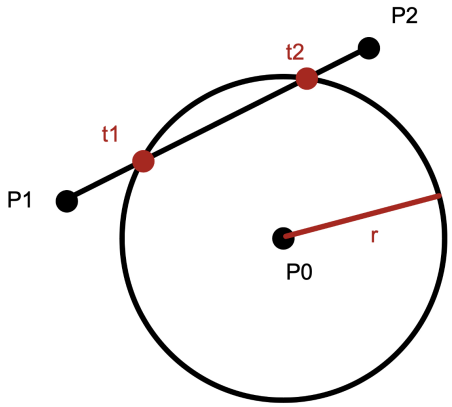

Figure 8: Geometrical representation of searching for intersecting points t_1 and t_2. P_1 and P_2 represent trajectory points. P_0 represents the estimated odometry with r as a lookahead distance. Only point t_2 is a candidate goal point because it directs the robot from P_1 to P_2.

We employ the variable t to represent any point on the line segment connecting points P_2 and P_1. As shown in **Figure 8**, we are in search of points t_1 and t_2 intersecting the segment and the circle around our robot. Let $v = P_2 - P_1$; through geometric relationships we formulate a quadratic formula with solutions t_1 and t_2.

$$t_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad t_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{2a}$$

$$a = v \cdot v \tag{2b}$$

$$b = 2(v \cdot (P_1 - P_0)) \tag{2c}$$

$$c = P_1 \cdot P_1 + P_0 \cdot P_0 - 2P_1 \cdot P_0 - r^2 \tag{2d}$$

If the discriminant of the quadratic function is negative, the circle misses the line segment entirely. This indicates that the robot has veered off the trajectory and is not able to track the succession of line segments. Otherwise, t_1 and t_2 are real. Three scenarios result for either t_1 or t_2:

- if $0 \leq t \geq 1$, then t intersects the segment.
- if $0 > t$ or $1 < t$, then t does not intersect the segment, but would intersect the segment if the segment had an infinite length.
- if $t = 0$, then there exists one intersection and the circle is tangent to the line segment

The value of t tells us the proportion along the segment which the intersecting point lies across. Because we are interested in moving along the segment from P_1 to P_2, we consider t_2 as a candidate goal point. If $0 \leq t_2 \geq 1$, then the three conditions mentioned above are satisfied and we begin pure pursuit on this point. If the discriminant is negative or either $0 > t_2$ or $1 < t_2$, then we search up the trajectory by considering the segments following the nearest segment for intersecting points satisfying the three criteria given above. If no intersecting point is found, the robot continues with its latest trajectory.

In one of our earlier implementations, the robot did not search up the trajectory when an intersecting point was not found. Instead, it issued a drive command with the last steering angle. This approach worked well enough, the robot was still able to track a trajectory, but it was limited to a smaller range of lookahead distance. The lookahead distance had to remain small enough to track short line segments, and this resulted in large oscillations, especially around corners. Switching to searching up the trajectory when an intersecting point was not found on the nearest segment allowed us to increase the lookahead distance, resulting in less oscillations without sacrificing tracking performance. A comparison of these two methods is shown in **Figure 9**.

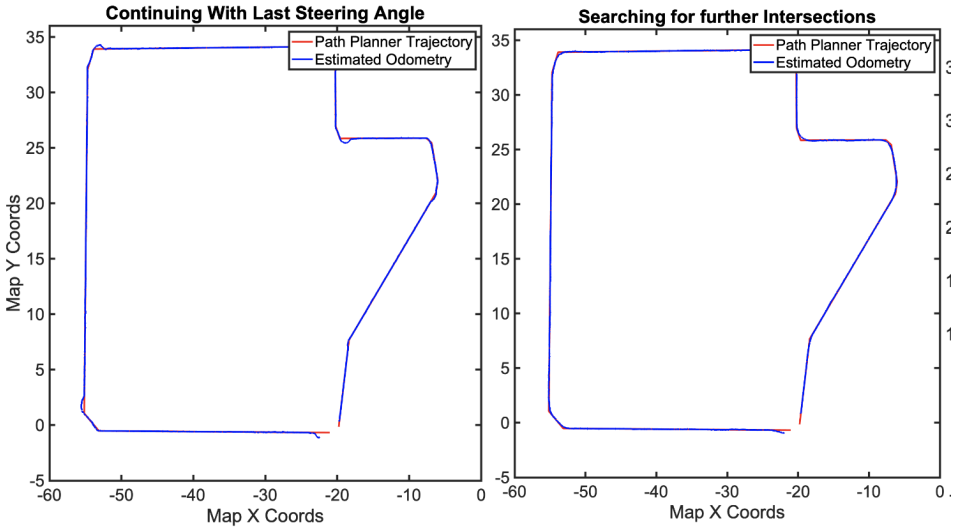

Figure 9: The left plot shows simulated pure pursuit performed on a trajectory that only searches for goal points on the nearest segment, while the right plot displays performance when subsequent segments on the trajectory are searched. The difference is greatest around corners. Searching subsequent sections performs better overall.

2.2.4 Performing Pure Pursuit

Provided a goal point, a steering angle S is calculated as the curvature resulting from driving along the circumference of a circle with a radius equal to the lookahead distance.

$$\eta = \arctan \frac{y}{x} \tag{3a}$$

$$S = \arctan \frac{2L \sin \eta}{L_r} \tag{3b}$$

where L = wheelbase, r = lookahead distance, and (y, x) is the goal point in the robot's frame.

3 Experimental Evaluation

3.1 Evaluating Tracking of our Pure Pursuit Algorithm

Authored by Henry Heiberger

To evaluate the performance of our Pure Pursuit algorithm, we first thoroughly tested the trajectory follower in isolation. We did this in simulation by having our robot follow a collection of smooth trajectories generated by the staff-provided trajectory loader. By plotting the coordinate points of the trajectory produced by the trajectory loader and plotting them along with the odometry data recorded by our robot as it attempted to follow the path, we were able to produce a visualization that displayed the error of the robot as it attempted to track the trajectory. An example of some of the visualizations produced by this method can be seen in **Figure 10**.

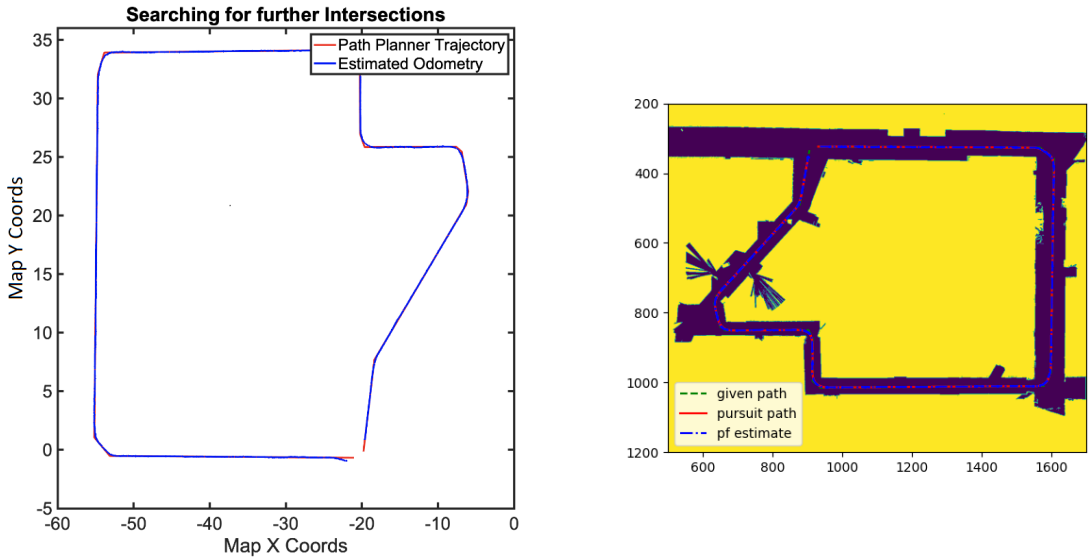

Figure 10: The left image shows a graphical depiction of the odometry of our robot as it attempted to follow the red depicted trajectory at speed 1 in simulation using our pure pursuit algorithm. The right image shows a similar test taken by the staff-provided autotester that was overlayed with the Stata basement map in which our robots were driving. The success of our pure pursuit can be seen by how, on a trajectory that follows the entirety of our robot's driving environment, our robot was able to tightly follow the path.

Looking at the provided figures, it is clear that our robot was able to adhere very well to the provided trajectory while following at speed 1. In fact, in all of our tested trajectories, our robot always maintained an error measurement near 0 m except when it was following a sudden curve. In the cases where a sudden curve did take place, the displacement of our robot from the provided trajectory never exceed 0.5 m and was able to re-converge on the trajectory almost immediately following the curve with no visible oscillation once tuned to the proper lookahead distance to fit the speed of the robot.

Once satisfied with the performance of our trajectory follower on the paths generated by the trajectory loader, we then performed the same tests and measurements as above using the paths generated by our search-based path planner. As mentioned earlier in our technical summary, we had already noticed that, even though our A* search path planning algorithm did not take driving feasibility into account, most of the trajectories produced by our path planner contained fairly mild turns, making the paths similarly navigable by our robot as the ones generated by the trajectory loader. This can be seen in **Figure 11**, with our robot still being able to tightly track the trajectory generated by our path planner. In fact, all

deviations our robot had from the planned trajectory occurred at a scale small enough that it would not substantially affect its position or course on the racetrack, showcasing that our pursuit goals had been achieved.

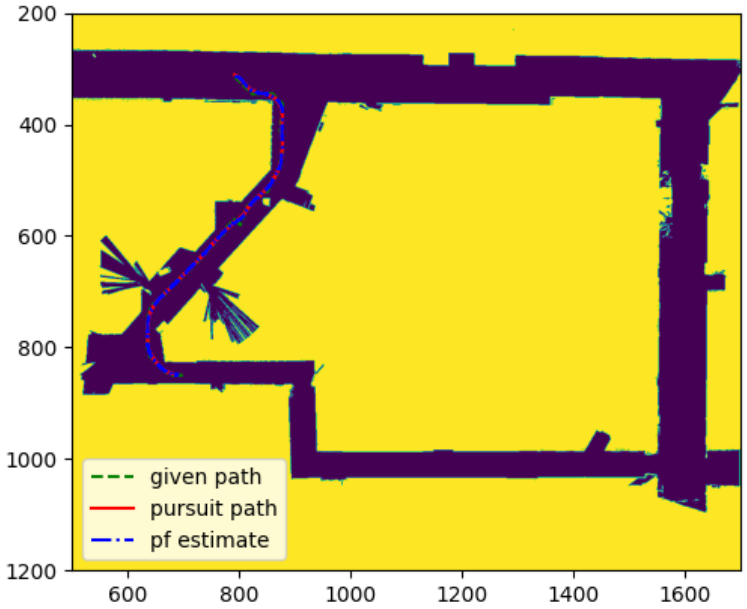

Figure 11: The figure shows a graphical depiction of the odometry of our robot as it attempted to follow the red depicted trajectory generated by our search-based path planning algorithm in simulation using our pure pursuit algorithm. The success of our pure pursuit can be further seen by how, even on a trajectory generated by a path planner which doesn't as strongly consider driving feasibility, our robot is still able to tightly adhere to the provided path.

3.2 Evaluating on the Physical Racetrack

Authored by Caden Moore and Alazar Lemma

Our real-world evaluation consisted of testing our car's ability to generate a feasible trajectory and to test the car's ability to follow this trajectory. We did this by running the robot's path planning code on similar starting and ending points with the same localization with varying lookahead distances and varying speeds. After confirming that the robot's path planning trajectory was feasible, we would run the car on the trajectory and record the physical behavior of the car along with the robot's perceived location and behavior on the computer and compared the two after the fact.

While testing on the racetrack, one of the things we noticed was that small lookahead distances resulted in an increase in oscillations with better tracking. Large lookahead distances, conversely, reduce oscillations but result in worse tracking. The trade off is especially apparent when the robot is attempting to track a curve in a trajectory. In physical terms, a short lookahead distance will enable the robot to track the straight line segments comprising the curve. However, the robot must issue larger steering angles which this results in turning inefficiently. On the other hand, a larger lookahead distance improves our performance around corners in the sense that smaller magnitude steering commands are issued earlier as the robot approaches a corner. However, this introduces the possibility that the robot may issue steering commands too early and end up crashing into a corner it was attempting to avoid.

We first initially tested with lower velocities and found that lower lookahead distances caused uncontrollable oscillation, so we ruled out using the lookahead distances that were too low for the lowest speeds for higher velocities. Then, we experimented with using higher lookahead distances and found that there was an optimum for our robot. At a speed of 2, we found that a lookahead distance of 2 meters resulted in the best performance, at speed 3, we noticed that 2.5 meters was the most optimum, and for speed 4

we found that 2.5 - 3.0 meters worked best. Our evaluation consisted on analyzing how closely the robot followed the trajectory, and how robust the robot was for oscillations and tight corners.

We performed tests on the race track in the Stata basement to determine how effective our algorithms worked, and we found that our robot performed its fastest with a speed of 4 and a lookahead distance of 2.8. Our fastest time from the start line at the North side of the basement to the finish line at the South side of the basement was 23.1 seconds. Overall, we consider this a very good score because it is the fastest time we could achieve, but by only a margin. Slight adjustment to lookahead distance did not make the biggest difference, and this time just represented an optimum. The car was able to perform adequately at a high velocity while still accurately following the planned path trajectory.

The recording of our robot is at this link: `http://youtu.be/EAgwfQ6LtGs`

3.3 Evaluating Performance of Search-Based Path Planning Against Sample-Based Path Planning

Authored by Vittal Thirumalai

We used the time library to compare the runtime of path generation and pure pursuit on the search-based A* vs sample-based RRT* algorithms. The data is listed below:

Search-based approach: A*
Path generation time: 1.59878182411
Pure pursuit time: 55.4175419807

Search-based approach: RRT* (1000 nodes)
Path generation time: 0.668648958206
Pure pursuit time: 60.5581967831

The data shows us that the path generation time for A* is longer than for RRT* by 139%, while the path is more optimal since the pure pursuit runtime is shorter by 9%. As expected, the search based approach creates a more optimal path but it

2000 nodes RRT* time: 0.832479953766 Pure pursuit time: 57.7650840282

4 Conclusion

Authored by Alazar Lemma

In this lab, we implemented a fully functioning path planning algorithm, and a pure pursuit controller along with it, to allow our car to find the most optimal path without bumping into walls or other obstacles portrayed on a map. For our path planning algorithm, we looked into two broad categories, search-based and sample-based. We tested our algorithms in simulation, and then pushed them onto the car, after which we fine-tuned them so that the algorithms would work optimally in the Stata basement. During the process, we ran into difficulties, but we cruised through them, given our past experience with debugging from previous labs.

Given our path planning works reasonably well, we do not necessarily believe that we need to make improvements, but for the future, we could consider looking into another path planning algorithm. Our current path planning algorithm, the A* search algorithm, is fairly decent the way we implemented it, but investing the time into finding a different algorithm is something that we could still do given more time. This change could save computing time on the racecar, which would help us during the competition as the car can make faster decisions. With this lab complete, our group will move onto the final competition, where we will integrate the knowledge obtained from this lab, and all the previous ones, to create a racecar that can navigate any challenge during the competition.

5 Lessons Learned

5.1 Henry Heiberger

This lab was really cool in that, by the end of it, we were essentially able to have a full autonomy stack. The process of developing this also provided many learning opportunities for me.

Regarding technical lessons, this lab was another instance where I had to implement an algorithm that was fairly new to me, allowing me to gain even more experience reviewing complicated technical resources about path planning and using them to improve my own implementations. Furthermore, given how open-ended this lab was and how there were several algorithm categories that could be applied, this lab also helped reinforce the importance of carefully creating a design plan and analyzing tradeoffs before jumping into the implementation itself. This is a very important skill to have when working with software systems of increased complexity, so I am happy that I was able to deal with it during this lab.

Regarding communication lessons, this lab especially gave me a lot of practice creating custom figures to better highlight the technical information that I was trying to explain. Given that visualizations do a great job of simplifying complicated information, this skill was very helpful to practice. Furthermore, it felt like the briefing for this lab had more information to cover than in past labs. This made pacing something we really had to practice with.

5.2 Benjamin Soria

I worked on implementing the pure pursuit control for our robot. To accomplish this, I brushed up on my linear algebra and geometry. Since this section requires various calculation-intensive procedures, I began by isolating the problem into three sections: identifying the nearest line segment, identifying a goal point intersecting the line segment and the circle of radius lookahead distance centered on the robot, and performing pure pursuit. I had already implemented pure pursuit given an (x, y) position in the robot's frame in our parking control implementation, so this problem was essentially solved. To start on the first two steps, I used MATLAB to test my understanding of the geometrical relationships on the loop2 sample trajectory points. This enabled me to quickly and easily visualize the output of each operation for a single odometry estimate. This new approach gave me the confidence to proceed with a ROS implementation in simulation and resulted in a nearly perfect implementation on the first try. Though, initially, the algorithm simply issued the last drive command whenever an intersecting point on the nearest line segment was not found. Though this implementation worked, the lookahead distance was constrained by the shortest line segment we expected to track. Short lookahead distances result in good tracking but increased oscillations. Thus, I revised the approach and configured our code to search up the trajectory for an intersecting point when one did not exist on the nearest line segment. This enabled our robot to track larger lookahead distances.

5.3 Caden Moore

I really liked this lab because it basically combined all of the work we had done in previous labs and when we integrated it all together, we essentially had a fully autonomous racecar that could navigate and traverse the Stata basement all by itself. We also implemented something new which was the path planning algorithm. This path planning algorithm was new to me and I learned a lot while working on it. I learned a lot technically about how the algorithm worked and in pursuing the objective of figuring out this algorithm, I learned a lot about other path planning algorithms along the way. Also in doing this I learned more about the physical behavior of pure pursuit and lookahead distances while working with it on the actual racecar rather than in simulation.

Regarding communication lessons that I learned during this lab, I learned that I need to make my points very concise and clear for the briefings. I needed to portray such a large amount of information in such a short amount of time, and because of this I really had to have my segment down pat. Trying to communicate many ideas fully in depth down to each little bug or issue we ran into would really make our briefing very long, so I needed to find a way to cut down on the less important details and only include the details that really needed to be communicated. This was a great exercise for me in learning how to communicate effectively.

5.4 Alazar Lemma

For technical lessons, this lab was really interesting in how it combined all the lessons from previous labs together. The challenges, the debugging both on the robot and in simulation, it had it all. We spent a while debugging the car for for error in hardware

The thing that I think was really eye opening is that our group really relied on communication mediums a lot more than in previous labs. Normally, we would just ask for help in our group chat and someone available would come, figure it out together. But in this lab, communication and coordination was really the key to our success. Not only in our group, but among other groups as well when we were stuck on certain aspects of the lab. Really cool lab overall, and I am looking forward to the final challenge.

5.5 Vittal Thirumalai

For the technical component, this lab tought me a lot about RRT* and sample based algorithms, as that was the main area that I implemented. I learned about the algorithm through videos and the technical paper provided. There were some difficulties with turning the pseudo code from the paper into the actual implementation, which I learned through by eventually solving. I also found the comparison between the RRT* and A* implementations pretty interesting, as it highlighted the trade-off that often exists between runtime and accuracy.

For the communication component, I gained more experience coordinating with team members to complete certain tasks. For example, I coordinated with Caden for part of the sample based planning. I also learned to make better slides and present the slides in a cohesive and effective manner.